An Analysis of the RADIUS Authentication Protocol

by Joshua Hill, InfoGard Laboratories

A current version of this article is maintained at:
http://www.untruth.org/~josh/security/radius/
Please send comments to josh-radius@untruth.org

Last Modified Sat Nov 24 23:58:41 PST 2001
See the Document Change History section for a history of modification

1 Introduction

RADIUS is a widely used protocol in network environments. It is commonly used for embedded network devices such as routers, modem servers, switches, etc. It is used for several reasons:

- The embedded systems generally cannot deal with a large number of users with distinct authentication information. This requires more storage than many embedded systems possess.
- RADIUS facilitates centralized user administration, which is important for several of these applications. Many ISPs have tens of thousands, hundreds of thousands, or even millions of users. Users are added and deleted continuously throughout the day, and user authentication information changes constantly. Centralized administration of users in this setting is an operational requirement.
- RADIUS consistently provides some level of protection against a sniffing, active attacker. Other remote authentication protocols provide either intermittent protection, inadequate protection or non-existent protection. RADIUS's primary competition for remote authentication is TACACS+ and LDAP. LDAP natively provides no protection against sniffing or active attackers. TACACS+ is subtly flawed, as discussed by Solar Designer in his advisory.
- RADIUS support is nearly omni-present. Other remote authentication protocols do not have consistent support from hardware vendors, whereas RADIUS is uniformly supported. Because the platforms on which RADIUS is implemented on are often embedded systems, there are limited opportunities to support additional protocols. Any changes to the RADIUS protocol would have to be at least minimally compatible with pre-existing (unmodified) RADIUS clients and servers.

RADIUS is currently the de-facto standard for remote authentication. It is very prevalent in both new and legacy systems.

1.1 Applicability

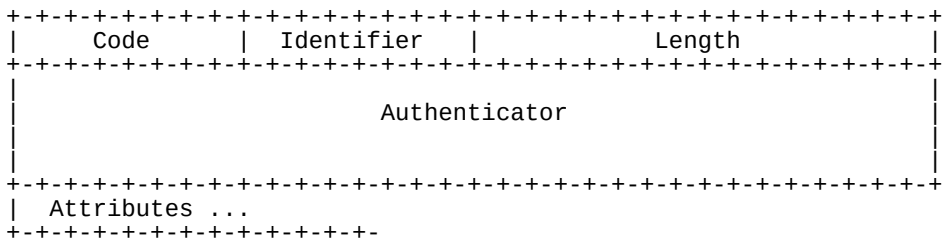
This analysis deals with some of the characteristics of the base RADIUS protocol and of the User-Password attribute. Depending on the mode of authentication used, the described User-Password weaknesses may or may not compromise the security of the underlying authentication scheme. A complete compromise of the User-Password attribute would result in the complete compromise of the normal Username/Password or PAP authentication schemes, because both of these systems include otherwise unprotected authentication information in the User-Password attribute. On the other hand when a Challenge/Response system is in use, a complete compromise of the User-Password attribute would only expose the underlying Challenge/Response information to additional attack, which may or may not lead to a complete compromise of the authentication system, depending on the strength of the underlying authentication system.

This analysis does not cover the RADIUS protocol's accounting functionality (which is, incidentally, also flawed, but normally doesn't transport information that must be kept confidential).

2 Protocol Summary

A summary of the RADIUS packet is below (from the RFC):

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```



The code establishes the type of RADIUS packet. The codes are:

Value	Description
1	Access-Request
2	Access-Accept
3	Access-Reject
4	Accounting-Request
5	Accounting-Response
11	Access-Challenge
12	Status-Server (experimental)
13	Status-Client (experimental)
255	Reserved

The identifier is a one octet value that allows the RADIUS client to match a RADIUS response with the correct outstanding request.

The attributes section is where an arbitrary number of attribute fields are stored. The only pertinent attributes for this discussion are the User-Name and User-Password attributes.

This description will concentrate on the most common type of RADIUS exchange: An Access-Request involving a username and user password, followed by either an Access-Accept, Access-Reject or a failure. I will refer to the two participants in this protocol as the client and the server. The client is the entity that has authentication information that it wishes to validate. The server is the entity that has access to a database of authentication information that it can use to validate the client's authentication request.

2.1 Initial Client Processing

The client creates an Access-Request RADIUS packet, including at least the User-Name and User-Password attributes.

The Access-Request packet's identifier field is generated by the client. The generation process for the identifier field is not specified by the RADIUS protocol specification, but it is usually implemented as a simple counter that is incremented for each request.

The Access-Request packet contains a 16 octet Request Authenticator in the authenticator field. This Request authenticator is a randomly chosen 16 octet string.

This packet is completely unprotected, except for the User-Password attribute, which is protected as follows:

The client and server share a secret. That shared secret followed by the Request Authenticator is put through an MD5 hash to create a 16 octet value which is XORed with the password entered by the user. If the user password is greater than 16 octets, additional MD5 calculations are performed, using the previous ciphertext instead of the Request Authenticator.

More formally:
Call the shared secret S and the pseudo-random 128-bit Request Authenticator RA. The password is broken into 16-octet blocks p1, p2, ... pn, with the last block padded at the end with '0's to a 16-octet boundary. The ciphertext blocks are c1, c2... cn.

c1 = p1 XOR MD5(S + RA)
c2 = p2 XOR MD5(S + c1)
.
.
.
cn = pn XOR MD5(S + cn-1)

The User-Password attribute contains **c1+c2+...+cn**, Where + denotes concatenation.

2.2 Server Processing

The server receives the RADIUS Access-Request packet and verifies that the server possesses a shared secret for the client. If the server does not possess a shared secret for the client, the request is silently dropped.

Because the server also possesses the shared secret, it can go through a slightly modified version of the client's protection process on the User-Password attribute and obtain the unprotected password. It then uses its authentication database to validate the username and password. If the password is valid, the server creates an Access-Accept packet to send back to the client. If the password is invalid, the server creates an Access-Reject packet to send back to the client.

Both the Access-Accept packet and the Access-Reject packet use the same identifier value from the client's Access-Request packet, and put a Response Authenticator in the Authenticator field. The Response Authenticator is the is the MD5 hash of the response packet with the associated request packet's Request Authenticator in the Authenticator field, concatenated with the shared secret.

That is, ResponseAuth = MD5(Code+ID+Length+RequestAuth+Attributes+Secret) where + denotes concatenation.

2.3 Client Post Processing

When the client receives a response packet, it attempts to match it with an outstanding request using the identifier field. If the client does not have an outstanding request using the same identifier, the response is silently discarded. The client then verifies the Response Authenticator by performing the same Response Authenticator calculation the server performed, and then comparing the result with the Authenticator field. If the Response Authenticator does not match, the packet is silently discarded.

If the client received a verified Access-Accept packet, the username and password are considered to be correct, and the user is authenticated. If the client received a verified Access-Reject message, the username and password are considered to be incorrect, and the user is not authenticated.

3 RADIUS Issues

The RADIUS protocol has a set of vulnerabilities that are either caused by the protocol or caused by poor client implementation and exacerbated by the protocol. The vulnerabilities that follow arose during a somewhat shallow exploration of the protocol; this is not expected to be a complete list of vulnerabilities of the RADIUS protocol, these are merely the vulnerabilities that presented themselves to the reviewer.

3.1 Response Authenticator Based Shared Secret Attack

The Response Authenticator is essentially an ad hoc MD5 based keyed hash. This primitive facilitates an attack on the shared secret. If an attacker observes a valid Access-Request packet and the associated Access-Accept or Access-Reject packet, they can launch an off-line exhaustive attack on the shared secret. The attacker can pre-compute the MD5 state for (Code+ID+Length+RequestAuth+Attributes) and then resume the hash once for each shared secret guess. The ability to pre-compute the leading sections of this keyed hash primitive reduces the computational requirements for a successful attack.

3.2 User-Password Attribute Cipher Design Comments

The User-Password protection scheme is a stream-cipher, where an MD5 hash is used as an ad hoc

pseudorandom number generator (PRNG). The first 16 octets of the stream cipher display the same properties as a synchronous stream cipher. After the first 16 octets, the stream cipher state integrates the previous ciphertext, and becomes more accurately described as a self-synchronizing stream cipher.

The security of the cipher rests on the strength of MD5 for this type of use and the selection of the shared secret. It is unclear what the requirements for this cipher are, so it is unclear if the MD5 function is appropriate for this use. MD5 is not designed to be a stream cipher primitive, it is designed to be a cryptographic hash. This sort of misuse of cryptographic primitives often leads to subtly flawed systems.

3.3 User-Password Attribute Based Shared Secret Attack

Because of the selection of a stream cipher for protection of the User-Password attribute, an attacker can gain information about the Shared Secret if they can observe network traffic and attempt an authentication. The attacker attempts to authenticate to the client with a known password. The attacker then captures the resulting Access-Request packet and XORs the protected portion of the User-Password attribute with the password they provided to the client. This results in the value of the MD5(Shared Secret + Request Authenticator) operation. The Request Authenticator is known (it is in the client's Access-Request packet), so the attacker can launch an off-line exhaustive attack on the shared secret. Note, though, that the attacker cannot pre-compute the MD5 state of the hash for the Request Authenticator, because the Request Authenticator is hashed second.

3.4 User-Password Based Password Attack

The use of a stream cipher to protect the User-Password attribute results in a vulnerability that allows an attacker to circumvent any authentication rate limits imposed by the client. The attacker first attempts to authenticate to the client using a valid username and a known (and likely incorrect) user password. The attacker then captures the resulting Access-Request packet and determines the result of the MD5(Shared Secret + Request Authenticator) operation (in the same way as in the previous attack). The attacker can then replay modified Access-Request packets, using the same Request Authenticator and MD5(Shared Secret + Request Authenticator) value, changing the password (and the associated User-Password attribute) for each replay. If the server does not impose user based rate limits, this will allow the attacker to efficiently perform an exhaustive search for the correct user password.

Note that the attacker can only use this method to attack passwords that are 16 characters or less, as the User-Password protection mechanism uses a chaining method that includes previous ciphertext in the state after the first 16 octets of output.

Any sort of strong data authentication in the Access-Request packet would make this attack impossible.

3.5 Request Authenticator Based Attacks

The security of RADIUS depends on the generation of the Request Authenticator field. The Request Authenticator must be both unique and non-predictable in order for the RADIUS implementation to be secure. The RADIUS protocol specification does not emphasize the importance of the Request Authenticator generation, so there are a large number of implementations that use poor PRNGs to generate the Request Authenticator. If the client uses a PRNG that repeats values (or has a short cycle), the protocol ceases to provide the intended level of protection.

The last two of these attacks require the attacker to cause the client to produce a particular identifier value. This is generally not particularly difficult, as identifiers were never meant as a security feature. The actual method of identifier generation is not specified by the protocol specification, but the most common method of generating the identifier is to increment a one octet counter for each request, and include the counter value as the identifier. Because the identifier generation is normally deterministic, it often doesn't increase the work factor very much at all. An attacker can insert a series of extra requests to the client, forcing the desired identifier to reoccur much more rapidly than it would normally. Even if the identifier were not generated in a readily attackable way, it would still only increase the work factor by 256 times.

3.5.1 Passive User-Password Compromise Through Repeated Request Authenticators

If the attacker can sniff the traffic between the RADIUS client and the RADIUS server, they can passively produce a dictionary of Request Authenticators, and the associated (protected) User-Password attributes.

If the attacker observes a repeated Request Authenticator, they can remove any influence of the Shared Secret from the first 16 octets of the passwords by XORing the first 16 octets of the protected passwords together. This yields the first 16 octets of the two (now unprotected) user passwords XORed together.

The impact of this attack varies according to how good the user passwords are. If the users all chose random passwords of the same length, the attacker can gain nothing because no information about either password can be extracted. Unfortunately, this is a somewhat unlikely occurrence. In reality, users choose passwords of varying lengths (generally less than 16 characters) and of varying quality.

The easiest problem for the attacker to exploit is the case where the two passwords are of different lengths. Ideally for the attacker, the passwords are both less than 16 characters long and are significantly different lengths. In this situation, one of the passwords has more padding than the other, so the non-overlapping characters of the longer password are XORed with '0' (the characters do not change). This results in the non-overlapping characters of the longer password being exposed to the attacker with no analysis.

More complex attacks are available if the attacker makes the assumption that the users chose low-entropy passwords. In this situation, the attacker can perform an intelligent dictionary attack guided by statistical analysis of the overlapping region. This dictionary attack can be further refined by noting the length of the two passwords and the trailing portion of the longer password, and then only trying passwords with this length and ending.

Even passwords longer than 16 characters are at risk from this attack, because the attacker still gains information about the first 16 characters of the password. This provides a firm basis for later attack, if nothing else.

3.5.2 Active User-Password Compromise through Repeated Request Authenticators

The attacker can attempt to authenticate many times using known passwords and intercept the generated Access-Request packets, extracting the Request Authenticator and User-Password attributes. The Attacker can then XOR the known password with the User-Password attribute and be left with the MD5(Shared Secret + Request Authenticator) value. The attacker generates a dictionary of Request Authenticator values and associated MD5(Shared Secret + Request Authenticator) values.

When the attacker sees a valid Access-Request packet that has a Request Authenticator value that is in the attacker's dictionary, the attacker can recover the first 16 octets from the protected region of the User-Password field by looking up the associated MD5(Shared Secret + Request Authenticator) value from the dictionary and XORing it with the intercepted protected portion of the User-Password attribute.

3.5.3 Replay of Server Responses through Repeated Request Authenticators

The attacker can build a dictionary of Request Authenticators, identifiers and associated server responses. When the attacker then sees a request that uses a Request Authenticator (and associated identifier) that is in the dictionary, the attacker can masquerade as the server and replay the previously observed server response.

Further, if the attacker can attempt to authenticate, causing the client to produce an Access-Request packet with the same Request Authenticator and identifier as a previously observed successful authentication, the attacker can replay the valid looking Access-Accept server response and successfully authenticate to the client without knowing a valid password.

3.5.4 DOS Arising from the Prediction of the Request Authenticator

If the attacker can predict future values of the Request Authenticator, the attacker can pose as the client and create a dictionary of future Request Authenticator values (with either the expected identifier, or with every possible identifier) and associated (presumably Access-Reject) server responses. The attacker can then masquerade as the server and respond to the client's (possibly valid) requests with valid looking Access-Reject packets, creating a denial of service.

3.6 Shared Secret Hygiene

The RADIUS standard specifically permits use of the same Shared Secret by many clients. This is a very bad idea, as it provides attackers with more data to work from and allows any flawed client to compromise several machines. All RADIUS clients that possesses the same shared secret can be viewed as a single RADIUS client for the purpose of all these attacks, because no RADIUS protection is applied to the client or server address.

Most client and server implementations only allow shared secrets to be input as ASCII strings. There are only 94 different ASCII characters that can be entered from a standard US style keyboard (out of the 256 possible). Many implementations also restrict the total length of the shared secret to 16 characters or less. Both of these restrictions artificially reduce the size of the keyspace that an attacker must search in order to guess the shared secret.

4 Conclusions

4.1 Summary Findings

The RADIUS protocol has several interesting issues that arise from its design. The design and policy characteristics that seem to be principally responsible for the security problems are as follows:

- The User-Password protection technique is flawed in many ways. It should not use a stream cipher, and it should not use MD5 as a cipher primitive. *(note 3.2; attacks 3.3, 3.4, 3.5.1, 3.5.2)*
- The Response Authenticator is a good idea, but it is poorly implemented. *(attack 3.1)*
- The Access-Request packet is not authenticated at all. *(attack 3.4)*
- Many client implementations do not create Request Authenticators that are sufficiently random. *(all attacks in 3.5)*
- Many administrators choose RADIUS shared secrets with insufficient information entropy. Many client and host implementations artificially limit the shared secret key space. *(note 3.6)*

4.2 Suggested Protocol Additions

Selection of a well understood symmetric block cipher to protect the user password would be good practice. A new User-Password like attribute that uses an alternate encryption scheme should be created. I suggest TDES (as specified in ANSI X9.52) used in CBC mode. If this new attribute is used, the User-Password attribute should not be.

Ideally the block cipher would be keyed independently from the shared secret, but this may prove unworkable for compatibility reasons. Another option would be to key the cipher from some derived value of the shared secret and the request authenticator. For instance the cipher could be keyed from the output of an HMAC of the Request Authenticator (where the HMAC is keyed by the shared secret) or by seeding a cryptographic PRNG with the shared secret and the request authenticator.

Instead of using an ad hoc keyed hash primitive in the Response Authenticator, an accepted Message Authentication Code (MAC) should be used. An HMAC would be an ideal choice for this primitive. In addition, the Access-Request packet would benefit from authentication.

Though MD5 is a cryptographic hash that could be used in the HMAC primitive, it has several significant attacks against it. The RADIUS protocol would benefit from using SHA-1 instead of MD5 for HMACs.

In order to protect the Access-Request, Access-Accept and Access-Deny packets, a new attribute should be created that contains a SHA-1-HMAC of the entire RADIUS packet (with the SHA-1-HMAC attribute data set to 0). If this attribute is present, the receiving client or server should compute the HMAC for the entire RADIUS packet (with the HMAC set to zeros) and verify that the result is the same as the stored HMAC. If the result is not the same, the packet should be discarded.

When the server generates a RADIUS Access-Accept or Access-Reject packet with a SHA-1-HMAC, it should set the Response Authenticator to the associated Request Authenticator. If a client receives a RADIUS Access-Accept or Access-Reject packet that has the SHA-1-HMAC attribute, it should not test for the validity of the Response Authenticator.

When a client generates a RADIUS Access-Request packet, it should include the SHA-1-HMAC attribute. When the server receives a RADIUS Access-Request packet, it should verify the SHA-1-HMAC attribute.

There is just such an attribute defined as a RADIUS Extension in RFC 2869, called the Message-Authenticator. This attribute contains the output from an MD5 based HMAC, keyed with the shared secret, of the entire RADIUS packet. This attribute adequately protects RADIUS packets that include this attribute. Unfortunately, this attribute is not required to be consistently used (in fact, it is only required to be used when the new EAP-Message attribute is used). RFC 2869 does suggest that this attributes be used in cases where the User-Password attribute is not included in the RADIUS Access-Request packet; unfortunately, the vulnerability seen in section 3.4 requires that the User-Password attribute is in use. Further, RFC 2869 does not suggest that the server and client should have a mode where packets received without the Message-Authenticator are discarded. Without this mode, the attacker can simply strip off the Message-Authenticator attribute from a RADIUS client Access-Request packet, modify the packet and then replay the resulting packet. (It should be noted that the attacker cannot strip off this attribute from a server Access-Accept or Access Reject packet, as that message is separately authenticated by the Response Authenticator).

The Message-Authenticator attribute could provide an effective defense if it were required to be more consistently used. Clients and servers should be able to be placed in a mode where RADIUS packets without the Message-Authenticator attribute are silently discarded.

4.3 Suggested Client Behavior Modifications

Authenticator Behavior

The RADIUS specification should require a strong cryptographic PRNG for generation of the Request-Authenticator, such as the PRNG specified in ANSI X9.17 appendix C or FIPS 186-2, appendix 3.

Shared Secret Behavior

The RADIUS specification should require each RADIUS client use a different Shared Secret. It should also require the shared secret to be a random bit string at least 16 octets long that was generated by a strong cryptographic PRNG.

In order to facilitate entry of this bit string, clients and servers should allow for input of arbitrary binary data. Quite likely, the easiest solution is to allow for the entry of hexadecimal digits.

4.4 General Comments

Both servers and clients should support the base RADIUS protocol and this extended RADIUS protocol. Both the server and the client should allow the administrator to enable the use of these RADIUS extensions on a client-by-client basis. This should be an explicit configuration option, not just an automatic determination made by the server. An automatic determination made by the server could lead to an attack where the attacker attempts to force the client/server interactions into the old RADIUS mode.

If it is not possible to change the RADIUS protocol, the system can still be made much more secure by just following the suggestions in section 4.3, which can all be implemented while still remaining completely compliant with the existing RADIUS protocol.

4.5 Why Modify RADIUS?

So, why attempt to modify RADIUS at all? Why not just go to another (presumably more modern, more secure) protocol? Well, for the most part, the answer is "Because such a protocol doesn’t currently exist." In the near future, however, Diameter is likely to be released by the IETF.

Diameter is the planned RADIUS replacement. The great majority of all the protocol work that has gone into Diameter has been directed to removing some of the functional limitations imposed by the RADIUS protocol. Effectively no work has been done as relates to the client/server security of the protocol. (CMS is defined, but this is a security layer for the proxy to proxy interaction, not the client to proxy/server interaction)

So, does this mean that they continue to use even RADIUS’s ad hoc system? No, they removed all security functionality from the protocol. They did the protocol designer’s equivalent of punting. Section 2.2 of the current Diameter protocol spec says:

"Diameter clients, such as Network Access Servers (NASes) and Foreign Agents MUST support IP Security, and MAY support TLS. Diameter servers MUST support TLS, but the administrator MAY opt to configure IPSec instead of using TLS. Operating the Diameter protocol without any security mechanism is not recommended."

So, all security aspects of the protocol are handled by IPSec and/or TLS. From a security aspect, this strikes me as a very good idea. Both IPSec and TLS are fully featured (sometimes too fully featured) protocols that many people have reviewed. (That's already much better than RADIUS ever did).

Examining this from a slightly different angle gives me some cause for concern, however. It strikes me that the overhead imposed by a full TLS/IPSec implementation is very significant for many current-day embedded devices. This would seem to indicate that (at least in the near future) manufactures are going to either continue to use RADIUS or ignore the Diameter standard and perform Diameter without TLS or IPSec.

Because of this, I suspect that it would be advantageous to push for at least minimal RADIUS protocol revision.

5 Document Change History

- *(2001-11-13)* Modified section 1.1 to remove references to CHAP, as CHAP is not sent using the User-Password attribute. (Thanks to Barney Wolff for pointing this out)
- *(2001-11-13)* Changed the bibliography reference from RFC 2138 to RFC 2865. The update in RFCs does not change any of the analysis in this document. Also added a reference to RFC 2869.
- *(2001-11-13)* Modified section 4.2 to include a small discussion about RFC 2869's Message-Authenticator attribute and associated guidance included in the RFC.
- *(2001-11-13)* Modified the document so that "DIAMETER" is more properly referred to as "Diameter". (Thanks to Barney Wolff for pointing this out)
- *(2001-11-14)* Reformatted some bibliography references and added some contact information at the beginning of the document.
- *(2001-11-24)* Modified the reference in section 4.4 that read "section 5.3" to correctly read "section 4.3".

6 Previous Work

There has been some independent previous work with the RADIUS protocol:

Attacks **3.5.3** and **3.5.4** are likely the attacks referred to in the RADIUS RFC.

The known password attack on the shared secret using the Access-Request packet (attack **3.3**) appears to have been first observed in September, 1996 by Thomas H. Ptacek.
Paper #1

The known password attack on the shared secret using the Access-Request packet (attack **3.3**), and the shared secret attack on the Access-Reject and Access-Accept packets (attack **3.1**) were independently observed in July, 1997 by Reilly (*rich.friedeman@ANIXTER.COM*)
Shared Secret Recovery in RADIUS

7 Acknowledgements

Thanks go to:

- Mark Smith (mark@halibut.com), who provided very useful comments regarding passwords greater than 16 bytes long.
- The Halibutians, for uncovering various grammatical and phrasing issues.
- Barney Wolff, who pointed me toward the updated RADIUS RFCs, allowing me to discuss the implications of the Message-Authenticator. Thanks also for setting me straight on a few DIA... errr... Diameter issues. :-)
- Andrew Pimlott, for pointing out some section numbering inaccuracies.

8 Bibliography

RFC 2865, "Remote Authentication Dial In User Service (RADIUS)", by C. Rigney, S. Willens, A. Rubens, W. Simpson. June 2000.

RFC 2869, "RADIUS Extensions", by C. Rigney, W. Willats, P. Calhoun. June 2000.

The DIAMETER Base Protocol, by Pat R. Calhoun, Haseeb Akhtar, Jari Arkko, Erik Guttman, Allan C. Rubens, Glen Zorn. July 2001.

DIAMETER CMS Security Application, by Pat R. Calhoun, Stephen Farrell, William Bulley. July 2001.

FIPS 186-2.

The Handbook of Applied Cryptography, by Alfred J Menezes, Paul C. van Ooschot, Scott A. Vanstone. Chapter 5, chapter 6 and chapter 9. Most notably:
The MD5 based stream cipher as a synchronous stream cipher (6.1.1, ii)
The use of cryptographic functions in pseudorandom number generation is discussed in section 9.2.6.
The use of a MDC in the creation of a MAC is discussed in 9.5.2.

An Analysis of the TACACS+ Protocol and its Implementations by Solar Designer. July 2000.