                   Classical versus Transparent IP Proxies

Status of this Memo

   This memo provides information for the Internet community.  This memo
   does not specify an Internet standard of any kind.  Distribution of
   this memo is unlimited.

Abstract

   Many modern IP security systems (also called "firewalls" in the
   trade) make use of proxy technology to achieve access control.  This
   document explains "classical" and "transparent" proxy techniques and
   attempts to provide rules to help determine when each proxy system
   may be used without causing problems.

Table of Contents

1. Background

   An increasing number of organizations use IP security systems to
   provide specific access control when crossing network security
   perimeters. These systems are often deployed at the network boundary
   between two organizations (which may be part of the same "official"
   entity), or between an organization's network and a large public
   internetwork such as the Internet.

   Some people believe that IP firewalls will become commodity products.
   Others believe that the introduction of IPv6 and of its improved
   security capabilities will gradually make firewalls look like stopgap
   solutions, and therefore irrelevant to the computer networking scene.
   In any case, it is currently important to examine the impact of
   inserting (and removing) a firewall at a network boundary, and to
   verify whether specific types of firewall technologies may have
   different effects on typical small and large IP networks.

   Current firewall designs usually rely on packet filtering, proxy
   technology, or a combination of both. Packet filtering (although hard
   to configure correctly in a security sense) is now a well documented
   technology whose strengths and weaknesses are reasonably understood.
   Proxy technology, on the other hand, has been deployed a lot but
   studied little. Furthermore, many recent firewall products support a
   capability called "transparent proxying". This type of feature has
   been subject to much more marketing attention than actual technical
   analysis by the networking community.

   It must be remembered that the Internet's growth and success is
   strongly related to its "open" nature. An Internet which would have
   been segmented from the start with firewalls, packet filters, and
   proxies may not have become what it is today. This type of discussion
   is, however, outside the scope of this document, which just attempts
   to provide an understandable description of what are network proxies,
   and of what are the differences, strengths, and weaknesses of
   "classical" and "transparent" network proxies.  Within the context of
   this document, a "classical" proxy is the older (some would say old-
   fashioned) type of proxy of the two.

   Also note that in this document, the word "connection" is used for an
   application session that uses TCP, while the word "session" refers to
   an application dialog that may use UDP or TCP.

2. Direct communication (without a proxy)

   In the "normal" Internet world, systems do not use proxies and simply
   use normal TCP/IP to communicate with each other. It is important
   (for readers who may not be familiar with this) to take a quick look
   at the operations involved, in order to better understand what is the
   exact use of a proxy.

   2.1 Direct connection example

      Let's take a familiar network session and describe some details of
      its operation. We will look at what happens when a user on a
      client system "c.dmn1.com" sets up an FTP connection to the server
      system "s.dmn2.com". The client system's IP address is
      c1.c2.c3.c4, the server's IP address is s1.s2.s3.s4.
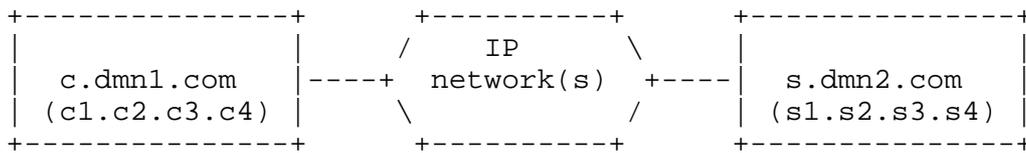
```
      +---------------+       +----------+      +---------------+
      |               |       |  /  IP  \ |      |               |
      |   c.dmn1.com  |----+  network(s)  +----|   s.dmn2.com  |
      | (c1.c2.c3.c4) |       \          /      | (s1.s2.s3.s4) |
      +---------------+       +----------+      +---------------+
```

      The user starts an instance of an FTP client program on the client
      system "c.dmn1.com", and specifies that the target system is
      "s.dmn2.com". On command-line systems, the user typically types:

          ftp s.dmn2.com

      The client system needs to convert the server's name to an IP
      address (if the user directly specified the server by address,
      this step is not needed).

      Converting the server name to an IP address requires work to be
      performed which ranges between two extremes:

       a) the client system has this name in its hosts file, or has
          local DNS caching capability and successfully retrieves the
          name of the server system in its cache. No network activity
          is performed to convert the name to an IP address.

       b) the client system, in combination with DNS name servers,
          generate DNS queries that eventually propagate close to the
          root of the DNS tree and back down the server's DNS branch.
          Eventually, a DNS server which is authoritative for the
          server system's domain is queried and returns the IP
          address associated with "s.dmn2.com" (depending on the case,
          it may return this to the client system directly or to an

intermediate name server). Ultimately, the client system
obtains a valid IP address for s.dmn2.com. For simplicity,
we assume the server has only one IP address.
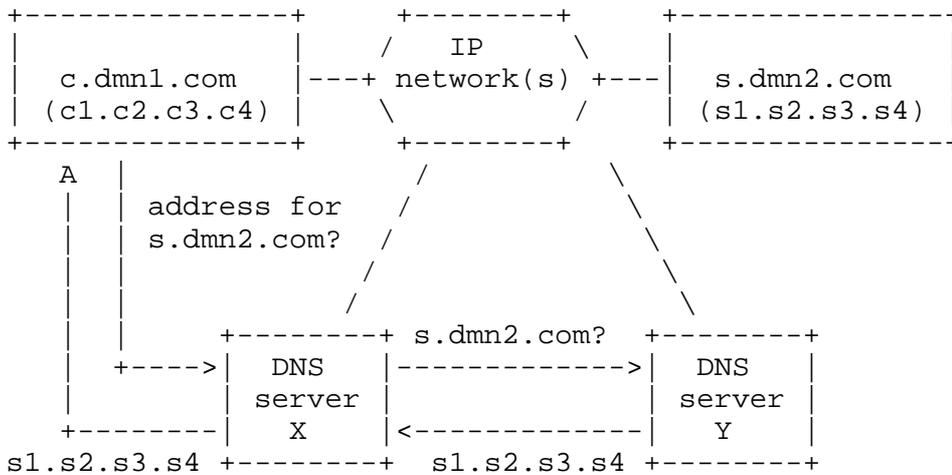
```
  +---------------+       +--------+       +---------------+
  |               |       /   IP    \      |               |
  |  c.dmn1.com   |---+ network(s) +---|   s.dmn2.com   |
  | (c1.c2.c3.c4) |       \        /      | (s1.s2.s3.s4) |
  +---------------+       +--------+       +---------------+
    A  |                      /            \
    |  | address for         /              \
    |  | s.dmn2.com?        /                \
    |  |                   /                  \
    |  |                  /                    \
    |  |     +--------+  s.dmn2.com?  +--------+
    |  +---->|  DNS   |-------------->|  DNS   |
    |        | server |               | server |
    +--------|   X    |<--------------|   Y    |
  s1.s2.s3.s4 +--------+  s1.s2.s3.s4 +--------+
```

Once the client system knows the IP address of the server system,
it attempts to establish a connection to the standard FTP
"control" TCP port on the server (port 21). For this to work, the
client system must have a valid route to the server's IP address,
and the server system must have a valid route to the client's IP
address. All intermediate devices that behave like IP gateways
must have valid routes for both the client and the server. If
these devices perform packet filtering, they must ALL allow the
specific type of traffic required between C and S for this
specific application.

```
  +---------------+                        +---------------+
  |  c.dmn1.com   |                        |  s.dmn2.com   |
  | (c1.c2.c3.c4) |                        | (s1.s2.s3.s4) |
  +---------------+                        +---------------+
    | |                                      |  |
    | | route to S              route to C  |  |
    | V                              V       |
    |                                        |
    | A                                      | A
    | | route to C                          | | route to S
    | |                                      | |
    | |       C          S         C        | |
  +----+  <-- +----+ -->      +----+  <-- +----+
  | G1 |--------| Gx |--------| Gy |---------| Gn |
  +----+ -->    +----+   <-- +----+ -->      +----+
       S                C          S
```
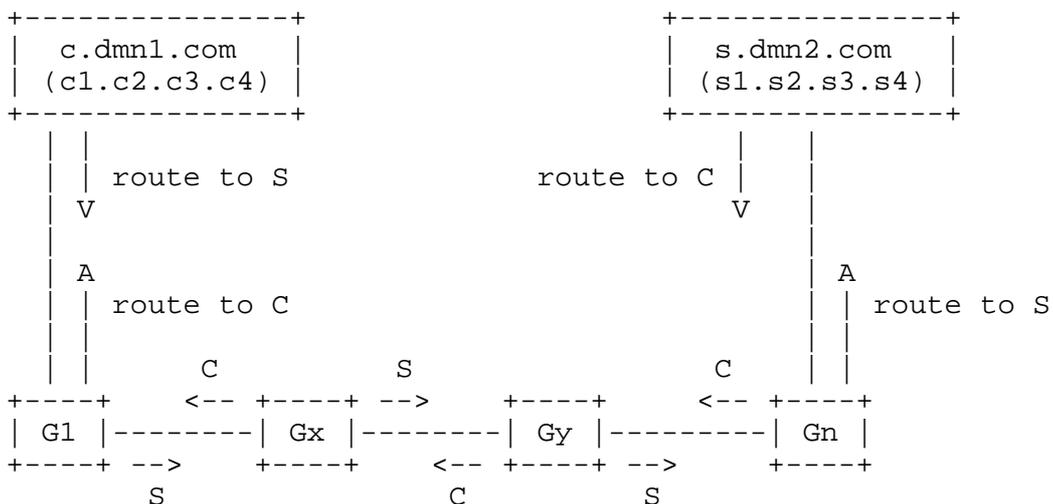
The actual application work for the FTP session between the client
and server is done with a bidirectional flow of TCP packets
between the client's and server's IP addresses.

The FTP protocol uses a slightly complex protocol and TCP
connection model which is, luckily, not important to the present
discussion. This allows slightly shortening this document...

2.2 Requirements of direct communication

Based on the preceding discussion, it is possible to say that the
following is required for a direct session between a client and
server to be successful:

a) If the client uses the NAME of the server to reference it,
   the client must either have a hardcoded name-to-address
   binding for the server, or it must be able to resolve the
   server name (typically using DNS). In the case of DNS, this
   implies that the client and server must be part of the same
   DNS architecture or tree.

b) The client and server must be part of the same internetwork:
   the client must have a valid IP route towards the server,
   the server must have a valid IP route towards the client,
   and all intermediate IP gateways must have valid routes
   towards the client and server ("IP gateway" is the RFC
   standard terminology; people often use the term "IP router"
   in computer rooms).

c) If there are devices on the path between the client and
   server that perform packet filtering, all these devices must
   permit the forwarding of packets between the IP address of
   the client and the IP address of the server, at least for
   packets that fit the protocol model of the FTP application
   (TCP ports used, etc.).

3. Classical application proxies

A classical application proxy is a special program that knows one (or
more) specific application protocols. Most application protocols are
not symetric; one end is considered to be a "client", one end is a
"server".

A classical application proxy implements both the "client" and
"server" parts of an application protocol. In practice, it only needs
to implement enough of the client and server protocols to accomplish
the following:

   a) accept client sessions and appear to them as a server;

   b) receive from a client the name or address of the final target
      server (this needs to be passed over the "client-proxy" session
      in a way that is application-specific);

   c) setup a session to the final server and appear to be a client
      from the server's point of view;

   d) relay requests, responses, and data between the client and
      server;

   e) perform access controls according to the proxy's design
      criteria (the main goal of the proxy, after all).

The functional goal of the proxy is to relay application data between
clients and servers that may not have direct IP connectivity. The
security goal of the proxy is to do checks and types of access
controls that typical client and server software do not support or
implement.

The following information will make it clear that classical proxies
can offer many hidden benefits to the security-conscious network
designer, at the cost of deploying client software with proxy
capabilities or of educating the users on proxy use.

Client software issues are now easier to handle, given the increasing
number of popular client applications (for Web, FTP, etc.) that offer
proxy support. Designers developing new protocols are also more
likely to plan proxy capability from the outset, to ensure their
protocols can cross the many existing large corporate firewalls that
are based at least in part on classical proxy technology.

3.1 Classical proxy session example

   We will repeat our little analysis of an FTP session. This time,
   the FTP session is passing through a "classical" application proxy
   system. As is often the case (although not required), we will
   assume that the proxy system has two IP addresses, two network
   interfaces, and two DNS names.

   The proxy system is running a special program which knows how to
   behave like an FTP client on one side, and like an FTP server on
   the other side. This program is what people call the "proxy". We
   will assume that the proxy program is listening to incoming
   requests on the standard FTP control port (21/tcp), although this
   is not always the case in practice.

```
+---------------+      +----------+
|               |     /    IP      \
|  c.dmn1.com   |----+  network(s)  +----------+
| (c1.c2.c3.c4) |     \            /           |
+---------------+      +----------+    +----------------+
                                       | (p1.p2.p3.p4)  |
                                       | proxy1.dmn3.com |
                                       |                |
                                       | proxy2.dmn4.com |
                                       | (p5.p6.p7.p8)  |
+---------------+      +----------+    +----------------+
|               |     /    IP      \           |
|  s.dmn2.com   |----+  network(s)  +----------+
| (s1.s2.s3.s4) |     \            /
+---------------+      +----------+
```

The user starts an instance of an FTP client program on the client
system "c.dmn1.com", and MUST specify that the target system is
"proxy1.dmn3.com". On command-line systems, the user typically
types:

        ftp proxy1.dmn3.com

The client system needs to convert the proxy's name to an IP
address (if the user directly specified the proxy by address, this
step is not needed).

Converting the proxy name to an IP address requires work to be
performed which ranges between two extremes:

  a) the client system has this name in its hosts file, or has
     local DNS caching capability and successfully retrieves the
     name of the proxy system in its cache. No network activity
     is performed to convert the name to an IP address.

  b) the client system, in combination with DNS name servers,
     generate DNS queries that eventually propagate close to the
     root of the DNS tree and back down the proxy's DNS branch.
     Eventually, a DNS server which is authoritative for the
     proxy system's domain is queried and returns the IP
     address associated with "proxy1.dmn3.com" (depending on the
     case, it may return this to the client system directly or
     to an intermediate name server). Ultimately, the client
     system obtains a valid IP address for proxy1.dmn3.com.

```
+---------------+            +--------+
|               |           /    IP    \
|  c.dmn1.com   |--------+  network(s)  +------------+
| (c1.c2.c3.c4) |          \           /             |
+---------------+            +--------+      +-----------------+
  A  |                         /      \      | (p1.p2.p3.p4)   |
     |   address for          /        \     | proxy1.dmn3.com |
     |   proxy1.dmn3.com?    /          \    |      ...        |
     |                      /            \   +-----------------+
     |                     /              \
     |                    /                \
     |          +--------+ proxy1.dmn3.com?  +--------+
     |  +------>|  DNS   |----------------->|  DNS   |
     |  |       | server |                  | server |
 +-----------|  |   X    |<-----------------|   Y    |
  p1.p2.p3.p4 +--------+    p1.p2.p3.p4    +--------+
```

   Once the client system knows the IP address of the proxy system,
   it attempts to establish a connection to the standard FTP
   "control" TCP port on the proxy (port 21). For this to work, the
   client system must have a valid route to the proxy's IP address,
   and the proxy system must have a valid route to the client's IP
   address. All intermediate devices that behave like IP gateways
   must have valid routes to both the client and the proxy. If these
   devices perform packet filtering, they must ALL allow the specific
   type of traffic required between C and P1 for this specific
   application (FTP).

   Finally, the proxy system must accept this incoming connection,
   based on the client's IP address (the purpose of the proxy is
   generally to do access control, after all).

```
+---------------+                   |      ...        |
|  c.dmn1.com   |                   | proxy1.dmn3.com |
| (c1.c2.c3.c4) |                   |  (p1.p2.p3.p4)  |
+---------------+                   +-----------------+
  | |                                    | |
  | | route to P1           route to C   | |
  | V                                 V  |
  |                                      |
  | A                                  A |
  | | route to C                         | | route to P1
  | |                                    | |
  | |   C          P1          C         | |
+----+  <-- +----+ -->    +----+  <-- +----+
| G1 |------| Gx |--------| Gy |---------| Gn |
+----+ -->  +----+  <-- +----+ -->    +----+
      P1            C           P1
```
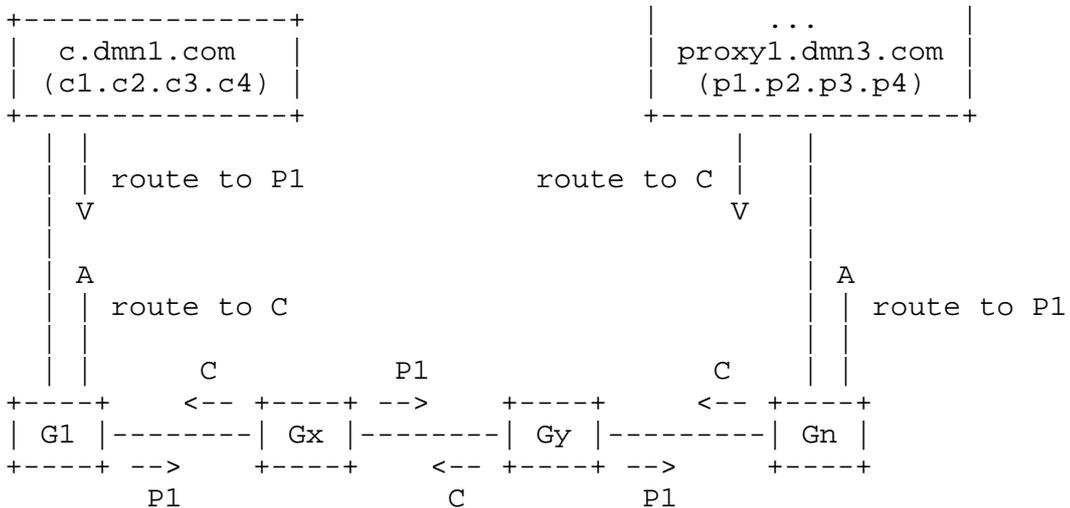
The actual application work for the FTP session between the client
and proxy is done with a bidirectional flow of TCP packets between
the client's and proxy's IP addresses.

For this to work, the proxy FTP application MUST fully support the
FTP protocol and look identical to an FTP server from the client's
point of view.

Once the client<->proxy session is established, the final target
server name must be passed to the proxy, since, when using a
"classical" application proxy, a way MUST be defined for the proxy
to determine the final target system. This can be achieved in
three ways:

 a) The client system supplies the name or address of the final
    target system to the proxy in a method that is compatible
    with the specific application protocol being used (in our
    example, FTP). This is generally considered to be the main
    problem with classical proxies, since for each application
    being proxied, a method must be defined for passing the
    name or address of the final target system. This method
    must be compatible with every variant of client application
    that implements the protocol (i.e. the target-passing
    method must fit within the MINIMUM functionalities required
    by the specific application protocol).

    For the FTP protocol, the generally popular method for
    passing the final server name to the proxy is as follows:

    When the proxy prompts the FTP client for a username, the
    client specifies a string of the form:

          target_username@target_system_name
          or
          target_username@target_ip_address

    The proxy will then know what is the final target system.
    The target_username (and the password supplied by the
    client) will be forwarded "as is" by the proxy to the final
    target system.

    A well-known example of an FTP proxy that behaves in this way
    is the "ftp-gw" program which is part of the Trusted
    Information System's firewall toolkit, available by anonymous
    FTP at ftp.tis.com. Several commercial firewalls also support
    this de-facto standard.

   b) If there is only one possible final destination, the proxy
      may be configured to know this destination in advance.
      Since the IP address of the client system is known when the
      proxy must make this decision, the proxy can (if required)
      select a different destination based on the IP address of
      the client.

   c) The client software may also support capabilities that allow
      it to present to the user the illusion of a direct session
      (the user just specifies the final target system, and the
      client software automatically handles the problem of
      reaching to the proxy system and passing the name or address
      of the final target system in whatever mutually-acceptable
      form).

      A well-known example of a system that provides modified
      client software, proxy software, and that provides the
      illusion of transparency is NEC's SOCKS system, available by
      anonymous FTP at ftp.nec.com.

      Alternatively, several FTP client applications support the
      "username@destination_host" de-facto standard implemented
      (for example) by the "ftp-gw" proxy application.

Once the FTP proxy application knows the name or IP address of the
target system, it can choose to do two things:

 a) Setup a session to the final target system, the more
    frequent case.

 b) Decide (based on some internal configuration data) that it
    cannot reach the final target system directly, but must go
    through another proxy. This is rare today, but may become
    temporarily common due to the current shortage of IP
    network numbers which encourages organizations to deploy
    "hidden" network numbers which are already assigned
    elsewhere. Sessions between systems which have the same
    IP network number but which belong to different actual
    networks may require going through two proxy systems.
    This is discussed in more detail in section 3.2.6,
    "Interconnection of conflicting IP networks".

If the FTP proxy decides to connect directly to the target system,
and what it has is the target system name, it will need to convert
the target system name into an IP address. If this process
involves DNS resolution, something like the following will happen:

```
          +-----------------+
          | proxy1.dmn3.com |
          |   (p1.p2.p3.p4) |          +--------+
          |                 |         /    IP    \
          | proxy2.dmn4.com |--------+ network(s) +------------+
          |   (p5.p6.p7.p8) |         \          /             |
          +-----------------+          +--------+    +---------------+
           A  |                        /      \      | (s1.s2.s3.s4) |
           |  | address for          /        \     | s.dmn2.com    |
           |  | s.dmn2.com?         /          \     |               |
           |  |                    /            \    +---------------+
           |  |                   /              \
           |  |                  /                \
           |  |     +--------+  s.dmn2.com?       +--------+
           |  +-------->| DNS    |------------------>| DNS    |
           |           | server |                   | server |
          +------------| X      |<------------------| Y      |
          s1.s2.s3.s4 +--------+    s1.s2.s3.s4    +--------+
```
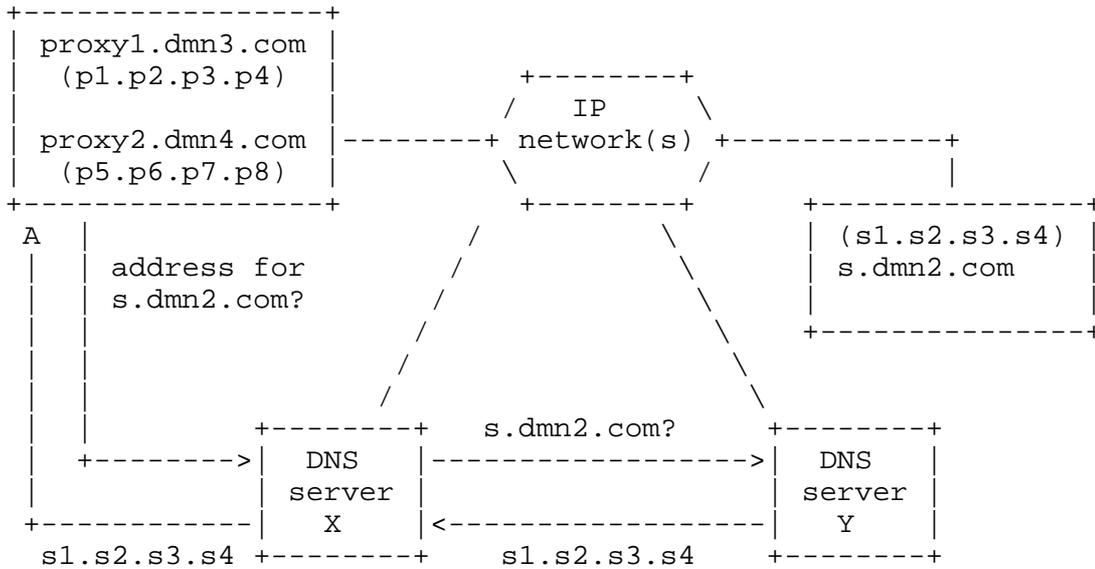
     Once the proxy system knows the IP address of the server system,
     it attempts to establish a connection to the standard FTP
     "control" TCP port on the server (port 21). For this to work, the
     proxy system must have a valid route to the server's IP address,
     and the server system must have a valid route to at least one of
     the proxy's IP address. All intermediate devices that behave like
     IP gateways must have valid routes to both the proxy and the
     server. If these devices perform packet filtering, they must ALL
     allow the specific type of traffic required between the proxy and
     S for this specific application.

```
+-----------------+
| proxy1.dmn3.com |
|   (p1.p2.p3.p4) |
|                 |                      +----------------+
| proxy2.dmn4.com |                      |   s.dmn2.com   |
|   (p5.p6.p7.p8) |                      |  (s1.s2.s3.s4) |
+-----------------+                      +----------------+
    | |                                       |  |
    | |                                       |  |
    | | route to S          route to P2 |  |
    | V                                    V  |
    |                                         |
    | A                                       | A
    | | route to P2                           | | route to S
    | |                                       | |
    | |      P2        S              P2      | |
  +----+   <-- +----+ -->      +----+   <-- +----+
  | G1 |-------| Gx |--------| Gy |---------| Gn |
  +----+ -->   +----+   <-- +----+ -->     +----+
        S              P2         S
```
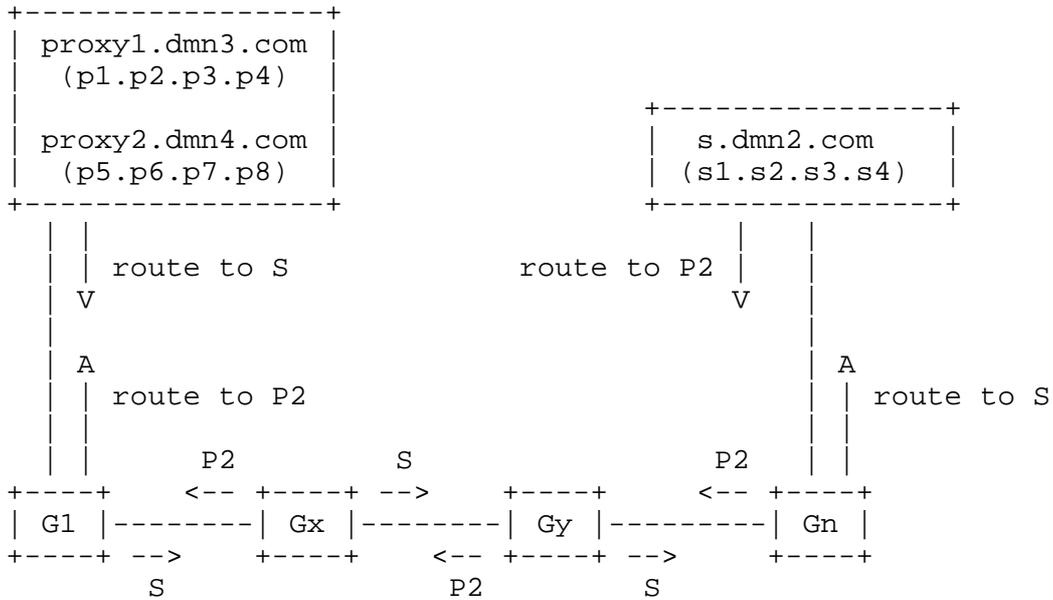
The actual FTP application work between the proxy and server is
done with a bidirectional flow of TCP packets between the proxy's
and server's IP addresses.

What actually happens BETWEEN THE CLIENT AND SERVER?  They both
send replies and responses to the proxy, which forwards data to
the "other" end. When one party opens a data connection and sends
a PORT command to the proxy, the proxy allocates its own data
connection and sends its PORT command to the "other" end. The
proxy also copies data across the connections created in this way.

3.2 Characteristics of classical proxy configurations

Several IP internetworks may be linked using only classical proxy
technology. It is currently popular to link two specific IP
internetworks in this way: the Internet and some organization's
"private" IP network. Such a proxy-based link is often the key
component of a firewall.

When this is done, several benefits and problems are introduced
for network administrators and users.

3.2.1 IP addressing and routing requirements.

   The proxy system must be able to address all client and server
   systems to which it may provide service. It must also know
   valid IP routes to all these client and server systems.

Client and server systems must be able to address the proxy
system, and must know a valid IP route to the proxy system. If
the proxy system has several IP addresses (and often, several
physical network interfaces), the client and server systems
need only to be able to access ONE of the proxy system's IP
addresses.

Note that client and server systems that use the proxy for
communication DO NOT NEED valid IP addressing or routing
information for systems that they reach through the proxy.

In this sense, it can be said that systems separated by a
classical proxy are isolated from each other in an IP
addressing sense and in an IP routing sense.

On the other hand, the classical proxy system (if running a
standard TCP/IP software stack) needs to have a single coherent
view of IP addressing and routing. If such a proxy system
interconnects two IP networks and two systems use the same IP
network/subnetwork number (one system on each network), the
proxy will only be able to address one of the systems.

This restriction can be removed by chaining classical proxies
(this is described later in section 3.2.6, "Interconnection of
conflicting IP networks").

Using a classical proxy for interconnection of IP
internetworks, it is also possible, with care, to achieve a
desirable "fail-safe" feature: no valid routing entries need to
exist for an internetwork which should be reached only through
the proxy (routing updates that could add such entries shout be
BLOCKED). If the proxy suddenly starts to behave like an IP
router, only one-way attacks become possible.

In other words, assume an attacker has control of the remote
internetwork and has found a way to cause the proxy to route IP
packets, or has found a way to physically bypass the proxy.

The attacker may inject packets, but the attacked internal
systems will be unable to reply to those packets. This
certainly does not make attacks infeasible (as exemplified by
certain holiday-period events in recent years), but it still
makes attacks more difficult.

3.2.2 IP address hiding

Application "sessions" that go through a classical proxy are
actually made of two complete sessions:

    a) a session between the client and the proxy
    b) a session between the proxy and the server

A device on the path sees only the client<->proxy traffic or
the proxy<->server traffic, depending where it is located. If
the two sessions actually pass through the same physical
network, a device on that network may see both traffics, but
may have difficulty establishing the relationship between the
two sessions (depending on the specific application and
activity level of the network).

A by-product of a classical proxy's behavior is commonly known
as "address hiding". Equipments on some side of a classical
proxy cannot easily determine what are the IP addresses used on
another side of the proxy.

Address hiding is generally viewed as a Good Thing, since one
of the purposes of deploying proxies is to disclose as little
information about an internetwork as possible.

People who are in charge of gathering network statistics, and
who do not have access to the proxy system's reports (if any)
may consider address hiding to be a Bad Thing, since the proxy
obscures the actual client/server relationships where the proxy
was inserted.  All IP activity originates and terminates on the
proxy itself (or appears to do so).

In the same way, server software that accepts connections that
have gone through a classical proxy do not see the IP address
of the incoming client, unless this information is included in
the application protocol (and even if it is, in many cases, the
proxy will replace this information with its own address for
the protocol to be consistent). This makes server access
control unusable if it is based on client IP address checks.

3.2.3 DNS requirements

In most classical-proxy configurations, client systems pass the
desired server name (or address) to the proxy system WITHOUT
INTERPRETING IT. Because of this, the client system DOES NOT
REQUIRE to be able to resolve the name of the server system in
order to access it through a classical proxy. It only needs to
be able to resolve the name of the proxy (if referencing the

proxy system by name).

Because of this, it can be said that a classical proxy system
can offer DNS isolation. If two IP internetworks use completely
separate DNS trees (each with their own DNS root servers),
client software in one IP internetwork may still reference a
server name in the other IP internetwork by passing its name to
the classical proxy.

The classical proxy itself will not be able alone to resolve
DNS names in both environments (if running standard DNS
resolution software), since it will need to point to one or the
other of the two DNS "universes".

A well-known technique called "split-brain DNS" can be used to
relax this restriction somewhat, but such a technique
ultimately involves prioritizing one DNS environment over
another. If a DNS query can return a valid answer in both
environments, only one of the answers will be found by the
proxy.

3.2.4 Software requirements

A classical proxy application is a fairly simple piece of
software, often simpler than either a real client
implementation or a real server implementation.  Such a program
may run on any system that supports normal TCP/IP connections,
and often does not require "system" or "superuser" privilege.

Classical proxy connections have no impact on normal server
software; the proxy looks like a normal client in most respects
except for its IP address and its "group" nature. All
connections from the network on the other side of the proxy
appear to come from the proxy, which poses problems if access
control by client system is desired.

Normal client software may access a classical proxy if the user
is willing or able to go through the extra steps necessary to
indicate the final server to the proxy (whatever they are).
Alternatively, modified (or newer) client software may be used
that knows how to negotiate transparently with the proxy.

3.2.5 Impact of a classical proxy on packet filtering

If packet filtering is needed around a classical proxy, the
packet filtering rules tend to be simplified, since the only
traffic needed and allowed will originate from or terminate on
the proxy (in an IP sense).

If the proxy starts behaving like an IP router, or if it is
physically bypassed, such filtering rules, if deployed
generally within an IP internetwork, will tend to prevent any
direct traffic flow between the "internal" internetwork and
"external" internetworks that are supposed to be only reachable
through the application proxy.

### 3.2.6 Interconnection of conflicting IP networks

By chaining classical proxies, it is possible to achieve some
interconnection of IP networks that have a high level of
conflict. In practice, this type of setup resolves IP
addressing conflicts much better than DNS conflicts. But DNS
conflicts are currently less of a problem because the DNS
"address space" is almost infinitely large (has anybody
calculated the possible DNS address space based on the RFC-
standard maximum host name length?).

Even though RFC 1597 was never more than an informational RFC,
many organizations have been quietly following its suggestions,
for lack of an easier solution. Now assume two organizations
each use class A network number 10 on their network. Suddenly,
they need to interconnect.  What can they do?

First possibility: one side changes network number (not as hard
as people think if properly planned, but this still represents
some work)

Second possibility: they merge the two numbers by renumbering
partially on each side to remove conflicts (actually harder to
do, but has the political advantage that both sides have to do
some work)

Third possibility: they communicate through chained classical
proxies:

```
   +--------+     +--------+    +--------+     +--------+
  /  Org. 1  \    | Proxy  |    | Proxy  |    /  Org. 2  \
 +  dmn1.com  +---+ system +---+ system +---+  dmn2.com  +
  \  net 10  /    |   1    |    |   2    |    \  net 10  /
   +--------+     +--------+    +--------+     +--------+
```
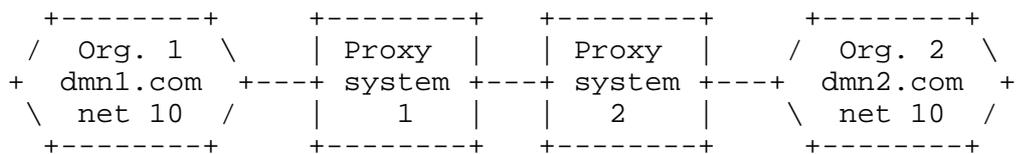
Both proxy 1 and 2 are standard systems running normal TCP/IP
software stacks. Their configuration is not typical, however:

a) The link between proxy 1 and proxy 2 may use any IP
   network number that is not used (or not needed) on
   either side. Nothing on Org.1 and Org.2's networks
   need to have an IP route to this network.

b) Proxy 1 has an IP route for network 10 that points to
   Organization 1's network, and does DNS resolution
   (if required) using dmn1.com's name servers.

c) Proxy 2 has an IP route for network 10 that points to
   Organization 2's network, and does DNS resolution
   (if required) using dmn2.com's name servers.

d) Proxy 1 and proxy 2 only require a host IP route to
   each other for communication.

e) For this to be convenient, the classical proxy
   applications must support the automatic selection of
   a destination based on the client IP address.

f) On proxy system 1, the proxy software treats incoming
   sessions from proxy system 2 in the normal way: the
   "client" (proxy system 2) will be prompted in an
   application-specific way for the final destination.
   However, incoming sessions from Org.1 addresses are
   immediately and automatically forwarded to proxy
   system 2.

   Proxy system 2 is configured similarly (that is,
   connections coming from proxy 1 are prompted for a
   target server name, connections from Org.2 addresses
   are immediately and automatically forwarded to
   proxy 1.

From a user's point of view, the behavior of such a chained
proxy system is not very different from a single classical
application proxy:

a) A user on a client system with address 10.1.2.3
   on Org.1's network wishes to do an anonymous FTP to
   "server.dmn2.com".

b) The user starts an FTP towards proxy 1. Proxy 1 sees
   an incoming connection from an address in network 10,
   so it immediately relays the connection to proxy 2.

c) Proxy 2 sees a connection coming from proxy 1, so it
   prompts the client. The user sees the username prompt

and types (assuming FTP proxies that behave like TIS's
ftp-gw):

    anonymous@server.dmn2.com

This will be resolved IN THE CONTEXT OF Org. 2'S
NETWORK. The user can then complete the dialog and
use the FTP connection.

d) Note that this setup will work even if the client and
   server have the EXACT SAME IP ADDRESS (10.1.2.3 in
   our example).

If the proxy applications support selecting another
proxy based on the destination supplied by the client,
and if DNS domains are unique, more than two conflicting
IP networks can be linked in this way! Here is an
example configuration:

a) Four IP networks that all use network 10 are linked
   by four proxy systems. The four proxy systems share a
   common, private IP network number and physical link
   (LAN or WAN).

b) A user on organization 1's network wishes to access
   a server on network 3. The user connects to its local
   proxy (proxy 1) and supplies that target system name.

c) Proxy 1 determines, based on a configuration rule,
   that the target system name is reachable by using
   proxy 3. So it connects to proxy 3 and passes the
   target system name.

d) Proxy 3 determines that the target system name is
   local (to itself) and connects to it directly.


Security Implications of chained proxies

Obviously, when such "chained" configurations are built,
access control rules and logging based on a
final-client/final-server combination are difficult to
enforce, since the first proxy in the chain sees a
final-client/proxy relationship and the last proxy in
the chain sees a proxy/final-server relationship.

Doing better than this requires that the proxies be
capable of passing the "original-client" and

"final-destination" information back and forth in the
proxy chain for access control and/or logging purposes.
This requires the proxies to trust each other, and
requires the network path to be trusted (forging this
information becomes an excellent attack).

Even if these problems were to be solved reliably, the
original goal of the proxy chains was to solve an IP
and possibly a DNS conflict. The "original-client" and
"final-destination" values may not have the same
meaning everywhere in the overall setup. Tagging the
information with a "universe-name" may help, assuming
it is possible to define unique universe names in the
first place. Obviously this topic requires more study.

## 4. Transparent application proxies

The most visible problem of classical application proxies is the need
for proxy-capable client programs and/or user education so that users
know how to use the proxies.

When somebody thought of modifying proxies in such a way that normal
user procedures and normal client applications would still be able to
take advantage of the proxies, the transparent proxy was born.

A transparent application proxy is often described as a system that
appears like a packet filter to clients, and like a classical proxy
to servers. Apart from this important concept, transparent and
classical proxies can do similar access control checks and can offer
an equivalent level of security/robustness/performance, at least as
far as the proxy itself is concerned.

The following information will make it clear that small organizations
that wish to use proxy technology for protection, that wish to rely
entirely on one proxy system for network perimeter security, that
want a minimal (or zero) impact on user procedures, and that do not
wish to bother with proxy-capable clients will tend to prefer
transparent proxy technology.

Organizations with one or more of the following characteristics may
prefer deploying classical proxy technology:

a) own a substantial internal IP router network, and wish to
   avoid adding "external" routes on the network
b) wish to deploy "defence in depth", such as internal firewalls,
   packet filtering on the internal network
c) wish to keep their DNS environment fully isolated from the
   "other side" of their proxy system, or that fear that their

          internal DNS servers may be vulnerable to data-driven attacks
     d) use some IP networks that are in conflict with the "other side"
          of their proxy system
     e) wish to use proxy applications that are easily portable
          to different operating system types and/or versions
     f) wish to deploy multiple proxy systems interconnecting them
          to the SAME remote network without introducing dynamic
          routing for external routes on the internal network

     4.1 Transparent proxy connection example

          Let us go through an FTP sesssion again, through a "transparent"
          proxy this time. We assume that the proxy system has two IP
          addresses, two network interfaces, and two DNS names.

          The proxy system is running a special program which knows how to
          behave like an FTP client on one side, and like an FTP server on
          the other side. This program is what people call the "proxy". This
          program, being a transparent proxy, also has a very special
          relationship with the TCP/IP implementation of the proxy system.
          This relationship may be built in several ways, we will describe
          only one such possible way.

          We will assume that the proxy program is listening to incoming
          requests on the standard FTP control port (21/tcp), although this
          is not always the case in practice.


        +---------------+       +----------+
        |               |       /    IP     \
        |   c.dmn1.com  |----+  network(s)   +----------+
        | (c1.c2.c3.c4) |       \           /           |
        +---------------+       +----------+   +----------------+
                                               | (p1.p2.p3.p4)  |
                                               | proxy1.dmn3.com |
                                               |                |
                                               | proxy2.dmn4.com |
                                               | (p5.p6.p7.p8)  |
        +---------------+       +----------+   +----------------+
        |               |       /    IP     \                  |
        |   s.dmn2.com  |----+  network(s)   +----------+
        | (s1.s2.s3.s4) |       \           /
        +---------------+       +----------+
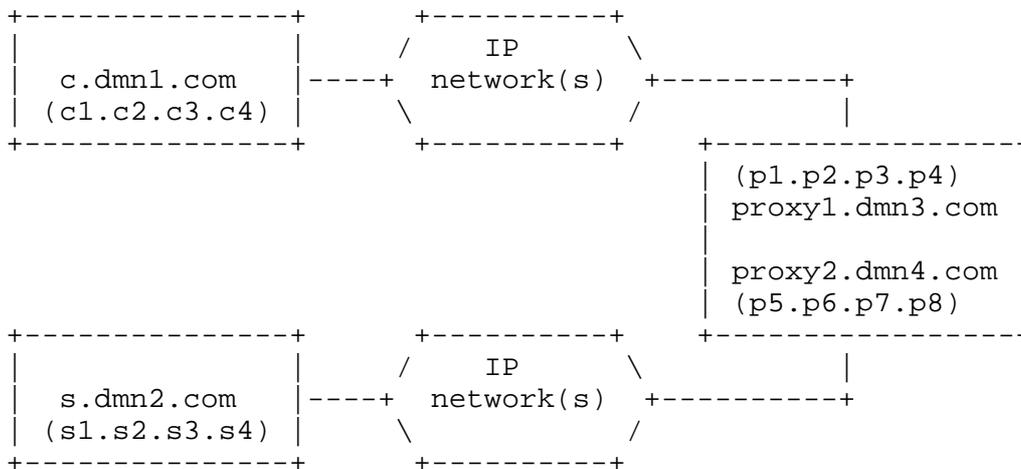
The user starts an instance of an FTP client program on the client
system "c.dmn1.com", and specifies a destination of "s.dmn2.com",
just like if it was reachable directly.  On command-line systems,
the user typically types:

        ftp s.dmn2.com

The client system needs to convert the server's name to an IP
address (if the user directly specified the server by address,
this step is not needed).

Converting the server name to an IP address requires work to be
performed which ranges between two extremes:

 a) the client system has this name in its hosts file, or has
    local DNS caching capability and successfully retrieves the
    name of the proxy system in its cache. No network activity
    is performed to convert the name to an IP address.

 b) the client system, in combination with DNS name servers,
    generate DNS queries that eventually propagate close to the
    root of the DNS tree and back down the server's DNS branch.
    Eventually, a DNS server which is authoritative for the
    server system's domain is queried and returns the IP
    address associated with "s.dmn2.com" (depending on the
    case, it may return this to the client system directly or
    to an intermediate name server). Ultimately, the client
    system obtains a valid IP address for s.dmn2.com.

```
     +---------------+             +--------+
     |               |            /   IP     \
     |  c.dmn1.com   |--------+  network(s) +-----------+
     | (c1.c2.c3.c4) |            \         /           |
     +---------------+             +--------+    +----------------+
      A |                          /             | (p1.p2.p3.p4)  |
        |  | address for          /    +-----+   | proxy system   |
        |  | s.dmn2.com?         /    /       \  | (p5.p6.p7.p8)  |
        |  |                    /    /         \ +----------------+
        |  |                   /    /           \        |
        |  |                  /    / s.dmn2.com? \       |
        |  |      +--------+  /                   |   +--------+
        |  +----->| DNS    |--+   +-------+       |  /   IP    \
        |         | server |     /         \     | + network(s) +
     +-----------|   X     |<---+           +    |  \         /
     s1.s2.s3.s4 +--------+     s1.s2.s3.s4|  |    +--------+
                                           |  |         |
                                           |  +         |
                                           |   \     +--------+
                                           +    +->|  DNS    |
                                            \      | server |
                                           +----|    Y     |
                                                 +--------+
```

NOTE: In practice, DNS servers that are authoritative for
      s.dmn2.com are highly likely to be located on the OTHER
      side of the proxy system. This means that DNS queries
      from the inside to the outside MUST be able to cross the
      proxy system. If the proxy system wishes to provide
      "address hiding", it must make these DNS queries
      (originating from the inside) appear to come from the
      proxy itself. This can be achieved by using a BIND-based
      DNS server (which has some proxy capabilities) or some
      simpler DNS proxy program.  For full RFC compliance,
      the proxy system must be able to relay TCP-based queries
      just like UDP-based queries, since some client systems
      are rumored to ONLY use TCP for DNS queries.

      The proxy system must be able to detect and block several
      classes of attacks based on DNS which (if nothing else)
      may cause denial of service:

      a) attempts from the outside to return corrupt cache
         entries to an internal DNS server
      b) attempts to return DNS bindings which have no
         relationship to the actual DNS query (some DNS
         servers are vulnerable to this). The attacker's goal
         may be to prime the cache of internal DNS servers with

interesting entries, including entries for internal
DNS names that point to external IP addresses...
c) data-driven stuff similar in style to the "syslog
buffer overrun" type attacks.

Once the client system knows the IP address of the server system,
it attempts to establish a connection to the standard FTP
"control" TCP port on the server (port 21). For this to work, the
client system must have a valid route for the server's IP address
THAT LEADS TO THE PROXY SYSTEM, and the proxy system must have a
valid route for the client's IP address and the server's IP
address. All intermediate devices that behave like IP gateways
must have valid routes for the client, the server, and usually the
proxy. If these devices perform packet filtering, they must ALL
allow the specific type of traffic required between C and S for
this specific application.

```
                                           A
                            route to S     |
                                           |
                                +-----------------+
    +---------------+           |  (p5.p6.p7.p8)  |
    |  c.dmn1.com   |           | proxy system    |
    | (c1.c2.c3.c4) |           |  (p1.p2.p3.p4)  |
    +---------------+           +-----------------+
       | |                            |   |
       | | route to S      route to C |   |
       | V                          V |
       |                              |
       | A                            | A
       | | route to C                 | | route to S
       | |                            | |
       | |    C         S         C   | |
    +----+  <-- +----+ -->   +----+  <-- +----+
    | G1 |--------| Gx |--------| Gy |---------| Gn |
    +----+ -->    +----+    <-- +----+ -->    +----+
          S              C            S
```
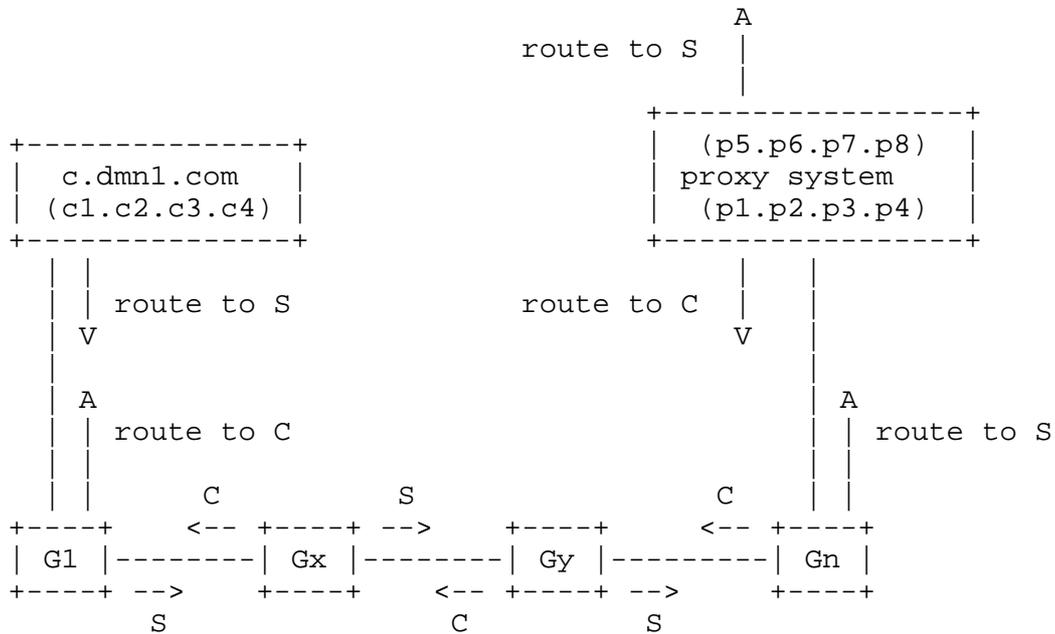
At the start of the FTP session, a TCP packet with a source
address of C and a destination address of S travels to the proxy
system, expecting to cross it just like a normal IP gateway.

This is when the transparent proxy shows its magic:

The proxy's TCP/IP software stack sees this incoming packets (and
subsequent ones) for a destination address that is NOT one of its
own addresses. Based on some criteria (a configuration file, for

example), it decides NOT to forward or drop the packet (which are
the only two choices an RFC-standard TCP/IP implementation would
have). The proxy system accepts the packet as if it was directed
to one of its own IP addresses.

In our example, the incoming packet is a TCP packet. Since
standard TCP/IP stacks store both a LOCAL and REMOTE IP address
field for each TCP connection, the transparent proxy may set the
LOCAL IP address field to the IP address that the client wants to
reach (s1.s2.s3.s4 in our example). The standard TCP/IP stack
probably needs to be modified to do this. UDP examples, although
not connection-based, could be handled in similar ways.

Once this is done, the actual FTP proxy application is invoked
since an incoming connection to TCP port 21 has occurred. It can
determine what is the final target destination instantly, since
the LOCAL IP address field of the connection contains the target
server's IP address.  There is no need for the proxy application
to ask the client what is the final target system.

Since the FTP proxy application knows the IP address of the target
system, it can choose to do two things:

 a) Setup a session to the final target system, the more
    frequent case.

 b) Decide (based on some internal configuration data) that it
    cannot reach the final target system directly, but must go
    through a "classical" proxy. This seems technically
    feasible, although no real transparent proxy system is
    known to offer this capability. The actual value of such
    a feature (if available) would need to be studied.

If the FTP proxy decides to connect directly to the target system,
it has the target system's IP address. It may choose to do a
reverse lookup on the target IP address to obtain a target system
name (possibly needed for access control). If this process
involves DNS resolution, something like the following will happen:

```
    +-----------------+
    | proxy1.dmn3.com |
    |   (p1.p2.p3.p4) |              +--------+
    |                 |             /   IP     \
    | proxy2.dmn4.com |--------+  network(s)  +------------+
    |   (p5.p6.p7.p8) |             \         /            |
    +-----------------+              +--------+    +---------------+
     A  |                        /          \    | (s1.s2.s3.s4) |
     |  |  name for            /            \   | s.dmn2.com    |
     |  |  s1.s2.s3.s4?       /              \  |               |
     |  |                    /                \ +---------------+
     |  |                   /                  \
     |  |                  /                    \
     |  |        +--------+   s1.s2.s3.s4?      +--------+
     |  +------->|  DNS   |------------------>|  DNS   |
     |           | server |                    | server |
    +-----------|   X    |<------------------|   Y    |
     s.dmn2.com +--------+   s.dmn2.com       +--------+
```
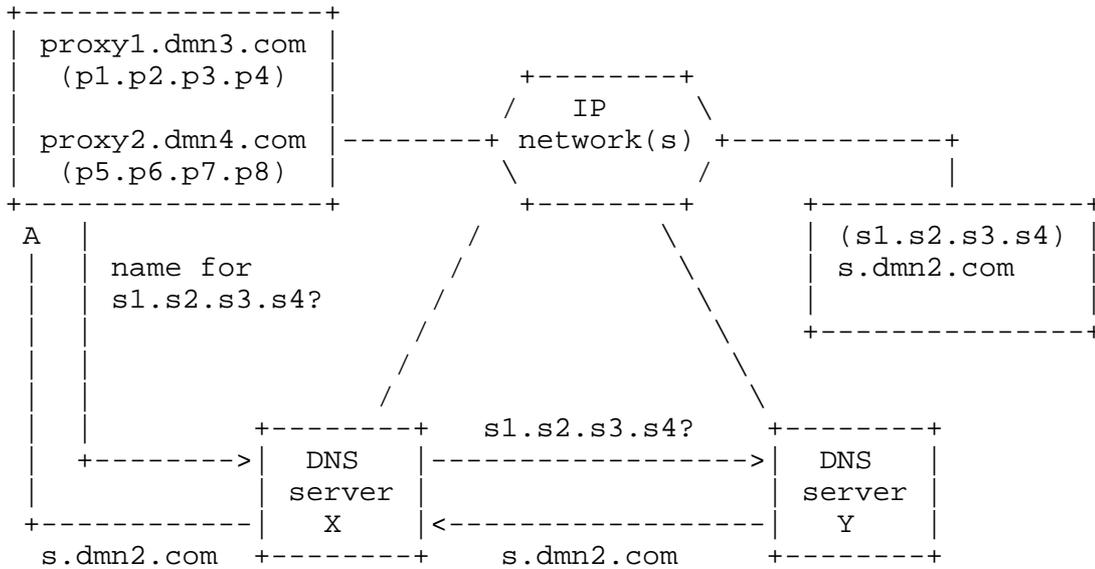
Once this is done and if the connection is allowed, the proxy
attempts to establish a connection to the standard FTP "control"
TCP port on the target server (port 21), using a technique
identical to a "classical" proxy. For this to work, the proxy
system must have a valid route to the server's IP address, and the
server system must have a valid route to at least one of the
proxy's IP address. All intermediate devices that behave like IP
gateways must have valid routes to both the proxy and the server.
If these devices perform packet filtering, they must ALL allow the
specific type of traffic required between the proxy and S for this
specific application.

```
        +-----------------+
        | proxy1.dmn3.com |
        |   (p1.p2.p3.p4) |
        |                 |                    +-----------------+
        |                 |                    |   s.dmn2.com    |
        | proxy2.dmn4.com |                    |  (s1.s2.s3.s4)  |
        |   (p5.p6.p7.p8) |                    |                 |
        +-----------------+                    +-----------------+
          | |                                      |  |
          | |  route to S          route to P2 |  |
          | V                                    V  |
          |                                         |
          | A                                     | A
          | |  route to P2                        | |  route to S
          | |                                     | |
          | |       P2          S          P2     | |
        +----+   <-- +----+ -->    +----+    <-- +----+
        | G1 |-------| Gx |--------| Gy |---------| Gn |
        +----+ -->   +----+   <-- +----+ -->      +----+
            S              P2          S
```
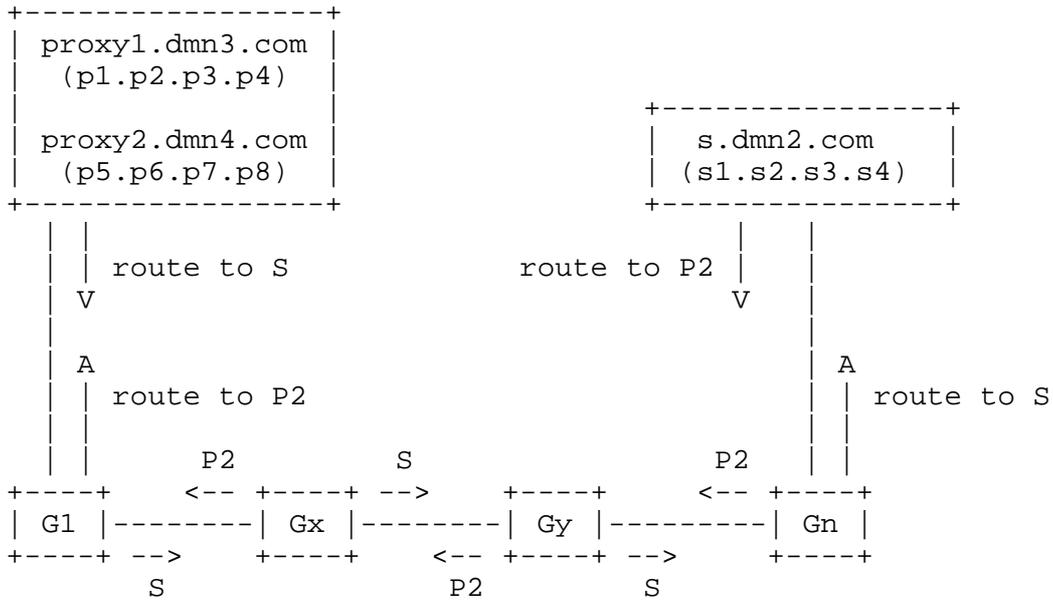
The rest of the transparent proxy's operation is very similar to
what would happen with a classical proxy.

4.2 Characteristics of transparent proxy configurations

   Transparent proxy technology can be used to build the key
   component of a "firewall", in a way quite similar to the way
   classical proxy technology may be used. Several important details
   of the architecture must be different, however.

   4.2.1 IP addressing and routing requirements

      The transparent proxy system must be able to address all client
      and server systems to which it may provide service. It must
      also know valid IP routes to all these client and server
      systems.

      Server systems must be able to address the proxy system, and
      must know a valid IP route to the proxy system. If the proxy
      system has several IP addresses (and often, several physical
      network interfaces), the server systems need only to be able to
      access ONE of the proxy system's IP addresses.

      Client systems MUST HAVE valid IP addressing and routing
      information for systems that they reach through the proxy. For
      example, in the common case where a transparent proxy is being
      used to interconnect a private network and the Internet, the

private network will effectively need to use a default route
that points to the transparent proxy system. This is a specific
need of transparent proxy configurations.

Interconnecting two internetworks with multiple transparent
proxies (for load sharing or fail-over) can be accomplished by
using different techniques from what would be done for
classical proxies:

>      a) with multiple classical proxies to the same remote
>         network, clients can be configured to access different
>         proxies manually, or DNS-based techniques, such as
>         DNS load-balancing may be used to make clients
>         access a different proxy at different times.
>
>      b) with multiple transparent proxies to the same remote
>         network, the internal network must be able to provide
>         dynamic routing towards the proxies (routing updates
>         may need to be supplied by the proxies themselves).
>         Client systems (depending on topology) may not need
>         to see the route changes, but internal backbone
>         routers probably do.

It is clear that internetworks linked by a transparent proxy
cannot be fully isolated from each other in an IP addressing
and routing sense. The network on which client systems are
located must have effective valid routing entries to the remote
internetwork; these routing entries must point to the proxy.

The transparent proxy system (if running a vaguely standard
TCP/IP software stack) needs to have a single coherent view of
IP addressing and routing. If a proxy system interconnects two
IP networks and two systems use the same IP network/subnetwork
number (one system on each internetwork), the proxy will only
be able to address one of the systems. Even if the proxy is
able to manage multiple conflicting IP universes (if, for
example, one instance of a complete TCP/IP stack and its data
structures is bound to each of the proxy network interfaces),
the client systems will still have a problem: Why should it
send packets with this network number to the proxy since this
network number exists also on the internal internetwork?

Chaining transparent proxies does not seem at first glance to
solve IP conflicts like it does for classical proxies.

From a "security" fail-safe point of view, the transparent
proxy has an undesirable characteristic: the network being
protected must have valid routing entries to the remote

network(s). If the proxy fails (starts behaving like a non-
filtering IP router) or is physically bypassed, it is likely
that the internal network will be immediately able to reply to
"attacker" packets. The attacker does not need to modify
routing tables or to spoof internal IP addresses.

This is important for organizations that do not wish to place
ALL their confidence and protection into a proxy system (for
whatever reason).

4.2.2 IP address hiding

Application "sessions" that go through a transparent proxy are
actually made of two complete sessions:

    a) a session between the client and the address of the
       server, the session being "intercepted" by the proxy
    b) a session between the proxy and the server

A device on the path sees either the client<->server traffic or
the proxy<->server traffic, depending where it is located. The
client<-"server" traffic is actually generated by the
transparent proxy. The two sessions SHOULD NEVER pass through
the same physical network, since in that case (due to the
routing requirements) a total bypass of the proxy at the IP
routing level may easily occur without being detectable.

Like classical proxies, transparent proxies accomplish a form
of IP address hiding. Client IP addresses are hidden from the
servers, since the servers see a session being initiated by the
proxy. Server IP addresses are NOT hidden from the clients
however, so that the illusion of transparency may be
maintained.

This difference implies that internal (client-side) network
statistics at the IP level will accurately reflect what outside
destinations are being accessed.  This can be useful for
analyzing traffic patterns.

4.2.3 DNS requirements

In transparent proxy configurations, client systems MUST be
able to resolve server names belonging to remote networks. This
is critical since the proxy will determine the target server
from the destination IP address of the packets arriving from
the client. Because of this, the "client" internetwork needs to
have some form of DNS interconnection to the remote network. If
internal client and name server IP addresses must be hidden

from the outside, these DNS queries must also be proxied.

Of course, remote host name/address relationships may be stored
locally on the client systems, but it is well known that such
an approach does not scale...

Because of this, it can be said that a transparent proxy system
cannot offer DNS isolation. If two IP internetworks use
completely separate DNS trees (each with their own DNS root
servers), client software in one IP internetwork will not have
a way of finding name/address relationships in the "other" DNS
tree, and this information must be obtained in order to pass
the desired address to the transparent proxy.

The classical proxy itself (if running standard DNS resolution
software) will not be able alone to resolve DNS names in both
environments, since it will need to point to one or the other
of the two DNS "universes".  Running multiple instances of DNS
resolution software can allow the proxy to do this, however.

Because of the requirement placed on some form of DNS
communication through the proxy, it is critical for the proxy
to be able to protect ITSELF, internal clients, and internal
name servers from data-driven attacks at the DNS level.

4.2.4 Software requirements

The big advantage of transparent proxies is that normal client
software may access remote servers with no modifications and no
changes to user procedures.

The transparent proxy application itself may not need to be
more complicated than a classical proxy application.

However, the proxy TCP/IP software stack cannot be a fully-
standard (well, today's standard at least) TCP/IP stack, and
requires specific extensions:

    a) the ability to specify ranges of IP addresses that
       do not belong to the proxy itself, but for which
       "intercept" processing will occur: if packets arrive
       at the proxy with a destination IP address in those
       ranges, the IP stack will not forward or drop the
       packets; it will pass them up to application layers.

    b) This mechanism requires that applications may obtain
       both the IP address from which the packets come, and
       the address to which the packets were going. Typical

IP stacks should already have the fields available
to store the info; it is a matter of updating them
properly for these "intercepted" packets.

c) In the case of "intercepted" TCP packets, the TCP
stack must support establishing TCP connections
where the "local" IP address is not one of the
proxy's IP address.

Any TCP/IP software implementation should be modifiable to
perform these tasks. If a standard API becomes widely available
to drive these extensions, and if this API is generally
implemented, transparent proxies may become "portable"
applications.

Until this occurs, it must be assumed that implementors have
chosen different ways of accomplishing these functions, so that
today's transparent proxy applications cannot be fully
portable. It also remains to be seen how much work is needed to
propagate these "extensions" to IPV6 software stacks.

4.2.5 Impact of a transparent proxy on packet filtering

The nature of a transparent proxy's functionality makes it
difficult to deploy good packet filtering on the "inside" (or
client-side) of the proxy. The proxy will "masquerade" as all
the external systems. Because of this, internal packet filters
WILL TYPICALLY NEED TO ALLOW IP traffic between internal and
external IP addresses.

Depending on the actual security policy of the network, it may
be possible to do filtering based on protocol type and/or on
TCP bits (to filter based on connection setup direction), but
filtering that blocks external IP addresses CANNOT be deployed.

If the proxy starts behaving like an IP router, or if
physically bypassed, the practical limitations imposed on
internal packet filtering imply that a lot of direct traffic
between the inside and outside network will be allowed to flow.
Furthermore, as we have seen previously, the internal network
will have valid routing entries for external network numbers
that point to the proxy.  If multiple proxies have been
deployed, the internal network may even HAVE TO TRUST routing
updates generated by the proxy.

In general, if an internal network wishes to communicate with
an external network through a transparent proxy, it MUST BE
FUNDAMENTALLY DESIGNED TO COMMUNICATE DIRECTLY with that

external network. This is true at the IP addressing level, at
the IP routing level, and at the DNS level.  A proxy security
failure in this type of environment is likely to result in
immediate, total, and undetected accessibility of the internal
network by the external network.

   4.2.6 Interconnection of conflicting IP networks

      Unlike classical proxies, transparent proxies do not readily
      seem useful in solving IP addressing conflicts.

      If two internetworks use the same network number(s), systems
      and routers in each internetwork will have valid routes to
      these network numbers. If these routes are changed to point to
      a transparent proxy, traffic that is meant to stay within the
      same internetwork would start to flow towards the proxy. The
      proxy will not be able to distinguish reliably between traffic
      between systems of the same internetwork, and traffic which is
      meant to cross the proxy.

      A possible solution to this problem is described in section 6
      of this document, "Improving transparent proxies".

5. Comparison chart of classical and transparent proxies

   For those who do not like longish discussions of technical details,
   here is a one-page summary of the strengths/weaknesses/differences of
   classical and transparent proxies:

```
    ----------------------------------------------------------------
   | Issue              | Classical Proxy    | Transparent Proxy    |
   |--------------------+--------------------+----------------------|
   | IP addressing      | systems/gateways on | systems/gateways on |
   |                    | each network need   | the "client" network |
   |                    | to address the proxy| need to address the  |
   |                    |                     | remote networks      |
   |                    |                     |                      |
   | IP routing         | systems/gateways on | systems/gateways on  |
   |                    | each network need a | the "client" network |
   |                    | valid routing entry | also need routing    |
   |                    | for the proxy       | entries for remote   |
   |                    |                     | entries              |
   |                    |                     |                      |
   | IP address hiding  | systems on each side| systems on the       |
   |                    | of the proxy are    | "client" side are    |
   |                    | hidden from each    | hidden from the      |
   |                    | other               | other sides          |
   |                    |                     |                      |
```

| | | |
|---|---|---|
| DNS | full isolation possible | resolution of outside names by inside systems is required |
| Proxy software requirements | runs on standard TCP/IP stack; can be portable | requires special TCP/IP stack; not 100% portable |
| Client software requirements | requires proxy-capable software or user education | nothing more than for a direct connection |
| User requirements | must use proxy-capable software or know how to use the proxy | nothing more than for a direct connection |
| Packet filtering | can filter out "external" addresses | cannot filter out "external" addresses |
| IP address conflict resolution | can be done with chained proxies that support auto-connect | no obvious way to get this to work |

6. Improving transparent proxies

   The main issues with transparent proxies seem to revolve around the
   need to force "client" systems to directly access external addresses.
   To some people, this characteristic makes a transparent proxy look
   too much like a complicated packet filter. Can this problem be
   solved?

   The first possibility that comes to mind is to use the flexibility of
   the DNS protocol to build new tricks. If we restrict the "internal"
   clients so that they MUST ALWAYS use DNS to resolve external host
   names AND THAT THEY MUST NEVER store permanent copies of external
   host addresses, the following technique would become theoretically
   possible (this is a very painful restriction, by the way):

   a) arrange for all internal queries for external DNS names to
      go to the transparent proxy system (this can be done in a
      number of ways).

   b) arrange for a routing entry to exist for a class A network
      number that is not used on the internal network. This IMPLIES
      that the internal network may not be part of the Internet. This
      routing entry will point to the transparent proxy system. For

the purpose of our discussion, this special network number will
be X.0.0.0.

c) when an internal system generates a query for an external
   address, the query (if no answer is cached on the internal
   network) will reach the proxy system. Assuming the query is to
   obtain the IP address corresponding to a domain name, the proxy
   will go through the following algorithm:

   - try to find a valid binding for this external domain name in
     its local cache

   - if not found, it will ITSELF launch an external DNS query
     for the domain name. When (and if) it receives a valid reply,
     it creates a local cache entry containing:

         Time To Live of the reply
         Expiry Time of the cache entry (based on the current time)
         External domain name
         External IP address
         Dynamically allocated IP address of the form X.x1.x2.x3.

   and returns to the client the dynamically allocated IP address
   in the range X.0.0.0, NOT THE REAL ONE.

   - the client may (or may not) store the IP address returned in
     its cache, and will then attempt to connect to the
     dynamically allocated IP address. This traffic will arrive at
     the proxy because of the routing setup.

   - The transparent proxy intercepts the traffic and can identify
     the actual desired target it should connect to based on the
     dynamically allocated IP address supplied by the client.

Such an approach, if workable, could improve many characteristics of
transparent proxies and may even make transparent proxies capable of
handling IP network number conflicts.

However, the algorithm above leaves many difficult questions
unsolved. Here is a list (by no means exhaustive) of these questions:

a) What is the percentage of client DNS resolver and DNS server
   implementations that conform to the RFC specifications in their
   handling of the Time-To-Live field?

b) How should the proxy handle other types of DNS queries for
   external domain names (inverse queries, queries for other
   resource record types)?

c) A client program may perform a DNS query once for an external
   name and then use the response for a long time (a large file
   transfer, or a permanent management session, for example).
   Should the proxy update the Expiry Time of cache entries based
   on the passing IP traffic, and if so, using what algorithm?

d) What new types of attacks would such a system introduce or
   make possible?

e) What data structures and resources (memory, disk) would be
   needed for an efficient implementation if the proxy must sustain
   a high rate of DNS queries for external names, and where a large
   number of different external names are referenced? The class A
   network number is used basically to reference cache entries.
   Would a 24-bit address space be sufficient for practical use?

f) What happens with the cache (and the functionality) if the proxy
   crashes or reboots?

Such a system would probably exhibit two types of intermittent
failures:

a) a client system is still using the result of an external name
   query (some X.x1.x2.x3 address dynamically allocated by the
   proxy), but this binding no longer exists in the proxy's cache.
   The client attempts a connection to this address, which fails.

b) a client's name cache contains a binding for X.x1.x2.x3, but the
   proxy has already reused this address for a different external
   host name. The client attempts a connection to this address,
   sees no obvious errors, but reaches a different system from the
   expected one.

If somebody has ever implemented such a scheme, information and live
experience in deploying it would be useful to the IP networking
community.

7. Security Considerations

   Most of this document is concerned with security implications of
   classical and transparent proxy technology.

8. Acknowledgements

   I could not have written this document without the support of Digital
   Equipment Corporation for whom I work as a consultant.

9. References

   [1] Cheswick, W., Bellovin, S., "Firewalls and Internet Security:
       Repelling the Wily Hacker", Addison-Wesley, 1994.

   [2] Chapman, B., Zwicky, E., "Building Internet Firewalls",
       O'Reilly and Associates, Inc., September 1995.

   [3] Comer, D., "Internetworking with TCP/IP volume 1: Principles,
       Protocols, and Architecture", Prentice-Hall, 1991.

   [4] Comer, D., Stevens, D., "Internetworking with TCP/IP volume 2:
       "Design, Implementation, and Internals", Prentice-Hall, 1991.

   [5] Postel, J., and J. Reynolds, "File Transfer Protocol (FTP)",
       STD 9, RFC 959, USC/Information Sciences Institute, October
       1985.

   [6] Huitema, C., "An experiment in DNS Based IP Routing", RFC 1383,
       INRIA, December 1992.

   [7] Rekhter Y., Moskowitz B., Karrenberg D., de Groot, G.,
       "Address Allocation for Private Internets", RFC 1597,
       IBM Corp., Chrysler Corp, RIPE NCC, March 1994.

   [8] The TIS firewall toolkit's documentation, available on
       Trusted Information System's anonymous FTP site, ftp.tis.com.

   [9] Many discussions in the last 18 months on the firewalls-digest
       mailing list maintained by Great Circle Associates. The
       archives of the list are maintained at ftp.greatcircle.com.

Author's Address

   Marc Chatel
   9, avenue Jean Monnet
   74940 ANNECY-LE-VIEUX
   FRANCE

   EMail: mchatel@pax.eunet.ch
   or at Digital Equipment:
   Marc.Chatel@aeo.mts.dec.com