

Exporting SSH Connectivity to a Client

Export access to our SSH daemon to some client's local port
2022: `ssh -R2022:127.0.0.1:22 user@client`

Connect back through an exported port forward, while verifying the server's identity: `ssh -O HostKeyAlias=backend_host user@127.0.0.1`

It's possible to both import and export, creating a "floating bastion host" both hosts meet at.

Other Things to Do with OpenSSH

Copy a file to a remote host: `scp file user@host:/path`

Copy a file over a local port forward: `scp -o 'HostKeyAlias backend_host' -o 'Port 2022' file user@backend_host:/tmp`

Synchronize a file with a remote host (only update what's necessary): `rsync -e ssh file user@host:/path/file`

Specify SSH1 for rsync: `rsync -e "ssh -1" file user@host:/path/file`

Rsync through a HTTP Tunnel

Start HTTP Tunnel Server: `hts 10080 -F 127.0.0.1:22`

Start HTTP Tunnel Client: `htc -F 10022 -P proxy_host:8888 host:10080`

Rsync entire directory through file, with details: `rsync -v -r -e "ssh -o HostKeyAlias=host path user@127.0.0.1:/path"`

Directly burn a CD over SSH: `mkisofs -JR path/ | ssh user@burning_host "cdrecord dev=scsi_id speed=# -"`

Burn a CD over SSH after caching the data on the remote host: `mkisofs -JR path/ | ssh user@host "cat > /tmp/burn.iso && cdrecord dev=scsi_id speed=# /tmp/burn.iso && rm /tmp/burn.iso"`

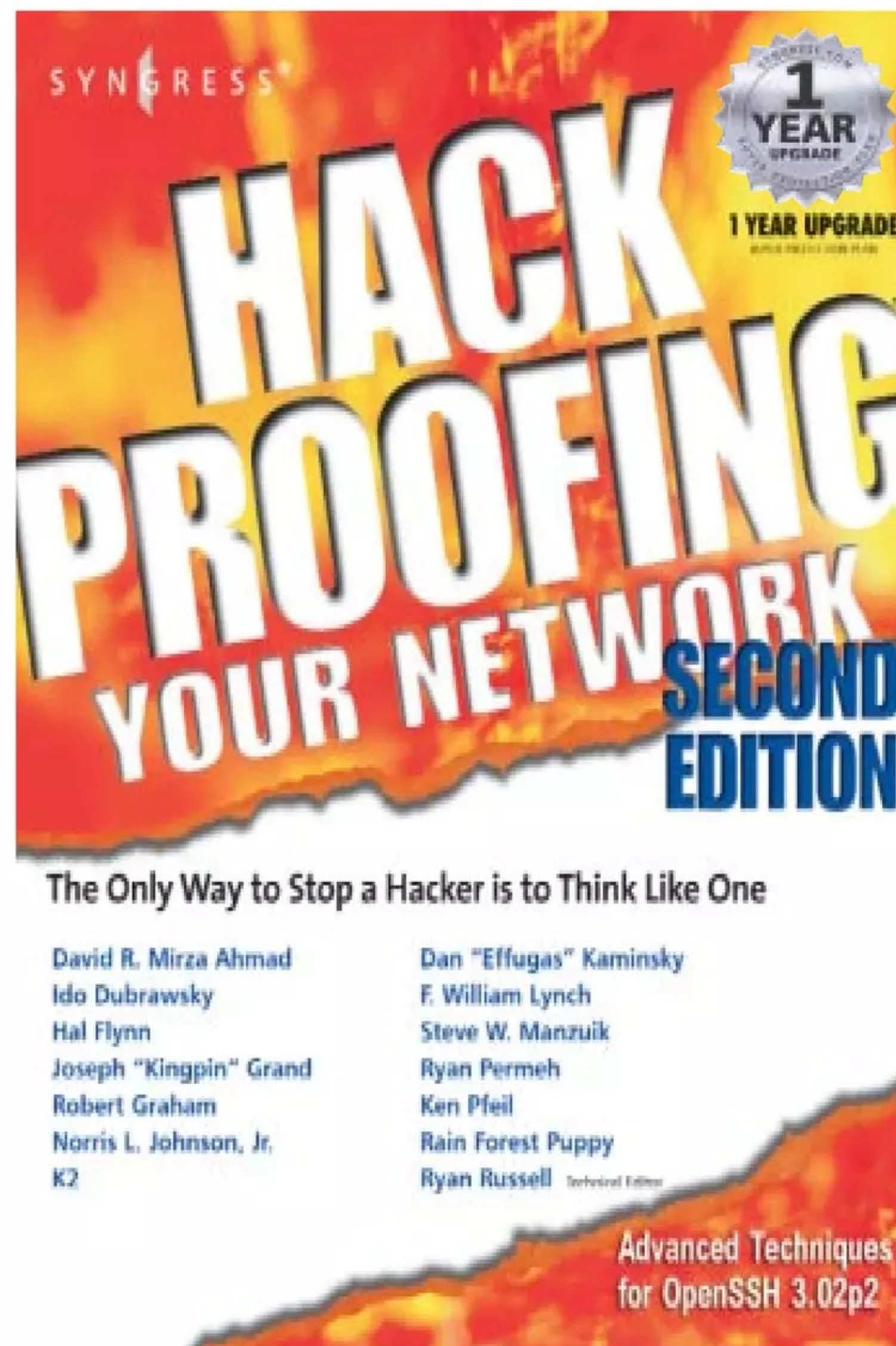
Forward all MP3 data sent to localhost:18001 to an mp3 decoder on a remote server: `ssh -L18001:127.0.0.1:18001 effugas@10.0.1.11 "nc -l -p 18001 -e ./plaympg.sh" (plaympg.sh contents: #!/bin/sh -c 'echo OK; exec mpg123 -')`

SYNGRESS®

Visit www.syngress.com/hackproofing for great prices on all our Hack Proofing titles

781.681.5151 Fax 781.681.3585 www.syngress.com

GIG-5295_4/02_5m



Basic SSH

Connect to host as user: `ssh user@host`

Connect to host as user, alternate port: `ssh -p port user@host`

OpenSSH Public Key Authentication

Generate SSH1 / SSH2 keypair: `ssh-keygen / ssh-keygen -t dsa`

Cause remote host to accept SSH1 keypair in lieu of password: `cat ~/.ssh/identity.pub | ssh -1 effugas@10.0.1.10 "cd ~ && umask 077 && mkdir -p .ssh && cat >> ~/.ssh/authorized_keys"`

Cause remote host to accept SSH2 keypair in lieu of password: `cat ~/.ssh/id_dsa.pub | ssh effugas@10.0.1.10 "cd ~ && umask 077 && mkdir -p .ssh && cat >> ~/.ssh/authorized_keys2"`

Add passphrase to SSH1 / SSH2 key: `ssh-keygen.exe -p / ssh-keygen.exe -d -p`

Start SSH key agent (prevents you from having to type passphrase each time): `ssh-agent bash`

Add SSH1 / SSH2 key to agent: `ssh-add / ssh-add ~/.ssh/id_dsa`

OpenSSH Command Forwarding

Execute command remotely: `ssh user@host command`

Pipe output from remote command into local command: `ssh user@host "remote_command" | "local_command"`

Get File: `ssh user@host "cat file" > file`

Put File: `cat file | ssh user@host "cat > file"`

List Directory: `ssh user@host ls /path`

Get Many Files: `ssh user@host "tar cf - /path" | tar -xf -`

Put Many Files: `tar -cf - /path | ssh user@host "tar -xf -"`

Resume a download: `ssh user@host "tail -c remote_filesize -local_filesize file" >> file`

Resume an upload: `tail -c local_filesize-remote_filesize file >> file`

Safely switch users: `ssh user@host -t "/bin/su -l user2"`

OpenSSH Port Forwarding

Forward local port 6667 to some random host's port 6667 as accessed through an SSH daemon: `ssh user@host -L6667:remotely_visible_host:6667`

Dynamically forward local port 1080 to some application specified host and port, accessed through an SSH daemon: `ssh user@host -D1080`

Forward remote port 5900 to some random host's port 5900 as accessible by our own SSH client: `ssh user@host -R5900:locally_visible_host:5900`

Using OpenSSH ProxyCommands

Basic Usage: `ssh -o ProxyCommand="command" user@port`

Use netcat instead of internal TCP socket to connect to remote host: `ssh -o ProxyCommand="nc %h %p" user@host`

Use Goto's connect.c to route through SOCKS4 daemon on proxy_host:20080 to connect to remote host: `ssh -o ProxyCommand="connect -4 -S proxy_user@proxy:20080 %h %p" user@host`

Use Goto's connect.c to route through SOCKS5 daemon on proxy_host:20080 to connect to remote host: `ssh -o ProxyCommand="connect -5 -S proxy_user@proxy:20080 %h %p" user@host`

Use Goto's connect.c to route through HTTP daemon on proxy_host:20080 to connect to remote host: `ssh -o ProxyCommand="connect -H proxy_user@proxy:20080 %h %p" user@host`

Using HTTP Tunnel with OpenSSH

Forward HTTP traffic from local port 10080 to the SSH daemon on local-host: `hts 10080 -F 127.0.0.1:22`

Listen for SSH traffic on port 10022, translate it into HTTP-friendly packets and throw it through the proxy on proxy_host:8888, and have it delivered to the httptunnel server on host:10080: `htc -F 10022 -P proxy_host:8888 host:10080`

Send traffic to localhost port 10022, but make sure we verify our eventual forwarding to the final host: `ssh -o HostKeyAlias=host -o Port=10022 user@127.0.0.1`

Importing Access from a Bastion Host

Set up a local forward to an SSH daemon accessible through a bastion host: `ssh L2022:backend_host:22 user@bastion`

Independently connect to the SSH daemon made accessible above: `ssh -o HostKeyAlias=backend_host -p 2022 root@127.0.0.1`

Set up a dynamic forwarder to access the network visible behind some bastion host: `ssh -D1080 user@bastion`

Connect to some SSH daemon visible to the bastion host connected to above: `ssh -o ProxyCommand="connect -4 -S 127.0.0.1:1080 %h %p" user@backend_host`

Set up no advance forwarder; directly issue a command to the bastion host to link you with some backend host: `ssh -o ProxyCommand="ssh user@bastion nc %h %p" user@backend_host`

(continued on back)