**[AkitaOnRails](https://www.akitaonrails.com)**

* explore por tag
* idioma
* **explore por tag**
  + [Iniciantes](https://www.akitaonrails.com/learning)
  + [Carrer](https://www.akitaonrails.com/career)
  + [Gerenciamento](https://www.akitaonrails.com/management)
  + [Filosofia](https://www.akitaonrails.com/philosophy)
  + [Ciência](https://www.akitaonrails.com/science)
  + [Startups](https://www.akitaonrails.com/startups)
* **idioma**
  + [Português](https://www.akitaonrails.com/?locale=pt-BR)
  + [English](https://www.akitaonrails.com/?locale=en)

**[Akitando] #125 - Burlando Proxies e Firewalls | Introdução a Redes Parte 5 - SSH**

**17 Agosto 2022, 10:32 h**

Ainda não é um episódio "de segurança" mas vou começar mostrando como é fácil injetar um bug de segurança tendo a melhor das intenções e depois como manipular tunelamento de SSH pra abrir buracos em firewalls e bypassar proxies. Coisas que podem ser muito úteis na vida real.

**Conteúdo**

* 00:00 - Intro
* 00:57 - todo programa é como uma função
* 02:52 - CAP 1 - fazendo um site com injection
* 06:34 - One Liners
* 08:25 - Sanitização
* 09:57 - CAP 2 - furando firewall
* 13:47 - criando um droplet na DigitalOcean
* 15:41 - reconfigurando sshd
* 18:06 - ssh remote forwarding
* 21:26 - CAP 3 - furando proxies
* 26:25 - ssh local forwarding
* 30:45 - CAP 4 - consertando a vaca
* 33:17 - conclusão
* 34:27 - bloopers

**SCRIPT**

Olá pessoal, Fabio Akita

Finalmente, agora que posso assumir que assistiram meus últimos quatro videos sobre introdução a redes e internet, posso começar a falar de coisas mais interessantes. Vocês já sabem como informação trafega na rede, como processos se ligam a portas, sockets e tudo mais. Então vamos falar rapidinho sobre alguns conceitos de segurança em rede. Só pra esclarecer, o tema de hoje não é o geral de segurança, só algumas brincadeiras que eu acho interessante saber.

A primeira parte da história vai envolver um pouco de programação web e um dos buracos de segurança mais comum. E na segunda parte vou mostrar alguns truques que dá pra fazer com SSH em alguns cenários. A idéia não é ser um video completo sobre segurança, só mencionar dois aspectos que podem ser úteis. Eu já fiz um video inicial de segurança e em episódios futuros talvez eu entre em mais detalhes. Mas por hoje são coisas que programadores iniciantes ainda nem sabem que dá pra fazer.

(...)

Pense assim, um programa é como se fosse uma função. Ele pode receber algum input seu, do usuário, processar alguma coisa dentro, e cuspir alguma resposta. Sempre são essas três grandes partes, input, processamento e output. Mesmo no design de um chip de CPU temos as três grandes partes de fetch, decode e execute. Se for um programa long lived, que persiste e fica rodando em background, como expliquei no episódio passado, é um Big Loop esperando algum input seu na forma de coisas que digita ou que clica na tela. Daí processa alguma coisa, e cospe alguma coisa na sua tela. Em resumo, programas mastigam seu input e cospem um output. Também podemos pensar em programas como alguma coisa que transforma o input em um output. Por exemplo, um Spotify transforma os cliques de mouse na interface em música, que é seu output.

Antes de internet e antes de redes, você era a única pessoa que entregava algum input pro programa que ia rodar na própria máquina. E a maioria de nós não tem interesse em digitar comandos que destruam a própria máquina. Ninguém vai digitar sudo rm -Rf / e dar a senha. Esse meme é antigo mas deve ter gente que ainda não sabe o que é isso. Sério, se você não sabe vai assistir meu episódio de Ubuntu. Tudo que você manda sua máquina fazer é responsabilidade sua saber o que vai acontecer. Se não sabe, Google tá aí pra isso.

Eu gosto de pensar nas minhas máquinas como extensões de mim. Da mesma forma que seu corpo vai estragar se só consomir um monte de refrigerante, doces e pizza, sua máquina também vai estragar se só sair dando copy e paste de comandos e rodando no terminal. Fast food é fast food. Mas o ponto que quero chegar é que partindo da premissa que quem vai executar alguma coisa sabe o que tá fazendo, os programas antigamente eram bem mais ingênuos. Até porque não tinha tanto poder de processamento pra fazer coisas consideradas supérfluas, então o código sempre era o mais simples possível.

Vamos pegar o exemplo de hello world de node.js que mostrei no episódio anterior e adicionar algumas coisas. Acho que foi no episódio de aprendizado na beira do caos que mencionei de um programinha besta que tem em Linux chamado cowsay lembram? Você instala e no terminal executa assim cowsay hello e ele cospe uma vaquinha desenhada com caracteres ASCII mais a sua mensagem. Aí nos primórdios da Web, alguém poderia pensar, "ulha, seria legal eu ter esse cowsay na web".

E por que não? Meu hello world do episódio passado é muito chato. Só imprime a mesma mensagem. Eu quero que o usuário consiga fazer a vaca dizer qualquer mensagem. Todo mundo precisa ver a carinha da vaca pra alegrar o dia. Eu quero poder digitar no navegador https://localhost:3000?message=bom+dia e ver a vaca me dar bom dia. Todo mundo precisa de um bom dia da vaca pra começar bem o dia.

A primeira coisa é conseguir pegar esse parâmetro de mensagem que vai na URL depois da interrogação. Em HTTP a URL não serve só pra passar o endereço do servidor, mas podem ir mais informações, como pares de chave e valor separado por & comercial depois de uma interrogação. Todo mundo já deve ter visto isso, especialmente se já copiou e colou links de produtos em ecommerces como a Amazon, que notoriamente tem uns links sujos pra cacete. Pra pegar esses parâmetros, primeiro vou importar a biblioteca url pra processar a URL. E lá embaixo, na função que recebe a estrutura de request, faço o parsing da URL pra me devolver um objeto de query strings.

Pronto. Com isso tenho acesso àquele parâmetro que chamei de "message" no exemplo da URL. Agora quero conseguir executar o comando cowsay que instalei no meu Linux. Pra isso todas as linguagens tem a capacidade de instanciar um novo processo filho, literalmente conseguem rodar outro programa. No caso do Node carrego a biblioteca child\_process e pego o comando exec dele. Já expliquei sobre forks, processos e coisas assim nos episódios de concorrência e paralelismo e no video de Ubuntu.

Agora vem o principal. Eu chamo a função exec com a query string "message" que tirei da URL e monto a string do comando de cowsay como fiz no terminal, usando interpolacão de strings. Toda linguagem tem o equivalente a interpolação de strings, se não sabem nem isso, estudem mais. Essa função exec vai executar o comando cowsay, e quando terminar chama essa outra função anônima que passei. Ela vai preencher pra mim o que o programa retornou, separado em objetos de erro, standard output ou stdout e standard error ou stderr.

Como é um exemplo besta, nem vou me preocupar com os casos de erro e ignorar completamente. Obviamente isso não é uma boa prática, mas é só pra simplificar neste exemplo. Quero só pegar o que o programa cuspiu no standard output e preencher a estrutura de response com ela. Pronto, agora salvo, volto pro terminal e executo o programa com o Node. Ele vai dar bind e ficar escutando na porta 3000 como antes. Podemos voltar pro navegador e digitar aquela URL que mostrei antes, localhost 3000 message igual bom dia.

E boom, olha que bonitinho, mostrou a vaquinha no meu navegador como eu queria. E era assim que muitos sites nos primórdios da Web faziam. Delegavam alguma coisa pra algum programa já instalado no seu sistema, pegavam o que cuspia no stdout, e devolvia no meio de um HTML pro usuário. No meu exemplo besta, nem me preocupei com HTML, só cuspi um texto puro mesmo. Iniciantes podem achar que acabou, mas agora que começam os problemas. Se você não viu nenhum problema no que eu fiz, sabemos que realmente ainda é um iniciante. Não se preocupe, vamos abrir seus olhos.

Vamos voltar pro terminal no Linux. Se não sabia disso, um shell como Bash ou ZSH ou todos eu acho, permite digitar múltiplos comandos numa mesma linha, um atrás do outro, basta separar por ponto e vírgula. Vamos tentar? Deixa eu digitar cowsay hello; ls -la e olha o que acontece. Ele executa o cowsay e printa na tela, mas logo em seguida executa o ls -la e printa embaixo. É um jeito de você fazer one-liners, que são programinhas que cabem numa única linha antes de dar enter. Pra scripts isso é muito útil. Todo mundo que mexe com Linux sabe disso.

Não precisa ser muito astuto pra pensar. “Hmm, será que eu consigo fazer isso do navegador?” Vamos tentar. Depois da mensagem posso digitar ponto e vírgula e algum comando. E olha só o que voltou!! O meu programa em Node executou o comando seguinte ao cowsay. E agora, eu consigo navegar por todos os arquivos do servidor e executar comandos arbitrários!!! E você pode pensar “ah, mas só dá pra listar coisas, por que isso seria um problema?”. Ok, vamos fazer outro exemplo https://localhost:3000?message=fodeu;cat ~/.ssh/id\_rsa.

Sabe o que eu fiz? Acabei de pegar a chave privada de SSH desse usuário, que é um arquivo que jamais deveria poder ser exposto fora desse diretório, por motivos de segurança. Com ela consigo invadir conta de GitHub, Amazon e tudo mais que essa chave conseguir usar. Eu posso até baixar arquivos binários, basta passar pelo comando base64 pra transformar num texto, vai aparecer no navegador, copio e colo com Notepad em algum lugar e faço decode do base64 de volta pra string. Expliquei isso no video de detecção e correção de erros. Consigo imaginar várias coisas que dá pra fazer só com isso. Se dá pra executar qualquer comando, eu já tenho controle sobre essa máquina inteira. Eu ownei essa máquina. É game over.

Esse meu programa é exatamente o que um programador amador poderia pensar em fazer: o mínimo de código pra fazer a idéia funcionar. E de fato funcionou, a vaquinha aparece como deveria. E normamente esse mínimo nunca assume que alguém teria a idéia de tentar coisas que o programa não foi desenhado pra fazer. Mas é assim que pensa todo hacker: como eu posso explorar os programas de formas que o autor não previu? E um programador minimamente mais experiente já sabe: eu deveria ter sanitizado o que veio como query string.

Hoje não é o dia de falar sobre vulnerabilidades, mas isso que acabei de fazer se chama injection, ou injeção, porque estou injetando comandos arbitrários num parâmetro que não foi devidamente tratado, ou sanitizado. Sanitizar significa considerar que tudo que vem do mundo externo de um usuário, por definição, é podre, sujo, e precisamos sanitizar, antes de usar. Sanitizar significa remover caracteres especiais, como esse ponto e vírgula, ou escapar esses caracteres pra serem tratados como string e não como comandos. Existem dezenas de bibliotecas que fazem isso.

Frameworks como o Express, que é feito em cima do Node, justamente adiciona coisas como sanitização já sabendo que a maioria dos programadores não se preocupa com isso. Mas o objetivo desse exemplo besta foi só pra demonstrar como nós fazemos programas ingênuos quando somos iniciantes, e se for só pra nós mesmos, pra usar no nosso próprio computador, tudo bem. Mas numa aplicação web que vai pra produção e vai ter gente de verdade usando, todo cuidado é pouco.

Mas o ponto é que programas podem ser usados de formas pras quais eles não foram desenhados. No final vou voltar nesse código, mas esta próxima parte é voltando pros conceitos de rede que vim explicando nos últimos episódios. Em particular a idéia de sockets e portas. Como disse, antigamente a gente era bem mais ingênuo e não parava pra pensar que existiam pessoas maliciosas ou simplesmente só curiosas. É o oposto do que se aprende em cursos e tutoriais.

Agora imagina. Povo tá empolgado com o nascimento da internet e começa a colocar programinhas ingênuos como o que eu fiz aberto em portas de servidores com endereço IP público válido, como nas universidades da época. Servidores que agora qualquer pessoa poderia acessar. E no meio deles tá esse site de vaquinha. O objetivo nem era deixar público, só queria mostrar pros colegas da faculdade, mas sem querer agora deixei um buraco de segurança enorme aberto. Por isso que nunca um código de junior pode ir pra produção sem ter passado pela revisão de um ou mais sêniors, no mínimo.

Existem várias formas de um administrador de sistemas bloquear essas tentativas de ataque mesmo sem a colaboração do autor da vaquinha. Uma delas e a mais comum é criar políticas como, pra estar na porta 80 ou 443, precisa ser um site feito por alguém de confiança da faculdade ou instituição. Sites feitos alunos ou outras pessoas, até podem rodar internamente em portas diferentes, dentro do servidor do departamento, mas não vão ser expostos na internet pública. Todo mundo pode subir sites nesse servidor hipotetico, em portas como a 3000 que eu usei ou qualquer outra como 4000 ou 8080.

O site oficial roda na porta 80 desse servidor. Todas as outras portas deveriam estar fechadas pro público. E assim nasce o conceito de firewalls. Firewall é um programa, como qualquer outro, que roda com privilégios de administrador ou root do sistema. Isso porque ele precisa interceptar todos os pacotes que chegam ou saem pela placa de rede e filtrar pra saber se podem ou não prosseguir. Sem entrar em detalhes, existem dois tipos básicos de regras, uma pra permitir coisas e outra pra rejeitar coisas, allow ou deny.

Sem nenhum firewall é como se tivesse um firewall com nenhuma regra de deny e allow asterisco, ou seja, aceita tudo. Segurança completa é deny tudo e nenhum allow, mas aí o servidor seria meio inútil. No caso do servidor dessa faculdade de exemplo, a regra poderia ser deny tudo mas allow in, ou entrada, na porta 80. Assim, eu poderia ter meu site de vaquinha interno na porta 3000, mas se alguém de fora, da internet pública, tentasse acessar a porta 3000, a regra do firewall diz que é pra rejeitar, então rejeita todos os pacotes endereçados pra 3000 e nunca chega no meu sitezinho.

Mesmo numa rede interna de faculdade ou empresa, isso ainda não é suficiente. Porque em segurança a gente suspeita de tudo e de todos, incluindo pessoas internas. De novo, não é porque eu acho que todo mundo tem más intenções, mas porque justamente as pessoas de boa intenção, que não enxergam mau nos outros, são os mais fáceis de enganar com coisas como malwares. Você nunca sabe se um aluno instalou um malware que veio por email sem querer e agora no notebook dele tem um malware pendurado numa porta qualquer e vasculhando por portas abertas.

Por isso os firewalls também bloqueiam que um notebook numa rede consiga tentar se conectar numa porta de outro notebook na mesma rede. Ele dá deny de todas as portas acima de 1024, por exemplo. Assim, mesmo se eu subir a vaquinha na porta 3000 da minha máquina, outro usuário na mesma rede não vai conseguir acessar, mesmo sabendo meu endereco IP. O firewall vai bloquear. E é assim mesmo até hoje em muito ambiente corporativo. As regras costumam ser as mais fechadas possível pra justamente evitar o acidente de alguém largar uma vaquinha dando bandeira e alguém malicioso se aproveitar disso.

Agora faz de conta, sou um aluno dessa faculdade. Eu fiz lá a vaquinha e subi no servidor na porta 3000 e sei que tá bloqueada do lado de fora. Mas eu queria porque queria poder mostrar isso pros meus amigos lá fora. Só que o administrador da rede nunca que vai deixar eu pendurar meu site publicamente do servidor da faculdade. Por acaso eu tenho uma máquina virtual de grátis que ganhei pra testar da DigitalOcean, que é um serviço de VPS ou servidores virtuais de aluguel. Eu poderia instalar lá, mas sabe? Tô com preguiça, porque já tá funcionando aqui no servidor da faculdade e só depois descobri que as portas tavam fechadas. O que eu posso fazer? Tá tudo bloqueado, mas ao mesmo tempo eu sei que consigo navegar na web. Então nem tudo tá bloqueado, no mínimo portas 80 e 443 tão abertas pra sair. Eu posso usar isso a meu favor.

De novo vou repetir, portas são só números. Por convenção 80 é pra HTTP e 22 é pra SSH. Vamos rapidinho criar uma nova máquina na DigitalOcean. Vou escolher um Ubuntu numa máquina fraquinha, no data center dos Estados Unidos. Espero terminar. E copio o endereço IP público valido que ele me dá. Depois de ter criado uma máquina numa DigitalOcean da vida, consigo logar usando o comando ssh root@ip da máquina e como não indiquei nenhuma porta, o ssh vai tentar na porta 22 padrão. Coloco minha senha, e pronto, entrei na máquina remota. Estou mostrando o endereço IP de verdade porque depois de hoje vou destruir essa máquina, então não vai mais funcionar se você tentar conectar agora.

A DigitalOcean fornece endereços válidos na internet pra máquina que eu alugo. Numa máquina baratinha ele fica mudando o endereço pra outro. Se eu quiser ter sempre o mesmo, posso pagar a mais pra reservar pra mim. Lembra minha explicação que não existem endereços IPv4 pra todo mundo? É por isso. Pra ter um válido, precisa pagar mais caro. Mas o ponto é que qualquer um na internet consegue enxergar, ou seja, rotear pacotes pra essa máquina.

Enfim, o programa em si que tá rodando no servidor pendurado na porta 22 se chama sshd ou ssh daemon. Daemon é todo programa que o sistema operacional gerencia, ele inicializa depois do boot e fica checando se ele crashear, daí tem regras se tenta carregar de novo e assim por diante. Estude sobre systemd, OpenRC ou runit, que são gerenciadores de daemons que a maioria das distros Linux roda. Mas enfim, saiba que eles que são responsáveis por garantir que programas como o SSH carreguem sozinhos após cada boot. Graças a esse programa daemon aberto e pendurado na porta 22 que do meu notebook consigo executar o programa ssh apontando pro endereço de lá e abrir uma conexão segura de terminal remoto.

Agora vai ter alguns detalhezinhos mas não é importante decorar, só acompanhar o raciocínio. O que eu vou fazer é o seguinte, de dentro dessa máquina remota, vou editar o arquivo /etc/ssh/sshd\_config e habilitar essas opções comentadas de Allow Tcp Forwarding e Gateway Ports pra yes, e lá embaixo vou mudar a porta de 22 pra 80. Viu, eu posso pendurar programas em qualquer porta e eu escolhi mudar o daemon de SSH pra porta 80. Agora salvo e uso o comando do systemd systemctl restart sshd pra reinicializar o daemon e carregar as novas configurações, agora posso sair do ssh com exit.

Se eu tentar o mesmo comando ssh pra conectar no servidor, vai falhar, porque como eu disse, vai tentar por padrão na porta 22. Só que mudei pra porta 80. E agora? Como faz? Normalmente esses programas de rede sempre tem uma opção pra manualmente escrever a porta, só colocar -p 80 no final e bingo, olha só, conectou direitinho. Agora que a brincadeira começa.

Aquela minha vaquinha tá rodando localmente no tal servidor hipotético da faculdade na porta 3000. Faz de conta que meu PC aqui de desenvolvimento é esse servidor da faculdade. Por isso vou no navegador e digito 127.0.0.1 ou localhost, que são os endereços locais da minha máquina.

Não adianta eu passar esse endereço pra ninguém, porque significa "minha máquina" como expliquei no episódio anterior. Na máquina do seu vizinho, se ele digitar a mesma coisa, não vai carregar nada porque não tem nada pendurado na porta 3000 da máquina dele. Mas eu quero que meus amigos de fora consigam ver a vaquinha. Pra isso preciso de um endereço IP público válido e que tenha portas abertas sem bloqueio de um firewall.

Pra isso vou usar um dos super poderes do SSH, o recurso chamado Remote Forwarding. No terminal do servidor da faculdade, eu digito ssh -R pra remote forwarding e agora fica complicadinho ip do servidor:7000 depois 127.0.0.1:3000. Estou falando o seguinte, toda vez que alguém tentar se conectar no endereço do meu servidorzinho na DigitalOcean, nessa porta 7000, faça forwarding, redirecione o tráfego, pra minha máquina local 127.0.0.1 porta 3000, que é onde tá rodando a vaquinha.

Daí termino com as mesmas informações de antes pro login, usuario root arroba endereço e -p pra porta 80 que é onde o sshd tá ouvindo. Eu sei, ficou um comandão enorme, mas depois revejam com calma e entendam cada parâmetro. O que esse comando diz é: conecte no meu programa de sshd na porta 80 do servidor da DigitalOcean e se pendure na porta 7000 de lá e fique escutando. Toda vez que vier alguma requisição por lá, faça forwarding, redirecione todo o trafego, pra porta 3000 da minha máquina local.

ssh

Explicando fica complicado. Então vamos mostrar na prática. Deixa eu abrir aqui meu celular que tá conectado via 4G, ou seja, ele não teria acesso ao tal servidor hipotético da faculdade. Vamos digitar o endereco IP público do servidor da Digital Ocean, válido na internet e colocar dois pontos 7000. E voilá, olha a vaquinha aqui, aparecendo em público! Como pode isso? O firewall não tava travando? O que aconteceu?

Vou repetir. No navegador do celular eu acessei um endereço IP válido da Digital Ocean. Essa máquina virtual tá vazia. É um Ubuntu recém instalado. A única modificação que eu fiz foi configurar o programa de sshd pra se pendurar na porta 80 em vez da 22 que seria padrão. Dessa forma, eu, pobre aluno da faculdade que está atrás de um firewall bem rígido que bloqueia portas como 22, consigo acessar via a porta 80, que está aberta porque a faculdade pelo menos permite as pessoas navegar na web. E pra isso, obrigatoriamente o firewall é obrigado a deixar passar saída na porta 80. Entenderam? O firewall bloqueia qualquer coisa de entrar na rede, mas precisa permitir pacotes saírem pras pessoas conseguirem navegar na Web, e pra isso no mínimo saída pela porta 80 estava aberta. Portanto, havia esse buraco que poderia ser explorado.

O firewall deixa pacotes sair, mas não deixa nada entrar de jeito nenhum, então mesmo o servidor da faculdade tendo um endereço IP valido, o firewall não deixaria ninguém se conectar na minha porta 3000. O que eu fiz com SSH é o que se chama de poking a hole, ou literalmente fazer um buraco no firewall, é assim que se fura um firewall do lado de dentro. O grande lance é que do servidor da DigitalOcean ele não consegue me achar na internet e se conectar em mim, mas eu consigo achar e me conectar nele, via a única porta aberta de saída, a 80. E uma vez aberta a conexão, daí meu servidor de fora consegue falar comigo via essa conexão que eu estabeleci com o programa SSH.

Se pra você isso é novidade, respire um segundo e reveja. O servidor da DigitalOcean bem como qualquer outra pessoa na internet não tem como se conectar no servidor porque o firewall bloqueia, mas se a conexão se inicia do lado de dentro, desse servidor, uma vez estabelecida a conexão pra fora, agora ambas as pontas conseguem se comunicar.

Esse é um truque que eu já usei inúmeras vezes ao longo dos anos. A primeira vez que aprendi a fazer isso acho que nem existia SSH. A gente usava um programinha chamado HTTP Tunnel, que faz o oposto do que acabamos de fazer, o recurso de Local Forwarding do SSH. Deixa eu dar um outro exemplo. Acontece que só fechar todas as portas mas manter a porta 80 e 443 de Web abertas, continua sendo um enorme risco de segurança, porque vai saber quem dentro da empresa que não tá entrando em site pornô, site de pirataria e uma hora esbarrando em sites com malwares. Pra evitar isso acho que até hoje tem empresas que obrigam todo mundo a usar um proxy.

Se você abrir um navegador recém instalado numa empresa mais segura, não vai conseguir navegar pra lugar nenhum. O administrador da rede da empresa vai te instruir a abrir a configuração de proxy do seu sistema operacional e colocar um endereço IP e uma porta que aponta pro programa de proxy. Isso pode ser no nível do sistema operacional ou nas configurações do navegador.

Lembra no episódio passado que fiz um exemplo de conectar num servidor web local usando telnet e digitando manualmente o cabeçalho de requisicão HTTP? Lembra? Comando GET seguido de barra e o nome da pagina html que quiser, terminando com HTTP 1.0? Um cabeçalho mais completo, que um navegador enviaria, seria parecido com isso aqui.

GET / HTTP/1.1 Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8 DNT: 1 Accept-Encoding: gzip, deflate, sdch Accept-Language: pt-BR,en;q=0.8

Ele dá bem mais detalhes pro servidor, como se apresentar dizendo que navegador que é, dizendo que línguas que aceita e coisas assim. É uma das formas de como serviços como Google Analytics conseguem saber coisas como qual navegador mais acessa seu site. Enfim. Se tiver um proxy configurado, o navegador não vai mais se conectar direto com o site que você quer, em vez disso sempre vai mandar todas as requisições pro servidor de proxy. E o pacote vai ser um pouquinho diferente. Assim.

GET https://www.mega.io/ HTTP/1.1 Host: www.mega.io Proxy-Connection: keep-alive Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8 DNT: 1 Accept-Encoding: gzip, deflate, sdch Accept-Language: pt-BR,en;q=0.8`

Olha só, no comando GET vai a URL inteira, com o domínio, uma nova linha dizendo que host estamos querendo nos conectar e uma configuração específica de Proxy. O Proxy vai receber essa requisição e modificar pra ficar igual à original sem os detalhes de proxy. E é proxy quem vai se conectar com o servidor do Mega, no caso. O site do Mega vai gerar um HTML, o pacote de response e devolver pro Proxy. Daí o proxy redireciona a resposta de volta pro meu navegador. Um proxy atua como um intermediário entre eu e os sites que quero navegar. Do ponto de vista do usuário, estou navegando normamente. Mas do ponto de vista dos sites que estou acessando, ele só enxerga o proxy como usuário.

Mais ou menos o efeito é parecido com o que eu expliquei de NAT e CGNAT no terceiro episódio da mini-serie. Internamente temos um endereço IP 192.168 que não é válido na internet. Todo pacote passa pelo roteador do provedor e é convertido num endereço válido, que é o que os sites enxergam. Mas todo mundo atrás do NAT vai ser visto com o mesmo endereço IP. Pros sites, é difícil dizer quem são os usuários só baseado nisso.

Mesma coisa com proxy, só que restrito ao tráfego de Web. Em vez de NAT os pacotes passam pelo Proxy, e os sites só enxergam um endereço IP público da empresa, sem saber com certeza os usuarios atrás. A gente rastreia usuários com outros dados, como o cookie do navegador, que tem o login de cada usuário. Outra hora explico sobre cookies, mas foi só pra dizer que em termos de conceito, NAT e Proxy tem resultados semelhantes do ponto de vista dos sites.

A vantagem disso pra empresa é óbvia. Proxies tem filtros com whitelists e blacklists. Meio como um firewall ele pode dizer o seguinte: ninguém pode navegar em nenhum site da internet, com exceção dos domínios que o administrador configurar na whitelist. Assim ele pode restringir em quais sites os funcionários podem navegar. Sites pornô e de entretenimento provavelmente estão na blacklist da empresa. E adivinha, se o administrador não tiver colocado stackoverflow.com, você se fodeu.

Mais do que isso, ele pode monitorar o conteúdo de tudo que tá passando e registrar. Isso é mais difícil hoje porque usamos HTTPS e aí ele tem mais dificuldades de vigiar o conteúdo do que você tá navegando porque vai tudo criptografado. Mas como expliquei antes, na requisição fica aberto que site você tá tentando acessar, e o Proxy consegue registrar isso. Só de restringir onde pode navegar já é um pé no saco. É compreensível, muita empresa não gosta de ver os funcionários perdendo tempo em redes sociais, então bloqueiam coisas como YouTube ou Facebook. É um pensamento super retrógrado, que era comum no começo dos anos 2000 e eu sei que tem empresas que pensam assim até hoje, infelizmente.

Pois bem. No exemplo da faculdade eu dei um cenário hipotético onde o administrador fez um firewall que bloqueia a saída de pacotes pra 100% das portas com exceção das de web. Mas na prática acho que quase ninguém faz isso. Seria muito pouco produtivo. Você acabaria bloqueando portas úteis como as acima de 49 mil até 65 mil que servem pra coisas como Zoom e outros software de reunião online ou mesmo portas como 25, 193 que são pra clientes de email. Mas as portas de web ficam fechadas e te obrigam a configurar proxy. Então dessa vez não temos portaa 80 de saída pra trabalhar.

Não vou mostrar na prática, mas a solução é simples. Com o mesmo servidor na Digital Ocean que eu tenho, posso configurar aquele daemon sshd não pra porta 80 mas sim pra uma porta como 50000. Uma porta alta que provavelmente não vai estar bloqueado nos firewalls. O remote forwarding que eu fiz, com aquela opção ssh -R, permite alguém de fora acessar um site meu rodando numa máquina privada dentro da rede. Mas o oposto também é possível. Eu posso rodar o seguinte comando, um pouco mais complicadinho: ssh -D 1337 -q -C -N root@ip do servidor de ssh -p 50000

Opções como -q é quiet mode, pra não ficar imprimindo nada no meu terminal, -Cmaiúsculo é pra comprimir o conteúdo que passar por esse programa. -N é pra não abrir uma linha de comando pro servidor. Mas o mais importante é esse -D 1337. Isso vai pendurar esse programa, fazer bind, nessa porta 1337 na minha máquina local. Poderia ser qualquer outro número de porta. O resto é a mesma coisa pra conectar no sshd apontando pra porta 50000 do servidor, o que provavelmente vai conseguir furar o firewall.

Agora, no meu sistema operacional ou direto no navegador, eu procuro a configuração de proxy de novo, mas em vez de colocar o endereço do proxy da empresa, coloco localhost e porta 1337. E pronto, agora posso navegar normalmente por qualquer site que quiser, mesmo se a empresa achar que está me bloqueando. O que fiz foi criar um túnel via SSH, que fica exporto como um serviço SOCKS5, que é o protocolo de um Proxy Web, e pendurado na porta local 1337 da minha máquina. Eu fiz um proxy pessoal, que recebe pacotes HTTP normalmente, como um servidor web, só que ele dá forward, redireciona o tráfego, pro meu servidor da Digital Ocean via esse túnel criptografado via ssh.

O serviço sshd de lá recebe os pacotes que vieram pelo túnel e navega usando a internet da Digital Ocean, que tá toda aberta. O site que eu quis navegar devolve a resposta HTTP pro servidor, e o ssd redireciona o pacote de volta pra mim pelo mesmo túnel. E assim eu burlo toda tentativa de me restringir de navegar. Posso navegar onde quiser. Mesmo a saída da porta 80 na empresa estando fechada por firewall, mesmo sendo instruído a usar o servidor de proxy da empresa, eu criei o meu próprio servidor de proxy saindo por uma porta alta que provavelmente tá aberta no firewall e passei a navegar sem restrição nenhuma. Isso é um exemplo do que se chama de tunelamento.

Mais uma curiosidade. Antes, se eu fosse num site como what is my ip, ia mostrar o endereço IP do proxy da empresa. É a mesma coisa como no caso de CGNAT de provedor. Meu computador tem um endereço privado tipo 192.168 da vida, que não é válido pra rotear na internet. Então todo mundo da rede da empresa vai ter o mesmo endereço IP público, seja por causa de NAT, seja por causa de Proxy. Agora, se navego via o túnel pro servidor da Digital Ocean, o que o site what is my ip vai ser é o endereço IP do servidor. Não só isso, se ele tentar encontrar a posição geográfica desse IP vai ver que está em Nova Iorque, porque eu escolhi montar meu servidor no data center de Nova Iorque. Como programador é importante entender isso: os dados de usuário que sua aplicação web recebe normalmente não representa o computador do usuário, porque a maioria está escondido em redes privadas de NAT ou atrás de Proxy de empresas.

De qualquer forma, com esse truque de tunelamento de SSH, significa que não só burlei as restrições da tal empresa hipotética, como agora posso assistir Netflix como se estivesse nos Estados Unidos, tendo acesso a conteúdo que não tem no Brasil. Então eu burlei também a restrição de região do Netflix. E se o que eu acabei de falar pareceu com aquelas propagandas sobre VPN, é porque tem a ver.

Esse é só um exemplo de coisas que conseguimos fazer quando sabemos um pouco sobre redes. No próximo episódio vou falar um pouquinho de VPN. Por hoje queria mostrar um pouco mais de que tipo de coisas você está perdendo quando se recusa a entender um pouco mais sobre redes. A pessoa hipotética que fez o site de vaquinha e conseguiu disponibilizar na internet com túnel de SSH eu poderia dizer que é o oposto da maioria dos programadores: um péssimo programador, que fez um site cheio de buracos de segurança, mas um bom cara de redes que soube ultrapassar as restrições de firewall e abrir buracos no bloqueio.

Mas no geral ele ganha nota negativa, porque alguém colocar um site que não entende que é inseguro, à força na internet o torna um péssimo profissional. Eu sou o tipo que gosta de quebrar as regras e sempre fiz isso, mas quando faço eu sei os riscos que estou correndo. Não seja esse cara irresponsável, aprenda de tudo um pouco e saiba onde pode ou não pode usar cada coisa. Por exemplo, lá no começo eu disse que faltou sanitizar a mensagem do usuário pra impedir ele de tentar injetar comandos arbitrários pra rodar no servidor. Como que faz isso? Antes de terminar o episódio vamos pelo menos fazer isso.

Existem dezenas de bibliotecas e frameworks como Express já vem preparado pra isso. Mas nesse exemplo besta, vamos fazer nossa própria função. Veja esta função sanitize, ela recebe uma mensagem e passar por uma expressão regular, uma regex. Se não conhecem regex, precisa estudar, é obrigatório saber. Essa expressão procura na mensagem tudo que não é letra minúscula, letra maiúscula ou números e com a função replace ele vai trocar por uma string vazia. Ou seja, coisas com o ponto e vírgula vai ser substituída por string vazia. Agora lá embaixo eu passo a mensagem que peguei da URL por essa função e só depois concateno com o comando cowsay.

Vamos no terminal, rodar essa nova versão do programa e voltar pro navegador. Chamamos http localhost interrogação mensagem com ponto e virgula e um comando como ls e olha só, ele não executa mais o comando, porque eu quebrei a sintaxe. Se alguém achar esse site no ar e tentar executar comandos, não vai mais conseguir, porque só com letras e números, não tem como concatenar outros comandos pra executar e com isso bloqueamos todas as tentativas de injeção. Essa função não é a melhor, porque ela apaga todo caracter especial, então não consigo usar pontuação nas mensagens. Isso é só um exemplo, tem jeitos melhores de fazer isso, mas foi só pra mostrar como é o jeito mais simples e mais drástico de sanitizar.

O recado é simples: tudo que um usuário manda, seja como parâmetros na URL, seja como campos num formulário, tudo deve ser sempre considerado suspeito e deve passar por sanitização. Todo usuário é mau e malicioso até prova em contrário, é assim que devemos considerar todo mundo que acessa seu site. Todo mundo que vai usar seu programa vai tentar quebrar ele. Como programador você tem que estar ciente disso. E é simples: em algum lugar no seu programa você permite o usuário enviar alguma informação? Essa informação é material radioativo, trate ele como tal.

Não tem a ver com o episódio, mas já que falei isso vale outro aviso. Tem gente que fica paranóico demais e faz programação defensiva em excesso e sai colocando tratamento de parâmetros em todas as funções que faz. E está errado. Toda função que é usada internamente não precisa sanitizar tudo toda hora. Só funções que explicitamente recebem dados vindos diretamente de um usuário que precisam disso. Cuidado pra não sair dando copy e paste de sanitização em todo lugar que não precisa e ficar redundante. Ter cuidado não significa atirar pra todo lado. Um bom programador usa sniper, mira e atira com precisão só um tiro e acerta o alvo. Quem atira pra todo lado é claramente amador.

Eu quis fazer esse episódio porque muita gente usa SSH pra se conectar em servidor mas não sabe o que o SSH realmente consegue fazer. Espero que isso tenha servido pra mostrar que você usa ferramentas sem saber pra que elas realmente servem. Interesse-se mais em explorar o que você usa no seu dia a dia. Se ficaram com dúvidas mandem nos comentários abaixo, se curtiram o video deixem um joinha, assinem o canal e não deixem de compartilhar o video com seus amigos. A gente se vê, até mais!

**tags:**

**Comentários**

**© 2006-2022 Fabio Akita. Todos os Direitos Reservados.**

[About Me](https://about.me/akitaonrails)

[YouTube](https://youtube.com/user/akitaonrails)

[Codeminer 42](http://www.codeminer42.com)