Windows Vista Network Attack Surface Analysis: A Broad Overview

Tim Newsham[1] and Jim Hoagland, *Principal Security Researcher, Symantec Corporation*

***Abstract*—A pre-release version of Microsoft Windows Vista was examined to determine the external security exposure and to identify how the networking components have changed from previous versions of Microsoft Windows. Changes in the networking stack, network services, and core protocols are enumerated and their effect on the external security of the system is analyzed.**

***Index Terms*—Computer network security, Computer security, Operating systems.**

I. Introduction

Windows Vista is Microsoft's much anticipated new client operating system. When released, it is due to replace Windows XP as Microsoft's premier desktop operating system. Windows Vista represents a significant departure from previous Windows systems both in terms of its emphasis on security and its many new features. As security has grown in importance, Microsoft has paid increasingly more attention to it, evidenced by the significant resource investment that has been made. Windows Vista provides Microsoft with its first opportunity to introduce security into the design process of the core operating system itself. Microsoft has also chosen Windows Vista as the platform on which to introduce many newly developed technologies.

The Windows Vista network stack is particularly interesting because many of its components are new. The TCP/IP network stack itself has been rewritten and is one of Vista's largest departures from previous versions of Windows. The new stack was written to allow easier maintenance, important new performance enhancements, and improved stability [30]. It integrates support for IPv6 and IPv4 into a single network stack and provides IPv6 support in the default configuration for the first time in the history of Windows. Many other new protocols are implemented and supported in Vista, either as part of the network stack or as separate components of the Windows operating system. These new protocols support features such as topology discovery, server-less name resolution, and NAT traversal. Even SMB, one of Microsoft's oldest technologies, received a revision with the introduction of the SMB2 variant. The amount of new code present in Windows Vista provides many opportunities for new defects. Each new protocol comes with its own collection of security implications that will need to be understood and considered.

Note that we have made an effort to discover public sources that may document or otherwise describe the new technologies in Windows Vista. In some cases, however, we may have overlooked these sources or new sources may have been published since this paper was written. It is not our intent to mislead the reader or to provide incomplete information.

Symantec evaluated the security of the network stack of public pre-release versions of Microsoft's Windows Vista operating system. Our investigation was broad and shallow, aiming to provide timely intelligence on the new system by covering as many aspects of the network stack as was practical in a small amount of time. Because of the limited amount of time given to the project, our analysis did not penetrate very deeply into any one aspect of the network stack and was forced to omit some key components. When omissions were made, we attempted to cover the most common configurations and components that were most likely to come under attack. We performed our analysis on the public beta builds 5231, 5270, and 5384 of Windows Vista. Since Windows Vista is still a work in progress, we expect many of our results to be invalidated by changes made prior to its public release.

The remainder of this paper will detail our analysis of the Windows Vista network stack. The discussion is ordered according to network layer. In section II we discuss link layer protocols. Section III covers network layer protocols and section IV covers transport layer protocols. Section V covers the Windows Firewall, a component whose design cuts across many protocol layers. Section VI covers the servers and clients that operate at the application layer. Finally, sections VIII and IX present our conclusions and suggestions for future work.

II. Link Layer Protocols

Windows Vista supports protocols at the Link Layer for transporting IP and IPv4 packets, for performing address resolution and auto configuration tasks, and for providing topology information for network diagnostics. For

[1] Tim Newsham conducted this research under contract with Symantec.

transporting IPv4 and IPv6 packets, Windows Vista uses protocols such as Ethernet, PPP, and PPPoE. In support of the IPv4 and IPv6 protocols, Vista supports ancillary protocols such as the Address Resolution Protocol (ARP) and Neighbor Discovery (ND), which are necessary to support the transmission of IPv4 and IPv6 packets. Windows Vista also introduces support for the new Link Layer Topology Discovery (LLTD) protocol which is used to provide network maps to assist in diagnosing networking problems.

We analyzed the ancillary support protocols ARP and ND to determine how they responded to redirection attacks. We also performed a cursory analysis of the undocumented LLTD protocol. As with all link layer protocols, attacks against these protocols are limited to the local network. In the interest of time, we did not perform an analysis of Ethernet due to its simplicity nor of PPP or PPPoE since those protocols are typically used on private links to which an attacker will be unlikely to have access. Analysis of PPP and PPPoE may be warranted in the future.

A. Address Resolution

The ARP[27] and ND[24] protocols provide Ethernet address resolution for IPv4 and IPv6, respectively. ARP operates at the link layer and provides mechanisms for querying the link layer address of an IPv4 node and for propagating address changes to other hosts on the link. ND is implemented using ICMPv6[5] packets above the IPv6 layer but provides necessary services to transmit packets at the link layer: querying for the link layer address of an IPv6 node, propagating address changes, and auto-configuring addresses and routes. ND makes use of well-defined IPv6 multicast addresses[14] with fixed link-layer addresses[6] to avoid bootstrapping problems. Both protocols are integral to the operation of the IP network stack and are enabled during installation.

ARP is susceptible to a redirection attack when an attacker sends a "gratuitous-ARP" packet (normally used to propagate address changes) to a target host [Appendix II]. After receiving such a packet, the target will forward future packets to the attacker's node rather than the intended target. We observed that the Windows Vista stack will accept the information in a gratuitous ARP packet sent to the broadcast link address or to the host's link address and create a new entry in the host's ARP table or overwrite an existing entry. When an existing ARP table entry is overwritten, no warnings are displayed to the user or logged in the system event log.

When processing an ARP packet that contains the same IP address as the host, the Windows Vista stack will log the address conflict in the system event log but will not otherwise notify the user of the conflict. After logging the conflict, the Windows Vista networking stack marks its network interface as having a conflict and stops initiating packets from that interface, effectively rendering the interface useless. Oddly enough, pre-existing connections on the host are not affected. The interface remains in this state until it is reset by reacquiring a DHCP lease.

The Neighbor Discovery implementation is much more robust to redirection attacks than the ARP implementation. We observed that the Windows Vista ignored all unsolicited ND responses [Appendix III]. However, it did update the neighbor table when receiving ND responses during the Probe phase of neighbor discovery. Two factors contribute to increase the practicality of redirection attacks: ND table entries timeout fairly quickly and the ND module keeps an address in the Probe phase for a significant amount of time. Taken together these factors make it easy to perform a redirection attack against ND by sending out spoofed ND replies periodically or in response to a legitimate request[2].

ARP packets are sent in Ethernet frames, but ND communicates using ICMPv6. However, ND is *prima facie* not vulnerable to remote attack since link-local addresses are used and the network stack verifies that an ND packet has a hop count of 255 before processing its contents. The hop count is decremented each time a packet is forwarded and discarded when its count reaches zero. Since 255 is its maximum value, ND packets cannot be received from a remote network with a hop count of 255[3].

B. LLTD

The Link Layer Topology Discovery protocol is a newly developed protocol designed by Microsoft for discovering hosts on the local network. LLTD is a core component in Microsoft's network diagnostic strategy. By providing high-quality topology information to end users, Microsoft hopes to make it easier for users to manage their home networks.

LLTD is implemented in Windows Vista as two components:

- a client program, which initiates and directs topology discovery
- a server ("responder"), implemented as a kernel driver, which responds to requests.

The client program is invoked when a user requests that a network map be generated from the networking control panel. The responder is configured during installation and is always running unless explicitly disabled.

We are unaware of any public documentation of the LLTD protocol. We performed a cursory analysis of the protocol and were able to decode many of the protocol fields [Appendix IV]. Figure 1 shows data from a typical LLTD packet decoded according to our best efforts. LLTD packets are transmitted using Ethernet type 0x88d9. The first 14 bytes of the figure represent the Ethernet header. Immediately following the Ethernet header are four bytes containing a version number, a flag field, and a message

[2] The IETF has defined SEND[1] as a way to provide secured Neighbor Discovery.

[3] Tunneling protocols may provide a way around this restriction. We have not investigated any ND attacks used in conjunction with tunneling.

type. We observed types varying from 0 to 9. The header contains several MAC address fields and a few unknown field types. The majority of the packet is made up of type-length-value encoded fields. TLV encoding allows easy decoding even without a full understanding of the packet format. Appendix IV contains a more detailed analysis of the protocol.

```
ff ff ff ff ff ff       dst (broadcast)
00 c0 9f d2 0c f8       src
88 d9                   type (LLTD)

01 01 00 01             version 1, ??, msg type 1
ff ff ff ff ff ff       mac (dst)
00 c0 9f d2 0c f8       mac (src)
00 00 00 0c
00 0c 9f d2 0c f8       mac
00 c0 9f d2 0c f8       mac

01 06 00 00 00 00       Type 1, Value 0
00 00
02 04 00 00 00 00       Type 2, Value: 0
03 04 00 00 00 06       Type 3, Value: 6
0c 04 00 0f 42 40       Type 12, Value 0x0f4240
0a 08 00 00 00 00       Type 10, Value 0x369e99
00 36 9e 99
0b 04 00 02 62 5a       Type 1, Value 0x026251
0f 12 61 00 63 00       Type 15, "acervista"
65 00 72 00 76 00
69 00 73 00 74 00
61 00
14 04 80 00 00 00       Type 20, value 0x80000000
00                      Type 0, End of TLVs.
```

Figure 1: Encoding of a typical LLTD packet.

We were able to construct an LLTD request that elicited a response from all LLTD responders. The response received back from each LLTD responder contains important information such as the name, Ethernet, IPv4 and IPv6 address of the host. We were not able to cause LLTD responders to generate other traffic on our behalf, although we know LLTD has this functionality[9]. The ability to cause other hosts to generate traffic on an attacker's behalf is often a useful tool in a denial-of-service attack. While some smarter switches may prevent it, in many cases an attacker can already saturate a link and forge source addresses without this mechanism.

III. NETWORK LAYER

Microsoft chose to rewrite the Windows Vista IP stack rather than derive it from the previous Windows XP stack. This new stack integrates support for IPv4 and IPv6 into a single network stack and, according to Microsoft, is easier to maintain, boasts increased performance, and is more stable than their previous network stack[30].

Windows Vista is the first Windows operating system to enable IPv4[28] and IPv6[7] during installation. The Vista stack integrates IPv4 and IPv6 into a single network stack where previous implementations offered a separate IPv6 stack as an optional component. Both stacks share many implementation characteristics as a result of this tight integration.

The inclusion of IPv6 support in Windows Vista is a major departure for Microsoft. IPv6 provides a lot of functionality, backed by a lot of code that has not been tried by extensive use in a hostile environment. To make matters worse, many of the defenses relied on to protect today's IPv4 networks either do not yet support IPv6 or are similarly immature. As IPv6 sees wider deployment, we expect that attackers will heavily scrutinize this protocol.

A. IP Behavior

We measured implementation characteristics of the IPv4 protocol layer and compared them to previous implementations. The characteristics we measured were IP ID generation and IP fragment reassembly behavior. We observed that the Windows Vista stack generates IP packet identifiers (used in fragment reassembly) sequentially [Appendix V]. This behavior is identical to that of the Windows XP stack. Sequential IDs can be used to measure the network activity of a host. When two packets are received from a host, the amount of traffic that was sent in the intervening time is the difference between the IDs in each packet. Sequential IDs are also useful in counting hosts behind a NAT firewall[4].

The Windows Vista networking stack behaved differently than the previous XP stack and other popular networking stacks when reassembling IPv4 fragments [Appendix VI]. Vista strictly discards all packets containing fragments with partial overlaps. In cases of total overlap, newer fragments are discarded in favor of previously received fragments. The Windows XP stack would allow partial overlaps and had a more complicated reassembly behavior. As a result of these differences, identical traffic sent to XP and Vista targets may be interpreted differently. Ambiguities in the interpretation of traffic provide opportunities for confusing network intrusion detection devices unless handled appropriately[25].

B. IP Protocols

Unlike the Windows XP stack, the Windows Vista stack responds to received packets containing an unhandled protocol type with an ICMP error message. We were able to make use of this behavior to enumerate the IP protocols that are configured during installation of Windows Vista [Appendix VII]. In build 5270, there was a response for both IPv4 and IPv6. However, in build 5384, we find that the ICMP error messages are no longer sent for IPv4. As such, it was necessary to turn off the firewall to update that protocol list.

The following IPv4 protocols are configured on a Windows Vista host: ICMP, IGMP, IPv4, TCP, UDP, IP6, GRE, ESP, AH, 43, 44, 249, and 251 (GRE and 249 are new with 5384). The following IPv6 protocols are configured: IPv4,

TCP, UDP, IPv6, ICMPv6, ESP, AH and 251 (IPv4 is new with 5384). IPv6 handles header options as protocol payloads and the IPv6 stack supports the Hop-By-Hop, Route, Fragment, and Destination options. These protocols represent core (IGMP, ICMP, TCP, UDP and ICMPv6), tunneling (IPv4, IP6), and security (ESP and AH) protocols.

Four protocols configured in Windows Vista are not commonly used: protocols 43, 44, and 249 in IPv4, and protocol 251 in IPv4 and IPv6. Protocols 43 and 44 coincide with the values used to represent IPv6 Route and Fragment options; we believe that their presence in the IPv4 stack is likely an oversight caused by the integration of the IPv4 and IPv6 stacks. In build 5270, we observed that sending random data to the host over protocol 43 causes the host to become unresponsive for a long period of time and that sending random data over protocol 44 causes Vista to crash with a blue screen. This is consistent with an implementation that does not expect to receive these protocol types over IPv4. This behavior has since been resolved in build 5384.

The remaining protocols, 249 and 251, are within the unassigned range[16]. Thus far we have not been able to elicit any responses when sending data to a Vista host using these protocol numbers, identify the service or driver making use of this protocol, or find any reference to this protocol online.

Analysis of all three exceptional protocols was light; further investigation will likely be rewarding. We also found it surprising that the network stack appears to support tunneling of IPv4 in IPv4 packets and IPv6 in IPv6, but we did not have time to verify that these features were functional or to explore their security implications.

C. Defects

We tested the stability of the Windows Vista TCP/IP stack using a suite of historic attacks and using random fault injection. Our testing uncovered several defects in earlier builds that leave the system vulnerable to denial-of-service attacks and possibly worse. These defects appear to be fixed by build 5384. We observed that Windows Vista is vulnerable to several historical packet-level attacks against the TCP/IP stack [Appendix VIII]. These well-known vulnerabilities have since been remedied in all popular network stacks. Two of the vulnerabilities observed in Windows Vista build 5231 have been addressed in build 5270; the other was addressed in 5384.

The first attack, called “Blat,” sends a TCP SYN packet to the target with an urgent pointer that points beyond the end of the packet. The effect this has on a Vista target is to cause the IPv4 stack to become unresponsive for a few seconds (the IPv6 stack continues to run independently). This vulnerability has been addressed by build 5270.

The second attack, called “Land,” sends a TCP SYN packet to the target’s address using the same source and destination address and ports. This can cause a target to reply to itself. The effect this attack has on a Vista target is to cause the IPv4 stack to become unresponsive for a few seconds. This vulnerability has been addressed by build 5270.

The third attack, called “Opentear,” sends a flood of improperly formatted UDP fragment packets to the target from a large number of forged source addresses. The effect this has on a Vista target is to cause the entire machine to be unresponsive until the flood of packets subsides[4]. This vulnerability was present in the 5231 and 5270 builds, but has been addressed as of build 5384.

In our testing we also discovered a number of new instabilities in the Windows Vista network stack that were present in earlier builds. To test for new vulnerabilities, we flooded the target host with randomly generated malformed traffic. We isolated and analyzed any packets that caused undesirable behavior on the target [Appendix IX]. This analysis uncovered three vulnerabilities in the network stack.

When a Vista target received an IP packet for protocol 43, the host crashed with a blue screen. Sending the target a protocol 44 packet caused the system to become unresponsive for a large amount of time. The presence of these protocols was also uncovered in our protocol enumeration testing. The last vulnerability involved the processing of IP options. When an option was received with an option length of zero, the machine locked up completely until reset. The Windows Vista stack likely enters an infinite loop while processing the option when it attempts to advance ahead by zero bytes and repeatedly processes the same IP option.

All three of these defects were fixed in build 5384. The presence, however, of these vulnerabilities is consistent with a rewrite of the network stack and suggests that Microsoft has repeated some of the mistakes others have made in the past and introduced some new vulnerabilities of their own. We believe that other defects are likely present in the stack and that further research in this area should be fruitful.

D. Tunneling Protocols

Windows Vista employs IPv6 transition technologies that allow IPv6 to be used in an IPv4 environment that has limited or no IPv6 infrastructure[19]. These protocols are configured when Windows Vista is installed and are available on all Vista hosts unless explicitly disabled. We investigated the new Teredo protocol, deferring analysis of the ISATAP protocol due to time constraints.

Teredo[20][15] is an IPv4-IPv6 transition technology that allows IPv6 traffic to be tunneled in IPv4 UDP packets in

[4] The Opentear attack was performed across a 100Mb/s Ethernet. The effect may be less pronounced if bandwidth is limited.

unmanaged networks. Windows Vista will enable Teredo tunneling as a last resort if there are no neighboring IPv6 routers or ISATAP servers. We expect this to be the most common environment among Windows Vista users until IPv6 sees wider deployment.

Teredo works by carrying IPv6 packets inside of UDP packets sent over IPv4 networks. What makes Teredo unique is its NAT traversal features. Teredo hosts establish and maintain a connection to one of a set of public Teredo servers. The IPv6 address assigned to a Teredo host encodes the public Teredo server that assigned it, as well as the public address and port assigned to the host (i.e. its address as seen outside of the NAT). A NAT-protected host can establish a direct connection to another such host with the assistance of the peer's Teredo server. The host can notify its peer that it wants to establish a connection by sending the packet to the peer's Teredo server, which is forwarded on to the peer host. The two peers may then send packets to each other, opening up mutual holes in their NAT gateways for return traffic to flow through. The two peers can maintain these NAT mappings indefinitely by periodically exchanging traffic.

Teredo restores global addressability and routing to hosts using private IPv4 addresses. This is a huge benefit to functionality but also has some serious security implications. Many individuals and companies use private addresses as a key part of their defense strategy and may find their Vista hosts externally reachable to an unexpected degree, unless strict egress filtering is in place.

In addition to Teredo, Vista supports ISATAP, 6to4, and 6over4. Surprisingly, Windows Vista seems to support the encapsulation of IPv4 packets in IPv4 packets (protocol 4) and IPv6 in IPv6 packets (protocol 41). We did not get the opportunity to investigate these tunneling technologies further, but expect them to provide leverage in performing network attacks. Further investigation will likely be fruitful.

IV. TRANSPORT

Windows Vista supports the TCP and UDP transport protocols over IPv4 and IPv6. We investigated the implementation characteristics of these protocols.

A. UDP

We tested the stability of the UDP transport protocol using random fault injection and did not observe any defects. We did notice that the Vista networking stack replied to UDP packets sent to unbound UDP ports with an ICMP unreachable error. The earlier Windows XP networking stack did not exhibit this behavior when Windows Firewall was active (the default configuration). This behavior allows the host to be "pinged" for aliveness and for the enumeration of UDP services even when Windows Firewall is active. We notice that the Vista stack seems to rate-limit both ICMPv4 and ICMPv6 error messages, which limits the rate at which UDP testing can be done. It might be interesting to evaluate the rate-limiting algorithm.

B. TCP

We tested the stability of the TCP transport using random fault injection and did not observe any defects. We also measured Windows Vista's TCP ISN generation, "fingerprint" and segment reassembly behaviors and observed several behavioral differences from the earlier Windows XP stack.

The choice of the Initial Sequence Number used when establishing a TCP connection has a profound impact on the security of a TCP connection[3][23][26]. We measured Windows Vista's ISN generation and observed that generation is done by random increment but appears random in practice [Appendix X]. When connections are made using the same connection identifiers (source and destination addresses and ports), each successive connection has an ISN that is only slightly larger than the previous connection. However, if connections having different identifiers are made, the sequence numbers appear uncorrelated. This is consistent with the generation algorithm recommended by RFC 1948[1], which generates ISN values by adding a system-wide counter to a secret hash of the connection identifier. This generation scheme offers strong protection against TCP attacks relying on poor ISN generation.

There are many network stack fingerprinting methods that identify an operating system through its network stack implementation details[12]. We looked at the TCP behavior measured by the nmap[11] utility. We observed that the Windows Vista networking stack behaves distinctly differently than the previous Windows XP version and other popular network stacks. We also observed differences between different Windows Vista builds, suggesting that the TCP stack is being actively developed. The details of these differences are noted in Appendix XI. Because all incoming TCP traffic is filtered by the Windows Firewall, these differences are observable only if a firewall exception is configured.

We measured the TCP reassembly behavior of Windows Vista. When reassembling a TCP stream, Windows Vista resolved any conflicts in overlapping TCP segments by preferring data received in earlier segments over data received in later segments [Appendix XII]. This behavior differs from the behavior observed in the earlier Windows XP networking stack. As a result of these differences, identical traffic sent to XP and Vista targets may be interpreted differently. Ambiguities in the interpretation of traffic provide opportunities for confusing network intrusion detection devices unless handled appropriately[25].

V. FIREWALL

Windows Firewall provides protection against attacks by filtering out protocol requests before they are processed. Windows Vista configures Windows Firewall during installation; unless explicitly disabled, Windows Firewall is running on all Windows Vista machines. We measured the firewall configuration of a Windows Vista machine after installation and after several common configuration changes. We also note methods that could be used to detect the presence of a Windows Vista host even when protected by Windows Firewall.

A. Configuration

In its default configuration, Windows Vista 5384 has two firewall exceptions. One was for "Remote Assistance" (the `msra.exe` program) [Appendix XIII]. This exception is active only when `msra.exe` is running (used when requesting remote assistance). However, `msra.exe` is not usually running. The other default exception was for "Network Discovery." Freshly installed Vista hosts will respond to LLTD requests, making it apparently the only service available by default.

Despite the protection of Windows Firewall, Windows Vista 5384 still processes TCP packets to inactive TCP ports, returning RST packets. It also processes IPv6-based UDP requests to inactive UDP ports, returning ICMPv6 unreachable errors. This error reporting allows for enumeration of the services in question and for remote aliveness testing as previously noted. Interestingly though, the 5270 build had eliminated the TCP RST responses that were seen in the 5231 build.

The earlier Windows Vista 5231 build had several firewall deficiencies that were addressed in the 5270 build. In addition to the RSTs sent for inactive ports, Vista 5231 allowed access to some TCP ports running RPC services (135 and some ports in the ephemeral port range) even though no exceptions were approved by the user or reported in the control panel.

Several common Windows configuration changes introduce filtering exceptions into the Windows Firewall configuration. Examples are turning on File and Print sharing (CIFS), opting into People Near Me, using Windows Collaboration, or enabling Windows Media Sharing. The user must authorize these changes by using the Windows Vista consent mechanism. The Teredo service also adds and removes exceptions from the firewall configuration while running, although without the user's consent. The details of these firewall configuration changes can be found in Appendix XIII.

We observed that the Windows Firewall APIs[21] and the Windows Firewall control panel did not always reflect the exceptions that were allowed through the firewall and did not report data consistent with each other. When exceptions were added to the firewall configuration, they were not always listed as active in data returned from the firewall API. When an exception was added for File and Printer Sharing, ICMP echoes were accepted by the firewall, but this change was not reflected in the control panel.

The Windows Firewall does not appear to distinguish between IPv4 and IPv6 exceptions. An attacker may be able to use an existing exception intended for an IPv4 port to allow traffic in to an IPv6 port, or vice-versa. Due to a lack of time, we did not explore this possibility.

B. Discovery

Windows Vista hosts protected by Windows Firewall can be discovered in several ways even though ICMP echoes (pings) are filtered. Hosts on the same network can effectively "ping" a host by querying for the host's hardware address using the ARP or ND protocols or by requesting all neighbors to respond to an LLTD request. Detection using LLTD is particularly attractive because it returns the host's Ethernet, IPv4 and IPv6 addresses, and host name. Hosts that are not on the same local network can elicit responses from a Windows Vista host remotely using routable IPv4 and IPv6 packets. As previously mentioned, Windows Vista responds to TCP packets sent to inactive TCP ports and to IPv6-based UDP packets sent to inactive UDP ports even when the firewall is enabled. It also responds to packets received using an unhandled protocol or with certain malformed fields[13] with ICMP errors. Windows Vista may be running Teredo, unbeknownst to its user, with firewall exceptions that allow a remote user to interact with it and elicit responses.

C. Tunneling

The tunneling protocols supported by Windows Vista have implications to firewalls protecting Vista hosts. If not blocked, tunnels may provide an attacker with an avenue to bypass all firewall restrictions. The tunneling protocols may also provide avenues for bypassing the Windows Firewall. Due to time constraints we did not explore tunnel-based attacks.

VI. NETWORK SERVERS

A. Active UDP Ports

We applied standard techniques to enumerate the network services using the UDP and TCP transports over IPv4 and IPv6 in Windows Vista [Appendix XIV]. We observed that Windows Firewall blocked access to all TCP services unless otherwise configured by the user[5]. However, in build 5384 (but not the earlier 5270) TCP services over both IPv4 and IPv6 could be mapped since RSTs were generated for ports with no service:

- MS-RPC (135)
- NBT (139) (IPv4 only)
- SMB (445)

[5] In the 5270 and 5384 builds. The earlier 5231 build allowed port scanning even when Windows Firewall was enabled.

- Six ephemeral ports (49152-49157)

Access to the UDP services were also blocked, but because Windows Vista replied to IPv6-based UDP packets sent to unused ports with ICMPv6 errors, the UDP services for IPv6 could still be enumerated.

The UDP ports that were found to be active over IPv6 were:

- NTP (123)
- ISAKMP (500)
- UPnP (1900)
- Web Services Discovery (3702)
- Link Local Multicast Name Resolution (5355)
- Two ephemeral ports (49767, 62133)

Because of the nature of UDP active port enumeration it is not possible to distinguish which of these active ports were being used as clients and which were being used as servers. It is likely that at least NTP is being used a client.

More details on active ports in the three Vista builds evaluated can be found in Appendix XIV.

B. RPC Services

Unlike the 5270 build, the earlier Vista 5231 build allowed remote enumeration of TCP services, and we document the results of such enumeration in Appendix XIV. The previous version also allowed unfiltered access to a few RPC services. We were able to enumerate the RPC interfaces supported on these TCP ports using a brute-force enumeration technique.

Windows Vista allows several services to share a single process. Due to a limitation of the RPC library, all RPC services in the same process are accessible via the same RPC endpoints. There are techniques to avoid providing unexpected network access to RPC services that were intended for local use, but these remedies rely on programmer diligence ([18] and section 4.11 of [17]). We observed that the available TCP ports answered to several RPC interfaces that should not have been reachable. We also observed that some of the interfaces were callable without any authentication. One such call, ServerAlive2 in the OXIDResolver interface, provided the host's name on demand. Windows Vista 5271 does not expose any of these services, but should an exception be added to the firewall configuration to allow access to any RPC services, we expect these issues to resurface.

C. File Sharing

Computer users with several machines commonly turn on File and Printer Sharing and we expect many Windows Vista users will do this. Windows Vista supports the SMB protocol and introduces the new SMB2[9] variant.

SMB2 is a new implementation of the SMB protocol that provides a clean slate for Microsoft. It eliminates many of the legacy SMB calls that were no longer used. It supports high-performance marshalling with fixed header sizes and better alignment rules. Finally, it provides larger field widths for many of the protocol fields to ensure support for larger disks and faster computers in the future. SMB2 is the preferred protocol when supported by both client and server (two Windows Vista hosts, for example), but support is included for legacy interoperability.

We performed some random fault injection of the SMB2 protocol to the stability of the new implementation. We also enumerated the named pipes that are accessible without authentication over the file-sharing protocol since these act as another transport mechanism.

We performed random fault injection on an SMB2 connection between two Windows Vista hosts by corrupting the data stream with a proxy. No resulting defects were discovered through this testing. We suspect that the protocol exchange is being protected by signatures, which would provide some immunity to network-based random fault injection by rejecting altered messages after minimal processing. Protocol-aware directed testing may prove more effective. There was some instability in the SMB implementation in the earlier Windows Vista 5321 build. We observed that sending a single string to port 445 was sufficient to cause the machine to crash with a blue screen. This defect had been addressed as of the 5270 build.

File sharing allows remote access to named pipes. These pipes are often used as a transport mechanism for application protocols. We enumerated the named pipes that were accessible over an anonymous connection to the "`IPC$`" share and found that we were able to open the `netlogon`, `lsarpc`, and `samr` pipes without any authentication [Appendix XV]. When using SMB2 we were also able to open the `protected_storage` pipe, which wasn't accessible via the legacy SMB protocol, suggesting that the two protocols are handled by different implementations. All of these pipes are aliases and refer to the pipe named "`lsass`." This pipe is used as a transport for several RPC-based interfaces, which we enumerated. We found that we were able to call several of the interfaces without any further authentication. The details of these RPC interfaces are given in Appendix XV.

VII. UNSOLICITED TRAFFIC

Windows Firewall offers protection against attacks aimed at servers by limiting the number of hosts that can interact with a server and the number of servers that can be accessed. However, it does nothing to protect against replies to traffic initiated from a Windows host. Similarly, many enumeration techniques will provide a list of services, but will not indicate which clients may be active. To enumerate clients and measure the exposure of a Windows Vista 5270 host to client attacks, we observed the unsolicited traffic initiated by a Windows Host during system startup, login, stasis and shutdown [Appendix XVI].

These unsolicited requests provide opportunities for well-placed adversaries to interact with the system despite the protections afforded by the Windows Firewall.

Most of the requests originating from a Windows Vista are standard such as address auto-configuration (DHCP) and support protocols (ARP, ND, DNS and NBNS). Some of the more noteworthy protocols in use are the newer protocols Web Service Discovery, Link Local Multicast Name Resolution, IPv6, and Teredo. All of the traffic we observed was unauthenticated, providing little resistance to malicious attack. Although most of the requests we observed were intended for the local network, a few requests were sent off-site to the public Internet, most notably DNS lookups and communication to a Teredo server. These protocols are most likely to be attacked.

When using the earlier Windows Vista 5231 build we observed that the host posted information back to a public Microsoft server during shutdown! This behavior had been removed as of the Windows 5270 build. The complete list of protocols we observed can be found in XVI.

VIII. Conclusion

The network stack in Windows Vista was rewritten from the ground up. In deciding to rewrite the stack, Microsoft has removed a large body of tried and tested code and replaced it with freshly written code, complete with new corner cases and defects. This may provide for a more stable networking stack in the long term, but stability will suffer in the short term. Despite the claims of Microsoft developers[30], the Windows Vista network stack as it exists today is less stable than the earlier Windows XP stack. We have identified several implementation flaws in the 5270 Windows Vista build and even more in earlier builds, though these have been fixed in 5384. While it is reassuring that Microsoft is finding and fixing these defects, we expect that vulnerabilities will continue to be discovered for some time. A networking stack is a complex piece of software that takes many years to mature.

Microsoft also chose Vista as a platform to introduce new protocols and new implementations of old protocols. IPv6 is enabled during installation for the first time in Windows Vista. The IPv6 protocol is not new, but it has yet to see widespread deployment. To support the process of transitioning from IPv4 networks to IPv6 networks and to increase the usefulness of peer-to-peer technologies, Microsoft has also enabled IPv6 tunneling support in Windows Vista. These tunneling protocols restore global addressability to hosts behind NAT firewalls, increasing the exposure of many users. In support of peer-to-peer communications, Microsoft Vista supports new server-less name-resolution protocols such as PNM and PNRP. Taken together, these technologies provide mechanisms to discover and deliver payloads between peers. These features are critical to the success of Microsoft's peer-to-peer initiative but are also the same features that attackers need to deliver malicious content to their victims. As these technologies see wider deployment, we expect IPv6 and the new peer-to-peer protocols to play an increasing role in the delivery of malicious payloads.

The SMB2 protocol introduces more new code into the Windows Vista environment. SMB2 is based heavily on the mature SMB protocols, but the new code provides new opportunities for defects as evidenced by the defect we found. This protocol is not available without manual configuration, but the service it provides is useful enough to be configured often. The exposure of this protocol is partially mitigated by the firewall since, when enabled, access is typically restricted to the local network. Due to this restriction, we don't expect this protocol to be widely used in remote attacks against home users.

Firewalls and IDSs will have to consider the presence of new Vista machines on their networks. If left unhandled and unchecked, IPv6 and its accompanying transition technologies allow an attacker access to hosts on private internal networks outside of the prevue of the administrator. Unwanted access can be prevented by analysis of IPv6 protocols in the firewall or IDS or by completely blocking all IPv6 protocols. Implementation-specific behavior of the new Vista stack allows an attacker to create ambiguous traffic that may be improperly interpreted by a passive intrusion detection device. IDSs will have to faithfully replicate Vista's behavior when analyzing data destined for Vista hosts. IDSs will also have to analyze new protocols and new versions of existing protocols or face being blind to their traffic.

IX. Future Work

We performed a fairly broad and shallow analysis of the networking technologies available in Windows Vista. Some significant omissions in our analysis should be investigated. Our work highlights many interesting implementation features, but does not penetrate very deeply into the subject matter. This provides a wealth of opportunities for future work at all layers of the network stack.

At the link layer, we did not investigate the Ethernet, PPP, or PPPoE protocols or any of the link-layer tunneling protocols, nor did we investigate all of the features of the ND protocol. While we did look into the LLTD protocol, we barely scratched the surface. There are fields of the protocol we did not fully decode and we were unable to exercise all of the protocol's features. The stability of the protocol should also be tested further to look for any implementation flaws.

At the network layer, we did not look into the fragmentation behavior of the IPv6 protocol, the ICMP and ICMP6 protocols, or into the ISATAP or 6to4 tunneling protocols. Tunneling protocols often invalidate some of the design decisions of other protocols. An analysis of attacks that could be performed in conjunction with tunneling

protocols is likely to be fruitful. All the tunneling protocols should also be tested more thoroughly for implementation flaws. Although we performed some fault injection against the IPv6 protocol, we believe the tools we used were sub-par and encourage further fault-injection testing of IPv6. The defects we observed in IPv4 while performing fault injection were isolated but not fully analyzed. A deeper analysis could determine whether any of these defects could be used for more interesting attacks than denial of service.

At the higher levels of the protocol stack, we left the LLMNR and PNRP protocols completely untouched. These protocols need to be analyzed, and their security implications should be understood and tested for implementation flaws. The SMB2 protocol was covered lightly, but could also benefit from a deeper analysis and better fault-injection testing. The firewall could also use more testing. Of particular interest is determining how the firewall behaves when the same port is reused in IPv4 and IPv6 by different programs. It may prove possible to bypass the firewall restriction intended for an IPv4-based transport by using the same port number over an IPv6-based transport, or vice versa.

REFERENCES

[1] J. Arkko, J. Kempf, B. Zill, P. Nikander, "SEcure Neighbor Discovery (SEND)," RFC 3971, March 2005. http://www.ietf.org/rfc/rfc3971.txt

[2] S. Bellovin, "Defending Against Sequence Number Attacks," RFC 1948, May 1996. http://www.ietf.org/rfc/rfc1948.txt

[3] S. Bellovin, "Security Problems in the TCP/IP Protocol Suite," *Computer Communications Review* 2:19, pp 32-48, April 1989. http://www.cs.columbia.edu/~smb/papers/ipext.pdf

[4] S. Bellovin, "A Technique for Counting NATted Hosts," *Proceedings of the Second Internet Measurement Workshop*, November 2002. http://www.cs.columbia.edu/~smb/papers/fnat.pdf

[5] A. Conta, S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 2463, December 1998. http://www.ietf.org/rfc/rfc2463.txt

[6] M. Crawford, "Transmission of IPv6 Packets over Ethernet Networks," RFC 2464, December 1998. http://www.ietf.org/rfc/rfc2464.txt

[7] S. Deering, R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," RFC 2460, December 1998. http://www.ietf.org/rfc/rfc2460.txt

[8] The Ethereal Project, "Ethereal: A Network Protocol Analyzer." http://www.ethereal.com/

[9] The Ethereal Project, "SMB2." http://wiki.ethereal.com/SMB2

[10] M. Frantzen, S. Xiao, "ISIC – IP Stack Integrity Checker 0.06." http://www.packetfactory.net/Projects/ISIC/

[11] Fyodor, "Nmap (`Network Mapper')." http://www.insecure.org/nmap/

[12] Fyodor, "Remote OS detection via TCP/IP Stack FingerPrinting," October 1998. http://www.insecure.org/nmap/nmap-fingerprinting-article.html

[13] V. Hauser, "Attacking the IPv6 Protocol Suite," CORE'05, November 2005. http://www.pacsec.jp/resources.html?LANG=ENGLISH

[14] R. Hinden, S. Deering, "IP Version 6 Addressing Architecture," RFC 2373, July 1998. http://www.ietf.org/rfc/rfc2373.txt

[15] C. Huitema, "Teredo: Tunneling IPv6 over UDP through NATs," Draft RFC, April 2005. http://www.ietf.org/internet-drafts/draft-huitema-v6ops-teredo-05.txt

[16] IANA, "Protocol Numbers." http://www.iana.org/assignments/protocol-numbers

[17] J. Marchand, Windows Network Service Internals, October 2003. http://www.hsc.fr/ressources/articles/win_net_srv/index.html

[18] Microsoft, "Be Wary of Other RPC Endpoints Running in the Same Process," Platform SDK. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/be_wary_of_other_rpc_endpoints_running_in_the_same_process.asp

[19] Microsoft, "IPv6 Transition Technologies," November 2002. http://www.microsoft.com/windowsserver2003/techinfo/overview/ipv6coexist.mspx

[20] Microsoft, "Teredo Overview," Jan 2003. http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/teredo.mspx

[21] Microsoft, "Windows Firewall Interfaces," Platform SDK, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ics/ics/inetfwprofile_authorizedapplications.asp

[22] D. Morgan, "Network Topology: Connectivity Visualized," WinHEC'05, April 2005. http://www.microsoft.com/whdc/winhec/Pres05.mspx

[23] R. Morris, "A Weakness in the 4.2 BSD Unix TCP/IP Software," Bell Labs Computer Science Technical Report 117, February 1985. http://pdos.csail.mit.edu/~rtm/papers/117.pdf

[24] T. Narten, E. Nordmark, W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)," RFC 2461, December 1998. http://www.ietf.org/rfc/rfc2461.txt

[25] T. Newsham, T. Ptacek, "Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection," Secure Networks, Inc. Technical Report, January 1998. http://citeseer.ist.psu.edu/ptacek98insertion.html

[26] T. Newsham, "The Problem with Random Increments," Guardent Technical Report, February 2001. http://www.lava.net/~newsham/random-increments.pdf

[27] D. Plummer, "An Ethernet Address Resolution Protocol," STD 37, RFC 826, November 1982. http://www.ietf.org/rfc/rfc0826.txt

[28] J. Postel, "Internet Protocol," STD 5, RFC 791. http://www.ietf.org/rfc/rfc791.txt

[29] K. Renard, "Security Issues in IPv6," March 2005. http://www.wareonearth.com/resources_whitepapers.html

[30] Scobleizer, "Abolade Gbadegesin and team – Networking in Windows Vista," Channel 9 interview. http://channel9.msdn.com/Showpost.aspx?postid=116349

[31] H. Seki, "Fingerprinting through RPC," BlackHat '04, July 2004. http://www.blackhat.com/presentations/win-usa-04/bh-win-04-seki-up2.pdf

[32] D. Song, DSniff 2.3, December 2000. http://naughty.monkey.org/~dugsong/dsniff/

[33] M. Zalewski, "Strange Attractors and TCP/IP Sequence Number Analysis," BindView Technical Report, April 2001. http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm

APPENDIX I – TESTED VERSIONS

All tests were performed against Windows Vista CTP Beta releases. The following versions were tested:

- Windows Vista CTP Build 5231 x86 (Beta 2, September 2005)
- Windows Vista CTP Build 5231 x64
- Windows Vista CTP Build 5270 x86 (Beta 2, December 2005)
- Windows Vista CTP Build 5270 x64
- Windows Vista CTP Build 5384 x86 (Beta 2, May 2006)
- Windows Vista CTP Build 5384 x64

Test machines were configured as stand-alone machines and not as a member of any managed network or Microsoft domain. We expect this setup to be typical of home users running Windows Vista.

APPENDIX II – ARP SPOOFING

We observed that Vista hosts will accept gratuitous ARP replies and overwrite existing ARP table entries with the data contained in the replies. Vista hosts are also vulnerable to a denial-of-service attack when they receive a gratuitous ARP for their own address.

We used the `arpspoof` tool from the dsniff[30] suite for our testing. Testing was performed using a Linux host as the attacker and two Vista 5270 machines acting as the target (that gratuitous ARPs were sent to) and the victim (whose ARP entry was being spoofed). (Testing could be performed using a single Vista host and any other host by performing the test once with Vista as the target and once with Vista as the victim). Our hosts were `10.200.200.1` (attacker), `10.200.200.124` (target), and `10.200.200.123` (victim).

We used a targeted gratuitous ARP with the following command line:

```
linux# arpspoof –t 10.200.200.124 10.200.200.123
```

We observed that an ARP entry for `10.200.200.123` was created on `10.200.200.124` (using the "`arp –a`" command to list the ARP entries). As expected, traffic from `10.200.200.124` to `10.200.200.123` was directed towards the attacking host (`10.200.200.1`). If we first created a legitimate ARP entry on the target (i.e. by pinging the victim), we observed that the ARP entry was overwritten when the gratuitous ARP was received. No warning message or error log is generated when the ARP entry is overwritten.

We repeated the tests using a broadcast gratuitous ARP:

```
linux# arpspoof 10.200.200.123
```

In this situation, the effect on the target was identical. However, when the victim host received the gratuitous ARP, it became confused. No warning was presented to the user (as is done in Windows XP when this situation occurs), but an error was logged in the event log:

```
The system detected an address conflict for IP address 10.200.200.1 with the system
having network hardware address 00-04-E2-0B-41-21. Network operations on this system
may be disrupted as a result.
```

Indeed, network operation was disrupted. The networking stack did not allow new connections to be made from the machine. For example, running "`ping 10.200.200.1`" resulted in an error `1232 (ERROR_HOST_UNREACHABLE)`. Existing TCP connections were not affected. This condition persisted until the network interface was reset by releasing and renewing the DHCP lease.

The denial-of-service attack was also observed to occur when a gratuitous ARP was directed at the victim:

```
linux# arpspoof –t 10.200.200.123 10.200.200.123
```

Because Vista accepted all gratuitous ARP requests we did not perform testing of forged replies to ARP requests. Any replies will be processed in the same manner as gratuitous ARPs are processed, with the latter replies overwriting any earlier reply.

APPENDIX III – NEIGHBOR DISCOVERY SPOOFING

We observed that Windows Vista hosts will not process unsolicited Neighbor Discovery replies. However, it is still possible to perform a redirect attack by sending spoofed replies in response to actual queries or by blindly sending out responses periodically.

We performed our testing with the `ndspoof.py` script that was constructed specifically for the purpose. Testing was performed from a Linux host using two Windows Vista hosts as a target (receiving the spoofed ND replies) and a victim (the address being spoofed). The hosts involved were `fe80::204:e2ff:fe0b:4121` (attacker, with MAC address 00:04:E2:0B:41:21), `fe80::214:c2ff:fed5:7e96` (target), and `fe80::2c0:9fff:fed2:cf8` (victim).

The `ndspoof.py` script was used to send ND response packets in a targeted manner with the following command line:

```
linux# ndspoof.py –a fe80::204:e2ff:fe0b:4121 –m 00:04:E2:0B:41:21 –t
fe80::214:c2ff:fed5:7e96 fe80::2c0:9fff:fed2:cf8
```

This causes `ndspoof.py` to periodically send ND packets with the solicit flag set directly to the target host. The `ndspoof.py` script was also used to send packets without the solicit flag to the all-nodes multicast address using the following command line:

```
linux# ndspoof.py –a fe80::204:e2ff:fe0b:4121 –m 00:04:E2:0B:41:21
fe80::2c0:9fff:fed2:cf8
```

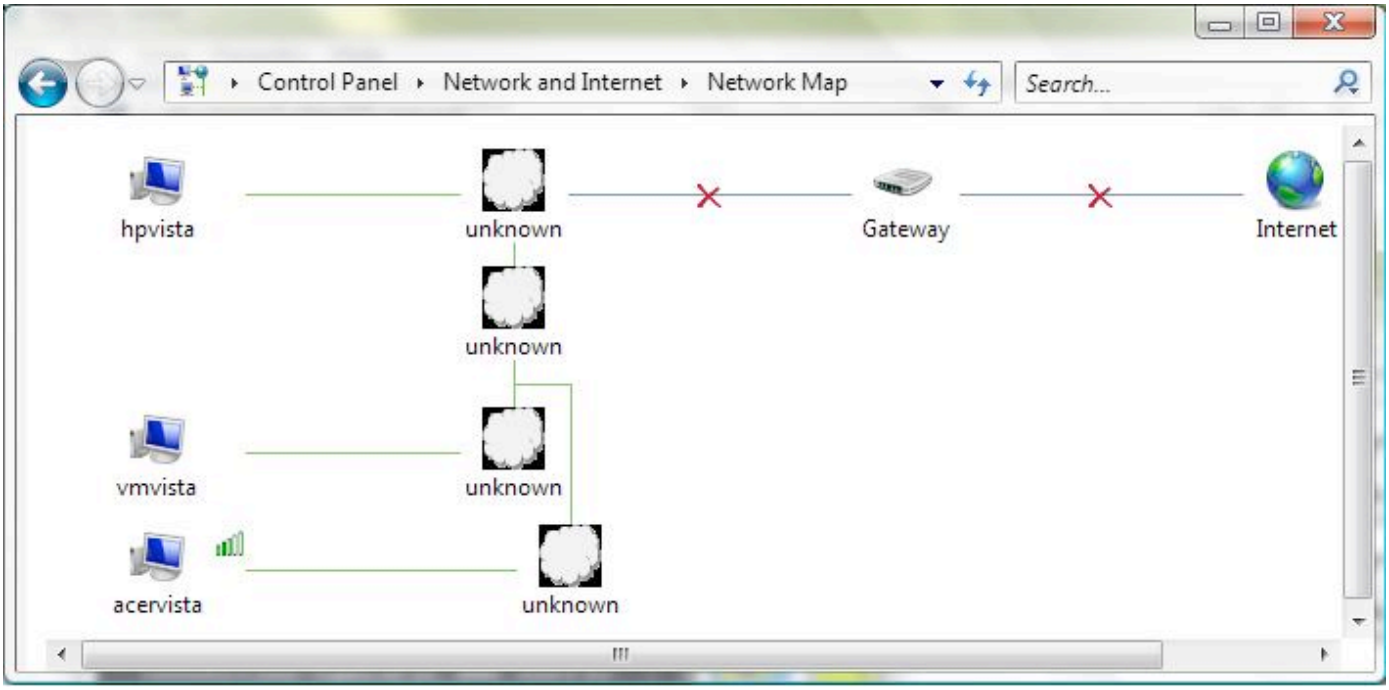
The state of the neighbor table was monitored on the victim host using the command line:

```
vista> netsh interface ipv6 show neighbors
```

We observed that sending ND packets did not cause a passive target to make any changes to its neighbor table. However, if the target was actively establishing a connection to the victim while `ndspoof.py` was sending targeted replies to it, it would use the information in the reply to update its table. Untargeted replies sent to broadcast or sent without the solicited bit set had no affect.

APPENDIX IV – LINK LAYER TOPOLOGY DISCOVERY PROTOCOL

We observed an exchange of LLTD packets between several Windows Vista machines. This exchange of packets, initiated by the `hpvista` machine, resulted in the following network map:



All packets are sent using the Ethernet type 0x88d9. Most packets are sent by the initiating host to the Ethernet broadcast address. The Ethernet payload contains some regularities:

- The first byte is always 1. This may be a version number.
- The second byte is either 0 or 1.
- The third byte is always zero. This may be the most-significant byte of a larger field.
- The fourth byte ranges from 0 to 9. This is likely the LLTD message type. The previous byte may be part of this field.
- The next six bytes contain an Ethernet address which is often the same as the destination address in the Ethernet header.
- The next six bytes contain an Ethernet address which is often the same as the source address in the Ethernet header.
- Ethernet addresses often appear elsewhere in the packet.
- Many packets contain a portion that is encoded using a Type-Length-Value encoding. This is fortunate in that TLV-encoded fields can be extracted without knowledge of the field type encoding.

Using this knowledge and some guess work we can partially decode packets. The following is a best-guess decoding of a typical LLTD packet:

- `Ethernet`
  - `ff ff ff ff ff ff` - dest = broadcast
  - `00 c0 9f d2 0c f8` - src
  - `88 d9` - type = LLTD
- LLTD
  - `01` - version 1
  - `01` -
  - `00 01` - type 1
  - `ff ff ff ff ff ff` - dst
  - `00 c0 9f d2 0c f8` - src
  - `00 00 00 0c`
  - `00 0c 9f d2 0c f8` - mac address
  - `00 c0 9f d2 0c f8` - mac address
  - Array of TLV entries

- `01 06 00 00 00 00 00 00` - Type 1, Length 6, Value 0
- `02 04 00 00 00 00` - Type 2, Length 4, Value: 0
- `03 04 00 00 00 06` - Type 3, Length 4, Value: 6
- `0c 04 00 0f 42 40` - Type 12, Length 4, Value 0x0f4240
- `0a 08 00 00 00 00 00 36 9e 99` - Type 10, Length 8, Value 0x369e99
- `0b 04 00 02 62 5a` - Type 1, Length 4, Value 0x026251
- `0f 12 61 00 63 00 65 00 72 00 76 00 69 00 73 00 74 00 61 00` - Type 15, Length 12, Value Unicode 'acervista'
- `14 04 80 00 00 00` - Type 14, Length 4, value 0x80000000
- `00` - Type 0, End of TLVs.

Type 1 is used to represent an Ethernet address, type 7 is used to represent an IP address, type 8 is used to represent an IPv6 address, and type 15 is used to represent a Unicode string.

Apparently, packets are first sent out by the originating host to solicit a response from hosts with LLTD responders. After an initial exchange, the LLTD initiator may cause responders to send out packets or to reply to them. Some packets are sent out with bogus Ethernet destination addresses. This is probably done to detect the presence of Ethernet switching.

We were able to craft LLTD packets and elicit responses from LLTD responders on the network. However, we were unable to cause LLTD responders to forward packets on our behalf. The `lltd.py` script sends out a broadcast query made up of NUL bytes. It then parses the return values and prints the contents of the decoded packet. The returned packet typically contains the responder's Ethernet, IPv4, and IPv6 addresses as well as its host name. An example invocation of `lltd.py` that elicited two responses is:

```
linux# lltd.py
1 0 0 00:04:e2:0b:41:21 ff:ff:ff:ff:ff:ff
'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00'

1 1 1 00:14:a4:3c:d3:6b ff:ff:ff:ff:ff:ff
LLTDsub1(mac1='\x00\x04\xe2\x0bA!', val=2L, mac2='\x00\x04\xe2\x0bA!',
data='\x01\x06\x00\x14\xa4<\xd3k\x02\x04\x00\x00\x00\x00\x03\x04\x00\x00\x00G\x04\x0
1\x01\x07\x04\n\xc8\xc8}\x08\x10\xfe\x80\x00\x00\x00\x00\x00\x00\x02\x14\xa4\xff\xfe
<\xd3k\t\x02\x00$\n\x08\x00\x00\x00\x00\x006\x9e\x99\x0b\x04\x00\x02bZ\x0c\x04\x00\x
08=`\r\x04\xff\xff\xff\xd4\x0f\x12a\x00c\x00e\x00r\x00v\x00i\x00s\x00t\x00a\x00\x14\
x04\x80\x00\x00\x00\x17\x08\x00\x00\x00\x00\x00Cj=\x00')
  0000:  01 06 00 14 a4 3c d3 6b 02 04 00 00 00 00 03 04  .....<.k........
  0016:  00 00 00 47 04 01 01 07 04 0a c8 c8 7d 08 10 fe  ...G........}...
  0032:  80 00 00 00 00 00 00 02 14 a4 ff fe 3c d3 6b 09  ............<.k.
  0048:  02 00 24 0a 08 00 00 00 00 00 36 9e 99 0b 04 00  ..$.......6.....
  0064:  02 62 5a 0c 04 00 08 3d 60 0d 04 ff ff ff d4 0f  .bZ....=`.......
  0080:  12 61 00 63 00 65 00 72 00 76 00 69 00 73 00 74  .a.c.e.r.v.i.s.t
  0096:  00 61 00 14 04 80 00 00 00 17 08 00 00 00 00 00  .a..............
  0112:  43 6a 3d 00                                      Cj=.
1 '00:14:a4:3c:d3:6b'
2 '\x00\x00\x00\x00'
3 '\x00\x00\x00G'
4 '\x01'
7 '10.200.200.125'
8 'fe80::214:a4ff:fe3c:d36b'
9 '\x00$'
10 '\x00\x00\x00\x00\x006\x9e\x99'
11 '\x00\x02bZ'
12 '\x00\x08=`'
13 '\xff\xff\xff\xd4'
15 u'acervista'
20 '\x80\x00\x00\x00'
23 '\x00\x00\x00\x00\x00Cj='
```

```
1 1 1 00:14:c2:d5:7e:96 ff:ff:ff:ff:ff:ff
LLTDsub1(mac1='\x00\x04\xe2\x0bA!', mac2='\x00\x04\xe2\x0bA!',
data='\x01\x06\x00\x14\xc2\xd5~\x96\x02\x04\x00\x00\x00\x00\x03\x04\x00\x00\x00\x06\
x07\x04\n\xc8\xc8{\x08\x10\xfe\x80\x00\x00\x00\x00\x00\x00\x02\x14\xc2\xff\xfe\xd5~\
x96\n\x08\x00\x00\x00\x00\x006\x9e\x99\x0b\x04\x00\x02bZ\x0c\x04\x00\x0fB@\x0f\x0eh\
x00p\x00v\x00i\x00s\x00t\x00a\x00\x14\x04\x80\x00\x00\x00\x17\x08\x00\x00\x00\x00\x0
17\xe0d\x00')
  0000:  01 06 00 14 c2 d5 7e 96 02 04 00 00 00 00 03 04  ......~.........
  0016:  00 00 00 06 07 04 0a c8 c8 7b 08 10 fe 80 00 00  .........{......
  0032:  00 00 00 00 02 14 c2 ff fe d5 7e 96 0a 08 00 00  ..........~.....
  0048:  00 00 00 36 9e 99 0b 04 00 02 62 5a 0c 04 00 0f  ...6......bZ....
  0064:  42 40 0f 0e 68 00 70 00 76 00 69 00 73 00 74 00  B@..h.p.v.i.s.t.
  0080:  61 00 14 04 80 00 00 00 17 08 00 00 00 00 01 37  a..............7
  0096:  e0 64 00                                         .d.
1 '00:14:c2:d5:7e:96'
2 '\x00\x00\x00\x00'
3 '\x00\x00\x00\x06'
7 '10.200.200.123'
8 'fe80::214:c2ff:fed5:7e96'
10 '\x00\x00\x00\x00\x006\x9e\x99'
11 '\x00\x02bZ'
12 '\x00\x0fB@'
15 u'hpvista'
20 '\x80\x00\x00\x00'
23 '\x00\x00\x00\x00\x017\xe0d'
```

APPENDIX V – IP ID GENERATION

We monitored packets sent out from a Windows Vista machine and observed that IP IDs are generated sequentially. The `ipid.py` script was used to observe packets sent out from a Windows Vista machine while traffic was generated on the Vista machine to several hosts. We observed that the packets had an ID field that incremented from one packet to another, independent of destination host:

```
linux# ipid.py 10.200.200.123
00d0    10.200.200.123 -> 64.65.64.17 proto 6
00d1    10.200.200.123 -> 64.65.64.17 proto 6
00d2    10.200.200.123 -> 10.200.200.1 proto 6
00d3    10.200.200.123 -> 10.200.200.1 proto 6
00d4    10.200.200.123 -> 10.200.200.1 proto 6
00d5    10.200.200.123 -> 10.200.200.1 proto 6
00d6    10.200.200.123 -> 10.200.200.1 proto 6
00d7    10.200.200.123 -> 10.200.200.1 proto 6
00d8    10.200.200.123 -> 10.200.200.1 proto 6
00d9    10.200.200.123 -> 10.200.200.1 proto 6
00da    10.200.200.123 -> 10.200.200.1 proto 6
00db    10.200.200.123 -> 64.4.25.80 proto 17
00dc    10.200.200.123 -> 10.200.200.1 proto 1
00dd    10.200.200.123 -> 10.200.200.1 proto 1
```

This behavior is identical to the ID generation employed in the earlier Windows XP stack.

APPENDIX VI – IP FRAGMENT REASSEMBLY

Vista's networking stack behaves differently than earlier versions in Windows XP or Windows 2000 when reassembling IPv4 Fragments. We observed that Vista was very strict in discarding IP packets with partially overlapping fragments. However, the stack did not discard packets with fragments that completely overlapped one another.

We performed our testing by sending out several out-of-order fragments of a UDP packet that contained conflicting data. The fragments were constructed such that the UDP header was sent once unambiguously with checksums disabled (using the distinguished value of zero). We then observed the payloads of the reassembled packets that were delivered to the application layer on the target machine.

All tests were performed using the `fragorder.py` tool. Fragorder listens for UDP packets to a special test network address and replies with a series of fragments of a UDP packet. The host being tested must be primed by adding a static ARP entry for the bogus test address which points to the MAC address that `fragorder.py` is running on (i.e. **`arp -s 10.200.200.199 00-04-e2-0b-41-21`**). To test a host, `netcat` is used to send out a packet to elicit a fragment, and the results received by `netcat` are recorded (i.e. **`nc -u 10.200.200.99 999`** and hitting the *Enter* key to initiate each test).

A. Test 1

Seven fragments were sent out with 16 bytes of data each (the fragment containing 1s also had the 8-byte UDP header, and the fragment containing 6s had a trailing newline). Each fragment overlapped at least one other fragment by 8 bytes. The following diagram indicates how the fragments' data overlapped:

```
Fragment #1                    2222222222222222
Fragment #2                                    5555555555555555
Fragment #3                                                    6666666666666666\n
Fragment #4                                4444444444444444
Fragment #5                                oooooooooooooooo
Fragment #6                          3333333333333333
Fragment #7  hhhhhhhh1111111111111111
```

After reassembly, the resulting UDP payload received was:

Ethereal	`1111111111111111222222223333333344444444555555556666666666666666`
Linux RedHat8	*no data received*
Windows XP	*no data received*
Windows Vista	*no data received*

B. Test 2

Six fragments were sent out with 16 bytes of data each (the fragment containing 1s also had the 8-byte UDP header, and the fragment containing 6s had a trailing newline). All fragments but the one containing the header overlapped at least one other fragment by 8 bytes. The following diagram indicates how the fragments' data overlapped:

```
Fragment #1                                    5555555555555555
Fragment #2                                                    6666666666666666\n
Fragment #3                                4444444444444444
Fragment #4                                oooooooooooooooo
Fragment #5                          3333333333333333
Fragment #6  hhhhhhhh1111111111111111
```

After reassembly, the resulting UDP payload received was:

Ethereal	`1111111111111111333333333333333344444444555555556666666666666666`
Linux RedHat8	`1111111111111111333333333333333ooooooooo5555555566666666`
Windows XP	*no data received*

Windows Vista	*no data received*

C. *Test 3*

Six fragments were sent out with 16 bytes of data each (the fragment containing 1s also had the 8-byte UDP header, and the fragment containing 6s had a trailing newline), except Fragment #5 which was shortened to 8 bytes. The following diagram indicates how the fragments' data overlapped:

```
Fragment #1                                          5555555555555555
Fragment #2                                                  6666666666666666\n
Fragment #3                                      4444444444444444
Fragment #4                                      oooooooooooooooo
Fragment #5                              33333333
Fragment #6    hhhhhhhh1111111111111111
```

After reassembly, the resulting UDP payload received was:

Ethereal	`1111111111111111333333334444444444444444555555556666666666666666`
Linux RedHat8	`1111111111111111333333330000000000000000555555556666666666666666`
Windows XP	`1111111111111111333333334444444455555555555555556666666666666666`
Windows Vista	*no data received*

D. *Test 4*

Five fragments were sent out with 16 bytes of data each (the fragment containing 1s also had the 8-byte UDP header, and the fragment containing 6s had a trailing newline). None of the fragments overlapped any other with the exception of fragments #2 and #3 which completely overlapped each other. The following diagram indicates how the fragments' data overlapped:

```
Fragment #1                                                  6666666666666666\n
Fragment #2                                      4444444444444444
Fragment #3                                      oooooooooooooooo
Fragment #4                              33333333
Fragment #5    hhhhhhhh1111111111111111
```

After reassembly, the resulting UDP payload received was:

Ethereal	`1111111111111111333333334444444444444444666666666666666666666666`
Linux RedHat8	`1111111111111111333333330000000000000000666666666666666666666666`
Windows XP	`1111111111111111333333334444444444444444666666666666666666666666`
Windows Vista	`1111111111111111333333334444444444444444666666666666666666666666`

E. *Test 5*

Test 5 repeats Test 3 but without any conflict between overlapping fragments. Six fragments were sent out with 16 bytes of data each (the fragment containing 1s also had the 8-byte UDP header, and the fragment containing 6s had a trailing newline) except Fragment #5 which was shortened to 8 bytes. The following diagram indicates how the fragments' data overlapped:

```
Fragment #1                                          4444444466666666
Fragment #2                                                  6666666666666666\n
Fragment #3                                      4444444444444444
Fragment #4                                      4444444444444444
Fragment #5                              33333333
Fragment #6    hhhhhhhh1111111111111111
```

After reassembly, the resulting UDP payload received was:

Ethereal	`1111111111111111333333334444444444444444666666666666666666666666`
Linux RedHat8	`1111111111111111333333334444444444444444666666666666666666666666`

Windows XP	`1111111111111111333333334444444444444446666666666666666`
Windows Vista	*no data received*

F. Analysis

These tests indicate that Windows Vista discards packets that contain fragments with partial overlaps (whether or not there is a conflict in these fragments). This might be done at fragment-receive time by noting an overlap with existing fragments, or at reassembly time by noting whether there was any overlap. In at least some cases, ICMPv4 reassembly timeout messages were sent after approximately 41 seconds.

Earlier versions, such as Windows XP, used a different reassembly strategy. Overlapping fragments are allowed in some instances and are dealt with by trimming the left-hand side (from high- to low-byte offset) from more recently received fragments.

APPENDIX VII – IP PROTOCOL ENUMERATION

The Windows Vista network stack responds to unknown IP protocols by sending back an ICMP packet indicating that the protocol is unreachable. This is for both IPv4 and IPv6 in build 5270 and for IPv6 in build 5384. This differs from the behavior of the Windows XP stack, which does not return such notification. This mechanism can be used to remotely "ping" a Vista machine, even if filtering is enabled. It can also be used to enumerate all of the IP protocols serviced by the target machine. We constructed a tool, `proto.py` which enumerates protocols using this method. When run against a Vista 5270 machine the results were:

```
linux# proto.py 10.200.200.123
1 icmp ICMP
2 igmp IGMP
4 ipencap IP-ENCAP                (IPv4 over IPv4)
6 tcp TCP
17 udp UDP
41 ipv6 IPv6                      (IPv6 over IPv4)
43 ipv6-route IPv6-Route          (IPv6 Route option)
44 ipv6-frag IPv6-Frag            (IPv6 Fragment option)
50 ipv6-crypt IPv6-Crypt          (IPSEC ESP)
51 ipv6-auth IPv6-Auth            (IPSEC AH)
251 ?? ??
```

(Under 5270 sending packets to protocol #43 caused the Vista target to blue screen, so this protocol was avoided by the test program. Sending packets to protocol #44 also caused problems and was also skipped.)

For Vista 5384, we temporarily turned off the firewall to update this enumeration, since IPv4 packets no longer produced Protocol Unreachable messages. We found the same result, but with two additional items:

```
47 gre GRE
249 ?? ??
```

Those protocols could have been added by 5384, or were previously masked by the Windows firewall.

We repeated the same process for IPv6 with another tool, `proto6.py` When run against a Vista 5270 machine, the results were:

```
linux# proto6.py fe80::214:c2ff:fed5:7e96%4 00:14:c2:d5:7e:96
0 ip IP                           (IPv6 Hop-by-Hop option)
6 tcp TCP
17 udp UDP
41 ipv6 IPv6                      (IPv6 over IPv6)
43 ipv6-route IPv6-Route          (IPv6 Route option)
44 ipv6-frag IPv6-Frag            (IPv6 Fragment option)
50 ipv6-crypt IPv6-Crypt          (IPSEC ESP)
51 ipv6-auth IPv6-Auth            (IPSEC AH)
58 ipv6-icmp IPv6-ICMP            (ICMPv6)
60 ipv6-opts IPv6-Opts            (IPv6 destination option)
251 ?? ??
```

Running against Vista 5384 added one item:

```
4 ipencap IP-ENCAP                (IPv4 over IPv6)
```

So, it appears that support was added for IPv4 nodes on IPv6-only networks. The results were the same with the firewall off. Unavailable protocols (actually, Next Header values) yield ICMPv6 parameter problem messages pointing to the Next Header field.

All of the reported protocols are standard protocols for TCP/IP, TCP/IP6 except for Protocols 249 and 251, which are in the reserved range.

APPENDIX VII – HISTORICAL STACK ATTACKS

We tested the Vista networking stack using a suite of historical network stack attack tools. While most modern networking stacks have long since fixed their handling of these deviant packets, we observed that the Vista 5231 stack is affected by three of these attacks, and the 5270 stack is affected by one.

The "Blat" attack sends a flood of SYN packets having an invalid urgent pointer that points past the end of the packet, each from a different forged source address. Performing this attack against the Vista stack causes the network stack to become irresponsive for a few seconds. After a few seconds, all affects of the attack seem to wear off. While the IPv4 stack is irresponsive, the IPv6 stack continues to perform properly.

The "Land" attack sends a SYN packet with the forged source address of the target using the same source and destination ports. This causes the target host to respond back to itself. Sending this single packet to an open port of the Vista stack causes the stack to become unresponsive for a few seconds. After a few seconds all affects of the attack seem to wear off. While the IPv4 stack is irresponsive, the IPv6 stack continues to perform properly.

The "Opentear" attack sends a flood of improperly fragmented UDP packets, each from a different forged source address. While packets are received, the entire Vista system becomes unresponsive. The affects of the attack disappear as soon as the packet stream stops. Tests were performed over a local 100Mbit Ethernet link; the attack may have less effect over a slower link.

The Vista 5270 stack seems to address all of these issues except the "Opentear" attack which still have the same effect. Opentear was addressed by 5384.

A. Tests

The following tests were performed against a target Vista host. When attack programs took an argument to specify a source address, the attack was run using both the real address of the attacking machine and using an unassigned address from the subnet.

All tests were performed over a single 100Mbit Ethernet subnet from a Linux host running Fedora Core 2. The original exploits were used. The exploits had to be modified to send packets out over a raw socket (using a modified `pcap` with `pcap_write()` functionality) whenever the exploit relied on fragmented traffic. The changes were necessary because the Linux kernel does not allow control of the fragment offset field of the IP header when using SOCK_RAW.

The following command lines were used to invoke the attack from target host 10.200.200.1 to victim host 10.200.200.123. Whenever a port number was given, it was against an open TCP or UDP port. When no open ports were available (as is the case with TCP ports for a freshly installed Vista machine), we created an exception in the Windows Firewall configuration on the target host.

```
linux# blat 10.200.200.1 10.200.200.123 445 100 0
linux# blat 10.200.200.1 10.200.200.123 445 100 1
linux# boink 10.200.200.1 10.200.200.123 400 500
linux# bonk 10.200.200.1 10.200.200.123
linux# land 10.200.200.123 445
linux# naptha 10.200.200.123
linux# neptune -s 10.200.200.99 -t 10.200.200.123 -p 445 -a 1000
linux# newtear 10.200.200.1 10.200.200.123
linux# opentear 10.200.200.123
linux# pingexploit 10.200.200.1 10.200.200.123
linux# syndrop 10.200.200.1 10.200.200.123
linux# synk4 10.200.200.1 10.200.200.123 400 500
linux# teardrop 10.200.200.1 10.200.200.123
```

The effects of the attack against our Vista target were:

Attack	**Observed Effect**
blat (SYN flood with optional URG flag, from separate IP addresses)	IP stack becomes unresponsive for a few seconds in the 5231 stack. In the 5270 stack the attack has

	no effect, even when directed at an open TCP port.
boink (strangely fragmented UDP)	no effect
bonk (strangely fragmented UDP)	no effect
land (SYN from self)	IP stack becomes unresponsive for 5 seconds with the 5231 stack! IPv6 is still responsive. sniffer seems to temporarily hang too. The 5270 stack is not affected by this attack.
naptha (random fragments)	no effect
neptune (SYN flood)	no effect
newtear (strangely fragmented UDP)	no effect
opentear (flood of strangely fragmented UDP)	5231 and 5270: machine console is completely locked up for the duration of the flood, but recovers instantly when it is terminated. The machine is still responsive to pings during the duration of the attack. 5384: no effect
pingexploit (oversized fragmented ICMP ping packet)	no effect
syndrop (SYN packet in fragments)	no effect (they have an intentionally bogus checksum?!)
synk4 (SYN flood)	no effect
teardrop	no effect

These tests were repeated against a Windows XP host (with port 445 left unfiltered) and no effects were observed.

APPENDIX IX – RANDOM FAULT INJECTION OF IP

We used the ISIC[10] suite of tools to perform random fault injection of the Ethernet, IP, ICMP, UDP, and TCP protocols on the Vista 5270 build. (The version of ISIC we used was modified because of limitations in the SOCK_RAW interface in Linux). When a vulnerability was found, ISIC was altered to stop generating that type of packet so that additional issues could be found without triggering known crash conditions. We found three issues, all at the IP layer, each of which seem to be fixed as of the 5384 build. Malformed traffic to higher protocol layers was handled properly; performance degradation was the only noticeable effect when a large amount of traffic was sent to the machine.

All testing was done using the ISIC tools. A fixed random seed (`-r`) was used and any observed issue narrowed down by using the skip (`-k`) and packet count (`-p`) flags until a single packet was isolated. The three issues found to cause crashes were[6]:

```
linux# isic -s 10.200.200.1 -d 10.200.200.123 -m 1 -r 1 -p 1018 -k 1017
```

This command causes a packet of type IP_PROTO_ROUTING (43) with a random payload to be sent to the target. The problem was reproduced in the `crash1.py` script. Upon reception of this packet, the target crashes with a blue screen. This protocol number is used to carry optional headers for IPv6 and is not usually used in conjunction with IPv4.

```
linux# isic -s 10.200.200.1 -d 10.200.200.123 -m 100 -r 1 -p 13399 -k 13398
```

This command causes a packet of type IP_PROTO_FRAGMENT (44) with a random payload to be sent to the target. The problem was reproduced in the `crash2.py` script. Upon reception of this packet, the target becomes partially unresponsive. The machine will still respond after a long delay, and if a ping flood is sent to the machine, it will recover sooner. This protocol number is used to carry optional headers for IPv6 and is not usually used in conjunction with IPv4.

```
linux# isic -s 10.200.200.1 -d 10.200.200.123 -m 1000 -r 1 -k 142595 -p 142596
```

This command causes a packet with a malformed IP option to be sent to the target. The option has an unknown type and a length field of zero. The problem was reproduced using the `crash3.py` script. Upon reception of this packet, the target becomes locked up until reset. The target is likely advancing zero bytes to get to the next option in the header and entering an infinite loop.

[6] ISIC's packet generation is endian-specific. These tests were run on a little-endian machine (Linux/IA32). ISIC will generate different packets on big-endian hosts given these same arguments. The Python scripts constructed to reproduce these issues should work identically on all platforms.

APPENDIX X – TCP INITIAL SEQUENCE NUMBER GENERATION

We observed the initial sequence number (ISN) generation of the Windows Vista stack for the 5270 build by sending SYN packets to an opened port and observing the sequence number in the returned SYN+ACK packet. A custom utility, `isn.py`, was used for this testing. The utility sends SYN packets to port 445 of the target (which must be accepted in Windows Firewall) either over IPv4 or IPv6. The source packet of each request was chosen sequentially with either 1 or 100 repeated requests from the same port. When sending a single request from each source port, the sequence numbers appear to be evenly distributed across the entire space:

```
linux# isn 1
src port 3340 5ad81d4d  (delta 1524112717)
src port 3341 d3bd2a61  (delta 2028277012)
src port 3342 39553588  (delta 1704463143)
src port 3343 468b81bf  (delta 221662263)
src port 3344 03c8b144  (delta 3174903685)
src port 3345 302a1713  (delta 744580559)
src port 3346 9f4198a7  (delta 1863811476)
src port 3347 25ff265d  (delta 2260569526)
src port 3348 e97ef158  (delta 3279932155)
src port 3349 ddb800de  (delta 4097380230)
src port 3350 107cf021  (delta 851767107)
src port 3351 5dbb8f79  (delta 1295949656)
src port 3352 2f9b161e  (delta 3521087141)
src port 3353 8bde781d  (delta 1547919871)
src port 3354 1baf75c5  (delta 2412838312)
src port 3355 acd5e355  (delta 2435214736)
src port 3356 6782370c  (delta 3131855799)
src port 3357 87e32d3e  (delta 543225394)
src port 3358 37fa3f02  (delta 2954301892)
src port 3359 8c51ef71  (delta 1415032943)
src port 3360 d584571a  (delta 1228040105)
```

However, when sending multiple requests using the same source port (which causes the TCP connection identifier to remain unchanged across requests), we observe that the ISN is being randomly incremented based on an internal timer:

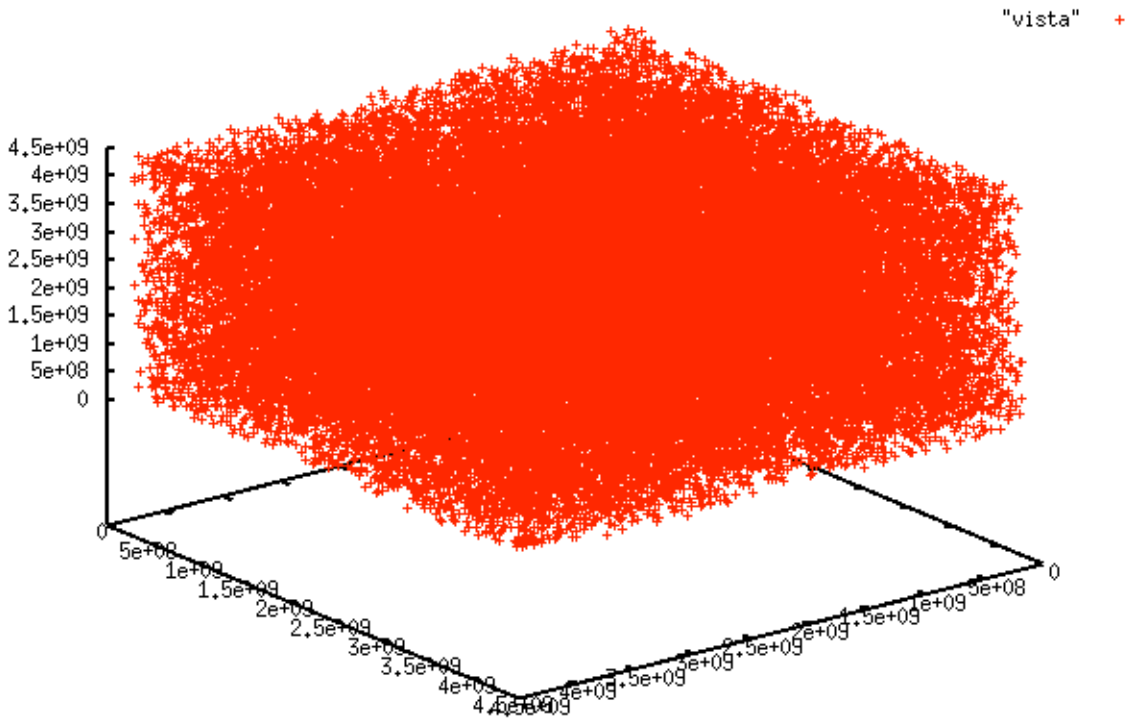
```
linux# isn
src port 3340
58570d02  (delta 1482099970)
58575cf6  (delta 20468)
5857b7b7  (delta 23233)
58581ef0  (delta 26425)
585888b1  (delta 27073)
5858f68a  (delta 28121)
58591ee0  (delta 10326)
58592ee0  (delta 4096)
5859a6de  (delta 30718)
585a050b  (delta 24109)
585a050b  (delta 0)
585a050b  (delta 0)
585a050b  (delta 0)
585a140b  (delta 3840)
[…]
avg dseq 000037bd

src port 3341
d151b6f2  (delta 2028280697)
d1521f06  (delta 26644)
d1526ecc  (delta 20422)
d1529983  (delta 10935)
```

```
d152a983  (delta 4096)
d152fbd1  (delta 21070)
d153629d  (delta 26316)
d153629d  (delta 0)
d153629d  (delta 0)
d153629d  (delta 0)
d153719d  (delta 3840)
d153bc47  (delta 19114)
d15425ca  (delta 27011)
[…]
avg dseq 00003346
```

Repeating these results with an additional “6” argument causes the tests to be performed with IPv6 instead of IPv4. The results are substantially the same as the results above and are omitted.

The Vista 5270 stack appears to be using random increments in its ISN generation while using the technique described in RFC 1948 to maintain a separation between the ISN generation of connections with different connection identifiers. This is typically done by adding the value of a secret hash of the connection identifier to a global ISN counter. This is the same behavior seen in the Windows XP stack, except that Windows XP seems to increment the ISN counter more often or by larger increments.

A rough measure of the strength of the ISN generation can be taken by making a state-space plot of the ISN deltas[33]. The values of (x[n] – x[n-1]), (x[n-1] – x[n-2]), (x[n-2] – x[n-3]) from a sequence, x, of ISN values are plotted in three dimensions. Patterns in this plot are often apparent for weaker generation schemes. The state-space plot of the Windows Vista generator, when different source ports are used, is shown below. The plot for Windows XP looks substantially similar and is not given. Note that the plot appears uniformly distributed across the available space, indicating the strength of the generator.



APPENDIX XI – STACK FINGERPRINT

We used the nmap[11] and its OS fingerprinting signature database to look at the behavior of the Vista stack and the earlier Windows XP stack. We tested both builds 5231 and 5270 and noticed changes between them as well as distinctive changes from the earlier Windows XP networking stack.

To get a fingerprint, we turned off Windows Firewall and used the following command line:

```
linux# nmap -sT -p 445,999 -O 10.200.200.124
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-12-28 13:09 HST
Interesting ports on 10.200.200.124:
PORT    STATE  SERVICE
445/tcp open   microsoft-ds
999/tcp closed garcon
MAC Address: 00:C0:9F:D2:0C:F8 (Quanta Computer)
No exact OS matches for host (If you know what OS is running on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.81%P=i686-pc-linux-gnu%D=12/28%Tm=43B31B36%O=445%C=999%M=00C09F)
TSeq(Class=TR%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=2000%ACK=S++%Flags=AS%Ops=MWNNNT)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=0%ACK=O%Flags=AR%Ops=)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

Uptime 0.130 days (since Wed Dec 28 10:02:20 2005)

Nmap finished: 1 IP address (1 host up) scanned in 7.064 seconds
```

The nmap signature for Longhorn and the previous Windows Vista 5231 build is:

```
Fingerprint Microsoft Windows Longhorn eval build 4051
Class Microsoft | Windows || general purpose
TSeq(Class=TR%gcd=<6%IPID=I%TS=100HZ)
T1(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=0%ACK=O%Flags=AR%Ops=)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=164%RIPTL=148%RID=E|F%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

The primary difference between the 5270 build and the earlier 5231 build is that 5270 will accept SYN packets with the ECN bit set (listed as test T1 in the signature). The signature for the 5384 build is almost identical to 5270 except that T1 has DF=N; this suggests that the stack is reaching a more stable state.

The following table summarizes our observations of the test packets sent out by nmap during fingerprinting and their elicited response. These observations where made by running nmap with the fingerprinting option (-O) while running Ethereal[8] on the target host.

#	dport	Sent to target	Reply from Vista
T1	open	SYN+ECN+W3072+opts	RST+ACK+W0,seq+1 (vista 5231) SYN+ACK+W8192,seq+1,opts2 (vista 5270)
T2	open	NULL+W1024+opts	RST+ACK+W0,seq

```
T3  open    FIN+SYN+PSH+URG+W3072+opts      RST+ACK+W0,seq+2
T4  open    ACK+W4096+opts                  RST,seq=0
T5  close   SYN+W3072+opts                  RST+ACK+W0,seq+1
T6  close   ACK+W3072+opts                  RST+W0,seq=0
T7  close   FIN+PSH+URG+W4096+opts          RST+ACK+W0,seq=0

opts = (20byte) 03 03 0a 01 02 04 01 09 08 0a 3f 3f 3f 3f 00 00 00 00 00 00
        WScale 10, NOP, MSS 265, TS 0x3f3f3f3f tsecr 0, EOL
opts2 = (20byte) 02 04 05 b4 03 03 08 01 01 01 08 0a 00 11 ed 42 3f 3f 3f 3f
        MSS 1460, WScale 256, NOP, NOP, NOP, TS 0x0011ed42 tsecr 0x3f3f3f3f
```

All tests consisted of a single TCP packet with some standard options. The first four tests were sent to an open port, and the last three to a closed port. The packets had different TCP flags and window sizes. The responses differed in the flags field and the sequence number.

Running the same tests against a Windows XP machine gives substantially different results. First, it should be noted that a Windows XP SP2 box does not have any reachable open ports. These tests were performed after disabling the firewall. The `nmap` signature for Windows XP is:

```
Fingerprint Microsoft Windows 2003 Server or XP SP2
Class Microsoft | Windows | 2003/.NET | general purpose
Class Microsoft | Windows | NT/2K/XP | general purpose
TSeq(Class=TR%gcd=<6%IPID=I)
T1(DF=Y%W=402E|FB8B%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=402E|FB8B%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=B0%RIPTL=148%RID=E%RIPCK=E|F%UCK=E|F%ULEN=134%DAT=E)
```

But our observations indicated that XP SP2 didn't answer most of the test probes:

```
#   dport   Sent to target                  Reply from Vista
-   -----   ----------------                ----------------
T1  open    SYN+ECN+W307                    SYN+ACK+W16430+opts2,seq+1
T2  open    NULL+W1024+opts                 no response
T3  open    FIN+SYN+PSH+URG+W3072+opts      no response
T4  open    ACK+W4096+opts                  no response
T5  close   SYN+W3072+opts                  no response
T6  close   ACK+W3072+opts                  no response
T7  close   FIN+PSH+URG+W4096+opts          no response

Opts2 = (20byte) 02 04 05 b4 01 03 03 00 01 01 08 0a 00 00 00 00 00 00 00 00
        MSS 1460, NOP, Wscale 0, NOP, NOP, TS 0 tsecr 0
```

This indicates that Windows XP is much more strict in not replying to strange packets. Windows XP and Vista 5270 will accept a SYN packet with the ECN bit set, while the 5231 build would not.

APPENDIX XII – TCP SEGMENT REASSEMBLY

Vista's networking stack behaves differently than earlier versions in Windows XP or Windows 2000 when reassembling TCP segments. Vista uses a policy that prefers previously received data to later received data. This preference is enforced on a byte-by-byte basis and not across entire segments.

.

We performed our testing by sending out several out-of-order segments in a TCP stream that contained conflicting data. We then observed the stream data that were delivered to the application layer on the target machine. Tests were performed using the `segorder.py` tool. `Segorder.py` listens for SYN packets sent to a test address. After receiving a SYN, it performs the three-way handshake to establish a connection with the sender followed by some overlapping segments and finally a RST packet. The host being tested must be primed by adding a static ARP entry for the bogus test address which points to the MAC address that `segorder.py` is running on (i.e. **`arp -s 10.200.200.199 00-04-e2-0b-41-21`**). To test a host, `netcat` is used to connect to the test address, and the results received by netcat are recorded (i.e. **`nc 10.200.200.99 999`**).

A. Test Data

Seven segments containing ambiguous data were sent out with four bytes of data each. Each segment overlapped at least one other segment by two bytes. The following diagram indicates how the segments' data overlapped:

```
Segment #1          2222
Segment #2                5555
Segment #3                  6666
Segment #4              4444
Segment #5              oooo
Segment #6            3333
Segment #7        1111
```

After reassembly the resulting TCP stream received was:

Ethereal	`111133335555`
Linux RedHat8	`11112233445566`
Windows 2000	`11112244445566`
Windows XP	`11112244445566`
Windows Vista	`11222244555566`

B. Analysis

Windows Vista resolved all conflicts by preferring bytes from segments that were received earliest. This behavior differs from the behavior of earlier versions such as Windows 2000 and Windows XP which seem to process segments in the order they are received by first trimming any excess from the left and then using the rest of the segment in its entirety, overwriting any existing data.

APPENDIX XIII – WINDOWS FIREWALL CONFIGURATION

We measured the Windows Firewall configuration on a clean 5270 machine and noted the changes that occurred when configuration changes were made to the Windows Vista system. We performed our testing by noting the settings in the Windows Firewall control panel under the Exceptions tab and Advanced tab. Although we constructed a program printout of this information in a uniform manner using documented APIs, we noticed that its output did not always agree with the information in the control panel or the observed behavior. The control panel wasn't completely accurate either – we noticed that after turning on File and Print Sharing we were able to ping the machine with ICMP echoes even though the "Allow incoming echo request" option remained unchecked in the control panel.

On a clean installation we were unable to reach any TCP services or elicit any replies from UDP services. In the 5231 build we received ICMP port unreachable messages when sending packets to unopened TCP ports, but in the later 5270 build no ICMP responses were elicited. Both 5231 and 5270 builds returned ICMP port unreachable messages when receiving packets on unopened UDP ports. This allows the machine's aliveness to be tested and UDP services to be enumerated.

When a user changes the Windows Firewall configuration, Windows Vista typically asks for the user's consent via the consent mechanism before making the change. There are instances when firewall configuration changes are made without any user consent. These exceptions are noted explicitly.

The Windows Firewall configuration on a clean installation of Windows Vista 5270 contains a single exception:

- Remote Assistance
  - Service: `C:\Windows\System32\msra.exe`
  - Scope: Any computer (including those on the Internet)

The `msra` executable is not running by default and this rule does not increase exposure except when `msra` is running.

If file or print sharing is turned on, the following exception is added to the firewall configuration:

- File and Printer Sharing
  - TCP ports 139, 445
  - UDP ports 137, 138
  - Scope: My network (subnet only)

Although not reported by the Windows Firewall control panel, we noticed that ICMP Echo messages were processed after File and Print Sharing was enabled.

If a user opts in to People Near Me in the network control panel, the following exceptions are added to the firewall configuration:

- Collaboration Infrastructure (People Near Me)
  - Service: `C:\Windows\System32\p2phost.exe`
  - Scope: Any computer (including those on the Internet)
- Collaboration Infrastructure (PNRP)
  - UDP port 3540
  - Scope: Any computer (including those on the Internet)

Opting in to People Near Me causes starts `p2phost` but this service is not restarted when the user reboots or logs back in. While `p2phost` is active, it listens on an ephemeral TCP port (such as 49161) and uses UDP port 3702 and a couple of ephemeral UDP ports for IPv4 (such as 51940 and 51943) and IPv6 (such as 59141 and 51944). All these ports will be exposed while `p2phost` is running.

If a user invokes Windows Collaboration, Windows Vista will direct the user to opt in to People Near Me (see the earlier exceptions related to PNM) and to accept the following firewall exceptions:

- Ad-hoc Meetings
  - Service: `C:\Program Files\AdHocMeetings\SpacesContainer.exe`
  - Scope: My network (subnet only)
- Ad-hoc Meetings (P2P Grouping)
  - TCP port 3587
  - Scope: My network (subnet only)

The `SpacesContainer.exe` binary is the Windows Collaboration program. While running, it listens on UDP port 3702 and an ephemeral port for IPv4 (such as 54744) and IPv6 (such as 54745).

While Teredo is running, it occasionally registers Windows Firewall exceptions without the user's consent. These exceptions may be later removed or may persist while Teredo is running.

- Teredo
  - UDP port 3544
  - Scope: Any computer (including those on the Internet)
- Teredo
  - UDP port 61587
  - Scope: Any computer (including those on the Internet)

Any number of Teredo exceptions may be present; most exceptions will be for ephemeral ports such as 61587, although an exception for the standard Teredo port (UDP 3544) or a configured Teredo client port is typically present.

APPENDIX XIV – EXPOSED SERVICES

We used traditional port-scanning techniques to identify exposed TCP and UDP services running on a default Windows Vista installation.

Running `nmap`[11] against a clean installation of Vista 5270 reported that all TCP ports over IPv4 were filtered. Running the same scan on the earlier 5231 beta gave the following results:

```
linux# nmap -P0 -sT -p1-65535 10.200.200.127

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-12-06 11:05 HST
Interesting ports on 10.200.200.127:
(The 65526 ports scanned but not shown below are in state: closed)
PORT      STATE    SERVICE
135/tcp   open     msrpc
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
3500/tcp  filtered unknown
49152/tcp filtered unknown
49153/tcp open     unknown
49154/tcp open     unknown
49155/tcp filtered unknown
49156/tcp filtered unknown
MAC Address: 00:14:C2:D5:7E:96 (Unknown)

Nmap finished: 1 IP address (1 host up) scanned in 1297.027 seconds
```

However, with the more recent build 5384, RSTs were generated for unserviced ports yielding:

```
Interesting ports on 192.168.0.200:
(The 65527 ports scanned but not shown below are in state: closed)
PORT      STATE    SERVICE
135/tcp   filtered msrpc
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
49152/tcp filtered unknown
49153/tcp filtered unknown
49154/tcp filtered unknown
49155/tcp filtered unknown
49156/tcp filtered unknown
49157/tcp filtered unknown
MAC Address: 00:C0:9F:D2:0C:F8 (Quanta Computer)

Nmap finished: 1 IP address (1 host up) scanned in 92.517 seconds
```

So, 3500 went away and 49157 appeared. In addition, no ports are open any longer. With the firewall off, we can see these nine ports are open.

Similar results are observed when using IPv6. This test was performed using a custom-written `tcpscan` utility, which works with both IPv4 and IPv6 addresses. Running `tcpscan` against Vista 5270 reported that all TCP ports were filtered (i.e. the connection timed out). The same scan against the earlier Vista 5231 gave the following results:

```
linux# tcpscan -p 1-65535 fe80::214:c2ff:fed5:7e96%6
135 open
445 Connection timed out
3500 Connection timed out
49152 Connection timed out
49153 Connection timed out
```

```
49154 Connection timed out
49155 Connection timed out
49156 Connection timed out
```

Interestingly, in build 5231, TCP ports 49153 and 49154 are filtered when reached over IPv6 but not when reached over IPv4. TCP over IPv6 experienced the same reversion in behavior as TCP over IPv4, so we got the following useful result, even with the firewall on:

```
135 Connection timed out
445 Connection timed out
49152 Connection timed out
49153 Connection timed out
49154 Connection timed out
49155 Connection timed out
49156 Connection timed out
49157 Connection timed out
```

Again, 3500 disappeared and 49157 appeared, as compared to build 5231. We also see that port 139 was available only over IPv4.

Vista 5384 did not respond on any UDP ports over IPv4, but using `nmap` to scan a clean Vista 5270 install gives the following results:

```
linux# nmap -sU -p1-65535 10.200.200.123
Interesting ports on 10.200.200.123:
(The 65526 ports scanned but not shown below are in state: closed)
PORT      STATE         SERVICE
123/udp   open|filtered ntp
137/udp   open|filtered netbios-ns
138/udp   open|filtered netbios-dgm
500/udp   open|filtered isakmp
1900/udp  open|filtered UPnP
3544/udp  open|filtered unknown
3702/udp  open|filtered unknown
4500/udp  open|filtered sae-urn
5355/udp  open|filtered unknown
49672/udp open|filtered unknown
Nmap finished: 1 IP address (1 host up) scanned in 19.492 seconds
```

With the firewall off on 5384, we see that ports 3544 and 49672 no longer appear but there are a couple variable ephemeral ports. There appears to be a trend towards removing UDP services.

The earlier 5231 build had similar results with a few extra ports open:

```
linux# nmap -P0 -sU -p1-65535 10.200.200.127

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-12-09 15:17 HST
Interesting ports on 10.200.200.127:
(The 65524 ports scanned but not shown below are in state: closed)
PORT      STATE         SERVICE
123/udp   open|filtered ntp
137/udp   open|filtered netbios-ns
138/udp   open|filtered netbios-dgm
500/udp   open|filtered isakmp
1900/udp  open|filtered UPnP
3544/udp  open|filtered unknown
3702/udp  open|filtered unknown
4500/udp  open|filtered sae-urn
49152/udp open|filtered unknown
57431/udp open|filtered unknown
```

```
57437/udp open|filtered unknown
MAC Address: 00:14:C2:D5:7E:96 (Unknown)

Nmap finished: 1 IP address (1 host up) scanned in 18.810 seconds
```

We performed a UDP scan using a custom written tool called `udpscan`. The `udpscan` tool works with both IPv4 and IPv6. Unlike for UDP over IPv4, UDP over IPv6 for 5384 still responds with an error on closed ports, so in one scan we can see:

```
123 opened or filtered
500 opened or filtered
1900 opened or filtered
3702 opened or filtered
5355 opened or filtered
49230 opened or filtered
49235 opened or filtered
```

However, a separate scan shows that the two high-numbered ephemeral ports can vary.

The IPv6 UDP scan for Vista 5270 reported:

```
linux# udpscan -p 1-65535 fe80::214:c2ff:fed5:7e96%4
123 opened or filtered
500 opened or filtered
1900 opened or filtered
3702 opened or filtered
5355 opened or filtered
```

While the IPv6 UDP scan for Vista 5231 reported:

```
linux# udpscan -p 1-65535 fe80::214:c2ff:fed5:7e96%6
123 opened or filtered
500 opened or filtered
1900 opened or filtered
3540 opened or filtered
3702 opened or filtered
53662 opened or filtered
```

In Vista 5384, the UDP ports reachable over IPv6 were a strict subset of those reachable over IPv4 (in particular 137, 138, and 4500 are not present). In build 5231 the changes were more drastic with less ports available, and, more interestingly, 3540 and 53662 are available only in IPv6. The 53662 port was being used for Teredo tunneling when this scan was performed.

There were no externally reachable TCP ports in Vista 5270 and 5384. The only externally reachable TCP ports in Vista 5231 are the MSRPC endpoint mapper (on port 135) and two MSRPC-based services on port 49154 and 49155. None of the open UDP ports elicited a response, but many were listening.

We enumerated the RPC services in Vista 5231 using the `epdump.py` utility available in the `dcerpc` suite. Querying the endpoint mapper gave the following results:

```
linux$ epdump.py -T -p 135 10.200.200.127
0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53 1.0:
  LPC: IUserProfile2
  LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
  LPC: senssvc

0b6edbfa-4a24-4fc6-8a23-942b1eca65d1 1.0: Spooler function endpoint
  LPC: spoolss

0b7edbfa-4b24-4fd6-9a23-952b1eca65d1 1.0: Spooler function endpoint
  LPC: spoolss
```

```
12345678-1234-abcd-ef00-0123456789ab 1.0: IPSec Policy agent endpoint
  LPC: LRPC-7270c5b1e46331efec

12345778-1234-abcd-ef00-0123456789ac 1.0:
  LPC: CastleLsa
  LPC: LRPC-44eac19cded0a9fea7
  LPC: audit
  LPC: protected_storage
  LPC: securityevent
  path: \PIPE\protected_storage
  path: \pipe\lsass
  tcp 49155

1ff70682-0a51-30e8-076d-740be8cee98b 1.0:
  LPC: IUserProfile2
  LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
  LPC: senssvc
  path: \PIPE\atsvc

24019106-a203-4642-b88d-82dae9158929 1.0:
  LPC: LRPC-0bd58ea0f222a8df72
  path: \pipe\6f7e04f732ea9f64

2eb08e3e-639f-4fba-97b1-14f878961076 1.0:
  LPC: IUserProfile2
  LPC: OLEC0ADE9115F00429FACDBF8C0EFCB

2fb92682-6599-42dc-ae13-bd2ca89bd11c 1.0: Fw APIs
  LPC: eventlog
  path: \pipe\eventlog

367abb81-9844-35f1-ad32-98f038001003 2.0:
  tcp 49156

378e52b0-c0a9-11cf-822d-00aa0051e40f 1.0:
  LPC: IUserProfile2
  LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
  LPC: senssvc
  path: \PIPE\atsvc

3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 1.0: DHCP Client LRPC Endpoint
  LPC: DNSResolver
  LPC: OLE66311F0901B64FE1B553CE7C99B5
  LPC: dhcpcsvc
  tcp 49153

3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 1.0: DHCPv6 Client LRPC Endpoint
  LPC: DNSResolver
  LPC: OLE66311F0901B64FE1B553CE7C99B5
  LPC: dhcpcsvc
  LPC: dhcpcsvc6
  tcp 49153

4b112204-0e19-11d3-b42b-0000f81feb9f 1.0:
  LPC: W32TIME_ALT
  path: \PIPE\DAV RPC SERVICE
  path: \PIPE\W32TIME_ALT

76f226c3-ec14-4325-8a99-6a46348418af 1.0:
  LPC: WMsgKRpc011FE90
  LPC: WMsgKRpc0123AD1
```

```
  LPC: WindowsShutdown
  path: \PIPE\InitShutdown

7ea70bcf-48af-4f6a-8968-6a440754d5fa 1.0: NSI server endpoint
  LPC: DNSResolver
  LPC: OLE66311F0901B64FE1B553CE7C99B5
  tcp 49153

86d35949-83c9-4044-b424-db363231fd0c 1.0:
  LPC: IUserProfile2
  LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
  LPC: senssvc
  path: \PIPE\atsvc
  tcp 49154

a398e520-d59a-4bdd-aa7a-3c1e0303a511 1.0: IKE/Authip API
  LPC: AudioClientRpc
  LPC: Audiosrv
  LPC: IUserProfile2
  LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
  LPC: SECLOGON
  LPC: keysvc
  LPC: keysvc2
  LPC: senssvc
  LPC: trkwks
  path: \PIPE\atsvc
  path: \PIPE\srvsvc
  path: \PIPE\wkssvc
  path: \pipe\keysvc
  path: \pipe\trkwks
  tcp 49154

ae33069b-a2a8-46ee-a235-ddfd339be281 1.0: Spooler base remote object endpoint
  LPC: spoolss

b58aa02e-2884-4e97-8176-4ee06d794184 1.0:
  LPC: AudioClientRpc
  LPC: Audiosrv
  LPC: IUserProfile2
  LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
  LPC: SECLOGON
  LPC: keysvc
  LPC: keysvc2
  LPC: senssvc
  LPC: trkwks
  path: \PIPE\atsvc
  path: \PIPE\srvsvc
  path: \PIPE\wkssvc
  path: \pipe\keysvc
  path: \pipe\trkwks
  tcp 49154

c9ac6db5-82b7-4e55-ae8a-e464ed7b4277 1.0: Impl friendly name
  LPC: AudioClientRpc
  LPC: AudioClientRpc
  LPC: Audiosrv
  LPC: Audiosrv
  LPC: CastleLsa
  LPC: IUserProfile2
  LPC: IUserProfile2
  LPC: IUserProfile2
```

```
    LPC: IUserProfile2
    LPC: IUserProfile2
    LPC: LRPC-44eac19cded0a9fea7
    LPC: LRPC-6368ba464290b26ef6
    LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
    LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
    LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
    LPC: OLEC0ADE9115F00429FACDBF8C0EFCB
    LPC: SECLOGON
    LPC: SECLOGON
    LPC: audit
    LPC: keysvc
    LPC: keysvc
    LPC: keysvc2
    LPC: keysvc2
    LPC: protected_storage
    LPC: securityevent
    LPC: senssvc
    LPC: senssvc
    LPC: senssvc
    path: \PIPE\atsvc
    path: \PIPE\atsvc
    path: \PIPE\protected_storage
    path: \PIPE\wkssvc
    path: \PIPE\wkssvc
    path: \pipe\keysvc
    path: \pipe\keysvc
    path: \pipe\lsass
    tcp 49154
    tcp 49154

  d95afe70-a6d5-4259-822e-2c84da1ddb0d 1.0:
    LPC: WMsgKRpc011FE90
    LPC: WindowsShutdown
    path: \PIPE\InitShutdown
    tcp 49152

  dd490425-5325-4565-b774-7e27d6c09c24 1.0: Base Firewall Engine API
    LPC: eventlog
    path: \pipe\eventlog
```

The following is a summary of the services bound to a network interface as reported by the endpoint mapper. When several RPC services share the same process, they will share the same endpoints. As a result, there may be interfaces bound to a network port that are not registered with the endpoint mapper.

```
  12345778-1234-abcd-ef00-0123456789ac 1.0:
    tcp 49155
  367abb81-9844-35f1-ad32-98f038001003 2.0:
    tcp 49156
  3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 1.0: DHCP Client LRPC Endpoint
    tcp 49153
  3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 1.0: DHCPv6 Client LRPC Endpoint
    tcp 49153
  7ea70bcf-48af-4f6a-8968-6a440754d5fa 1.0: NSI server endpoint
    tcp 49153
  86d35949-83c9-4044-b424-db363231fd0c 1.0:
    tcp 49154
  a398e520-d59a-4bdd-aa7a-3c1e0303a511 1.0: IKE/Authip API
    tcp 49154
  b58aa02e-2884-4e97-8176-4ee06d794184 1.0:
```

```
  tcp 49154
c9ac6db5-82b7-4e55-ae8a-e464ed7b4277 1.0: Impl friendly name
  tcp 49154
d95afe70-a6d5-4259-822e-2c84da1ddb0d 1.0:
  tcp 49152
```

We wrote a tool, `identify.py` (also part of the `dcerpc` suite), that attempts to brute-force the interfaces actually available over a network port. The tool sends malformed requests to a port using a large list of known UUIDs. It is able to distinguish which interfaces are available by the error message sent back. Using this tool, we were able to enumerate most RPC services that were bound to a port. Additionally, the error message indicates which services require authentication. Not all interfaces available on a network port are actually usable; there are RPC mechanisms for blocking requests arriving over the network ([17] and section 4.11 of [15]). This is useful for services that do not wish to be available over the network but share a process with another service that uses a network transport.

The `identify.py` script identified the following interfaces available on the open TCP ports:

```
linux$ identify.py -p 135 -T 10.200.200.127
00000136-0000-0000-c000-000000000046 0 0 ACCESS_DENIED ISCMLocalActivator
000001a0-0000-0000-c000-000000000046 0 0 ACCESS_DENIED ISystemActivator
0b0a6584-9e0f-11cf-a3cf-00805f68cb1b 1 1 ACCESS_DENIED localpmp
1d55b526-c137-46c5-ab79-638f2a68e869 1 0 ACCESS_DENIED ???
412f241e-c12a-11ce-abff-0020af6e7a17 0 2 ACCESS_DENIED ISCM
64fe0b7f-9ef5-4553-a7db-9a1975777554 1 0 ACCESS_DENIED ???
99fcfec4-5260-101b-bbcb-00aa0021347a 0 0 RANGE_ERROR IOXIDResolver 0-6
afa8bd80-7d8a-11c9-bef4-08002b102989 1 0 ACCESS_DENIED rpcmgmt (ifids)
b9e79e60-3d52-11ce-aaa1-00006901293f 0 2 ACCESS_DENIED IROT
c6f3ee72-ce7e-11d1-b71e-00c04fc3111a 1 0 ACCESS_DENIED IMachineActivatorControl
e1af8308-5d1f-11c9-91a4-08002b14a0fa 3 0 RANGE_ERROR epmapper 0-9
e60c73e6-88f9-11cf-9af1-0020af6e72f4 2 0 ACCESS_DENIED ILocalObjectExporter

linux$ identify.py -p 49153 -T 10.200.200.127
18f70770-8e64-11cf-9af1-0020af6e72f4 0 0 ACCESS_DENIED ole32 (IOrCallback)
3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 1 0 ACCESS_DENIED dhcpcsvc (RpcSrvDHCPC)
3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 1 0 ACCESS_DENIED dhcpcsvc6
7ea70bcf-48af-4f6a-8968-6a440754d5fa 1 0 ACCESS_DENIED nsisvc
aa411582-9bdf-48fb-b42b-faa1eee33949 1 0 ACCESS_DENIED ???
afa8bd80-7d8a-11c9-bef4-08002b102989 1 0 ACCESS_DENIED rpcmgmt (ifids)
c33b9f46-2088-4dbc-97e3-6125f127661c 1 0 ACCESS_DENIED ???

linux$ identify.py -p 49154 -T 10.200.200.127
000001a0-0000-0000-c000-000000000046 0 0 ACCESS_DENIED ISystemActivator
0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53 1 0 ACCESS_DENIED taskeng (idletask)
0d72a7d4-6148-11d1-b4aa-00c04fb66ea0 1 0 ACCESS_DENIED ICertProtect
12b81e99-f207-4a4c-85d3-77b42f76fd14 1 0 ACCESS_DENIED seclogon (ISeclogon)
18f70770-8e64-11cf-9af1-0020af6e72f4 0 0 ACCESS_DENIED ole32 (IOrCallback)
1ff70682-0a51-30e8-076d-740be8cee98b 1 0 ACCESS_DENIED atsvc
2eb08e3e-639f-4fba-97b1-14f878961076 1 0 ACCESS_DENIED ???
300f3532-38cc-11d0-a3f0-0020af6b0add 1 2 ACCESS_DENIED trkwks
326731e3-c1c0-4a69-ae20-7d9044a4ea5c 1 0 ACCESS_DENIED profsvc (IUserProfile)
378e52b0-c0a9-11cf-822d-00aa0051e40f 1 0 ACCESS_DENIED sasec
3faf4738-3a21-4307-b46c-fdda9bb8c0d5 1 1 ACCESS_DENIED AudioSrv
629b9f66-556c-11d1-8dd2-00aa004abd5e 3 0 ACCESS_DENIED SENSNotify
63fbe424-2029-11d1-8db8-00aa004abd5e 1 0 ACCESS_DENIED SensApi
68b58241-c259-4f03-a2e5-a2651dcbc930 1 0 ACCESS_DENIED ???
6bffd098-a112-3610-9833-46c3f87e345a 1 0 ACCESS_DENIED wkssvc
86d35949-83c9-4044-b424-db363231fd0c 1 0 ACCESS_DENIED ???
a398e520-d59a-4bdd-aa7a-3c1e0303a511 1 0 ACCESS_DENIED IKEEXT
afa8bd80-7d8a-11c9-bef4-08002b102989 1 0 ACCESS_DENIED rpcmgmt (ifids)
b58aa02e-2884-4e97-8176-4ee06d794184 1 0 ACCESS_DENIED sysmain
c386ca3e-9061-4a72-821e-498d83be188f 1 1 ACCESS_DENIED ???
```

```
c9ac6db5-82b7-4e55-ae8a-e464ed7b4277 1 0 ACCESS_DENIED sysntfy
f50aac00-c7f3-428e-a022-a6b71bfb9d43 1 0 ACCESS_DENIED ICatDBSvc

linux$ identify.py -p 135 -T6 fe80::214:c2ff:fed5:7e96%6
00000136-0000-0000-c000-000000000046 0 0 ACCESS_DENIED ISCMLocalActivator
000001a0-0000-0000-c000-000000000046 0 0 ACCESS_DENIED ISystemActivator
0b0a6584-9e0f-11cf-a3cf-00805f68cb1b 1 1 ACCESS_DENIED localpmp
1d55b526-c137-46c5-ab79-638f2a68e869 1 0 ACCESS_DENIED ???
412f241e-c12a-11ce-abff-0020af6e7a17 0 2 ACCESS_DENIED ISCM
64fe0b7f-9ef5-4553-a7db-9a1975777554 1 0 ACCESS_DENIED ???
99fcfec4-5260-101b-bbcb-00aa0021347a 0 0 RANGE_ERROR IOXIDResolver 0-6
afa8bd80-7d8a-11c9-bef4-08002b102989 1 0 ACCESS_DENIED rpcmgmt (ifids)
b9e79e60-3d52-11ce-aaa1-00006901293f 0 2 ACCESS_DENIED IROT
c6f3ee72-ce7e-11d1-b71e-00c04fc3111a 1 0 ACCESS_DENIED IMachineActivatorControl
e1af8308-5d1f-11c9-91a4-08002b14a0fa 3 0 RANGE_ERROR epmapper 0-9
e60c73e6-88f9-11cf-9af1-0020af6e72f4 2 0 ACCESS_DENIED ILocalObjectExporter
```

The exposure of shared-process servers gives some insight into what other services are running on the machine. This information can be used for fingerprinting purposes[31].

APPENDIX XV – ANONYMOUS ACCESS TO NAMED PIPES

We determined which named pipes were remotely accessible without providing authentication. This was done by first enabling file sharing on the remote machine (a prerequisite to remote access of named pipes). We then enumerated all of the named pipes locally using the `pipelist.exe` tool[7].

```
vista> pipelist > pipes.txt
vista> type pipes.txt
PipeList v1.01
by Mark Russinovich
http://www.sysinternals.com

Pipe Name                                      Instances       Max Instances
---------                                      ---------       -------------
InitShutdown                                       2               -1
lsass                                              4               -1
protected_storage                                  2               -1
ntsvcs                                             2               -1
scerpc                                             2               -1
net\NtControlPipe1                                 1                1
plugplay                                           2               -1
net\NtControlPipe2                                 1                1
Winsock2\CatalogChangeListener-350-0               1                1
epmapper                                           2               -1
Winsock2\CatalogChangeListener-208-0               1                1
net\NtControlPipe3                                 1                1
LSM_API_service                                    2               -1
net\NtControlPipe4                                 1                1
eventlog                                           2               -1
5373d71f34c9a06b                                   2               -1
Winsock2\CatalogChangeListener-3c0-0               1                1
net\NtControlPipe5                                 1                1
net\NtControlPipe6                                 1                1
net\NtControlPipe7                                 1                1
net\NtControlPipe8                                 1                1
net\NtControlPipe9                                 1                1
net\NtControlPipe0                                 1                1
net\NtControlPipe10                                1                1
Winsock2\CatalogChangeListener-42c-0               1                1
atsvc                                              2               -1
Winsock2\CatalogChangeListener-400-0               1                1
89199f2923fa487c011aa0740144fd14                   1                1
net\NtControlPipe11                                1                1
3edbd0599284422b00a7783c0006fdbc                   1                1
DAV RPC SERVICE                                    2               -1
Winsock2\CatalogChangeListener-244-0               1                1
wkssvc                                             3               -1
srvsvc                                             3               -1
browser                                            2               -1
net\NtControlPipe12                                1                1
net\NtControlPipe13                                1                1
keysvc                                             2               -1
net\NtControlPipe14                                1                1
net\NtControlPipe15                                1                1
net\NtControlPipe16                                1                1
net\NtControlPipe17                                1                1
trkwks                                             2               -1
net\NtControlPipe18                                1                1
```

[7] The `pipelist.exe` tool was previously released online at www.sysinternals.com, but appears to no longer be available.

```
Winsock2\CatalogChangeListener-730-0              1                1
W32TIME_ALT                                       2               -1
INETINFO                                          2               -1
SMTPSVC                                           2               -1
SQLLocal\SQLEXPRESS                               2               -1
MSSQL$SQLEXPRESS\sql\query                        2               -1
Winsock2\CatalogChangeListener-234-0              1                1
USearch                                           2              254
a991e6c5c51e4d83011adbb40144fc68                  1                1
1566e57c69534197011af1940144fc68                  1                1
e174b15fda70473300cf78940006fdbc                  1                1
95c9fc10b11d4e9400cf788c0006fdbc                  1                1
PIPE_EVENTROOT\CIMV2SCM EVENT PROVIDER            2               -1
net\NtControlPipe19                               1                1
Winsock2\CatalogChangeListener-bac-0              1                1
40abfd5a2fc5467e011ad70c0144fd14                  1                1
9afc8331c39d458700a778440006fdbc                  1                1
Winsock2\CatalogChangeListener-594-0              1                1
```

We also enumerated the pipes in the HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\lanmanserver\Parameters\NullSessionPipes registry key using regedit:

- SQL\QUERY
- SPOOLSS
- netlogon
- lsarpc
- samr
- browser

We then established an anonymous connection to the IPC$ share of the target machine from another Windows host and used the `trypipes.py` script to establish connections to each of these named pipes to see which ones were actually reachable. When we ran the script from a Windows XP machine, we observed the following results:

```
xp> net use \\10.200.200.123\ipc$ /u:"" ""
xp> c:\python24\python trypipes.py -m 10.200.200.123 pipes.txt
\\10.200.200.123\PIPE\netlogon
\\10.200.200.123\PIPE\lsarpc
\\10.200.200.123\PIPE\samr
```

Unexpectedly, we were able to access more pipes when running the same tests from another Vista machine:

```
vista> c:\python24\python trypipes.py -m 10.200.200.123 pipes.txt
\\10.200.200.123\PIPE\lsass
\\10.200.200.123\PIPE\protected_storage
\\10.200.200.123\PIPE\netlogon
\\10.200.200.123\PIPE\lsarpc
\\10.200.200.123\PIPE\samr
```

These path names all refer to the same named pipe. The HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Npfs\Aliases\lsass key lists the following names:

- protected_storage
- netlogon
- lsarpc
- samr

We then enumerated the RPC services using these pipes (if any) by running the `identify.py` script from the `dcerpc` suite from a Vista host. The result was the same for all five pipe names:

```
vista> c:\python24\python identify.py -P -f netlogon 10.200.200.123
```

```
11220835-5b26-4d94-ae86-c3e475a809de 1 0 error -1 ICryptProtect
12345778-1234-abcd-ef00-0123456789ab 0 0 RANGE_ERROR LSA access (lsarpc) 0-95
12345778-1234-abcd-ef00-0123456789ac 1 0 RANGE_ERROR samsrv 0-70
3919286a-b10c-11d0-9ba8-00c04fd92ef5 0 0 RANGE_ERROR LSA DS access (lsarpc) 0-1
afa8bd80-7d8a-11c9-bef4-08002b102989 1 0 RANGE_ERROR rpcmgmt (ifids) 0-5
c681d488-d850-11d0-8c52-00c04fd90f7e 1 0 RANGE_ERROR efsrpc 0-25
c9ac6db5-82b7-4e55-ae8a-e464ed7b4277 1 0 error -1 sysntfy
```

APPENDIX XVI – UNSOLICITED TRAFFIC

We observed the network traffic of a clean installation of Windows Vista 5270 and 5231 during startup, in an idle period, and at shutdown. The machine was placed on a subnet with other Windows machines with a server that provided a DHCP lease and Internet connectivity. We observed that Windows Vista sent out a number of unsolicited protocol requests to other machines on the local network and elsewhere on the Internet. These observations were made by running Ethereal[8] on another host on the same network (a non-switching hub was used, but a switch could also be used if properly configured). The following is a summary of all traffic observed:

- IPv6 ICMP neighbor and router solicitation messages and multicast notifications.
- IPv4 ARP requests and responses
    - The first few arps were for the private address 169.254.126.150. When no reply was received, the machine started to use this address until a DHCP lease was acquired.
- IGMP membership
- LLMNR (Link Local Multicast Name Resolution, UDP 5355) traffic via IPv6 and IPv4 for
    - <hostname>
    - _ldap._tcp.dc._msdcs.<domain>
    - isatap, isatap.<domain>
    - wpad.<domain>
- DHCP
    - Request address
    - Inform during shutdown
- NBNS
    - registration <hostname>
    - registration WORKGROUP
    - lookup ISATAP
    - registration __MSBROWSE__
    - lookup WPAD.<domain>
    - query WORKGROUP
    - (5231 build used to lookup SQM)
- DNS
    - PTR lookup for <ip address>.in-addr.arpa
    - SRV lookup for _ldap._tcp.dc._msdcs.<domain> and _LDAP._TCP
    - A lookup for teredo.ipv6.microsoft.com
    - A lookup for isatap and isatap.<domain>
    - A lookup for time.windows.com
    - AAAA lookup for wpad.<domain> (during shutdown?)
    - (5231 build performed AAAA lookup for pnrpv2.ipv6.microsoft.com)
    - (5231 build performed AAAA and A lookup for sqm.msn.com)
- Teredo (IPv6 tunneled in UDP)
    - ICMP router solicitations
    - ICMP ECHO to pnrpv2.ipv6.microsoft.com
    - And other unanalyzed traffic (perhaps normal Teredo maintenance traffic?)
- EAPOL 802.1x Authentication (Ether type 0x888e) Start request
- Windows Browser (UDP 138) announcement and backup list.
- WSD (Web Services Discovery, UDP Port 3702 multicast traffic with XML SOAP payload) via IPv4 and IPv6.
- (5231 build sent HTTP POST to port 80 of sqm.msn.com during shutdown)
    - POST /sqm/windows/sqmserver.dll