MAC

Analysis and improvement of message authentication codes in OpenSSH

Abstract

In this paper, we will analyze the current MAC algorithms used in OpenSSH in light of recent attacks to HMAC functions. We will evaluate the suitability of UMAC as an alternative to the existing algorithms, in terms of both security and efficiency.

Introduction

A message authentication code (MAC) is a keyed cryptographic hash function designed to ensure data integrity (message has not been altered) and source authentication (message originated from the purported sender). Suppose Alice and Bob share a secret key. When Bob sends a message to Alice, he computes the MAC of the message using the key, and sends the message and MAC to her using the communication channel. Alice then uses the shared key to compute the MAC on the message received from Bob and compares the computed MAC with the MAC received. If these match, then Alice is assured that the message was not altered and that it came from Bob, because only someone possessing the shared key can compute a valid MAC for the message. MAC algorithms are widely used in Internet protocols (SSH, SSL/TLS, IPsec) for providing data integrity and source authentication.

OpenSSH is a free implementation of the SSH protocol [1] which allows the establishment of a secure communication channel between two users. The first version of the protocol did not use MAC algorithms [2]. The SSH-2 design [1] uses public key cryptography to authenticate the remote user and provides a secure and reliable way to exchange data using encryption and message authentication codes.

The goals of this project will be to assess the current MAC implementations in OpenSSH. Each of these algorithms are based on HMAC [1] and we will show the HMAC construction briefly [5]. Recent results in cryptographic research have presented theoretical limitations of HMAC [7, 8, 4]. We will evaluate the suitability of UMAC [12] as an alternative to these existing algorithms, in terms of both security and efficiency, and implement it in OpenSSH.

Previous Work

The current MAC algorithms implemented in OpenSSH [1] are based on HMAC, whose functionality and security are based on a cryptographically secure hash function [5]. OpenSSH uses HMAC based on the MD5, SHA-1 and RIPEMD-160 hash functions. There have been recent security results concerning these algorithms [7, 8, 4] that help better understand their limitations. For example, distinguishing and forgery attacks on HMAC are presented that can be executed when HMAC employs hash functions that have slow difference propagations [8]. These attacks have been applied to HMAC based on MD4, MD5, SHA-0 and a reduced version of SHA-1 [7]. While the attacks presented in this work do not diminish the security of HMAC as it is typically implemented, it demonstrates a potential insecurity if the hash function the HMAC algorithm depends on is weak. There are other known attacks, including birthday paradox and collision attacks discussed in [11].

UMAC is a relatively new message authentication code described in [12, 6, 10]. It is developed with the goal of being very fast - on a Pentium II PC used to perform the tests,

[1]The SSH-2 protocol has been in wide use since 1996, and has been proposed as an Internet standard in 2006.

it requires 0.98 cycles/byte, compared to a 12.6 cycles/byte for HMAC-SHA1 [6]. It is also provably secure - it is proven that forging a UMAC authenticated message requires breaking the cryptographic primitive used in its construct [6]. The "U" in UMAC stands for *universal hashing*, since the algorithm uses a special family of hash functions called *universal*. A hash function is ϵ-universal if for any pair of distinctive messages, the probability that they give a hash collision is at most ϵ (the probability is over the random choice of hash function). A UMAC is generated by hashing the message with a secretly chosen universal hash function, and then encrypting the resulting hash. It is shown that when the hash function is strongly universal and the encryption is done by a one-time pad, a third party cannot forge the message with a probability better than that by randomly chosing a string for the MAC [6]. In real world usage any symmetric key cryptosystem can be used instead of the one-time pad for a high level of security. For example, it has been shown that 96-bit UMAC tags cannot be forged with probability greater than $1/2^{90}$, assuming that the probability of a successful attack against the encryption function used is low [12].

The description of UMAC in [10] is accompanied with an implementation in C [9]. This implementation uses AES at the encryption step. The VIA C7 CPU which includes the VIA PadLock Security Engine can do AES encryption and decryption operations in hardware at a sustained rate of 12.8 gigabits per second [3]. This could lead to a significant performance improvement for UMAC operations.

Solution

We will first assess the MAC algorithms currently used in OpenSSH. The recent results on the security of HMAC will be studied, and their impact on OpenSSH will be determined.

We will then assess the suitability of UMAC as an alternative. We will research the current state-of-the-art of the security of UMAC and the methods for implementing it in comparison to HMAC. In order to compare the performance of UMAC with HMAC, we will implement UMAC in OpenSSH. We will benchmark the performance of the UMAC implementation and compare it with the existing HMAC implementations, and time permitting, include benchmarking results on the VIA C7 with hardware AES.

Project Timeline

February 1 - 14: Research the construct of HMAC, the known attacks and assess the implications on the algorithms used in OpenSSH.
February 15 - 28: Research UMAC and its security results.
March 1 - 20: Implement UMAC in OpenSSH.
March 20 - 31: Experiment with VIA C7 (with AES operations in hardware), collect results and compare to HMAC performance.
April: Finish paper.

Conclusion

After exploring the HMAC construct and the known attacks, we will justify why UMAC is an alternative to consider. In addition to having strong security properties, it is designed for very high performance, as will be shown by implementing it in the OpenSSH codebase.

References

[1] OpenSSH SSH client. ⟨`http://www.OpenSSH.com/`⟩.

[2] SSH Frequently Asked Questions. ⟨`http://www.snailbook.com/faq/ssh-1-vs-2.auto.html`⟩.

[3] VIA PadLock Security Engine. `http://www.via.com.tw/en/initiatives/padlock/hardware.jsp`.

[4] Mihir Bellare. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. Cryptology ePrint Archive, Report 2006/043, 2006.

[5] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message authentication using hash functions: the HMAC construction. *CryptoBytes*, 2(1):12–15, Spring 1996.

[6] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. *CRYPTO '99: Proceedings of Crypto, 19th annual international cryptology conference & Lecture Notes in Computer Science*, 1666:216–233, 1999.

[7] Scott Contini and Yiqun Lisa Yin. Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2006.

[8] Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (extended abstract). In Roberto De Prisco and Moti Yung, editors, *SCN 2006*, volume 4116 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2006.

[9] Ted Krovetz. UMAC C Source Code. Web site. ⟨`http://fastcrypto.org/umac/2004/code.html`⟩.

[10] Theodore D. Krovetz. *Software-optimized universal hashing and message authentication.* PhD thesis, University of California, Davis, 2000.

[11] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, New York, 1996.

[12] Ted Krovetz. UMAC: Message Authentication Code using Universal Hashing, March 2006. RFC 4418, ⟨`http://fastcrypto.org/umac/rfc4418.txt`⟩.