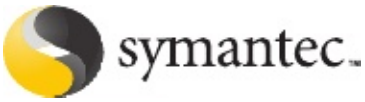
SYMANTEC ADVANCED THREAT RESEARCH



The Teredo Protocol:
Tunneling Past Network Security and Other Security Implications

Dr. James Hoagland
Principal Security Researcher
Symantec Advanced Threat Research

Symantec Advanced Threat Research

The Teredo Protocol

Tunneling Past Network Security and Other Security Implications

Contents

Contents *(cont'd)*

Abstract: This report examines the security implications of Teredo. Teredo is a platform-independent protocol developed by Microsoft®, which is enabled by default in Windows Vista™. Teredo provides a way for nodes located behind an IPv4 NAT to connect to IPv6 nodes on the Internet. However, by tunneling IPv6 traffic over IPv4 UDP through the NAT and directly to the end node, Teredo raises some security concerns. Primary concerns include bypassing security controls, reducing defense in depth, and allowing unsolicited traffic. Additional security concerns associated with the use of Teredo include the ability of remote nodes to open the NAT for themselves, how it may benefit worms, ways to deny Teredo service, and the difficulty in finding all Teredo traffic to inspect.

Introduction

IPv6 is the next version of the Internet Protocol, and many hosts and networks are being upgraded to support this version and take advantage of its features. A part of the Internet that is expected to lag behind in IPv6 availability are the IPv4 Network Address Translation (NAT) devices used in many household and organizational networks. They are only infrequently updated or replaced, especially on small networks such as those found in residences. However, transition mechanisms that tunnel IPv6 directly over IPv4, such as the Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) and 6to4, do not typically work through NATs.

Microsoft is making a strong push for IPv6, and in response has developed a transition mechanism to address this issue. Fortunately, the mechanism was routed through IETF channels, and the IETF has published RFC 4380 as a standards-track individual submission. Originally the protocol was called Shipworm (after a species of mollusk that digs holes in ship hulls, analogous to what the protocol does with NAT devices). But the protocol has been renamed Teredo, after a common genera of shipworms (perhaps to avoid any negative connotation).

Teredo is already in use on the Internet. It is available in Windows Vista and Longhorn, where it is enabled by default. Teredo is also available in Windows XP SP2 and Windows 2003 SP1, although disabled by default.[4] At least one third-party implementation of Teredo is available for UNIX and Mac® OS X.[2]

Teredo is specified to be an IPv6 provider of last resort, not to be used when a native IPv6 connection or ISATAP/6to4 is available. It is also meant to be a temporary solution, with its retirement intended to be automatic due to disuse. (However, we anticipate that the availability of Teredo will to some extent slow down the deployment of other IPv6 methods, because it reduces the incentive for ISPs to provide native IPv6 connectivity and for users to upgrade their NAT and other perimeter devices.) While the use of Teredo will eventually diminish, Teredo services will certainly be available on the Internet for longer than actual use would necessitate.

For an IPv6-capable node behind an IPv4 NAT, the barrier to sending and receiving packets from IPv6 peers is that at least a portion of the network between the IPv6-capable node and the peer does not support IPv6. This includes at least the NAT. To resolve the problem, Teredo establishes an open-ended tunnel from the client, through the NAT, to a dual-stacked node on the Internet. IPv6 packets are tunneled through a single User Datagram Protocol (UDP) port on the NAT.[1] Thus, each IPv6 packet is inside a UDP header, which is in turn inside an IPv4 header.

Recall that NATs map internal ports and addresses to external ports and addresses. A pure cone NAT passes all packets through a mapped port, but a restricted NAT accepts them only from past recipients, which introduces extra work for Teredo. A symmetric NAT exists, but does not work with Teredo unless specifically configured. (For more details on NAT types, refer to "Security of NAT types" section.)

We feel that the use of Teredo has important security implications, and these implications are the focus of this report. Little published research exists on this topic, other than the "Security Considerations" section of the Teredo RFC itself. John Spence of Command Information includes a brief mention of Teredo in the "IPv6 Security and Security Update,"[6] and suggests disabling it since it "defeats IPv4 NAT." This report is based on the RFC and does not consider specific Teredo implementations. In the future, we plan to review Teredo on Windows Vista.

The report is organized as follows: an overview of how Teredo works; our analysis and discussion of Teredo security considerations; our conclusions; and future work.

Overview: How Teredo works

This section is meant to help the reader understand the material in this report. For more details and authoritative information, review RFC 4380. We have interpreted some of the RFC terms to make the content easier to understand, but reference the corresponding RFC terms as well.

Teredo works by tunneling IPv6 over an IPv4 UDP port for at least the portion of the network that is IPv4 only. Teredo has a high degree of automatic tunnel setup.

Teredo components

The Teredo framework consists of three basic components: clients, relays, and servers. *Teredo clients* are nodes seeking to use Teredo to reach a *peer* on the IPv6 Internet. For example, a node may need to reach an IPv6-only server. Clients are dual-stack (IPv4 and IPv6) nodes that are "trapped" behind one or more IPv4 NATs. Teredo clients always send and receive Teredo IPv6 traffic tunneled in UDP over IPv4 (see Figure 1).

[1] In this paper, ports refer specifically to IPv4 UDP ports unless otherwise noted.

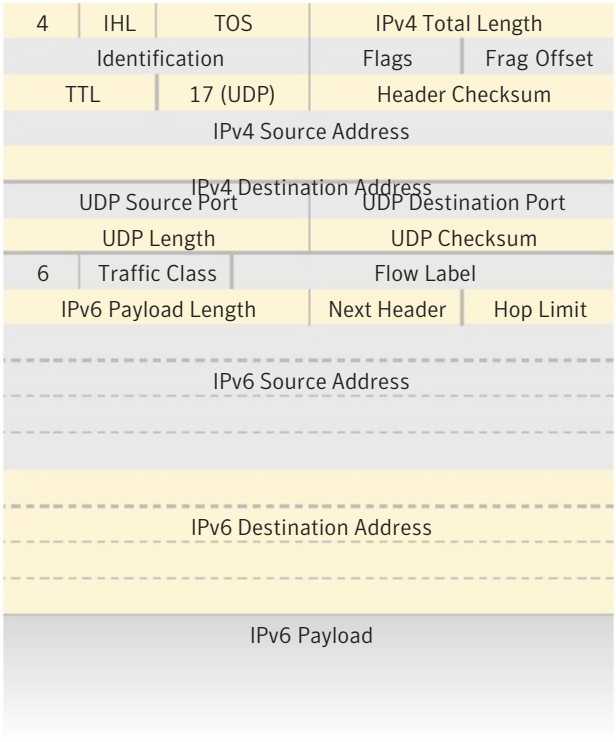


Figure 1. Teredo encapsulates IPv6 packets in UDP over IPv4 when packets are routed as IPv4.

The Teredo component on a client adds the tunnel headers on IPv6 packets being sent out by an application (encapsulation) and removes the tunnel headers from application-bound incoming traffic (decapsulation), thereby abstracting away the IPv6 connectivity method from the application.

Teredo relays serve as routers to bridge the IPv4 and IPv6 Internets for Teredo nodes. IPv6 native packets are encapsulated for transmission over the IPv4 Internet (including the client); when packets are received from the IPv4 Internet, they are decapsulated into native IPv6 packets for the IPv6 Internet. The peers need not know that the node they are communicating with is using Teredo. A special case is a host-only relay, which serves as a relay for the local host only. Connections between a client and a peer use the relay closest to the peer.

Teredo servers help clients set up tunnels to IPv6 nodes, determining their Teredo address and whether their NAT is compatible with Teredo. Like relays, Teredo servers sit on both the IPv4 and IPv6 Internets, but do not serve as a general relay. Teredo servers pass along packets to and from the client, but only messages that pertain to the functioning of the Teredo protocol; they do not pass along data packets.

The Teredo servers are generally statically configured on the client. For example, Windows nodes by default use "teredo.ipv6.microsoft.com" as their server; this currently resolves to four servers (or at least four IPv4 addresses) that Microsoft maintains. There may not be many Teredo servers on the Internet due to the need for static configuration, and due to the seemingly limited benefit creating their own server would provide to organizations and ISPs.

Figure 2 illustrates examples of these components and where they could be situated.

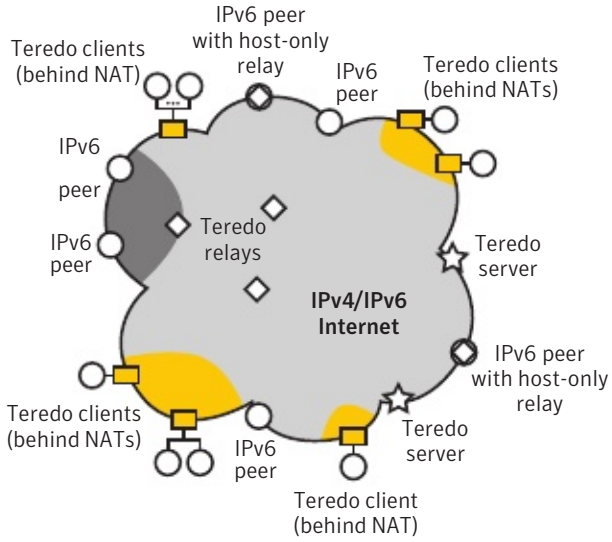


Figure 2. A Teredo microcosm, including key Teredo components, native IPv6 nodes, and IPv4 NATs. The cloud represents the Internet, where the yellow areas are IPv4 only, the dark gray area is IPv6 only, and the mixed gray area supports both. The interior of the cloud represents Internet routers and infrastructure.

The standard port on which the Teredo servers listen is UDP port 3544. Both clients and relays can use any UDP port for their Teredo service, so their UDP service port could be ephemeral. Because the client is behind an IPv4 NAT, the external port number of its Teredo service is, in general, not the same as the local port that is listened on. However, the Teredo protocol tries to keep that external port number stable since it is the port to which the relays need to connect.

Servers are specifically designed to be stateless, so a large number of clients can be accommodated. Clients and relays, by contrast, are stateful and maintain several state variables, as described in RFC 4380.

For example, clients and relays maintain a cache of recent peers and even a queue of packets to be sent when possible.

Teredo setup

Before packets can be sent to and from remote IPv6 nodes, some tunnel setup communication occurs. The phases are as follows:

1. The client completes a qualification procedure (see “Qualification procedure” section) to establish a Teredo address.
2. The client determines which relay to use (see “Packet relaying and peer setup for non-Teredo peers” section) for a given IPv6 peer node. This phase may involve a procedure to set up the NAT for traffic from the relay (“Bubble packets and creating a NAT hole” section).
3. A packet is sent via a relay.

The first phase needs to be conducted only once (for each time Teredo is activated on the client). The next two phases are completed for each peer that was not recently used. After that setup, it is just a matter of sending the packet via the relay. The relaying and per-peer setup take a special form when the remote

peer is also a Teredo IPv6 address ("Packet relaying and peer setup for Teredo peers" section). A special provision (outside the scope of this report) allows IPv6 nodes behind the same NAT to find each other by using an optional local client discovery procedure.

Teredo addresses

Teredo clients (and only Teredo clients) receive a specially formatted IPv6 address called a Teredo address. Addresses contain enough information for a relay to reach a client (see Figure 3).

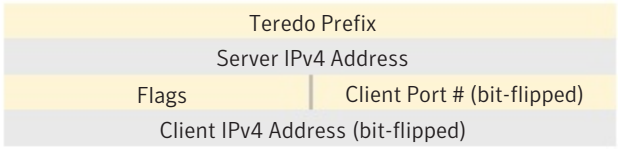


Figure 3. The format of a Teredo address. Like all IPv6 addresses, it is 128 bits (16 octets) long.

The prefix is standard for Teredo addresses; 2001:0000::/32 was recently assigned. You might see other prefixes, such as 3ffe:831f::/32, used in Teredo components that predate the current assignment.

The second 32 bits of the address correspond to the IPv4 address of the client's Teredo server. This part of the address tells remote nodes which server is assisting the client with communication setup.

The bottom 48 bits correspond to the client's external address and Teredo service port. This part of the address indicates to relays where to send packets destined directly for the client. To protect these two fields from any NAT translation, all of the bits in these fields are reversed.

The flags field is 16 bits, but only 1 bit is assigned by the RFC. The top bit is the "cone bit." If set, the cone bit indicates that the node is behind a pure cone NAT; if unset, it indicates the node is behind a restricted NAT. The rest of the bits in the field should be set to 0.

An example Teredo address is 2001::4136:E37E:8000:EEFB:3FFF:DD59. This format corresponds to a Teredo client behind a pure cone NAT that is using the server at address 65.54.227.126 (4136:E37E), and to which its NAT has assigned (mapped) the address 192.0.34.166 (3FFF:DD59 with each bit reversed) and port 4356 (0xEEFB with each bit reversed) for its Teredo service port.

Origin data

When a Teredo server sends an IPv6 packet to one of its clients on behalf of an IPv4 host, it adds additional data between the UDP encapsulation and the IPv6 packet. This is the origin data (see Figure 4) and reflects the IPv4 address and port number that it acts on behalf of. (The RFC calls this origin encapsulation.)

As in Teredo addresses, the port number and address have all their bits reversed. The client concludes that extra data is present, as the first nibble after the UDP header is 0 instead of 6 (the version number from the IPv6 header).

Figure 4. The format of the origin data, which is located below the encapsulated IPv6 packet.

Qualification procedure

The qualification procedure determines if a client can use the Teredo service and establishes the Teredo address. For example, a client cannot use the Teredo service if it is behind a symmetric NAT. A portion of the Neighbor Discovery Protocol (NDP, RFC 2461) is used, with the Teredo server acting as the router.

During qualification, the client sends Router Solicitations (RSs); the server then sends back Router Advertisements (RAs) plus an origin data block (see "Origin data" section) in response. Both the RA and RS messages are encapsulated ICMPv6 packets. Since the RA is sent in response to an RS from the client's Teredo service port, the origin data reveals to the client its external Teredo address and port number. That data becomes part of the client's Teredo address.

Qualification begins with the client sending an RS to the server with the cone bit set. Setting the cone bit means the client is trying to determine if it is behind a pure cone NAT. When it sees the cone bit is set, the server sends the RA from a different IPv4 address to the one that it received the packet on. If the client is indeed behind a pure cone NAT, the NAT passes the packet to the client. However, if the client is behind a restricted NAT, the NAT will not pass the packet to the client because the source is not a previous destination.[2]

If the client receives the RA, it knows it is behind a pure cone NAT and concludes qualification. The client forms a Teredo address with the cone bit on.

However, if the RA is not received, it could be due to packet loss. So after T seconds of waiting (default is 4 seconds), the client tries again, up to N times total (default is 3).

If the client still doesn't receive the RA, it tentatively assumes it is behind a restricted NAT and sends the RS with the cone bit unset. Since the cone bit is off, the server responds from the same address as it received the RA from. If this attempt does not succeed after N times of waiting T seconds, the client gives up, assuming a server connectivity problem.

However, if the client does receive an RA, it knows its Teredo address (cone bit off), but needs to do another check. The client sends the RS again, but to a different server address. Assuming the client receives a response, it compares the origin data on that response with the origin data from the previous server. If they differ (i.e., the NAT used a different external port), the client concludes it is behind a symmetric NAT and cannot use Teredo. If the data matches, the client concludes qualification. If there is no response after N times of waiting T seconds, the client gives up.

Note that the NAT would eventually discard the mapping between the client's Teredo service port and the external address and port that is represented in the client's Teredo address. To keep the address valid, if no communication with the server has occurred recently (as tracked by a state variable on the client), the client sends an RS to the server with the same cone bit status as in the Teredo address. The origin data on

[2] One hopes it is not a recent destination, at least. We could see this leading to some confusion.

the resulting RA will be cross-checked against what is currently in use. The amount of time that passes before the client sends the RS varies. The duration is calculated as a random percentage (between 75 and 100 percent) of the client's Teredo refresh interval. This interval is 30 seconds by default, but can be adjusted using the optional refresh interval determination procedure (not covered in this report).

Secure qualification

Teredo provides an option for the qualification procedure to be "secured" by adding authentication data (the RFC calls this authentication encapsulation) between the UDP header and the origin data with the encapsulated packet. Without this data, the client would not know that the response is sent from the real server (versus having received a randomly sent RA). The authentication data takes the format shown in Figure 5.

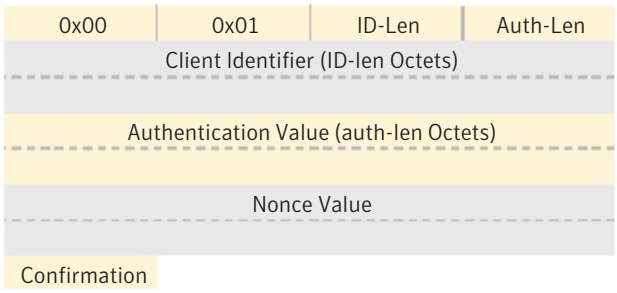


Figure 5. The general format of the authentication data. In secure qualification, this data is positioned after the UDP header.

Here, the client identifier and authentication value are optional and have their specific length indicated in one-octet fields. The nonce value is always present and always 8 octets in length; it is a random number chosen by the client and repeated by the server in the response. This simple measure establishes (with high probability) that if there is an attacker, it is at least on-path between the client and the server. Figure 6 shows the layout of the authentication data in this simple case.

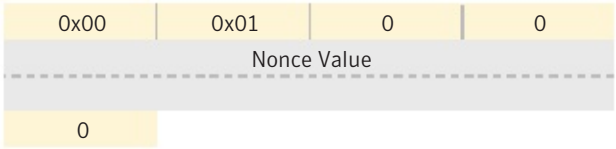


Figure 6. Authentication data at it simplest, when there is no client identifier or authentication value.

The authentication value (if present) is a keyed cryptographic hash of most of this header, the origin header, and the IPv6 packet. By default, the hash is based on HMAC and SHA1. This measure provides stronger protection against tampering and can help ensure that the server is the one intended. The RFC is not specific on the value of the client identifier, but it can relate to the authentication value. The confirmation byte is non-0 if the client should obtain a new key.

Bubble packets and creating a NAT hole

Teredo makes use of what the Teredo RFC refers to as bubble packets. These are simple IPv6 packets with no IP payload; that is, the IP payload length is 0, and the Next Header field has the value 59 (No Next Header).

These packets manipulate a NAT into allowing the real traffic. A typical use is when a relay needs to send a packet to a Teredo client, but the client is behind a restricted NAT (as evidenced by the cone bit being unset), and the relay is not a previous (recent) peer with that client. This circumstance prevents direct communication, so the following bubble-to-open procedure (see Figure 7) takes place:

1. The relay sends an encapsulated bubble packet to the Teredo client's server with the IPv6 destination set to the Teredo peer. The server address is extracted from the client's Teredo address.
2. The server passes the bubble along to the Teredo client, adding origin data (the IPv4 address and port of the relay).
3. The NAT receives the packet and passes it on to the client. The NAT allows this because the client and server communicate on a regular basis.
4. Upon receipt of the bubble, the client sends an encapsulated bubble to the address and port in the origin data (the relay).
5. The encapsulated bubble is received by the NAT and forwarded to the relay. The NAT now sees the relay as a recent peer and allows incoming packets from it.

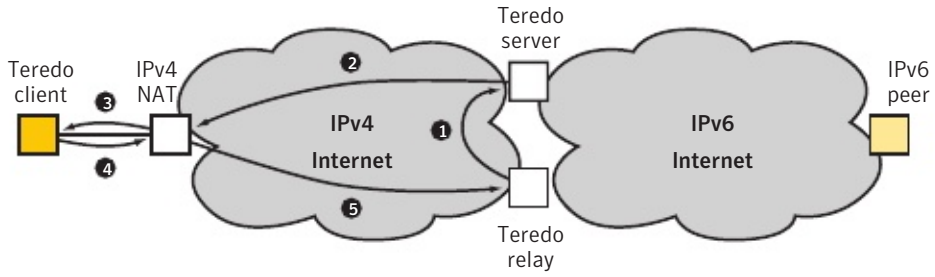


Figure 7. The bubble-to-open procedure opens a restricted NAT's port to a relay. To do this, the relay asks the server to ask the client to send it a bubble packet

Thus, Teredo provides an on-demand service that allows packets from arbitrary Internet hosts to be passed to the client. For a Teredo client's service port, the service makes a restricted NAT resemble a pure cone NAT. This concept is explored further in "Teredo implications on ability to reach a host through a NAT" section.

In any case, the RFC requires rate limiting of the bubbles sent to a specific peer, to protect against flooding. A bubble SHOULD NOT be sent if one was sent in the last 2 seconds or if four were sent in the past 5 minutes without receiving any direct responses.

Packet relaying and peer setup for non-Teredo peers

In this section, we discuss how relaying is set up and performed when the remote peer is not a Teredo node. The "Packet relaying and peer setup for Teredo peers" section covers the client-to-client case.

Relays serve as a bridge between the IPv4 Internet (including the Teredo client) and the IPv6 Internet (native hosts and 6to4 nodes). Relays encapsulate traffic in the direction of the Teredo client and decapsulate traffic in the direction of the plain IPv6 node. Figure 8 shows this behavior in the general case; Figure 9 shows the behavior for a host-only relay.

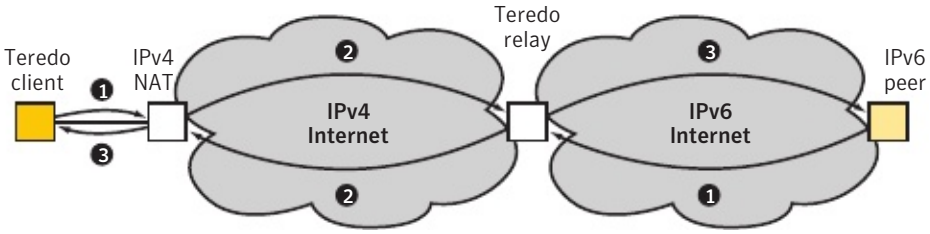


Figure 8. Sending and receiving a packet through a network-based Teredo relay. Encapsulation and decapsulation take place when passing through the relay (and on the client).

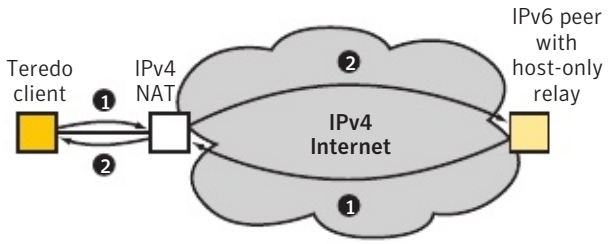


Figure 9. Sending and receiving a packet through a host-only Teredo relay. Encapsulation and decapsulation take place on the Teredo-aware remote peer (and on the client).

In the direction of the client, the relay may need to use the bubble-to-open procedure to open the client's NAT to allow packets to reach the client. However, a relay must be located before the communication can occur. The following two subsections describe how this is accomplished for the IPv6 and IPv4 sides.

Finding a relay from IPv6

From the IPv6 side, relay location is automatic since relays advertise that they have a route to 2001::/32. Thus, normal IPv6 routing takes place, though relays might be configured to serve only specific networks (e.g., those for a particular ISP).

In the special case of a host-local relay, the internal relay does not broadcast its route, but instead sets up a host-internal route for directly relaying. Such hosts would need to be dual-stacked with both IPv4 and IPv6 connectivity to the Internet. Note that in this case there is no native IPv6 traffic sent on the network at all; it exists only within the peers.

Ping test and finding a relay from IPv4

The ping test is a procedure used for a couple purposes in Teredo (the RFC refers to it as the Direct IPv6 Connectivity Test). In the procedure (shown in Figure 10), the Teredo client sends an ICMPv6 echo request (ping) to a remote IPv6 peer via the client's server.

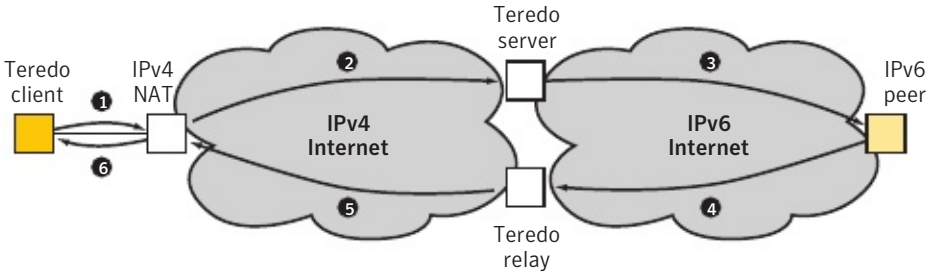


Figure 10. The ping test establishes which relay to use for a peer. The Teredo client sends an ICMPv6 ping to the peer via the client's server, and the peer responds back through the closest relay.

The server decapsulates the request and sends the ping directly over the IPv6 Internet to the peer. The peer then replies (assuming that it normally responds to pings). The reply is destined for the client's Teredo address and finds the closest Teredo relay (itself, in the case of a host-only relay). The relay then encapsulates the reply using the information found in the Teredo address. If necessary, the relay first employs the bubble-to-open procedure (see Figure 11) described in the "Bubble packets and creating a NAT hole" section and queues the reply until that completes.

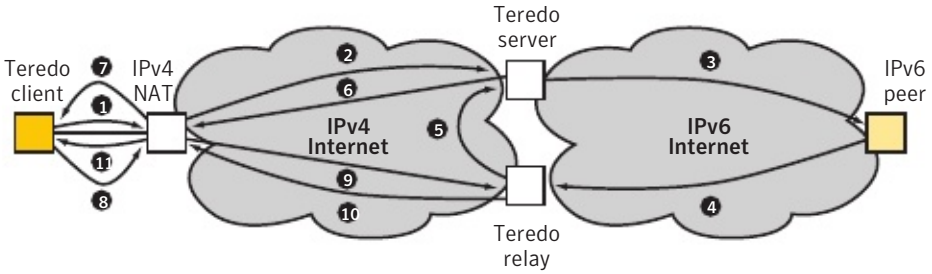


Figure 11. A ping test that required a bubble-to-open, which can be needed if the client is behind a restricted NAT.

In any (successful) case, the packet gets sent and makes its way via IPv4 routing to the NAT, which forwards it on to the client.

In creating the ping, the client sets the ping payload to a large random number, which the RFC suggests should be at least 64 bits in length. That value is checked in the reply as an assurance against spoofing. To spoof a reply, someone would need to either guess the random number used or be on-path. (Anti-spoofing measures in Teredo are explored in the "Teredo anti-spoofing measures" section.)

The ping test is used when a client is communicating with an IPv6 peer for the first time (recently). This applies when sending an outgoing packet or receiving an incoming packet. In both cases, it is the source IPv4 address and port from the ping reply that the client stores as the relay. The relay is used for outgoing packets to the peer, while incoming packets from the peer are checked against that address and port (to protect against spoofing). Note that since the relay address and port came from the ping reply, it would be difficult to spoof a different address as the Teredo relay closest to the peer.

Packet relaying and peer setup for Teredo peers

A specialized process allows Teredo peers to communicate with each other (see Figure 12). The client uses this process when it is sending a packet to a peer with an address starting with the Teredo prefix. In this process, both Teredo clients essentially act as their own host-only relays, sending packets over the IPv4 Internet to the peer's external IPv4 address and port.

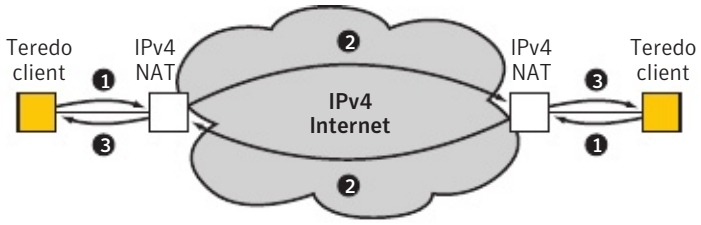


Figure 12. Sending and receiving a packet when the peer is a Teredo client. Each client directly encapsulates packets for the other.

Although there is no need to find a relay (and the appropriate IPv4 address can be predicted from the Teredo address), the NATs involved may require some preparation for the first (recent) communication.

If the destination is behind a pure cone NAT (as indicated by the cone bit in the address), the packet can be sent immediately. The remote NAT passes the packet, and if the local NAT is restricted, the destination IP becomes a previous destination, so return packets are allowed.

If the destination is behind a restricted NAT, the packet needs to be queued until a packet is seen from the remote address. It is not sent immediately because the remote NAT might not allow the packet in, and seeing a packet from the remote side suggests that it will now be allowed. If the local client is located behind a restricted NAT, a bubble is sent directly to the remote host; this may not succeed in reaching the remote client, but the main purpose of this packet is to set up the local NAT to allow packets in from the remote IP. In any case, the bubble-to-open procedure (see "Bubble packets and creating a NAT hole" section) is used to set up the remote NAT.

Trusted state

Both clients and relays maintain a state called "trusted" for each peer in their recent peer cache. What is meant by trusted varies by client and relay, and by Teredo peer and non-Teredo peer.

For non-Teredo peers on clients, having a trusted entry as well as a relay address that matches the client's records is the only way to guarantee that an incoming packet is accepted (RFC 4380 section 5.2.3). Were either of these not the case (there is no trusted entry or the relay address differs), the packet may be

accepted at the client's discretion, but the client is advised to do a ping test on the peer. The only way a peer becomes trusted is by receiving a ping reply that matches a previously sent ping. In this case, trusted means that anti-spoofing information is available and may be used.

For Teredo peers on clients, all well-formed packets from a trusted Teredo address are accepted by the Teredo client. If a Teredo address is trusted, the outgoing packet bypasses any setup-related bubbling that might otherwise be required. The state is set to trusted upon receipt of any packet from the peer (if the IPv4 source IP and port match the Teredo address). In this situation, trusted simply means that the address is already set up.

Relays maintain records for Teredo peers only. On relays, if a Teredo address is trusted, the packet bypasses any setup-related bubbling that might otherwise be required. The address becomes trusted upon receipt of any packet from that address. In addition, a peer entry created for an outgoing packet is set to trusted only if the address has the cone bit set (not a restricted NAT). Thus, in this case, trusted also means set up.

The term "trusted" may be confusing as there is no reputation-based reason for a peer to be considered trusted—particularly for relays and Teredo peers, for which the state simply means the NATs have been set up.

Required packet filtering

The RFC requires the Teredo components to conduct a number of checks on incoming (and sometimes outgoing) packets. While there are too many to list in this report, here are some notable ones:

- Does the authentication data check out?
- Are the UDP packet and contained IPv6 packet well formed?
- Is the RA or RS well formed?
- Is the IPv4 address in a Teredo address or elsewhere a global unicast address (not broadcast/multicast, loopback, RFC 1918, etc.)?
- Are non-Teredo IPv6 addresses global-scope unicast addresses?
- For servers: Is the IPv6 packet a bubble or ICMPv6?
- For relays: Is the destination IP a Teredo address?

Failure to pass these checks results in a silent discard of the packet in question and eliminates a number of possible attacks using Teredo.

Teredo security considerations

In this section, we discuss various Teredo security considerations. We focus on areas of concern, as well as on what the Internet would be like with and without Teredo. We often compare the reality that Teredo creates to what is possible on IPv4 and IPv6 networks. Note that this report is looking at the Teredo protocol itself (based primarily on the RFC), and not at any specific implementation. Additionally, no experimental results are included here.

Where noted, additional details on a topic may be available in the Security Considerations section (section 7) of the Teredo RFC.

Security of NAT types

This section compares the security of different NAT types specified in RFC 3489: pure cone, restricted cone, port restricted cone, and symmetric. This serves both as an introduction to the different NAT types (referred to in the Teredo analysis) and as a possible Teredo security implication.

Because symmetric NATs do not work with Teredo, there may be a move away from them to some form of cone NAT that Teredo does work with. The significance to security of that shift will depend on the extent to which the move happens, and the extent to which there is a security difference between symmetric and the new form. Besides Teredo, there may be other reasons for such a move, such as interactive games, VoIP, and peer-to-peer applications.

Some debate exists as to whether there is a difference in security properties between a symmetric NAT and a port restricted cone NAT. This report contends that a port restricted cone NAT is slightly weaker security-wise, at least with respect to inbound packets.[3] (RFC 4787 and "NAT Classification Results using STUN," [4] present an alternate view.)

To better relate to our perception of the relative differences in NAT security, in this section we assign a rating to different NAT types. This rating is based on our arbitrary scale, for which no range or mathematical definition exists.

NATs should not be considered security devices, but the restrictions they place on inbound traffic include a security benefit as a side effect. Since there are (very often) more IP addresses in use on the inside of the NAT than on the outside, a dynamic mapping of internal ports to external ports needs to be maintained by the NAT device. As a result, an inbound packet targeted at a port that has no mapping will not be routed inward to any port.[4] So someone wanting to route a packet in would need to find a mapped port. We give this NAT type a rating of 3 on our scale because of the amount of security-through-extra-work required.

The different degrees of restriction associated with different cone NAT types differentiate them in terms of security protection. A *pure cone NAT* places no restrictions on inbound traffic beyond the necessity of finding a mapped port. This NAT type has the rating of 3.

A *restricted cone NAT* limits incoming traffic to only traffic from IP addresses to which the internal address previously sent packets. Thus, an attacker would have to either own that peer IP address or be able to spoof it. With the ownership approach, the attacker's location is narrowed. While it is possible in general to spoof a source IP, this places some limits on the type of attack that could be attempted, and requires that the attacker discover or guess a previous (recent) peer IP. The additional burden that this imposes increases the security rating on our informal scale to 6.

[3] Since symmetric NATs cycle though external port numbers quicker, it is easier for a denial of service to be completed from inside the NAT with a symmetric NAT.
[4] Exceptions may exist.

Port restricted cone NATs have all the restrictions of a restricted cone NAT, plus require that the incoming source port is one that was used with the incoming source address in an earlier packet, sent out from the internal IP address. If either the source address or source port does not match a previous destination, the packet will not be routed in. It would not be difficult for an attacker to change a source port, but it does require the attacker to find one that works. Our designated rating is 8.

A type of NAT could exist where the set of previous outgoing IPs and ports (the basis of the inbound source restriction) is associated with a specific port and not with the internal IP as a whole. Following the RFC 3489 definitions, this is also a port restricted cone NAT, but we will coin the term "double port restricted cone NAT" to refer to this type of NAT. This places an additional burden on the attacker, so we move this up to a 9.

The difference between cone NATs and *symmetric NATs* is defined in the case where the same internal source address and port are sending a packet to a different destination IP or port than the one to which a previous packet was sent, and where the previous packet still has its associated map. Here, a cone NAT uses the same external IP and port as it did for the previous connection. However, a symmetric NAT maps the new quadruple (source IP, source port, destination IP, and destination port) to a different external port or IP address. From an external perspective, this means that by observing a packet being sent to one IP address, with symmetric NATs one cannot deduce the external port (and address) that was used for a different IP address. Thus, additional work is involved, and we rate the increase in difficulty to be worth about 0.5, for a rating of 9.5.

The definition of symmetric in RFC 3489 allows for the inbound restrictions to be similar to those found in restricted, port restricted, or double port restricted cone NATs. As a result, the symmetric rating would be between 6.5 and 9.5. However, we expect that typical symmetric NATs will behave like the double port restrictive case, resulting in a 9.5 rating. Here, in fact, at most one remote IP and port is accepted on any given mapped port. Note that even with a rating of 9.5, it is easy to craft a packet that will be routed to an internal port.

If NATs are moved from symmetric to cone, this would imply some drop in their security properties—to what degree depends on the cone type they are converted to. It would be reasonable to expect the NATs to retain the same incoming routing restrictions as before. If that is the case, the security decrease would be relatively small, but could be larger depending on the situation.

Teredo's open-ended tunnel (a.k.a. extra security burden on end host)

Teredo creates an open-ended tunnel through the NAT to the client. Teredo is designed as an IPv6 tunneling mechanism for end nodes behind a NAT. It works without the cooperation of any non-Teredo components. Additionally, since it is a new mechanism, pre-existing network-based security controls (for example, firewalls and IPSs) on the client's network do not see through the tunnel to apply the controls to the traffic being tunneled. One could therefore say that Teredo is evading those controls, which has to be a concern for those who set them up, since those controls are supposed to adequately regulate all traffic. In addition, it might be difficult to monitor or block Teredo traffic, as discussed in "Teredo mitigation" section.

If network controls are bypassed due to the use of IPv6 via Teredo, the burden of controls shifts to the Teredo client host. Since the host may not have full control over all the nodes on the network, security administrators sometimes prefer to implement security controls on the network. In addition, having both network controls and host controls provides defense in depth, a basic security principle.

Ingress filtering (the sanity-checking of incoming destination addresses) and egress filtering (sanity-checking outgoing source addresses) are most naturally done by the network. However, unlike what is presumably available for native IPv4 and IPv6 traffic, Teredo offers no opportunity for this filtering on IPv6 without special provisions on the network device. This applies as well to any other IPv6-based routing controls that sit between the relay and the client.

The "Allowed packet" section describes how the tunnel to the client is open-ended. "Teredo and IPv6 source routing" section shows how the failure of the client host to regulate incoming traffic can be used to forward traffic to other internal hosts (including those not using Teredo). "IPv4 ingress filtering bypass" section discusses an IPv4 ingress filtering bypass scenario enabled by Teredo. "Teredo and bot networks" section discusses Teredo and bots. And "Teredo implications on ability to reach a host through a NAT" section discusses the impact of Teredo on the ability to reach a host through a NAT.

Allowed packets

RFC-compliant Teredo clients allow packets to be sent to any IPv6 global unicast addresses and hence put few restrictions on outgoing packets. (Of course, a rouge client could send packets to any address type, and a node could also just pretend to be behind a NAT.)

Since Teredo is designed as a tunnel and provides that functionality, there are few restrictions on externally initiated packets reaching the Teredo client. Teredo provides a tunnel to which any peer can connect. As described in "Trusted state" section, when a client has the peer in a trusted state (and the relay address and port match what is expected), the client accepts any well-formed incoming packets. After the client receives a packet from a Teredo peer, it will be in a trusted state. Non-Teredo peers are trusted only after a successful ping test. At the client's discretion, packets from untrusted sources or where the relay address does not match may also be accepted.

Accepting incoming packets, of course, allows an attacker to exchange packets with the Teredo client host (and its associated network-facing components, such as the firewall). The easiest thing for the attacker to do here is to be a Teredo client itself—or at least pretend to be—in order to send the packet.[5] This results in the packet being accepted by Teredo. It is also not difficult for an attacker to use a non-Teredo IPv6 address, but it may require responding to a ping challenge.

In the following quote, the Teredo RFC discusses this security concern; we have interspersed the quote with our comments.

> "The very purpose of the Teredo service is to make a machine reachable through IPv6. By definition, the machine using the service will give up whatever firewall service was available in the NAT box, however limited this service may be [RFC2993]. The services that listen to the Teredo IPv6 address will become the potential target of attacks from the entire IPv6 Internet. This may sound scary, but there are three mitigating factors.
>
> "The first mitigating factor is the possibility to restrict some services to only accept traffic from local neighbors, e.g., using link-local addresses. Teredo does not support communication using link-local addresses. This implies that link-local services will not be accessed through Teredo, and will be restricted to whatever other IPv6 connectivity may be available, e.g., direct traffic with neighbors on the local link, behind the NAT."

[5] If cone bit of the target is 0, the attacker may be able to cheat in doing its own set up work by sending the packet via a relay (assuming the relay accepts packet from a Teredo source address).

We feel that this is asking a lot of other services because of a new service arriving. We also note that, while we have some concerns with it, the "IPv6 Transition/Co-existence Security Considerations" Internet Draft[1] offers contradictory advice, saying that the acceptability and use of link-local addresses should be limited.

The RFC continues:

> "The second mitigating factor is the possible use of a 'local firewall' solution, i.e., a piece of software that performs locally the kind of inspection and filtering that is otherwise performed in a perimeter firewall. Using such software is recommended."

This is definitely something that should be employed, and one can hope the firewall does a thorough job enforcing all desired security controls. However, defense in depth for the network has still been lowered unless all network-based controls are Teredo-aware, and all Teredo traffic can be feasibly identified. The RFC quote concludes:

> "The third mitigating factor is the availability of IP security (IPsec) services such as IKE, AH, or ESP [RFC4306, RFC4302, RFC4303]. Using these services in conjunction with Teredo is a good policy, as it will protect the client from possible attacks in intermediate servers such as the NAT, the Teredo server, or the Teredo relay. (However, these services can be used only if the parties in the communication can negotiate a key, which requires agreeing on some credentials; this is known to be a hard problem.)"

IPsec would help with security to the extent it is available and used. An additional possible mitigating factor, not mentioned in the RFC, is that an attacker would need to learn or guess the Teredo address of the client it wants to reach; "Finding a Teredo address for a host" section discusses this.

Teredo and IPv6 source routing

The source routing capability is built into IPv6 and is specified via the Routing header (RFC 2460), which sits between the IPv6 base header and the IPv6 payload. As in IPv4 source routing, IPv6 source routing allows a packet's sender to specify what nodes a packet should pass through on the way to its final destination. When a packet reaches the destination specified in the designation address field, there is a check for a Routing header; if there is one and this is not the final destination, the next destination is copied into the destination address field, and the packet is sent back out to the network. As in IPv4, best practice dictates blocking any packets containing source routing.

Teredo combined with IPv6 source routing opens up some attack mechanisms not mentioned in the RFC's Security Considerations section. The following discusses one mechanism, and "Selecting a relay source routing" section lists another.

Consider the case where a Teredo packet reaches a Teredo client (and is accepted) and the encapsulated IPv6 packet contains a Routing header. The Routing header indicates that this is not the intended destination (just a hop in a source route). By initial appearance, this packet could be normal inbound traffic, or it could be from qualification or a ping test. Unless the Teredo client has source routing disabled, it would pass the IPv6 packet on to the next hop. One way to use this for an attack would be to have the next hop be a node internal to the network the client is on. Another is to pass the packet back outside, using the Teredo node as a reflection point.

This behavior is not specific to Teredo packets; it works in the same way for all IPv6 packets. However, with native IPv6 packets, a gateway prohibition of source-routed packets would have prevented the packet from even reaching the internal host. With Teredo active, the burden is placed upon the end hosts, at least those running Teredo. Source routing post-Teredo may also be a surprising possibility (packets on an end-to-end tunnel not stopping at the end) that might not have been anticipated in network controls, especially given that a NAT was traversed in the process.

IPv4 ingress filtering bypass

For networks containing a Teredo client that has one or more non–RFC 1918 addresses behind the NAT, Teredo provides a novel way to bypass ingress IPv4 address filtering, although the technique requires specific circumstances to be of particular use. Even if the address in the low 32 bits of an incoming source Teredo address is an address on the internal network, a peer entry will be created for it (and marked as "trusted"). Now if the client decides to respond to the incoming packet (e.g., the packet is a request), the response will be an encapsulated IPv6 packet send to the internal address listed in the forged Teredo address. RFC 1918 addresses will not work here because the RFC requires that the client check if the destination IPv4 address is a global unicast address. The destination UDP port will be one the attacker chose in creating the Teredo address, but the attack would need to be one in which an encapsulated reply and some bubbles could be used as a vector. The Teredo client might be able to guard against this by judicious address checks.

Teredo and bot networks

The recently released 10th edition of the *Symantec Internet Security Threat Report* [7] states:

> "If [bot creators] begin to exploit an attack vector that bypasses firewalls and perimeter defenses [in order to create new bot-infected machines], the population of bot-infected computers could increase rapidly. This could be particularly dangerous because bot network owners have become more organized and experienced."

Teredo could provide a way for the firewalls and perimeter defenses to be bypassed. The attacker would then need only a usable vulnerability in the Teredo component or in something that is reachable inbound.

Teredo implications on ability to reach a host through a NAT

This subsection examines what impact Teredo has on the ability to reach a host through an IPv4 NAT, as compared with the non-Teredo case. The analysis is separated between pure cone NATs and restricted NATs, with the general situation followed by the Teredo-related impact. Note that the only IPv4 UDP ports that Teredo causes to be accessible are those that correspond to the Teredo service.

In order to ensure that a packet will reach a specific internal host behind a pure cone NAT, one needs to find an external port on the NAT that the internal host has recently had a local port mapped to. This may take up to 65,536 packets multiplied by the number of external IPv4 addresses (assuming there is even such a port available)—although the port reached on the internal host would be more likely to have something listening on the port than if one were to somehow scan a random internal port; in fact, any externally usable listeners would have a port open on the NAT. (Note that no knowledge of the internal IP address is required.)

With knowledge of the NAT or of the internal ports that may be open, the possible number of guesses could be reduced. For example, the NAT may not use all the available ports on an external IP in practice, or it may have a predictable mapped port for an internal port. This is especially true for port-preserving NATs that attempt to use the same external port number as the internal one. One carefully chosen port may be sufficient. Also, if one sees a packet coming from the NAT, the IPv4 source port is known to be a mapped port on the NAT (although the internal host it corresponds to would not necessarily be known).

A Teredo client active on the internal host has a couple of effects in this situation. First, there is a NAT mapping that is intentionally being kept open indefinitely. Depending on how the client chooses a local port number and how the NAT maps it, the port that it is on may be predictable as well.

Another effect is that this port number and corresponding IPv4 address are being made widely visible as part of the Teredo IPv6 address of the client. While the Teredo protocol itself distributes this address only on packets, peers and even network components such as Teredo relays may record the Teredo address in, for example, log files; the address may even make its way onto, for example, peer-to-peer host advertisements. This is an incremental concern over the non-Teredo case due to the fact that addresses are recorded more often than addresses plus their corresponding ports. In addition, the Teredo protocol contains more messages that are exchanged and with more parties, offering more chance for visibility into the source port and address in use, when compared with the straightforward IPv4 case.

With a restricted cone and a port restricted cone, the NAT does not allow a packet with just any external source address and source port to be forwarded into the network. Instead, the attacker must get the source address or port correct, as discussed in "Security of NAT type" section. The sources for a packet must match something that was previously used outbound for that external port. There may not be many of these that are accepted.

So if the attacker had to guess, the space from which to guess for restricted cone would be 2^{32} (since there is no port number filtering) and 2^{48} ($2^{32} \cdot 2^{16}$) for port restricted cone. In some cases, the set of correct guesses would be different for the different NAT ports tried on an IP address as well. The number of packets the attacker would need to try with this approach would be so large that the most realistic scenario is one in which the attacker already knows a recent source due to seeing the original packet or some record of the packet, or having knowledge of the target's habits.[6]

Introducing a Teredo client on a host behind the restricted NAT provides a significant advantage to the attacker. First, as described for the pure cone case, the mapped Teredo port and IP address are much more exposed than another mapped port would be. Additionally, Teredo provides a way for external hosts to connect into a client behind a restricted NAT. Specifically, the attacker can use bubble-to-open ("Bubble packets and creating a NAT hole" section), thereby opening a hole in the restricted NAT for the attacker. (You might say the attacker is pretending to be a relay here.) The client's server is located in the client's Teredo address, for additional convenience. Note that the need to spoof an address is eliminated as well.

[6] In either case, spoofing an arbitrary source address has its limitations, the main one being that it is difficult to see any packets sent as a result of the original packet. In some cases, this does not matter. However, for port scanning to be useful, an attacker must be in a position to see the responses to his probes. For this reason, scanning is often conducted without using spoofed source addresses (for probes, decoy packets are sometimes used as a distraction). The alternative is to be in a network location between the target and the network location of the source address used. The opportunistic attack in which a host is attacked after it contacts a malicious or compromised host does not face these problems.

Information revealed to third parties

Teredo does not result in much client information being revealed to third parties, beyond what direct communication over IPv4 or IPv6 would provide.

The Teredo address, which is easily identified and shared to all involved parties except the server, contains a few pieces of information:

- **Server address:** The server being used by the client may reveal a small amount of interesting information about the client. For example, if the server is a Microsoft server, one might guess that an address corresponds to a Windows client. One could envision this being used to select Teredo addresses to attack.
- **Client IPv4 address:** The address could be the same one used if the communication were over IPv4, so this is not very revealing. If the NAT only has one external IP address, then the address would definitely be the same.
- **Client port number:** This is somewhat sensitive because the Teredo protocol is keeping this port mapped to a Teredo client. In IPv4 communication, the source port is often not that way. IPv6 nodes with direct Internet connectivity inherently reveal a way to reach the node via IPv6, but no additional information is revealed by the port.
- **Cone bit:** Whether the NAT is pure cone or restricted cone is revealed. The existence of the Teredo address suggests that the NAT is not a symmetric NAT. This sensitive information may help attackers select and optimize attacks with Teredo. Moreover, if attackers sees a Teredo address with the cone bit off, they might assume that network would be easier to attack due to the weaker inbound restrictions.

The Teredo server has reliable access to all of these address components except the flags field. The server may have to guess the final state of the cone bit. (See “Finding a Teredo address for a host” section for more on the “guessability” of a Teredo address.) The other item shared with the server is a link-local address that the client included with its RA. It is not clear what would be in the host part of that address; conceivably, it could be something moderately interesting.

Note that the server does not see any data packets (unless it is also operating as a relay). However, the server could learn all the addresses of the client’s intended IPv6 non-Teredo peers, since the client uses the server for the ping test. A malicious user or program (e.g., spyware) could secretly change a client’s Teredo server setting to a malicious server, for the purpose of monitoring connections (at least those over IPv6). If the server indeed provided correct service, the user probably would not notice the switch. The closest layer-3 analog to the attack for native IPv4 or IPv6 is changing the router in use, but that is less likely due to proximity requirements. This is most similar to changing one’s HTTP proxy setting; although in that case the scope is a single protocol.

Teredo anti-spoofing measures

This section discusses Teredo and address/host spoofing for both peers and servers. On the Internet, it often is not difficult to spoof an address. Also, without extra measures on a local network, it is not possible to distinguish traffic from a remote host from traffic pretending to be remote; for Teredo this means that, notwithstanding the following, a local host can send traffic while pretending to be an external Teredo server, relay, or client.

Peer address spoofing

Teredo has some protection against spoofed peer IPv6 source addresses. The basis of the mechanism depends on whether the peer IPv6 address is Teredo or non-Teredo. For non-Teredo addresses, the mechanism is based on the ping test that is completed for new peers (section "Ping test and finding a relay for IPv4). That establishes (in a fairly secure way) the peer relay's IPv4 address and port; in practice, the security of the association depends on the nonce (ping payload) length and the difficulty in predicting the nonce value. The client has the option to ignore packets purporting to be from a peer IPv6 address if the IPv4 address or port does not match, or to hold off delivery of packets for which the ping test has not been completed. The ping test would fail, so it is necessary that the spoofed source host be a live host (and willing to respond to pings).

The Teredo peer address scenario is simpler. An algorithmic relationship exists between the IPv6 address and relay IPv4 address. (Recall that Teredo clients serve as their own relays when communicating with Teredo peers.) The client has the option to not accept packets from the IPv6 address unless the IPv4 address and port match what is encoded in the IPv6 address.

There are a couple of realistic ways around this:

- In the non-Teredo case, a host behind the same relay as the address to be spoofed would have the same IPv4 and port, and hence be successful in spoofing. This requires knowledge of the relay for the spoofed source and a specific location in the network (unless source routing could be used; see "Selecting a relay via source routing" section).
- An IPv4 node that can spoof source addresses can craft a packet that appears to come from the relay (i.e., it has the right source IPv4 address and port). In non-Teredo cases, this requires knowledge of the address and port of the relay for the spoofed source.

Both of these cases are analogous to spoofing possibilities often present in native IPv4 or IPv6 cases, but the bar has been raised a little for the non-Teredo case because the relay address must be known and a live host must be used as the source.

Stronger anti-spoofing could be achieved by using IPsec, which is compatible with Teredo. In fact, the Teredo RFC's Security Considerations section states:

> "The Teredo nodes can use IP security (IPsec) services such as Internet Key Exchange (IKE), Authentication Header (AH), or Encapsulation Security Payload (ESP) [RFC4306, RFC4302, RFC4303], without the configuration restrictions still present in 'Negotiation of NAT-Traversal in the IKE' [RFC3947]. As such, we can argue that the service has a positive effect on network security."

That is a rather narrow view, of course. It also assumes the availability of the infrastructure required to support authentication, and does not help when communication with previously unknown parties is acceptable. It could, however, help with confidentiality and data integrity.

Server spoofing

For communications between the client and server, a good degree of anti-spoofing protection is provided by the authentication data used by secure qualification (see "Secure qualifications" section). For example, the nonce plays the same role as the random payload of the ping in the ping test. This protection requires that secure qualification be used. However, there do not seem to be any barriers to using secure qualification, at least for nonce-only.

Server spoofing is discussed in section 7.2.1 of RFC 4380, which points out that it is possible to spoof the server, even if the authentication and client ID are used. The attacker could set up as a man-in-the-middle. However, the gain does not seem worth the effort.

Denial of Teredo service

This section discusses methods for creating a denial of Teredo service and its impact. Servers, relays, and the remote node are key components in communication and can obviously cause a denial of service (DOS) if they are malicious or compromised. Our remaining discussion is on external parties causing the denial of service. From our analysis, we conclude that Teredo should not be relied upon to always be available.

Storage-based attacks

In a couple of situations, Teredo processing requires queuing up packets for possible later transmission. If attackers are able to force a Teredo component to queue up many packets—especially large packets—they may be able to cause a denial of service, perhaps taking the form of legitimate packets not being queued or delivered, or new peers not being reachable.

Teredo relays can queue up packets destined for Teredo clients behind restricted NATs for which setup is not complete. This optional (expected to be implemented) behavior allows time for a bubble (designed to set up the possibility of inbound communications though the NAT) to make its way back to the relay. In the case of a failure, this can take 6 seconds (three tries with a 2-second timeout each). The RFC suggests that relays limit their queuing to guard against such a DOS. Attackers may have the best chance of success if they generate several large packets (perhaps pings) for each of several targets, in a short amount of time. The targets would be nonexistent Teredo hosts with the cone bit unset.

Clients also maintain a queue of packets destined for untrusted destination addresses (IPv6 addresses for which the client does not know what relay to use or for which the NAT has not been prepared). This is required for non-Teredo peers and for Teredo peers that are behind restricted NATs. As described in "Inducing a client to make external connections" section, it probably is not difficult to induce a client to connect to multiple destinations. Nonexistent and nonresponsive addresses would be most effective here, and queuing is up to 6 seconds.

Clients and relays are also required to maintain a cache of recent peers, along with specific data about each peer. If someone were to exceed the number of peers that can be maintained, then a denial (or at least degradation) of service would result. For a client, "Inducing a client to make external connections" section discusses ways to do this, and per the Teredo RFC (section 7.3.3), this would essentially prevent direct connections with peers, but would last only as long as it was sustained.

The number of peers on a relay could also be exceeded by one or more IPv6 nodes sending packets via the relay to many Teredo destinations. The result is that for any destination not currently in the cache, the relay sends a bubble via a server to the client and will probably hold the packet until it is returned, introducing a significant delay. The attacker would not care so much about this happening to their packets, but legitimate users of the relay would notice the delay, possibly for each packet they send to a Teredo host. For local host relays, the request would have to be initiated locally, by techniques similar to those described in "Inducing a client to make external connections" section. Teredo servers are stateless and obviously not subject to these storage-based attacks.

Relay DOS

If some means (such as the previous example or even brute force) is used to create a denial of service condition on a network-based relay (or the relay is unavailable for some other reason), communication using that relay will fail. Per section 7.3.5 of the RFC, this will continue for at least as long as the relay continues to announce the reachability of 2001::/32.

If the client were trying to send a packet when the relay was unavailable, the RFC does not seem to have a provision for the client to try to establish a new relay. On the other hand, the peer would normally have no awareness of a Teredo relay being in use and would send a packet to the Teredo address. When the routing system recognizes that the relay is no longer usable, the next closest relay would be found. When the packet arrives at the client (the new relay may need to bubble-to-open first), the client will notice that the IPv4 address and port do not match what is expected. The RFC leaves it up to the client whether to accept that packet immediately, perform a ping test first, or discard the packet.

To guard against recovery though moving on to the next relay, the attacker may try to take out multiple relays. It is not clear what depth of relaying is likely to be commonly available, or even if another relay is likely to be available at all.

Server DOS

It may be possible to achieve denial of service through a brute-force attack on the server bandwidth or processing speed. If the server supports the authentication value as part of security qualification, it needs to compute this in response to any valid qualification request. Multiple clients could make requests at the same time, possibly causing a DOS due to the expense of computing it. The server is stateless, so the same request could probably be sent repeatedly, reducing the load requirement on the attacker side.

If a DOS is successful against a server (or the server is otherwise unavailable), its clients will not be able to requalify their address, nor will they be able to establish communications with new peers (except for incoming in some situations). To recover from this, RFC 4380 (section 7.3.5) indicates that the client would need to be ready to fail over to a new server. That means the client would need to obtain a new Teredo address to communicate with that server. (The client might decide to keep the old address in case it receives packets.)

Scanning Teredo addresses compared with native IPv6 addresses

This section presents an analysis of the difficulty of scanning for Teredo addresses relative to the difficulty of scanning for native IPv6 addresses. As a first step, we look into the difficulty of finding a live Teredo address in different scenarios: for a specific host, within an external IPv4 address, and on the Internet. There are two general approaches: The first is to find a Teredo address somewhere on the Internet, as discussed in "Teredo implications on ability to reach a host through a NAT" section. The other is to guess and verify the address. For verification to work, one would need to find a packet that produces a different result for a valid Teredo address than for an invalid (unused) one. If possible, it would be desirable to use one's own IPv6 address as the source, so results could easily be seen.

Finding a Teredo address for a host

This subsection considers the difficulty of finding the Teredo address for a particular host inside a NAT. We have discussed the chance of finding the whole Teredo address through its use; here the focus is on deriving or guessing the address. We assume the seeker already knows the external IP address of the host. If not, there would be a large number of external IP addresses to scan, and then determine if the correct host was found. If the NAT has several IP addresses, the following approaches could be tried for each.

Recall the format of a Teredo address: The first 32 bits (the global Teredo prefix) is always a given and the last 32 bits represent the NAT's external IP address. The middle 32 bits are the Teredo client's server. Depending on the client, this may be fairly easy to guess. For example, Microsoft provides four servers, and these are the defaults for the Windows Teredo component. In any case, the number of Teredo servers, especially popular ones, is likely to be a couple score at most.[7] So, this field has low entropy, say no more than five.[8]

Assuming the flags in the address are set per the RFC, then the 16-bit flag field has only one bit to guess—the cone bit. Next is the 16-bit external port number that corresponds to the client's Teredo service. As previously discussed, this may be predictable; even in the worst case, there are 2^{16} possibilities. So an attacker may have to guess $32 \cdot 2 \cdot 2^{16} = 4{,}194{,}304$ addresses, but there are realistic cases where the number may be more around 8 or 16.

The expected number of guesses required would be half that, assuming that the fields cannot be independently tested or narrowed down. However, there may be ways to achieve such a reduction. For example, one could start with a guess that the host is behind a cone NAT. Then, the NAT's open ports could be scanned, perhaps trying best guesses at the Teredo service port first. This would use some likely-to-exist packet that produced a different result if the NAT had the port unmapped or if the packet reached a service. Only when a port was open would Teredo addresses with that port number be tried. Even better would be if the packet could specifically distinguish Teredo service ports. If that did not find the host, then non-cone addresses could be tried as normal.

Microsoft's *Teredo Overview*[4] indicates that the Vista and Longhorn implementations of the Teredo client use the flag fields a bit differently than the RFC specifies. Whereas the RFC indicates that the unassigned bit fields must be set to zero and ignored upon receipt, the Microsoft implementations have the client randomize the value of 12 of those bits when determining its address, as a measure against address scans by malicious users. Assuming those bits have no predictability, that should help by increasing the address space to guess by $2^{12} = 4096$, in all guessing cases.

[7] If small or mid-sized ISPs turn out to frequently provide Teredo servers, this may not be a good assumption. That does not seem likely, but at this point there is no way to know.
[8] One way to get an entropy of 5 is to have 32 possible server IPv4 addresses that are equally probable. Regardless of the actual distribution, it also means that if one ordered guesses by decreasing probability, the average number of guesses, before one found the correct answer, would be 16.

Finding any Teredo address for an external IPv4 address

The difficulty of finding the address of any Teredo client behind a given IPv4 address depends on the number of Teredo clients behind the address, which shall be designated N. N might represent one or two for home networks, but be much more for large organizations and for ISPs that provide only RFC 1918 addresses to customers. Obviously, if N is 0, it would be impossible to find any such address. The chance of coming across any Teredo address for an IPv4 address through its usage would seem to be approximately N times higher than in the case where we were looking for the Teredo address of a particular internal host and knew the external IPv4 address ("Finding a Teredo address for a host" section).

For the guessing case, the situation is similar to the scenario described in "Finding a Teredo address for a host" section, and the address space to guess is also the same. However, the expected number of guesses could be less, although the adjustment factor is not entirely clear. The guessing for all the bits except the external port number would be unaffected. In the worst case, it is not possible to predict what external port number a given Teredo client will be using; in which case, the space from which to guess is 2^{16}; looking for any address instead would cut down the expected number of guesses required by N to $2^{16}/N$.

In the case of a port preserving NAT and Teredo clients that favor a certain source port number, there is a high likelihood that there will be mapping from that source port number on the outside, regardless of the size of N (if N>0). For all other cases, the ease of guessing would fall in between.

In short, the more optimized the guessing of external port numbers is for finding a particular host, the less the advantage gained by looking for any 1 of N hosts. The range then is $2^{16}/N$ to 1 for expected number of guesses required for external port number. In the absence of other optimizations, that would bring the expected number of guesses required to find a Teredo address behind an IP address to between $32 \bullet 2 \bullet 2^{16}/N = 4{,}194{,}304/N$ and 4.

Finding any Teredo address on the Internet

Let us now focus on finding one or more live Teredo addresses on the Internet, without concern for the particular IPv4 node that an address corresponds to. This is the type of searching that attackers use whenever they are not concerned about a specific target.

Finding a precomposed Teredo address through its normal usage and normal address storing is one approach to finding one or several Teredo addresses. Assuming that the data is fresh, then nearly any Teredo address found in such a manner will match what we are looking for (unlike in the previous two cases, in which we were looking for a specific host or address, and therefore had to sort through the addresses). However, this approach will not work if one is looking for all Teredo addresses—or even for a sufficiently large subset of all Teredo addresses.

The other approach is to combine the guessing approach from the last section with any other approach to finding live IPv4 addresses. This makes the address space 2^{32} times as large; but the number of different IPv4 addresses that need to be tried to find the next Teredo address would not be nearly as large, especially if one knew which IPv4 addresses are more likely to have Teredo clients. For example, specific types of ISPs (e.g., consumer ISPs or ISPs that provide only RFC 1918 addresses) could become known for having a higher density of Teredo clients.

Scanning difficulties compared

Our research has shown the difficulty (lack of feasibility, even) of performing a blind scan of native IPv6 unicast addresses, due to the address size. Much of the difficulty is due to the low address density enabled by the size of the host part of the addresses (the last 64 bits). In Teredo addresses, the network part of the address comes from the fixed Teredo prefix and the Teredo server used. As mentioned in "Teredo components" section, the number of popular servers is likely to be small and fairly well-known. This means there are many hosts with addresses represented within the address space of a few servers, thereby increasing the density. In addition, all Teredo addresses have a well-defined format, allowing for additional optimization. Hence, it is much easier to find a Teredo address than a native IPv6 address (although this method would not find native IPv6 addresses). The work required to find Teredo addresses on the Internet is discussed in "Finding any Teredo address on the Internet" section.

While the resulting address space for Teredo addressees is larger than the address space that IPv4 requires to be scanned (since an IPv4 address is included in the process), it may bring the scanning within the realm of possibility. Nevertheless, alternative methods of IPv6 host discovery may prove more efficient in many cases.

The effect of Teredo service on worms

In some cases, the existence of the Teredo service will make the development of effective worms easier, specifically those worms that choose their targets at layer 3 or 4.

The main benefit that worms gain from Teredo is that it provides a means of traversing NATs. This is especially true for IPv6-based worms, which can reach IPv6 nodes even if a NAT would normally prevent IPv6 service. However, it is expected that all hosts will have direct IPv6 connectivity eventually, so Teredo cannot be blamed for reducing the timeframe for this eventuality.

Even with IPv4, Teredo specifically holds open a UDP port mapping in NATs, so that NAT would route packets through to the internal host over that port, at least from specific source addresses. This will only reach a Teredo service port, which may restrict which packets are useful.

Even if a host firewall would normally block the IPv6 packet (say, because there was no service to receive it), an alarming scenario exists. Specifically, if there is a vulnerability in the Teredo service or in any other processing before the firewall decision is made (e.g., vulnerability in IPv4 options or in a firewall) that allows remote code execution, it could be used to create a Slammer-style worm. Like Slammer, the worm could propagate in a single UDP packet; but unlike Slammer, it would bypass NATs.

Meta-server worms[8] find their next targets by querying a pre-existing repository of targets such as search engines and peer-to-peer servers. The only way in which this particular form of worm would benefit is if the query could be one that is intended to yield Teredo IPv6 addresses, from which an IPv4 address and port number can be extracted as an IPv4 target. Topological worms,[8] which find their next target by examining information available on the current target, would benefit in the same way. (A famous worm with a major topological vector is the 1988 Morris Worm, which obtained target addresses from files such as /etc/hosts.equiv and /.rhosts.[5])

While IPv4 blind scanning worms gain no specific benefit from Teredo, IPv6 blind scanning worms do. As discussed in "Scannning Teredo addresses compared with native IPv6 address" section, scanning Teredo addresses may make the IPv6 scanning approach feasible. Wanting to reach inside NATs or needing to use IPv6 may even provide reason enough to want to try it.

Attack pieces

The descriptions in this section are not attacks in themselves, but what could be a part of an attack.

Getting Teredo components to send packets to third parties

Teredo components often forward packets. In fact, the job of a relay—and to a more limited extent, the server—is to forward packets. In following Teredo procedures, components also create new packets based on unverified information in an incoming packet. In fact, the anti-spoofing measures "Teredo anti-spoofing measures" section) cannot be employed unless there is a response like that. So, there are several ways (as mentioned in section 7.4 of the Teredo RFC) that Teredo components can send packets to a third party. Packets can go from IPv4 to IPv4, from IPv6 to IPv4, and from IPv6 to IPv4. This is not a novel capability on the Internet; for example, a forged TCP SYN packet often gets a response sent to the selected third party.

Packets sent in response to packets can be used as part of a DOS attack or to mislead the recipient about the source of a packet. For a DOS, one of the key things to consider is amplification. The RFC claims that none of the Teredo possibilities can lead to "noticeable" amplification. (The specific checks against broadcast and multicast addresses help here.) In addition, the RFC points out that messages from Teredo components tend to have a high degree of regularity (the Teredo address prefix, if nothing else) that can be used in filtering out a flood. The concern, though, is that all Teredo addresses might be blocked along with those associated with the denial of service.

Inducing a client to make external connections

Some attacks rely on the ability to convince a Teredo client to send one or more packets out through the Teredo interface, or at least to create an entry in the client's cache of recent peers. This is not difficult to achieve.

Sending the client a packet labeled with a Teredo source address that requires a response is the surest way to create a cache entry and produce outgoing packet(s). IPv6 pings and packets that generate ICMPv6 error messages are a couple of options. It may be necessary to send some bubble packets prior to sending a packet from a source, and that may in itself be sufficient to create an entry and cause a bubble or ping to be sent out.

Packets with a non-Teredo source address may also serve for creating a cache entry. Per section 5.2.3, item 6, of RFC 4380 (and private communication with the RFC author), it is up to the client whether to accept that packet, but the client "SHOULD" start a ping test if it does.

One way for an attacker to convince the client to send out multiple packets, or at least to create multiple cache entries, is to repeat the above multiple times with different IPv6 source addresses. Another approach is to use an upper-layer application, such as a web or email client. For example, an HTML page could be crafted that in-lines images from multiple hosts that are specified by its IPv6 address or an equivalent hostname. This would cause an outgoing connection to be established to each of the different IPv6 addresses and a cache entry for each one. JavaScript in HTML could also loop to cause connection attempts to random destination addresses.

Selecting a relay via source routing

A possible attacker use of IPv6 source routing that ties in with Teredo is using it to select a specific Teredo relay to send packets through. The attacker, on a native IPv6 peer, crafts the packet with a list of specified hops that allow it to use the relay. This requires that (1) there be a node that accepts source routing requests whose closest Teredo relay (routing-wise) is the targeted relay; (2) no source-routed packet filtering take place between the attacker and that node, or that node and the relay, at least not any that cannot be source-routed around, and (3) the relay does not pay attention to the fact that the IPv6 packet it is transferring to the IPv4 Internet involved source routing. Targeting a specific relay can sometimes be an enabler for other attacks, such as a DOS against the relay (and its users).

Finding the IPv4 side of an IPv6 node's relay

Some attacks can be facilitated if the attacker knows the IPv4 address and port of the relay that would be used in sending a packet via Teredo. One easy way for an attacker to do this, if they have both IPv4 and IPv6 access to the Internet, is to send an arbitrary IPv6 packet with a special Teredo address as the destination. The address would have its own IPv4 address as the server address, the cone bit set, and arbitrary values for the encoded address and port. When the packet reaches the closest relay (IPv6 routing-wise), the relay will send a bubble to what it thinks is the server address. On the IPv4 address, the attacker will receive the bubble whose source IPv4 address and port is the closest relay.

Teredo mitigation

To mitigate security concerns about Teredo, one may want to block or at least inspect Teredo traffic. The ideal way to block Teredo is by disabling it at the client, but that is often not a thorough solution. This section explores detecting or blocking Teredo traffic at a gateway.

There may not always be an easy and reliable method for detecting all external Teredo traffic from the perspective of a Teredo client's network. Only the server sits on a well-known port, 3544. Any outbound packets destined for port 3544 are likely to be Teredo-related, though they may not be to a server. (Inside or even outside of a NAT, inbound packets to port 3544 are likely Teredo traffic, but there is no guarantee; the same holds true for packets sourced from port 3544.)

All other UDP ports would need to be screened for Teredo traffic as well, both inbound and outbound. Payload inspection and comparing against the expected form of UDP payload seem the simplest approach. As a first pass, the following algorithm could be completed on UDP packets (reassembled if needed):

1. The packet is UDP over IPv4.
2. The UDP payload is at least 40 octets (the length of an IPv6 base header); assume this is the start of an IPv6 packet.
3. If the start of the presumed IPv6 packet is 0x0001, assume this is authentication data; also assume the start of the IPv6 packet is at offset 13 bytes plus the lengths of the client identifier and the authentication value.

4. The size of the presumed IPv6 packet is still at least 40 octets long.
5. If the start of the IPv6 packet is 0x0000, move the presumed start of the IPv6 packet ahead by 8 bytes (this assumes that was origin data).
6. The presumed IPv6 packet is still at least 40 octets long.
7. The payload length field of the presumed IPv6 header is exactly 40 octets less than the actual length of the IPv6 packet.
8. The internal-side IPv6 address from the presumed IPv6 header (the source address on outgoing packets and the destination address on incoming packets) starts with 0x20010000 (the Teredo prefix).

If concerned about possible evasion, one might need to repeatedly loop over steps 3 to 6 for as long as 0x0000 or 0x0001 continues to appear; this is because some hypothetical Teredo recipient might not care about the order of authentication data and the origin data, and might even tolerate multiple copies.

If this algorithm produces too many false positives, an additional check can be made if the packet inspection is on the outside of all NATs. This check involves looking further at the internal IPv6 Teredo address in the presumed IPv6 header. Specifically, the IPv4 address and the port encoded in the Teredo address can be compared against the internal-side address and port from the IPv4 header. If those addresses match, one can be sure it is a Teredo packet. A similar check can be performed on the inside of a NAT, but only if the set of external IPv4 addresses that the NAT uses is available; the address part of the Teredo address would need to match one of those.

The feasibility of adding the previously described checks into a network device depends on the device. There could be performance considerations, especially if this requires all UDP packets to be taken off the fast-path. As a simple first check, see if the first 4 bits of the UDP payload are either 0 or 6. However, with DNS those 4 bits are the most significant bits of a 16-bit application-chosen identifier, so these could be frequent matches—especially 0.

In the case where external UDP packets to port 3544 are being blocked, fewer Teredo packets should be seen, since a Teredo address cannot be obtained (at least in the normal way). In addition, if the blocking occurs before the final NAT, NAT keep-opens would be blocked, so the Teredo address would stop working if the Teredo map is idle for sufficiently long. If the blocking is done on the outside of the NAT, the Teredo tunnel stays open as long as the client decides it does not need a reply from the Teredo server during keep-open. So blocking outbound port 3544 may not be a sufficient control on Teredo traffic for many environments.

However, a client and server could use an out-of-band mechanism to agree to use a different server port (e.g., 53). This is another reason why blocking outbound port 3544 may not be a sufficient control on Teredo traffic; the previously described payload inspection would work for server traffic as well. In general, it is difficult to stop the flow of IPv6 packets (or any other data) between two cooperating parties. However, for their initial attack, attackers need to rely on some packet processing already being done by their intended target; Teredo could be one such avenue unless sufficient safeguards are in place past the end of the Teredo tunnel.

Conclusion

We have completed an analysis of the Teredo protocol based on reading the RFC (and apart from any implementation). In this summary, some of the significant security implications of the protocol are highlighted; that is, ways in which Teredo positively or negatively impacts the IPv4 and IPv6 portions of the Internet.

Teredo provides a way for dual-stack nodes that do not have direct IPv6 connectivity (due to being located behind an IPv4 NAT) to communicate with remote IPv6 nodes. This approach promotes the earlier use of IPv6 for the large number of hosts "stuck" behind NATs. The protocol essentially must bypass the NAT; Teredo accomplishes this by establishing a fixed UDP port for each client, over which IPv6 is tunneled to the end client—however, in so doing, more than just the NAT is bypassed. Existing network-based security controls (e.g., firewalls, IPSs), even those that support IPv6, are bypassed as well. Although the controls can continue to provide IPv4 inspection, the real traffic is occurring over the UDP tunnel. Unless those controls are upgraded to be Teredo-aware, they will not be properly applying IPv6 or higher-level controls to this traffic. (We also found that it may be difficult to inspect all Teredo traffic due to the lack of fixed port numbers.)

An opportunity exists to apply security controls on the Teredo client past the end of the tunnel. That is advisable since Teredo essentially puts the client directly on the Internet (any IPv6 node can send packets that will reach the client). However, Teredo does not require such controls, and any controls that are unique to the network are bypassed. Even network-based controls that have an analog (comparable control) on the client have had their defense in depth reduced. Teredo even allows unsolicited incoming packets to be passed through the tunnel. End-to-end connectivity like this is expected to be the norm under native IPv6, but proper security controls are more likely in place there.

A situation in which end-host security controls are important for Teredo clients—especially when network security controls have been bypassed—is with IPv6 source routing. Source routing is quite often disabled via network controls. If it is not disabled on the client as well, an IPv6 source-routed packet sent over Teredo will be forwarded by the Teredo client to its next destination, which may be inside the client's network.

Teredo provides a bubble-to-open function, which allows arbitrary IPv4 nodes to set up a Teredo client's NAT so they can send unsolicited traffic to the client (i.e., they can poke a hole in the NAT for themselves). This turns a restricted NAT into an unrestricted (pure cone) one, for each port maintained by a Teredo client. Even the accessibility through a pure cone NAT is improved, because Teredo is both including the port number and address in the client's Teredo address and actively keeping the port open. The appropriate security posture may need to be rethought due to this.

Something else revealed in the Teredo address is the client's NAT type, and this can facilitate attacks. An attacker may interpret the fact that the cone bit is on as a sign of network weakness and preferentially target such nodes. Servers see a client's intended IPv6 peers, so one should use only trusted servers; this is a concern mainly if the server setting is secretly switched to a malicious server.

Worms that target layer 3 or 4, such as blind IP address-scanning worms, benefit from increased "reachability," since they can reach hosts even if they are behind a NAT. Even if a firewall were in place on the host, if a vulnerability exists in the Teredo client or some other pre-firewall component that permits remote code execution, a worm that spreads with a single UDP packet (like Slammer) may be possible.

To some extent, Teredo components on the Internet provide an easier means of denying service when one of the peers is a Teredo client than when native IPv4 or IPv6 connectivity is in use. The reason varies with the attack. If enough IPv6 packets with different Teredo destination addresses are sent through a Teredo relay, the number of peer address records that the relay can store at one time will be exceeded; this would cause at minimum a significant degradation of service. We suspect this to be a worse problem than with IPv4 or IPv6 routers, because it is expected that this limit will easily be reached, especially since the relay may be queuing packets for up to a 6 seconds while waiting for NATs to be set up.

A similar problem exists on the client, since it is probably easy to cause a client to connect to many different destinations and thus exceed the maximum number of peers it can maintain at one time. If a Teredo server is subjected to a brute-force denial of service or is compromised, the impact may be more widespread than in the native IPv4 or IPv6 case, because a large number of clients may depend on it for IPv6 access.

On the positive side, Teredo has peer anti-spoofing measures that are automatically applied. Though not foolproof or as strong as IPsec, these measures provide more peer validation than is typically applied, especially in the case of IPv4. In addition, IPsec is compatible with Teredo, but might not be compatible with other transition mechanisms. Sanity checks required of Teredo components by the Teredo RFC prevent many potential attacks.

Future work

In the future, we plan to examine Teredo's implementation on Windows Vista to assess the specific security implications. It may be worthwhile to also look into Teredo's optional "local client discovery" and "refresh interval determination" procedures.

Acknowledgments

The author would like to thank Ollie Whitehouse, Chris Wee, Roel Jonkman, and Nishant Doshi of Symantec for their suggestions related to Teredo security, and Matt Conover and Ollie Whitehouse for their comments on this report. Thanks also to Christian Huitema and Rémi Denis-Courmont for their valuable feedback on an earlier version of this paper, and to Oliver Friedrichs and Symantec for supporting this research and its publication.

References

1. Microsoft. “Teredo Overview.” microsoft.com. http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/teredo.mspx

2. Denis-Courmont, Rémi. Miredo. http://www.simphalempin.com/dev/miredo/
3. Spence, John. “IPv6 Security and Security Update.” NAv6TF/ARIN XV IPv6 Conference, April 2005: http://www.nav6tf.org/documents/arin-nav6tf-apr05/6.IPv6_Security_Update_JS.pdf

4. Jennings, Cullen. “NAT Classification Results using STUN.” Internet Draft draft-jennings-midcom-stun-results-00.txt (work in progress). February 2004: http://www.employees.org/~fluffy/ietf/draft-jennings-midcom-stun-results-00.html

5. Davies, E., S. Krishnan, and P. Savola. *IPv6 Transition/Co-existence Security Considerations.* Internet Draft draft-savola-v6ops-security-overview-04.txt (work in progress). March 2006: http://ietfreport.isoc.org/idref/draft-ietf-v6ops-security-overview/

6. Symantec. *Symantec Internet Security Threat Report:* Trends for January 06–June 06. Symantec white paper, Volume X, Sept 2006: http://www.symantec.com/specprog/threatreport/ent-whitepaper_symantec_internet_security_threat_report_x_09_2006.en-us.pdf

7. Weaver, N. “How Many Ways to 0wn the Internet?: Towards Viable Worm Defenses: http://www.cs.berkeley.edu/~nweaver/wormdefense.ppt

8. Seely, Donn. *A Tour of the Worm* In Proceedings of the 1989 Winter USENIX Technical Conference, January 1989: http://securitydigest.org/phage/resource/seely.pdf

About Symantec

Symantec is a global leader in infrastructure software, enabling businesses and consumers to have confidence in a connected world. The company helps customers protect their infrastructure, information, and interactions by delivering software and services that address risks to security, availability, compliance, and performance. Headquartered in Cupertino, Calif., Symantec has operations in 40 countries. More information is available at www.symantec.com.

For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 (800) 745 6054.

Symantec Corporation
World Headquarters
20330 Stevens Creek Boulevard
Cupertino, CA 95014 USA
+1 (408) 517 8000