Zach Grace

Musings on security, mostly offensive.

twitter github

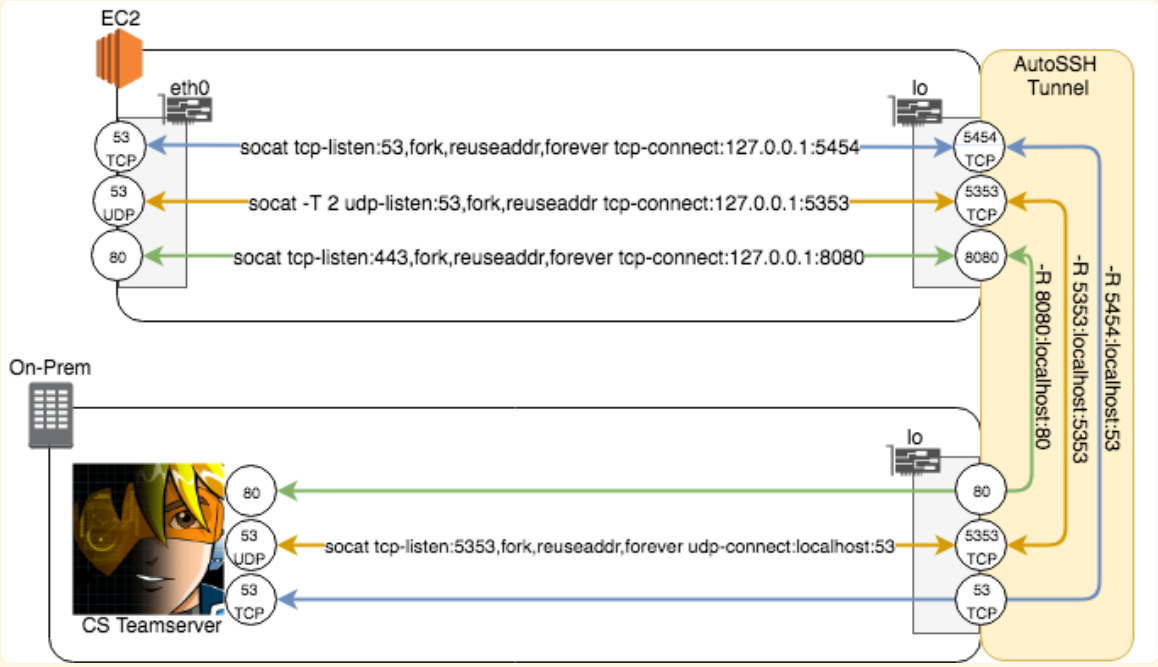
Hybrid Cobalt Strike Redirectors

2018-02-20

Working for an organization with a strict data security policy puts a few challenges on a Red Team, especially when it comes to building robust infrastructure. m0ther_ and I set out to build a robust, multi-redirector infrastructure similar to what Raphael Mudge described in his blog post, Cloud-based Redirectors for Distributed Hacking, except we wanted to host the team server on-prem. The post below describes two iterations of infrastructure we built to meet our needs.

Iteration One - socat & Autossh

Our first iteration was quick and dirty, and worked…until it didn't. We decided to use a combo of SSH remote forwarding and socat relays to shovel the traffic around.



Moving TCP data around in this fashion is relatively straight forward. We used autossh to automatically log into the redirector servers using remote port forwarding commands, `autossh -M 20000 -o "ServerAliveInterval 15" -o "ServerAliveCount 4" -R 5454:localhost:5454 user@redirector0`. We then configured socat to listen on a specified port and relay the traffic back to the remote forwarded port using a command like, `socat tcp-listen:53,fork,reuseaddr,forever tcp-connect:127.0.0.1:5454`. **Update:** It turns out the use of socat for the TCP redirectors is unnecessary (we didn't figure this out until after we implemented iteration two). Configuring the redirector's sshd_config `GatewayPorts` to either `yes` or `clientspecified` along with a remote forward directive like `-R 0.0.0.0:5353:localhost:5353` will make the remote port forward listen on all interfaces.

If you're familiar with Cobalt Strike, much of it's power is in the DNS Beacon payloads which means we needed to move around UDP data too. The challenge is that SSH remote port forwards can only forward TCP traffic. Luckily for us, socat can natively convert UDP traffic to TCP by specifying the appropriate parameters. On the redirector, we used `socat -T 2 udp-listen:53,fork,reuseaddr tcp-connect:127.0.0.1:5353` to start a UDP listener for the beacon to connect to and forwarded that traffic to the TCP `127.0.0.1:5353` SSH remote port forward. Once that traffic has passed through the tunnel, it needs to get converted back to UDP using `socat tcp-listen:5353,fork,reuseaddr,forever udp-connect:localhost:53`.

Tip: The `-T 2` in the redirector listener terminates connections after a specified interval. Unless you like hundreds or thousands of forked socat listeners waiting around for doomsday, use it to clean up your UDP listeners.

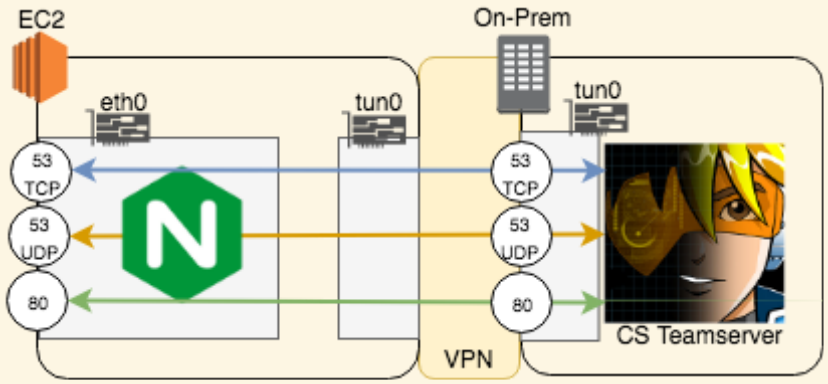
Drawbacks

The biggest drawback was debugging where connection failures occurred. The setup ran fine for a few weeks, but eventually the connections would get hosed. Because of all of the packet shenanigans we were doing, it tended to be time-consuming to determine where the real failures occurred. We typically resorted to turning it off and on again to resolve the issues (restarting supervisord on the redirectors which managed the socat relays and autossh on the team server).

Iteration Two - Nginx & OpenVPN

After enough messing around with the original architecture, we went back to the whiteboard and came up with a new plan. We decided to use Nginx's TCP and UDP load balancing to accept and forward the incoming beacon communication. And to simplfy the traffic forwarding, we decided to place the redirectors and

team server on the same subnet using a VPN. We made the primary redirector an OpenVPN server and all redirectors became clients. The team server was also a VPN client, but was given a static address.



To accomplish the TCP and UDP forwarding, we created a stream config in `/etc/nginx/tcpconf.d/passthrough`:

```
stream {
    upstream ssl {
        server 192.168.99.5:443;
    }

    upstream http {
        server 192.168.99.5:80;
    }

    upstream dns {
        server 192.168.99.5:53;
    }

    server {
        listen 443;
        proxy_pass ssl;
    }
```

```
    server {
        listen 80;
        proxy_pass http;
    }

    server {
        listen 53;
        proxy_pass dns;
    }

    server {
        listen 53 udp;
        proxy_pass dns;
    }

}
```

From there we simply set up the initial redirector server as an OpenVPN server. I won't cover the OpenVPN setup as there are many good tutorials available. The only post-configuration task for the OpenVPN server is to set up the team server with a static IP address. We created a `client-config-dir` and added a simple config for the team server, `ifconfig-push 192.168.99.5 255.255.255.0`, to assign it a static ip address.

Drawbacks

The primary drawback to this method is that in the Cobalt Strike UI, the beacon address shows up as the VPN address of the redirector that received the beacon communication. But you can stitch this info back together using the Nginx logs.

References

- Cobalt Strike Cloud-Based Redirectors https://blog.cobaltstrike.com/2014/01/14/cloud-based-redirectors-for-distributed-hacking/
- Nginx - TCP and UDP Load Balancing - https://www.nginx.com/resources/admin-guide/tcp-load-balancing/
- How To Set Up An OpenVPN Server On Ubuntu 16.04 - https://www.digitalocean.com/community/tutorials/how-to-set-up-an-openvpn-server-on-ubuntu-16-04

Tags: cobaltstrike, redirector, nginx, openvpn, socat, supervisord

If you'd like to support this blog, please consider donating a few CPU cycles by clicking Start below and see this link for more info: Coinhive Authedmine. Cheers!

HASHES/S	TOTAL	THREADS	SPEED
0	0	2 + / −	100% + / −

START MINING

powered by coinhive