Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP)

Bruce Schneier
schneier@counterpane.com
Counterpane Systems,
101 East Minnehaha Parkway,
Minneapolis, MN 55419

Mudge
mudge@l0pht.com
L0pht Heavy Industries
P.O. Box 990857
Boston, MA 02199

Abstract

The Point-to-Point Tunneling Protocol (PPTP) is used to secure PPP connections over TCP/IP links. In this paper we analyze Microsoft's Windows NT implementation of PPTP. We show how to break both the challenge/response authentication protocol (Microsoft CHAP) and the RC4 encryption protocol (MPPE), as well as how to attack the control channel in Microsoft's implementation. These attacks do not necessarily break PPTP, but only Microsoft's implementation of the protocol.

1 Introduction

Many organizations and institutions are not centralized. Branch offices, virtual corporations, and traveling employees make the notion of running dedicated network connections to each location logistically impossible. The concept of virtual networking provides a solution to this problem by tunneling cojoined network space over other, transitory and insecure, networks (such as the Internet), thus enabling remote locations to appear to be local. This is done without the expense incurred from running leased lines or dedicated cabling to each location, and is sometimes called a "tunnel."

While virtual networks solve the problem of decentralized machines, they create a new problem. They open up traffic that was previously considered internal to the company, to any prying eyes on the networks it traverses. Authentication and encryption are required to keep this virtual network traffic not only tamperproof but private. The result, virtual networking connections combined with cryptographic protections, is a Virtual Private Network (VPN).

The security of a VPN is based on the security of its authentication and encryption protocols. If a VPN's cryptography is weak, then its security is no better than a non-private virtual network routed over the Internet. Since companies are relying upon VPNs to extend trusted internal perimiters to remote offices, breaking the security around such a tunnel is tantamount to defeating all of the security around the internal perimiter. Breaking into a VPN is often the same as penetrating the firewall.

The Point-to-Point Tunneling Protocol (PPTP) was designed to solve this problem of creating and maintaining a VPN over a public TCP/IP network using the common Point-to-Point Protocol (PPP). Although the protocol leaves room for every type of encryption and authentication imaginable, most commercial products use the Microsoft Windows NT version of the protocol. This is the implementation that we cryptanalyze in this paper.

We have found Microsoft's authentication protocol to be weak and easily susceptible to a dictionary attack; most passwords can be recovered within hours. We have found the encryption–both 40-bit and 128-bit–to be equally weak, and have discovered a series of bad design decisions that make other attacks against this encryption possible. We can open connections through a firewall by abusing the PPTP negotiations, and can mount several serious denial-of-service attacks on anyone who uses Microsoft PPTP.

The remainder of this paper is divided into sections as follows: In Section 2 we describe PPTP, both the generic protocol and Microsoft's implementation. In Section 3, we describe the two password hashing functions in Microsoft PPTP and describe how to attack them. In Section 4 we cryptanalyze Microsoft's authentication protocol, and in Section 5 we cryptanalyze Microsoft's encryption protocol. We look at other attacks against Microsoft PPTP in Section 6. Finally, in Section 7 we attempt to reach some conclusions.

2 Point-to-Point Tunneling Protocol

PPTP [HPV+97] is a protocol that allows PPP connections [Sim94] to be tunneled through an IP network, creating a VPN. Thus, a remote machine on network X can tunnel traffic to a gateway machine on network Y and appear to be sitting, with an internal IP address, on network Y. The gateway machine receives traffic to this internal IP address, and sends it back to the remote machine on network X. There are two primary ways of using PPTP, either directly over the Internet or through dial up services. This paper focuses on the use of PPTP as a Virtual Private Network where the client is directly attached to the Internet.

PPTP works by encapsulating the virtual network packets inside of PPP packets, which are in turn encapsulated in Generic Routing Encapsulation (GRE) [HLFT94] packets sent over IP from the client to the gateway PPTP server and back again. In conjunction with this encapsulated data channel, there is a TCP-based control session. The control session packets are used to query status and to convey signaling information between the client and the server. The control channel is initiated by the client to the server on TCP port 1723. In most cases this is a bi-directional communication channel where the client can send requests to the server and vice-versa.

PPTP does not specify specific algorithms for authentication and encryption; instead it provides a framework for negotiating particular algorithms. This negotiation is not specific to PPTP, and relies upon existing PPP option negotiations contained within CCP, CHAP, and other PPP extensions and enhancements [LS92, Mey96, Ran96, Sim96, BV98]. Just as PPP sessions have been able to netogiate compression algorithms, they can negotiate authentication or encryption algorithms. Appendix A provides details of this negotiation process.

2.1 Microsoft PPTP

Microsoft PPTP [Mic96a, Mic96b] is part of Windows NT Server, and can be downloaded free from the Microsoft website [Kli98] and enabled using the Windows Network Control Panel and the Registry Editor. This implementation of PPTP is used extensively in commercial VPN products precisely because it is already a part of the Microsoft operating systems.

The Microsoft PPTP server can only be run under Windows NT, although client software exists for Windows NT, Windows some, and Windows 98. There are three authentication options supported in the Microsoft implementations:

1. Clear Password: The client sends the server a password in the clear.
2. Hashed Password: The client sends the server a hash of the password, as described in Section 3.
3. Challenge/Response: The client and the server authenticate using the MS-CHAP challenge/response protocol, as described in Section 4.

The third option is called "Microsoft Authentication" in the user documentation, and must be enabled in order for PPTP packets to be encrypted. With either of the other two authentication options, no encryption is possible. Additionally, encryption (either 40-bit or 128-bit) is only guaranteed to be possible if the client is running Windows NT; some Windows 95 clients cannot establish encrypted sessions.[1]

3 Cryptanalysis of Windows NT Password Hash Functions

Microsoft Windows NT uses two one-way hash functions to protect passwords: the Lan Manager hash and the Windows NT hash [ZC98]. The Lan Manager hash function was developed by Microsoft for IBM's OS2 operating system, and was integrated into Windows for Workgroups and optionally in Windows 3.1, and is used in several additional pre-Windows NT authentication protocols. The Windows NT hash was developed by Microsoft specifically for Windows NT. The Lan Manager hash is based on the DES encryption algorithm [NBS77]; the Windows NT hash is based on MD4 one-way hash function [Riv91]. Both of these hash functions are used in many Windows NT authentication protocols, not just PPTP.

The Lan Manager hash is calculated as follows:

1. Turn the password into a 14-character string, either by truncating longer passwords or padding shorter passwords with nulls.
2. Convert all lowercase characters to uppercase. Numbers and non-alphanumerics remain unaffected.
3. Split the 14-byte string into two seven-byte halves.
4. Using each seven-byte string as a DES key, encrypt a fixed constant with each key, yielding two 8-byte encrypted strings.
5. Concatenate the two strings together to create a single 16-byte hash value.

Dictionary attacks are easy against the Lan Manager hash for the following reasons [L97b]:

- Most people choose easily guessable passwords [Kle90].
- All characters are converted to upper case, making the number of possible passwords even smaller.
- There is no salt; two users with the same password will always have the same hashed password. Thus, it is possible to precompute a dictionary of hashed passwords and compare an unknown password

[1] According to our experiments, some Windows 95 clients support Microsoft Authentication and some do not. We do not know what the difference is, or how to guarantee that a particular Windows 95 system support Microsoft Authentication. If the protocol is not supported, the option is greyed out on the dialog box. This restriction is consistent with Microsoft's claim that Windows 95 does not provide security, and that users who are interested in security should upgrade to Windows NT. However, Microsoft has claimed that Windows 95 clients cannot perform the Windows NT hash, and require the Lan Manager hash. As it turns out, Windows 95 clients transmit both the Windows NT hash and the Lan Manager hash. From our examination of the Windows 95 code, there is no reason encryption cannot be enabled in all Windows 95 clients.

against the dictionary. With this time/memory trade-off, passwords can be tested as fast as disk I/O allows.

- The two seven-byte "halves" of the password are hashed independently. Thus, the two halves can be brute-forced independently, and the complexity of the attack is at most the complexity against a seven-byte password. Passwords longer than seven characters are no stronger than seven-character passwords. Additionally, passwords of seven characters or less can be immediately recognized since the second half of the hash is always the same constant: encryption of the fixed constant with seven nulls as the key.

The Windows NT hash is calculated as follows:

1. The password, up to 14 characters long and case-sensitive, is converted to Unicode.[2]
2. The password is hashed using MD4, yielding a 16 byte hash value.

The Windows NT hash is an improvement over the Lan Manager hash–case sensitivity, allowing passwords longer than 14 characters, and hashing the entire password together instead of in small sections–although there are no provisions for salt. Thus, two people with the same password will have the same Windows NT hashed password; comparing a file of hashed password with a precomputed dictionary of hashed passwords is still a very fruitful attack.

Another, more serious, implementation problem makes attacking the passwords even easier. Even though the Lan Manager hash was included for backwards compatibility, and is not required in Windows NT-only networks, both hashes are always sent together.[3] Therefor, it is possible to brute-force the password using the weaker Lan Manager hash, and then test various lower-case alternatives to find the Windows NT hash.[4]

[2] Microsoft documentation claims that Windows NT passwords can be up to 128 characters, and the Windows NT hash function accepts passwords of that length. However, the NT User Manager limits passwords to 14 characters or less [MB97]. The MS-CHAP documentation references this limitation [ZC98], which is also borne out by experimentation.

[3] In fact, since the only backwards compatibility required for Microst PPTP is Windows 95, there is no reasson to include it at all.

[4] A popular hacker tool, L0phtcrack [L97a], automates the process of recovering passwords from these hash values. On a Pentium Pro 200, L0phtcrack 2.0 can check a 200-entry password file against an 8 Megabyte dictionary of popular passwords in under a minute. Testing the entire 26-character alphabet space takes 26 hours, and the 36-character alphanumeric space takes about 250 hours. Adding non-alphanumerics significantly increases the difficulty of this search. Precompting the Lan Manager hash values of these passwords can increase the speed of these attacks by several orders of magnitude.

In 1997, Microsoft attempted to modify Windows NT authentication in response to L0phtcrack. They prevented the Lan Manager hash from being transmitted in a Windows NT-only environment. This patch appeared on their web site, but was removed when they discovered during regression testing that many NT protocols broke. Despite claims from Microsoft, the Lan Manager hash is required for many NT-to-NT communications. Still, Microsoft strongly recommends disabling the Lan Manager hash in instances where this is possible [Mic98].

4 Cryptanalysis of MS-CHAP

PPP has in it several methods for handling authentication. One of these is the Challenge Handshake Authentication Protocol (CHAP). Microsoft's PPP CHAP implementation (MS-CHAP) [ZC98] is almost identical to the authentication method that it uses for client authentication on its Windows-based networks.

MS-CHAP works as follows (see Appendix B for details):

1. Client requests a login challenge.
2. Server sends back an eight-byte random challenge.[5]
3. The client calculates the Lan Manager hash, and adds five nulls to create a 21-byte string, and partitions the string into three seven-byte keys. Each key is used to encrypt the challenge, resulting in a 24-byte encrypted value. This is returned to the Server as a response. The client does the same with the Windows NT hash.
4. Server looks up the hash in its database, encrypts the challenge with the hash, and compares it with the encrypted hashes it received. If they match, the authentication completes.

   The server could make the comparison on the Windows NT hash or the Lan Manager hash; the results would be the same. Which hash the server uses depends on a particular flag in the packet. If the flag bit is set, the server tests againts the Windows NT hash; if the flag bit is not set, the server tests against the Lan Manager hash.

On the surface, the challenge/response protocol is standard; the use of a random login challenge makes precomputed dictionary attacks impossible against MS-CHAP as they are against the file of stored password hashes. Still, because both the Lan Manager and Windows NT hashes are transmitted even in a Windows NT-only environment, it is possible to attack the weaker Lan Manager hash in every case. And because the client's reply is divided into thirds, and each third is encrypted independently, it is possible to attack the MS-CHAP protocol itself.

The last eight bytes of the Lan Manager hash is a constant if the password is seven characters or less. This is true despite the random challenge. Therefore, the last eight bytes of the Client's reply will be the challenge encrypted with that constant. It is easy to test whether a given password is seven characters or less. After an attacker finds the Lan Manager hash, he can use that information to recover the Windows NT hash.[6]

[5] We did not investigate the pseudo-random number generator used to generate this challenge nor its cryptographic strength [KSWH98].

[6] L0phtcrack 2.0 has automated this process as well, and can recover passwords after eavesdropping on a login session. Time estimates are longer than for brute-forcing the password, although large dictionary attacks are still feasible. Microsoft's response to L0phtcrack [Mic97] was to remind system administrators to protect the file of password hashes; this particular attack relies on communications across the public network, and does not require access to the hashed password file.

This attack can be sped up considerably through judicious use of precomputation and carefully exploiting the weaknesses of both the Lan Manager hash and the MS-CHAP protocol. Details of ths optimized attack follow:

P_0 through P_{13} are the bytes of the password. H_0 through H_{15} are the bytes of the Lan Manager hash, which becomes a 21-byte key: K_0 through K_{20}. S is the fixed constant used in the Lan Manager hash. The challenge is C and the 24-byte response is R_0 through R_{23}. An eavesdropper can learn C and R, and wants to find P.

1. Try all possible values of K_{14}, K_{15}. The right value can be recognised when C encrypts to $R_{16}, \ldots, R_{23}$ under the key $K_{14}, K_{15}, 0, 0, 0, 0, 0$. This takes an average of 2^{15} operations.

2. Try likely values of $P_7, \ldots, P_{13}$. Wrong values can be quickly disgarded by encrypting S under the guess and checking if the last two bytes of the ciphertext equal K_{14} and K_{15}. (This will eliminate all but 1 in 2^{16} of the wrong ones.) Each remaining guess of $P_7, \ldots, P_{13}$ yields a candidate for $K_8, \ldots, K_{13}$. To check the candidate, try all possible values for K_7 to see if there is any for which C encrypts to $R_8, \ldots, R_{15}$ under the $K_8, \ldots, K_{13}$ candidate. If there is such a K_7, then the guess for $P_7, \ldots, P_{13}$ is almost certainly correct. If not, try another candidate for $P_7, \ldots, P_{13}$. If there are N likely values of $P_7, \ldots, P_{13}$, this recovers the correct value with about N trial encryptions.

   Note that since there is no salt used in the protocol, this attack can be sped up considerably using a time/memory tradeoff. With N precomputed trial encryptions, recovering the correct value of $P_7, \ldots, P_{13}$ takes $N/2^{16}$ work.

3. Once $P_7, \ldots, P_{13}$ has been found, recovering values for $P_0, \ldots, P_6$ takes M trials, where M is the number of likely values of $P_0, \ldots, P_6$. Again, since there is no salt, the attack can be completed in $N/2^8$ trials with M precomputations.

Additionally, in this protocol only the Client is authenticated. An attacker who hijacks a connection can trivially masquerade as the Server. If encryption is enabled, the attacker will not be able to send and receive messages (unless he breaks the encryption), but by reusing an old challange value he can obtain two sessions of ciphertext encrypted with the same key (see atacks based on this below).

5 Cryptanalysis of MPPE

5.1 Description of MPPE

Microsoft Point-to-Point Encryption (MPPE) protocol [PZ98] provides a methodology of encrypting PPTP packets. It assumes the existence of a secret key shared by both ends of the connection, and uses the RC4 stream cipher [Sch96] with either a 40-bit key or a 128-bit key. The method for negotiating the use of MPPE is through an option in the PPP Compression Control Protocol (CCP) [Ran96, Pal96a] and is described in Appendix C. After these negotiations, the PPP session begins passing payload packets of encrypted data. It is important to note that only PPP packets whose protocol numbers are in the range `0x0021` to `0x00fa` are encrypted. All other packets are passed in the clear, even if the encryption option is enabled. RFC 1700 [RP94] lists the types of packets that are and are not encrypted.[7] There is no authentication provided for any packets.

In MPPE, the 40-bit RC4 key is determined as follows:

1. Generate a deterministic 64-bit key from a Lan Manager hash of the user's password (shared by both the user and the host) using SHA [NIST93].
2. Set the high-order 24 bits of the key to `0xD1269E`.

The 128-bit RC4 key is determined as follows:

1. Concatenate the Windows NT hash of the user's password and a 64-bit random nonce created by the host during the MS-CHAP protocol. This nonce was sent to the client during the protocol, so is known by both the client and the server.
2. Generate a deterministic 128-bit key from the results of the previous step using SHA.

The resulting key is used to initialize RC4 in the usual manner, and then to encrypt data bytes. After every 256 packets—MPPE maintains a "coherency count" that records the packet number—a new RC4 key is generated using the following procedure:

1. Generate a deterministic key—64 bits long for 40-bit encryption and 128 bits long for 128-bit encryption—by hashing the previous key and the original key with SHA.
2. If the required key is 40 bits, set the high-order 24 bits of the key to `0xD1269E`.

A typical PPTP packet is about 200 bytes long, including header data.

In the event of synchronization loss, RC4 is reinitialized with the current key. There is also an option to update the RC4 key after every packet; this option reduces the efficiency of encryption by about half because of the time required to execute the RC4 key schedule.

5.2 Recovering the Key

In MPPE, the security of the key is no greater than the security of the password. Most passwords have much less than 40 bits of security and are susceptible to dictionary attacks [Kle90]. The Lan Manager hash is even more

[7] While MPPE will encrypt PPP packets containing IP, Novell IPX, Van Jacobsen Compressed/Uncompressed TCP/IP and other packets within this range, many important PPP packets will not be encrypted. Examples include LCP, PAP, CBCP, CHAP, IPCP, among others.

vulnerable; because of the maximum sie, limited alphabet and lack of lower-case characters, it is impossible to generate a 128-bit key even if the user wanted to. The MPPE documentation includes a flag for calculating the 40-bit RC4 key based on the Windows NT hash instead of the Lan Manager hash, but this feature has not yet been implemented. There are no provisions for calculating the 128-bit RC4 key using the Windows NT hash, even though this would be more secure (but still much less secure than a random 128-bit key).

In either case, the overall security of the encryption is not 40 bits or 128 bits, but the number of bits of entropy in the password. Experimentally, English has about 1.3 bits of entropy per character [CK78]; case variations, numbers, and non-alphanumeric characters increase that value significantly. Any attack that tried a dictionary of weak passwords would be able to read most encrypted MPPE traffic. Additionally, the stylized headers in the PPP packet make it easy to collect known plaintext and test whether a particular key guess is correct.

The 40-bit RC4 suffers from even more serious weaknesses. Because there is no salt, an attacker can precompute a dictionary of ciphertext PPP headers, and then quickly look-up a given ciphertext in this dictionary. When looking for known plaintext locations inside the MPPE packets, an attacker can take advantage of the abundance of SMB and Netbios communications that occurs in standard Microsoft communications [Hob97, MB97].

Moreover, the same 40-bit RC4 key is generated every time the same user initializes the PPTP protocol. Since RC4 is an output-feedback mode cipher, it is trivial to break the encryption from the ciphertext from two sessions. This severe security weakness is mentioned in the most recent MPPE specification [PZ98], although it is missing from the previous version [Pal96b]. No version of the Microsoft documentation mentions that the same key is used in both the forward and backwards direction, guaranteeing that the same keystream is used to encrypt two different plaintexts.

The 128-bit RC4 uses a 64-bit nonce in the key generation process; this makes precomputed dictionary attacks impractical. Still, brute-force against the password is much more efficient than brute force against the keyspace. The nonce also means that two sessions using the same password will have two different 128-bit RC4 keys, although the same key will be used to encrypt the plaintext in both directions.

5.3 Bit Flipping Attacks

RC4 is an output-feedback mode stream cipher, and does not provide any authentication of the ciphertext stream. Since there is no other authentication mechanism provided for in MPPE, an attacker can undetectably flip bits in the ciphertext. If the underlying protocol is sensitive to particular bit toggles–to enable or diable features, turn on or off options, reset parameters–this attack can be very fruitful. Note that this attack does not require the attacker to know the encryption key or the client's password. Of course, higher-level protocols might detect or prevent these sorts of attacks.

5.4 Attacking Resynchronization

If a packet is dropped in transit or arrives with an unexpected coherency count in the MPPE header, a resynchronization of keys takes place. The end that recieved the inconsistent packet sends a message to the sender requesting resynchronization. Upon receipt of this requst, the sender end re-initializes the RC4 tables and sets the "flushed" bit in the MPPE header. When a system sees the flushed bit set in a packet, it re-initializes its RC4 tables and sets the coherency count to the match the one it just recieved.

This creates the problem whereby an attacker can either spoof resynchronization requests or forge MPPE packets with incorrect coherency counts. If this is done continuously just prior to the 256th packet exchange, where the session key would normally be updated, an attacker can succeed in forcing the communications channel to never re-key.

This can be used to recover encrypted plaintext. All an attacker needs to do is to force a resynchronization. A simple XOR of the original stream and the resynchronized stream results in an XOR of the two plaintexts.

6 Other Attacks Against MS-PPTP

Even though the attacks that break the MS-CHAP and MPPE protocols completely negate the usefulness and security of MS PPTP, there are several other interesting attacks worth mentioning.

6.1 Passive Monitoring

A tremendous amount of information can be gleaned by just watching PPTP sessions traverse across the net. This sort of information is invaluable for traffic ananlysis and should be protected. However, the server publicly announces information such as the maximum number of channels that it has available. This information can be used to asses the approximate size of the PPTP server, and to monitor its load. By querying repeatedly with `PPTP_START_SESSION_REQUEST` packets, an attacker can see when new connections are made and when existing connections are closed. In such a fashion the attacker can gain information about the system and its usage patterns without necessarily being directly in a promiscuous location.[8]

By setting up a standard sniffer and eavesdropping on public communications, the following information was recovered from Microsoft PPTP servers:

- Client Machine IP address.
- Server Machine IP address.
- Number of PPTP virtual tunnels the Server has available.
- Client Machine RAS version.

[8] We have built automatic programs that query Microsoft PPTP servers on the Internet every few minutes, and have built graphs showing usage over time.

- Client Machine Netbios name.
- Client Vendor Identification.
- Server Vendor Identification.
- Internal Virtual Tunnel IP address handed to the client.
- Internal DNS servers handed to the client.
- Client Username.
- Enough information to retrieve the users password hash.
- Enough information to retrieve the initialization value used inside of MPPE.
- Current value of the encrypted packet for the Client before RC4 is re-initialized.[9]
- Current value of the encrypted packet for the Server before RC4 is re-initialized.

In any scenario where communications are encrypted and the user assumes some level of confidentiality, the above information should not be so easily obtainable. There is no easy way for Microsoft PPTP to encrypt this information, since the leaks come from outside the channel that MPPE controls. In some cases, these packets are the configuration and setup for the stream cipher inside of MPPE, and must be transmitted before encryption can begin. The only solution is to encrypt the control channel, or severely reduce the information being sent over it.

6.2 Spoofing PPP Negotiations

The PPP negotiation packets occur before and after the encryption can be applied. Since the method for resynchronization of keys is done via PPP CCP packets, these communications can never be encrypted in the same sleeve. Conjoined with this is the fact that there is no real authentication of the packets. This configuration stage is thus entirely open to attack.

Spoofing the configuration packet containing the DNS server could be used to force all name resolution to happen through a compromised name server.

Similarly, spoofing the configuration packet containing the internal tunnel IP address could be used to circumvent stateful packet filtering firewalls by forcing the client to connect to external machines from inside the private network.

6.3 Control Channel and Server Denial of Service

In this paper, not a tremendous amount of attention has been directed towards the control channel portion of PPTP. Part of the reason is that it is not clear why this channel exists. Everything this out-of-band channel accomplishes could be done via PPP negotiations or inside of unused portions of the GRE header.[10]

The other major stumbling block was Microsoft's actual implementation of the Control Channel. We quickly found that it is trivial to make a Windows NT machine running a PPTP server crash with kernel panics of varying types, sometimes referred to as the dreaded Blue Screen of Death (BSOD). In fact, it became very difficult to test the control channel without crashing the PPTP server. So difficult in fact that most of the attacks we attempted, in order to exploit theorized control channel problems, crashed the server before the attacks could complete. The following is a small subsection of tests that crashed a Windows NT Server with Service Pack 3 installed:

- Cycling through `PPTP CLEAR_CALL_REQUEST` packets in an attempt to step through the 16-bit space for call ID's.
- Iterating through all valid and non-valid values that could be held in the Packet Type field inside the PptpPacketHeader.
- Sending invalid values in the PPTP Control Packet headers.

All of the above packets can be sent to the PPTP server from outside a firewall, without any authentication. This, of course, assumes that there is no firewall configuration that only allows PPTP to the PPTP server from particular IP addresses or networks. However, if the users have the ability to access the PPTP server from anywhere in the world, then an attacker can send these queries in from anywhere in the world too.

6.4 Potential Client Information Leaks

The Windows 95 client does not properly sanitize its buffers, and information leaks in the protocol messages. Although the PPTP documentation states that characters after the hostname and vendor string should be set to the value of `0x00` in the `PPTP_START_SESSION_REQUEST` packet, Windows 95 does not do this.

```
 80: 0000 6c6f 6361 6c00 0000 3e1e 02c1 0000
 96: 0000 85c4 03c1 acd9 3fc1 121e 02c1 2e00
112: 0000 2e00 0000 9c1b 02c1 0000 0000 0000
128: 0000 88ed 3ac1 2026 02c1 1049 05c1 0b00
144: 0000 3978 00c0 280e 3dc1 9c1b 02c1 041e
160: 02c1 0e00 0000 121e 02c1 2e00 0000 2e00
176: 0000 3dad 06c1 74ed 3ac1 1c53 05c1 9c1b
192: 02c1 041e 02c1 0e00 0000 121e 02c1 2e00
208: 0000
```

```
 80: ..local...>.....
 96: ........?.......
112: ................
128: ....:. &...I....
```

[9] This value can tell an attacker when the RC4 key is re-initialized. By modifying this packet, an attacker may be able to prevent RC4 from being re-initialized.

[10] Microsoft did not implement a full GRE implementation [HLFT94].

```
144: ..9x..(.=.......
160: ................
176: ..=...t.:..S....
192: ................
208: ..
```

The above trace displays the garbage characters that appear after the hostname and vendor string. The 82nd through 86th byte contains the hostname, which the Windows 95 client seems to always set to "local". The 113th byte is where the vendor string would be located. A trace of a similar packet from a Microsoft NT PPTP client shows all of these garbage bytes set to nulls.

There is a distinct possibility for information leaks depending upon how and where these structures are being allocated from and what was happening internally on the client system. Further analysis of the Windows 95 code is necessary to determine the full extent of these information leaks.

7 Conclusions

Microsoft's PPTP implementation is fragile from an implementation perspective, and seriously flawed from a protocol perspective. The authentication protocol has known flaws in it that have been pointed out not only here, but by groups such as the L0pht. The encryption is improperly deployed, and this implementation uses an output-feedback-mode stream cipher whereas a cipher-block-chaining-mode block cipher would have been more appropriate. To tie the weak authentication together with the poor encryption, Microsoft makes the encryption key a function of the user password instead of using a strong key-exchange algorithm like Diffie-Hellman or EKE. Finally, the control channel is neither authenticated nor strongly protected.

We did not spend serious time looking at the clients local IP forwarding mechanisms and how Microsoft attempted, or failed to take into account, the vulnerability that this, now dual-homed, client presents. This, of course, is a potential problem with any virtual private network scheme, not just PPTP. We did, however, document problems with non-standard subnet masks and internal tunnel IP traffic being sent from the local network interface card as opposed to from the PPTP server. Implementors beware!

Finally, we wish to stress that our cryptanalysis does not break the PPTP protocol [HPV+97], but only Microsoft's implementation of it. While Microsoft uses their own extensions (MS-CHAP, MPPE, MPPC) inside the PPP section of PPTP, the PPTP specification does not require this. Vendors may wish to include these Microsoft extensions for compatibility purposes, but are not restricted to their use and are encouraged to implement more secure security extensions. Of course, any new extensions would have to be understood and implemented on both the client and server for correct operation.

8 Acknowledgments

We would like to thank Mark Chen, Chris Hall, Brad Kemp, Paul Jones, Ben McCann, Mark Seiden, Inderpreet Singh, David Wagner, and Wray West for their helpful comments. David Wagner was especially helpful in navagating the minutae of the optimized attack against MS-CHAP.

References

[BV98] L. Blunk and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)," Network Working Group, RFC 2284, Mar 1998. ftp://ftp.isi.edu/in-notes/rfc2284.txt.

[CK78] T.M. Cover and R.C. King, "A Convergent Gambling Estimate of Entropy," *IEEE Transactions on Information Theory,* v. IT-24, n. 4, Jul 1978, pp. 413–421.

[HLFT94] S. Hanks, T. Li, D. Farinacci, and P. Traina, "Generic Routing Encapsulation (GRE)," Network Working Group, RFC 1701, Oct. 1994. ftp://ftp.isi.edu/in-notes/rfc1701.txt.

[Hob97] Hobbit, "CIFS: Common Insecurities Fail Scrutiny," Avian Research, Jan 1997. http://www.avian.org.

[HPV+97] K. Hamzeh, G.S. Pall, W. Verthein, J. Taarud, and W.A. Little, "Point-to-Point Tunneling Protocol," Internet Draft, IETF, Jul 1997. http://www.ietf.org/internet-drafts/draft-ietf-pppext-pptp-02.txt.

[KSWH98] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic Attacks on Pseudorandom Number Generators," *Fast Software Encryption: 5th International Workshop*, Springer-Verlag, 1998, pp. 168-188

[Kle90] D.V. Klein, "Foiling the Cracker: A Security of, and Implications to, Password Security," *Proceedings of the USENIX UNIX Security Workshop*, Aug 1990, pp. 5–14.

[Kli98] S. Klinger, "Microsoft PPTP and RRAS for Windows NT Server 4.0," *LAN Times*, Feb ??, 1998. http://www.lantimes.com/98/allowbreak 98feb/802b065b.html.

[L97a] L0pht Heavy Industries Inc, L0phtcrack, 1997. http://www.L0pht.com/L0phtcrack/.

[L97b] L0pht Heavy Industries Inc, "A L0phtCrack Technical Rant," Jul 1997. http://www.l0pht.com/l0phtcrack/rant.html.

[LS92] B. Lloyd and W. Simpson, "PPP Authentication Protocols," Network Working Group, RFC 1334, Oct 1992. ftp://ftp.isi.edu/in-notes/rfc1334.txt.

[Mey96] G. Meyer, "The PPP Encryption Control Protocol (ECP)," Network Working Group, RFC 1968, Jun 1996. ftp://ftp.isi.edu/in-notes/rfc1968.txt.

[Mic96a] Microsoft Corporation, *Advanced Windows NT Concepts*, New Riders Publishing, 1996. Relevent chapter at http://www.microsoft.com/communications/nrpptp.htm.

[Mic96b] Microsoft Corporation, "Pont-to-Point Tunneling Protocol (PPTP) Frequently Asked Questions," Jul 1996. http://premium.microsoft.com/sdn/library/bkgrnd/html/pptpfax.htm.

[Mic97] Microsoft Corporation, "Response to Security Issues Raised by the L0phtcrack Tool," Apr 1997, http://www.microsoft.com/security/l0phtcrack.htm.

[Mic98] Microsoft Corporation, "Clarification on the L0phtcrack 2.0 Tool." Mar 1998. http://www.microsoft.com.security/l0pht20.htm.

[MB97] P. Mudge and Y. Benjamin, "Deja Vu All Over Again," *Byte*, Nov 1997. http://www.byte.com/art/9711/sec6/art3.htm.

[NBS77] National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[NIST93] National Institute of Standards and Technology, "Secure Hash Standard," U.S. Department of Commerce, May 93.

[Pal96a] G.S. Pall, "Microsoft Point-to-Point Compression (MPPC) Protocol," Network Working Group Internet Draft, Jul 1996.

[Pal96b] G.S. Pall, "Microsoft Point-to-Point Encryption (MPPE) Protocol," Network Working Group Internet Draft, Jul 1996.

[PZ98] G.S. Pall and G. Zorn, "Microsoft Point-to-Point Encryption (MPPE) Protocol," Network Working Group, Internet Draft, IETF, Mar 1998. http://www.ietf.org/internet-drafts/draft-ietf-pppext-mppe-00.txt.

[Ran96] D. Rand, "The PPP Compression Control Protocol (CCP)," Network Working Group, RFC 1962, Jun 1996. ftp://ftp.isi.edu/in-notes/rfc1962.txt.

[RP94] J. Reynolds and J. Postel, "Assigned Numbers," Networking Group, Std 2, RFC 1700, Oct 1994. ftp://ftp.isi.edu/in-notes/rfc1700.txt.

[Riv91] R.L. Rivest, "The MD4 Message Digest Algorithm," *Advances in Cryptology — CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 303–311.

[Sch96] B. Schneier *Applied Cryptography, 2nd Edition*, John Wiley & Sons, 1996.

[Sim94] W. Simpson, "The Point-to-Point Protocol (PPP)," Network Working Group, STD 51, RFC 1661,, Jul 1994. ftp://ftp.isi.edu/in-notes/rfc1661.txt.

[Sim96] W. Simpson, "PPP Challenge Handshake Authentication Protocol (CHAP)" Network Working Group, RFC 1334, Aug 1996. ftp://ftp.isi.edu/in-notes/rfc1994.txt.

[ZC98] G. Zorn and S. Cobb, "Microsoft PPP CHAP Extensions," Network Working Group Internet Draft, Mar 1998. http://www.ietf.org/internet-drafts/draft-ietf-pppext-mschap-00.txt.

A Microsoft PPTP Details

TCP port 1723 on the server listens for PPTP Control Channel packets, which are sent before anything travels over the GRE tunnel. Through this mechanism, the server is alerted to new connections that wish to begin and existing connections that wish to be terminated.

A condensed version of the control channel communications looks like the following:

1. Start Session (client to server)
2. Start Session Reply (server to client)
3. Out Call Request (client to server)
4. Out Call Reply (server to client)
5. Set link info (server to client)

The first packet sent from the client to the server (note that we have excluded the standard TCP setup packets and are examining only the packets directly related to the PPTP Control channel protocol) is a `PPTP_START_SESSION_REQUEST` packet.

The server responds to the `PPTP_START_SESSION_REQUEST` packet with a `PPTP_START_SESSION_REPLY`. This packet contains information about the server, and a result status for the previous request packet. In this packet there are several items of interest. The NT server properly sanitizes the hostname and vendor string arrays. The result states whether the `START_SESSION_REQUEST` packet was accepted and if the client may proceed. The server also announces the maximum number of channels that it has available.

At this point the client makes a `PPTP_OUT_CALL_REQUEST` to the server. The phone number and sub address arrays are not filled in for connections directly over the Internet. Packet processing delay information is included as is the minimum and maximum connection speeds allowed.

After receiving the `PPTP_OUT_CALL_REQUEST`, the server responds with a `PPTP_OUT_CALL_REPLY` packet, conveying to the client the status of the processing of the request.

The final communication that occurs in the control channel, before GRE packets are sent, is to set link info. We do not know why this was included in the control channel, and not in the PPP Link Control Protocol section.

The control channel stays open for the duration of the client to server connection. Periodically, `PPTP_ECHO_REQUEST` packets and `PPTP_ECHO_REPLY` packets are exchanged to ensure that both sides are still active. When the client disconnects from the server the control channel sends back and forth one of the following packets: CALL_DISCONNECT_NOTIFY, CLEAR_CALL_REQUEST, or STOP_SESSION_REQUEST, depending upon the situation.

In a full-fledged connection the above communications take place, followed by further communications and negotiation at the PPP layer.

After the PPTP Control Channel Set Link Info packet, GRE (IP prototype 47) packets with PPP payloads are transmitted. The first part of the setup entails PPP setting up negotiations for how subsequent packets will be treated through standard PPP Link Control Protocol (LCP) and Network Control Protocol (NCP) negotiations.[11]

B MS-CHAP Details

A typical MS-CHAP protocol exchange looks like the following inside of a PPP LCP packet. This is the initial client message:

```
c0 21 01 00 00 13 03 05 c2 23 80
[extra negotiation removed]

0xc021   - LCP packet
0x01     - Configure Request
0x00     - ID 0
0x13     - length 19 bytes
0x03     - Authentication
0x05     - CHAP option length 5 bytes
0xc223   - CHAP
0x80     - MS-CHAP
```

As can be seen from the above, the LCP configuration is identical to standard CHAP except for the change in algorithm type to 0x80 to represent MS-CHAP.

The actual challenge comes across in a reply packet from the server:

```
c2 23 01 00 00 0d 08 cf 4f 0e 72 89 04 3b

0xc223   - CHAP packet
0x01     - challenge
0x00     - ID 0
0x000d   - length 13 bytes
0x08     - value size of the challenge
0xcf4f0e7289043b0c    - challenge value
```

The Client response follows:

```
c2 23 02 00 00 53 31 41 77 45 5b d1 d8
60 68 fd d3 8e 4d 68 aa 24 6f 0c d6 95
34 7b 8c 9a 31 19 6c 45 57 78 77 a0 d0
4a 47 7a 36 a1 8a 57 8e 76 c6 36 78 a1
14 79 0f 01 41 64 6d 69 6e 69 73 74 72
61 74 6f 72

0xc223  - CHAP
0x02    - Response
0x00    - ID
0x0053  - length 53 bytes [The ascii
          string has been changed to protect
          the innocent]
0x31    - Value length of challenge response
0x4177455bd1d86068fdd38e4
d68aa246f0cd695347b8c9a31 - LANMAN response
0x196c45577877a0d04a477a3
6a18a578e76c63678a114790f - NT response
0x01    - use Windows NT compatible challenge
          response flag
"Administrator"   - account name
```

The Microsoft PPP CHAP Extensions document describes the "use Windows NT compatible challenge response flag" as telling the end system to use the NT response in preference to the LANMAN response. The LANMAN response can be set to all 0's in this case.

C MPPE Details

Microsoft Point-to-Point Encryption protocol is negotiated inside the PPP Compression Control Protocol as type 18 (0x12).

If the least significant octet in the option field is 0x20, then a 40 bit session key is being requested. Similarly, 0x40 requests a 128 bit session key.

A sample exchange would appear as follows. The Client sends the Server:

```
80 fd 01 05 00 0a 12 06 00 00 00 20

0x80fd     - Compression Control Protocol
0x01       - Configure Request
0x05       - ID 5
0x000a     - Length 10
0x12       - type 18 MPPE
0x06       - Length 6
0x00000020 - 40 bit session key
```

The Server sends the Client:

```
80 fd 02 05 00 0a 12 06 00 00 00 20

0x80fd     - Compression Control Protocol
0x02       - Configure Acknowledgement
0x05       - ID 5
0x000a     - Length 10
0x12       - Type 18 MPPE
0x06       - Length 6
0x00000020 - 40 bit session key
```

At this point, the client and server have succesfully agreed upon MPPE 40 bit encryption. If the client

[11] In our test environment, standard PPP LCP and NCP packets are exchanged with the following negotiations being agreed upon: Protocol Field Compression is on, Address and Control Field Compression is on, Call Back numbers are negotiated via PPP Call Back Control Protocol Packets and subsequently set to empty values as we are connected over the internet, MS-CHAP is used as the CHAP authentication, Microsoft Point to Point Compression is not used, the tunnel IP address for the client, the DNS server for the client, IP header compression, and Microsoft Point to Point encryption using a 40-bit session key.

or server wished to refuse the CCP Configure request packet, a CCP Configure Reject packet would have been sent instead of a Configure Acknowledgement.