vtTLS: A Vulnerability-Tolerant Communication Protocol

André Joaquim Miguel L. Pardal Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Lisbon, Portugal
{andre.joaquim, miguel.pardal, miguel.p.correia}@tecnico.ulisboa.pt

***Abstract*—We present VTTLS, a vulnerability-tolerant communication protocol. There are often concerns about the strength of some of the encryption mechanisms used in SSL/TLS channels, with some regarded as insecure at some point in time. VTTLS is our solution to mitigate the problem of secure communication channels being vulnerable to attacks due to unexpected vulnerabilities in encryption mechanisms. It is based on diversity and redundancy of cryptographic mechanisms and certificates to provide a secure communication channel even when one or more mechanisms are vulnerable. VTTLS relies on a combination of k cipher suites. Even if $k-1$ cipher suites are insecure or vulnerable, VTTLS relies on the remaining cipher suites to maintain the channel secure. We evaluated the performance of VTTLS by comparing it to an OpenSSL channel.**

***Keywords*—network protocol, secure communication channels, diversity, redundancy, vulnerability-tolerance.**

I. Introduction

Secure communication protocols are fundamental building blocks of the current digital economy. *Transport Layer Security* (TLS) alone is responsible for protecting most economic transactions done using the web, with a value so high that it is hard to estimate. These protocols allow entities to exchange messages or data over a secure channel on the Internet, that provides *confidentiality* and *integrity* of communications. However, vulnerabilities may exist in the specification, the cryptographic mechanisms used, or in the implementation.

VTTLS is a new protocol proposed in this paper that provides *vulnerability-tolerant communication channels*. These channels do not rely on individual cryptographic mechanisms, so if one of them is found vulnerable (or possibly a few of them), the channels remain secure. The idea is to leverage *diversity* and *redundancy* of cryptographic mechanisms and keys, i.e., the use of different and multiple sets of mechanisms/keys, respectively. More specifically, diversity is employed in pair certificate/private key, key exchange mechanism, authentication mechanism, encryption mechanism and message authentication code (MAC). This use of diversity and redundancy is inspired by previous works on computer immunology [1], diversity in security [2], [3], and moving-target defenses [4].

VTTLS is configured with a parameter k, the *diversity factor* ($k > 1$). This parameter indicates the number of different cipher suites and different mechanisms for key exchange, authentication, encryption, and signing. This parameter means also that VTTLS remains secure as long as only $k-1$ vulnerabilities exist. We expect k to be usually small, e.g., $k = 2$ or $k = 3$, because vulnerabilities, even if unknown (zero-day), do not appear in large numbers in the same components [5].

The main contribution of this paper is the VTTLS protocol and an experimental evaluation that shows that it has an acceptable overhead when compared with the TLS implementation in which our prototype is based: OpenSSL [6].

II. Background and Related Work

This section presents related work on diversity (and redundancy) in security, provides background on TLS, and discusses vulnerabilities in cryptographic mechanisms.

A. Diversity in Security

The term *diversity* is used to describe multi-version software in which redundant versions are deliberately created and made different between themselves [2]. Without diversity, all instances are the same, with the same implementation vulnerabilities. Using diversity it is possible totw present different versions to the attacker, hopefully with different vulnerabilities. Software diversity targets mostly software implementation and the ability of the attacker to replicate the user's environment. Diversity does not change the program's logic, so it is not helpful if a program is badly designed. According to Littlewood and Strigini [2], multi-version systems on average are more reliable. They also state that the key to achieving effective diversity is that the dependence between the different programs needs to be as low as possible.

B. SSL/TLS Protocol Vulnerabilities

TLS is composed by the Handshake Protocol and the Record Protocol. The Handshake Protocol is used to establish or resume a secure session between two communicating parties – client and server. The Record protocol is responsible for processing all the messages to sent and received.

TLS vulnerabilities discovered in the past can be classified in two types: *specification vulnerabilities* that concern the protocol itself and can only be fixed by a new protocol version or an extension; and *implementation vulnerabilities* that exist in the code of some of the implementations of SSL/TLS.

One of the most recent attacks against a *specification vulnerability* is Logjam, a man-in-the-middle attack exploiting several Diffie-Hellman key exchange weaknesses [7]. *Heartbleed* is one of the most recent *implementation vulnerabilities*. It was a bug in OpenSSL 1.0.1 through 1.0.1f, when the heartbeat extension was introduced and enabled by default which allowed an attacker to perform a buffer over-read [8].

C. *Vulnerabilities in Cryptographic Schemes*

The Advanced Encryption Standard (AES) is the current American standard for symmetric encryption [9]. AES can be employed with different key sizes – 128, 192 or 256 bits. The number of rounds corresponding to each key size is, respectively, 10, 12 and 14. The most successful cryptanalysis of AES was published by Bogdanov *et al.* in 2011, using a biclique attack. Ferguson et al. [10] presented the first known attacks on the first seven and eight rounds of AES.

Regarding public-key cryptography, Kleinjung et al. [11] performed the factorization of RSA-768, a number with 232 digits. The researchers spent almost two years in the whole process, which is clearly a non-feasible time for most attacks.

Some generic attacks to hash function include brute force attacks, birthday attacks, and side-channel attacks. SHA-1 is a cryptographic hash function which produces a 160-bit message digest. Although there is no public knowledge of collisions for SHA-1, it is no longer recommended for use [12]. Stevens et al. [13] presented a freestart collision attack for SHA-1's internal compression function, where the attacker can choose the initial chaining value, known as initialization vector (IV). Regarding SHA-2 and its security, Khovratovich et al. [14] presented a biclique attack against SHA-2's preimage resistance. For several years, other researchers have also tried differential attacks for finding collisions and pseudo-collisions [15], [16].

III. VULNERABILITY-TOLERANT TLS

VTTLS is a protocol for diverse and redundant vulnerability-tolerant secure communication channels. Unlike TLS, it negotiates more than one cipher suite between client and server. Diversity and redundancy appear firstly in VTTLS in the Handshake protocol, in which client and server negotiate the k cipher suites to be used in the communication. The server chooses the best combination of k cipher suites according to the cipher suites server and client have, and the available certificates. VTTLS uses a subset of the k cipher suites to encrypt the messages.

A. *Protocol Specification*

The VTTLS Handshake Protocol is similar to the TLS Handshake Protocol. The first message to be sent is CLIENTHELLO containing a list of the client's available cipher suites. The server responds with a SERVERHELLO message containing the k cipher suites to be used in the communication. The server proceeds to send a (SERVER) CERTIFICATE message containing its k certificates. The SERVERKEYEXCHANGE message is then sent to the client. For every k cipher suites using ECDHE or DHE, the server sends a SERVERKEYEXCHANGE messages containing the server's DH ephemeral parameters for that cipher suite. Instead of computing all the ephemeral parameters and sending them all on a single larger message, the server sends each one immediately.

After sending its certificates, the client sends k CLIENTKEYEXCHANGE messages to the server. The content of these messages is based on the k cipher suites chosen. Client and server now exchange CHANGECIPHERSPEC messages. Just like in the Cipher Spec Protocol of TLS 1.2, from that moment on, they use the previously negotiated cipher suites for encrypting messages. In order to finish the Handshake, the client and server send each other a FINISHED message. These are the first encrypted messages sent using the k cipher suites.

B. *Combining Diverse Cipher Suites*

Regarding integrity, all of VTTLS' cipher suites use either AEAD (Authenticated Encryption with Associated Data) (MAC-then-Encrypt mode), SHA-2 (SHA-256 or SHA-384), SHA-1 or MD5. The choice starts from the current security status of each hash function. While AEAD (which is not an hash function) and SHA-2 are considered secure, SHA-1 is being deprecated and MD5 is considered insecure. Therefore, we excluded SHA-1 and MD5 from the possible combinations of hash functions. Our choice for creating maximum diversity relies on AEAD plus a variant of SHA-2. As AEAD is a different approach to MACs, it is expected to be vulnerable to different attacks than the ones targeting hash functions, such as SHA-2. Using a combination of two SHA-2 variants would not create maximum diversity, because even though they have different digest sizes and rounds, their structure is identical. If SHA-3 was available in TLS, it could also be used.

As we want to increase diversity in order to increase security, we prioritize mechanisms which grant perfect forward secrecy (PFS) instead of mechanisms with disjoint mathematical hard problems. After comparing several public-key encryption mechanisms, we concluded that the best three combinations are: `RSA + ECDH(E)`, `ECDSA + ECDH(E)` and `ECDSA + RSA`. VTTLS uses public-key encryption mechanisms for key exchange and authentication. Regarding authentication, the preferred combination is `ECDSA + RSA`. Regarding key exchange, the preferred is `RSA + ECDH(E)`.

The symmetric mechanisms chosen for comparison were `AES`, `Camellia`, `SEED`, and `3DES EDE`. We also included `ARIA` in our comparison, even though it is not available in VTTLS. Symmetric-key encryption mechanisms' three most important metrics are structure, mode of operation and common known attacks. In a certain sense, these metrics are related: attacks target a specific structure or mode of operation. Therefore, the combinations of symmetric encryption mechanisms we consider to be the most diverse are: `AES + CAMELLIA` (using a different mode of operation and key size), and `AES256-GCM + SEED128-CBC`. We consider the most diverse combination to be `AES256-GCM + CAMELLIA128-CBC` due to the fact that their structure is different, as its mode of operation and the set of known attacks is disjoint. Although, using a cipher suite that contains `Camellia` or `SEED`, in order to maximize diversity in symmetric encryption would force reduced diversity and security essentially in MAC. In the end, the best combination is `AES256-GCM + AES128-CBC` which can be considered diverse, although it is not the most diverse of all the ones considered.

Concluding, the best combination of cipher suites is arguably: `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` and `TLS_RSA_WITH_AES_128_CBC_SHA256`. The least diverse mechanisms are MAC and symmetric encryption, due to the fact that TLS 1.2 does not support SHA-3 and OpenSSL does not support `Camellia` keyed-hash message authentication code (HMAC) based cipher suites.

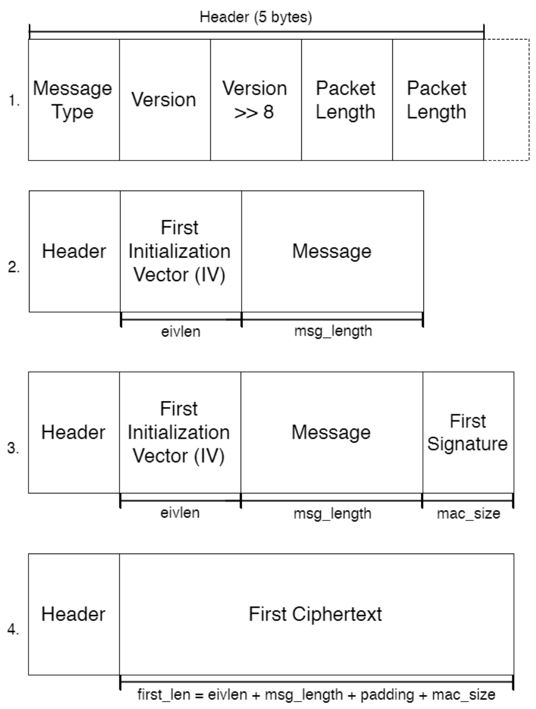

Fig. 1: First four steps regarding the ordering of the encryption and signing of vTTLS using a diversity factor $k = 2$.

IV. Implementation

vTTLS' implementation is a modified version of OpenSSL v1.0.2g. Existing software such as OpenSSL has the advantage of being extensively tested. Furthermore, creating a new secure communication channel, and consequently a new API, would create adoption barriers to programmers otherwise willing to use our protocol. Therefore, we chose to implement vTTLS based on OpenSSL and keeping the same API. Nevertheless, OpenSSL is a huge code base (currently 438,841 lines of code) and modifying it so support diversity was quite a challenge. vTTLS adds a few functions to the OpenSSL API. They allow defining additional certificates, keys, cipher suites, etc. The signing and encryption ordering is very important for vTTLS. Figure 1 shows the ordering for one cipher and one MAC in the OpenSSL implementation.

The approach taken was the following, ordered from first to last: apply the first MAC to the plaintext; encrypt the first message and its MAC with the first encryption mechanism; apply the second MAC to the ciphertext; encrypt the ciphertext and its MAC with the second encryption mechanism. Figure 2 shows the final ordering of vTTLS communication data.

In relation to the *Record Protocol*, signing and encrypting k times has a cost in terms of message size. Figures 1 and 2 show also the expected increase of the message size due to the use of a second MAC and a second encryption function (for $k = 2$). For OpenSSL, the expected size of a message is $first\_len = eivlen + msg\_length + padding + mac\_size$, where $eivlen$ is the size of the initialization vector (IV), msg_length the original message size, $padding$ the size of the padding in case a block cipher is used, and mac_size the size of the MAC. For vTTLS, the additional size of the message is $eivlen\_sec + first\_len + padding\_sec + mac\_size\_sec$, where $eivlen\_sec$ is the size of the IV associated with the second cipher and mac_size_sec the size of the second MAC.

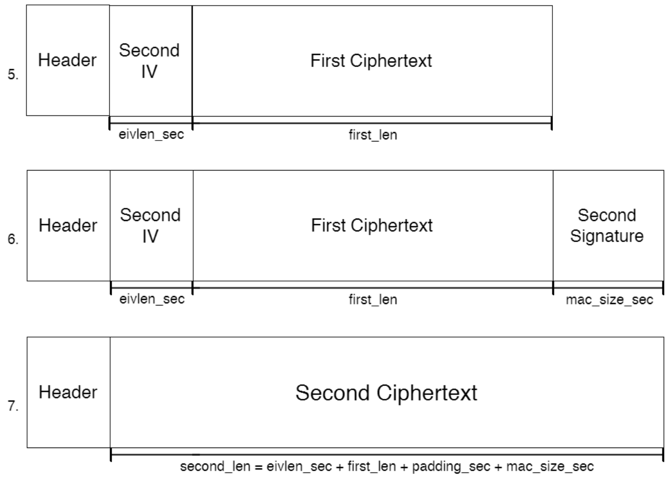

Fig. 2: Final three steps regarding the ordering of the encryption and signing of vTTLS using a diversity factor $k = 2$.

In the best case, the number of packets is the same for OpenSSL and vTTLS. In the worst case, one additional packet may be sent if the encryption function requires a fixed block size and the maximum size of the packet is exceeded by, at least, one byte after the second MAC and the second encryption. In this case, an additional full packet is needed due to the constraint of having fixed block size.

V. Experimental Evaluation

Implementing diversity has performance costs as it creates overhead in the communication. Every message sent needs to be ciphered and signed $k - 1$ times more than using a TLS implementation and every message received needs to be deciphered and verified also $k - 1$ times more. In the worst case, users should experience a connection k times slower than using OpenSSL. We considered $k = 2$ in all experiments, as this is the value we expect to be used in practice (we expect vulnerabilities to appear rarely, so the ability to tolerate one vulnerability per mechanism is sufficient). All the tests were done in the same controlled environment and same geographic locations in order to maintain the evaluation valid, exact and precise. We evaluated vTTLS's *performance* and *costs* and considered the OpenSSL implementation of TLS as the baseline.

A. Performance

In order to evaluate vTTLS performance, we executed several tests in order to understand if the overhead of vTTLS is lower, equal, or bigger than k comparing to OpenSSL. We configured vTTLS to use the following cipher suites: `TLS_RSA_WITH_AES_256_GCM_SHA384` and `TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384`. The suite used with OpenSSL was the latter.

To evaluate the performance of the handshake, we executed 100 times the Handshake Protocol of both vTTLS and

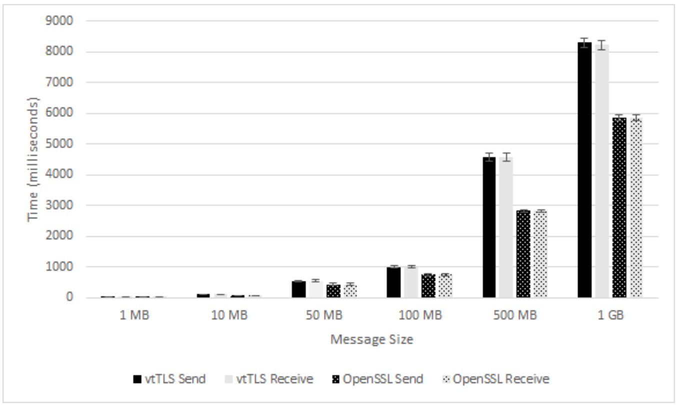

Fig. 3: Comparison between the time it takes to send and receive a message using vtTLS and OpenSSL.

OpenSSL. In average, the vtTLS handshake took 3.909 milliseconds to conclude whereas the OpenSSL handshake took 2.345 milliseconds. Therefore, the vtTLS handshake takes 66.7% longer than the OpenSSL handshake which is better than the worst case (that would be 100% longer).

B. Data Communication

We also performed data communication tests to assess the overhead generated by the diversity and redundancy of mechanisms. The communication is expected to be, at most, $k = 2$ times slower than using TLS. For this test, we considered a sample of 100 messages sent and received using vtTLS, and other 100 messages sent and received using OpenSSL. Figure 3 shows the comparison between the time it takes to send and receive a message using vtTLS and OpenSSL.

In average, a message sent through a vtTLS channel takes 22.88% longer than a message sent with OpenSSL. For example, a 50 MB message takes, in average 534.55 ms to be sent with vtTLS. Using OpenSSL, the same message takes 435.01 ms to be sent. The overhead generated is much smaller than the worst case.

We evaluated the message size increase of the ciphertext of several plaintexts with different sizes. A 1 MB plaintext message corresponds to a ciphertext of $1,029,054$ bytes using vtTLS, while using OpenSSL the same message converts into a message of $1,025,856$ bytes i.e. sending 1 MB through a vtTLS channel costs an additional $3,198$ bytes over using an OpenSSL channel.

VI. Conclusions

vtTLS is a diverse and redundant vulnerability-tolerant secure communication protocol designed for communication on the Internet. It aims at increasing security using diverse cipher suites to tolerate vulnerabilities in the encryption mechanisms used in the communication channel. In order to evaluate our solution, we compared it to an OpenSSL communication channel. While expected to be $k = 2$ times slower than an OpenSSL channel, the evaluation showed that using diversity and redundancy of cryptographic mechanisms in vtTLS does not generate such a high overhead. vtTLS takes, in average, 22.88% longer to send a message than TLS/OpenSSL, but considering the increase in security, this overhead is acceptable. Overall, considering the additional costs of having an extra certificate, the time increase, and potential management costs, vtTLS provides an interesting trade-off for a set of critical security applications.

Acknowledgements This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] S. Forrest, S. A. Hofmeyr, and A. Somayaji, "Computer immunology," *Communications of the ACM*, vol. 40, no. 10, pp. 88–96, Oct. 1997.

[2] B. Littlewood and L. Strigini, "Redundancy and diversity in security," in *Computer Security – ESORICS 2004, 9th European Symposium on Research Computer Security*, 2004, pp. 227–246.

[3] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "OS diversity for intrusion tolerance: Myth or reality?" in *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks*, 27–30 June 2011, pp. 383 –394.

[4] M. Carvalho and R. Ford, "Moving-target defenses for computer networks," *IEEE Security and Privacy*, vol. 12, no. 2, pp. 73–76, 2014.

[5] L. Bilge and T. Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2012, pp. 833–844.

[6] J. Viega, M. Messier, and P. Chandra, *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly, 2002.

[7] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. Vandersloot, E. Wustrow, and S. Paul, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, October 2015. [Online]. Available: https://weakdh.org/imperfect-forward-secrecy.pdf

[8] M. Carvalho, J. DeMott, R. Ford, and D. A. Wheeler, "Heartbleed 101," *IEEE Security Privacy*, vol. 12, no. 4, pp. 63–67, July 2014.

[9] V. Rijmen and J. Daemen, "Advanced Encryption Standard," *U.S. National Institute of Standards and Technology (NIST)*, vol. 2009, pp. 8–12, 2001.

[10] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved cryptanalysis of Rijndael," in *Proceedings of Fast Software Encryption*, G. Goos, J. Hartmanis, J. van Leeuwen, and B. Schneier, Eds. Springer, 2001, vol. LNCS 1978, pp. 213–230.

[11] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, H. Te Riele, A. Timofeev, and P. Zimmermann, "Factorization of a 768-bit RSA modulus," in *Proceedings of the 30th Annual Conference on Advances in Cryptology*, vol. LNCS 6223, 2010, pp. 333–350.

[12] ENISA, "Algorithms, key size and parameters report – 2014," nov 2014.

[13] M. Stevens, P. Karpman, and T. Peyrin, "Freestart collision on full SHA-1," Cryptology ePrint Archive, Report 2015/967, 2015.

[14] D. Khovratovich, C. Rechberger, and A. Savelieva, "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family," Cryptology ePrint Archive, Report 2011/286, 2011, http://eprint.iacr.org/.

[15] C. Dobraunig, M. Eichlseder, and F. Mendel, "Analysis of SHA-512/224 and SHA-512/256," Cryptology ePrint Archive, Report 2016/374, 2016, http://eprint.iacr.org/.

[16] M. Eichlseder, F. Mendel, and M. Schlffer, "Branching Heuristics in Differential Collision Search with Applications to SHA-512," Cryptology ePrint Archive, Report 2014/302, 2014, http://eprint.iacr.org/.