Forward Pinhole Requirements and Options

Forwards an inbound packet sent to a configurable port on the WAN interface of the FT to an *internal address and port*.

This could be impl. as a network address translation (NAT) function mapping between a FT's *{external address, external port}* tuple and a specified *{internal address, internal port}* tuple.

As an alternative, TCP and UDP support could be implemented as a proxy socket that accepts connections on a FT's WAN interface:port and connects to a specified *internal address, internal port.*

Options, for each option letter we must choose at least one # to provide a working impl:

A) packet IP source validation

1. forward all inbound packets regardless of source IP
   or
2. only forward inbound packets that match a specific IP address/mask or range

B) NAT of inbound packet source IP

1. none (straight port forward does not hide source IP) – this is a more pure form
   or
2. change the source IP to the FT's LAN IP (essentially a dual NAT translation on the packet)
   - N/A to non-router AP's

C) pinhole duration

1. mission
   or
2. mission start + timeout
   or
3. window

- connection timeout (device/linux/impl. Specific keep alive workarounds?)

D) kernel (netfilter based) or application layer pinhole impl

1. netfilter based impl.
   Pros
   impl. could be as simple as applying current proxy code to inbound connections

   Cons
   ?
2. Application layer impl:
   Pros
   - Could be as simple as mission cmd exec support. e.g. exec netcat –args
   -simple impl. leads to control of alternative tools that may be available on some FTs

Cons
- not as reliable and as good of performance as NAT translation (marginal considering use cases)

- may be easier to provide an application layer impl on some devices
- even application layer impl. still requires firewall rule checking/override
- Note: we must override existing firewall config specified by a user on the FT-- which means that we'd have to either modify IP tables periodically or place our hooks in before iptables can filter out/drop the packets.

E) Support to forward to FT LAN IP or localhost

1. Required
2. If it works for a FT without effort, great.
3. No support to forward to FT LAN IP or FT localhost

Assumptions:

- no validation or security on pinhole
  i.e. a packet from ANY IP would be forwarded through the firewall (option A-1)
  or
  the 'right' IP (option A-2)
- must be able to configure a FT's firewall to open up the hole without interfering with the FT's normal firewall configuration.

Reverse Pinhole

Note: We already have a form of reverse pinhole, but is for all outbound traffic for a specific IP. I don't think that it will be difficult to perform a NAT translation from a outbound *destination IP*, that corresponds to one of the following options, to a translated outside IP and port. Where the *destination IP is:*

1. the FT's LAN IP
   or
2. an arbitrary IP/network mask
   or
3. a domain, (requires looking into DNS requests).

Hole punching:

Technique of establishing and outbound TCP connection or UDP packet (or UDP handshake) in order to open up a hole in Firewalls between a non public addressable FT and an outside destination IP. This is useful in order to

Pinhole Estimate of Work

This estimate is for kernel/netfilter based impl (Option D-1) on FT's that already support HTTP

Proxy (i.e. outbound NAT dest IP translation)

Firewall Manipulation:

We will have to open up the firewall to allow for inbound or outbound pinhole (packet with original IP destination must be allowed, regardless of user configured firewall).
As simple as programatically (preferred) or via an exec on the cmd line, adding an accept rule to every IP table. This estimate is for support on one device FT, FT's with a similar kernel would also likely be supported with little or no effort.

netfilter firewall configuration:

1-2 weeks

static/hardwired pinhole test for forward and reverse pinhole assuming options A-1, B-1, C-1:

2-5 days

UDP support (depends on previous TCP test working)

1 day

mission protocol configuration/support:

1-2 days

CW support:

target action/activated pinhole: + 5-10 days
static mission FW pinhole: 2-4 days if added to generic MissionProperties page

Note: sponsor has not requested Reverse Pinhole (RP)
domain-> IP pinholing / redirection (option RP-3): + 3-5 days
global/mission (option RP-1,2) : +2 days

Testing:

1 week minimum

Windex Connection Negotiation over HTTPS

Main Requirement: application layer proxy app (hereto referred as '**wxpx**') that servers as a HTTP connection proxy and can hand off a connection to windex. See white board for pseudo code impl. stages.

Significant Development tasks:

1. HTTP request and response inspection and domain lookup (already have kernel level functions for HTTP request and response inspection)
   or

   Including destination IP (port optional) in packet before sending over local interface or over NETLINK. If forwarding multiple connections from **gf**, local interface is preferred.

2. SSL support (many libs available, we need to choose a small footprint one that is portable to most Fts)

- use libtomnet ... not mature enough...
- netcat SSL patch... good example of patch to netcat that adds SSL support:

    patch_netcat_ssl-20040224.diff, based on OpenSSL support

        #include <openssl/err.h>

        #include <openssl/x509v3.h>

-see http://xyssl.org/docs/ for alternative SSL impl.s, e.g.: peersec 50K

3. **wxpx** app communication IO with **gf**:

   - **wxpx** notify ready (after starting from signal to mm from **gf** after target acquisition)

   - **wxpx** notify success/failure (we could restrict **gf** to only forward one connection per client until success)

–

Transparent Proxy / Route

Proxy or route all traffic between a client to various destination IP(s) through an alternative proxy server or tunneled gateway.

Assumption: Traffic should not be encrypted, it will only raise a red flag.

General Issues:

- **Authentication**

  The FT firmware already ships with a auth protocol that is encrypted and can be used as the basis for authenticating with a proxy server.

  Note: rather than re-use the same pre-shared key as with mm crypto, we should just deploy another key for proxy authentication and encryption.

  - CherryTree based auth and handoff
  Pros: no additional crypto lib required to hide pre shared key or credentials.
  CT protocol could determine the client proxy type and select an appropriate server impl.
  CB Server could host a local openvpn or socat server or 'pipe' the connection to another server.
  means of quickly adding support for proxy with minimal overhead to Frontend.

  Cons: VPN client software on the FT must be customized to integrate with an initial handshake prior to after establishing a TCP tunnel. i.e. Customization of socat or openvpn client app. Susceptible to replay attack.

  - ssl certificate auth
  Pros: socat and openvpn already support ssl integration
  Cons: requires a significant extra space on FT firmware
  still requires a means of authentication as part of establishing the ssl connection
  - pre deploy server or root certificate so that the client can avoid a MITM attack
  - deploy proxy IP, certificate or credentials in the mission

- **DNS request handling**

  If a client's traffic is routed through a proxy server, then the client's nameserver(s) may not be accessible.

  Potential Solutions:

- Do not route any DNS traffic.
Pros: simple and best solution
Cons: more complex target based IP routing rules and manipulations...may not be a factor depending on how we inevitability manipulate, or short-circuit, routing rules.

- Map or translate all outbound DNS request dest IPs to the proxy server's nameserver.
Pros: allows for remote manipulation of DNS resolution
Cons: if the target is attempting to resolve an address that is only bound in the FT's default nameserver, then the proxy server will fail to resovlve the address. e.g. hotel dns redirection of first lookup to authentication or waiver EULA page.
e.g.

```
iptables -t nat -I PREROUTING 1 -p udp --destination-port 53 -j DNAT --to 4.2.2.1
```

- **DHCP lease renewal**

  If a client is using a DHCP lease, then it's request to renew the lease traffic should not be proxied or routed.

  I think this case would be rare, since in most use cases the FT itself would be acting as a DHCP server and not another server on or outside the LAN.

  From openvpn FAQ:

  > "Many OpenVPN client machines connecting to the internet will periodically interact with a DHCP server to renew their IP address leases. The **redirect-gateway** option might prevent the client from reaching the local DHCP server (because DHCP messages would be routed over the VPN), causing it to lose its IP address lease."

A) Traffic Types to Proxy

1. All protocols, excluding the following:
   TCP established connections
   ICMP inbound (TTL traceroute replies)
   Traffic from a target to an outbound Private IP address
   http://tools.ietf.org/html/rfc1918
   ```
   10.0.0.0        -   10.255.255.255  (10/8 prefix)
   172.16.0.0      -   172.31.255.255  (172.16/12 prefix)
   192.168.0.0     -   192.168.255.255 (192.168/16 prefix)
   ```

2. Mission configurable protocols and ports

B) Proxy/Route Transport

The choices between which proxy/VPN/traffic encapsulation strategy to use is a primarily a trade off between
a) more reliable/dynamic VPN IP protocols and implementations
b) firewall negotiation.
e.g. A TCP tunnel could hang or be lost, and then all traffic from the client is blocked or dropped until the tunnel is re-established vs. a firewall only allowing TCP port 80 traffic.

Assumption:
Avoiding firewall show stopping issues is more important than tunnel reliability for our sponsor.

1. Application Layer/User Space Tunnel

   Pro: an unencrypted TCP port 80 tunnel shouldn't raise too many flags, and avoid many VPN FW issues between the FT and the proxy server.

   Options: TCP is not the sole tunnel transport option among application layer/user space tunnel applications, it is merely the best option for getting through a firewall without manual testing or punching a pinhole.

2. Multi channel / Non TCP based VPN tunnels

   Con: VPN kernel support likely limited on some FTs, may require a significant amount of image space.

   IPSEC – requires pre-shared key or cert, or radius server auth

   PPTP – sends regular PPP session with GRE, requires two network sessions
   "The system uses TCP (i.e., port 1723) to send the PPTP control channel packets. On the data channel, PPTP uses a protocol called Generic Routing Encapsulation (GRE—IP protocol number 47) to securely encapsulate the Point-to-Point Protocol (PPP) packets in an IP packet."