1.1 Key Derivation Functions and MAC Algorithms

Current standards for implementation of TCP-AO are specified in RFC 5925[2] and RFC 5926[3]. RFC 5925 outlines the structure of TCP-AO, the general requirements for its implementation as well as numerous other security considerations. In contrast, RFC 5926 details more implementation specific requirements for the first implementations of the protocol. As such it mandates the support of SHA-1 and AES-128 for the Key Derivation functions (KDF) as well as for the MACs. More formally, it lists the algorithms as follows:

MAC Algorithms:

• HMAC-SHA-1-96 [RFC2104][FIPS-180-3]

• AES-128-CMAC-96 [NIST-SP800-38B][FIPS197]

KDF Algorithms:

• KDF\_HMAC\_SHA1

• KDF\_AES\_128\_CMAC [RFC 5926]

In revisiting the algorithms listed above, it must be noted that ﻿SHA-1 was deprecated by the National Institute of Standards and Technology (NIST) in 2011.[NIST-SHA1] In fact, The NIST Policy on the usage of SHA-1 as a hash function stated that all federal agencies should stop using SHA-1 as a hash function in any applications that require collision resistance. [4] Therefore ﻿we recommend support for SHA-1 be replaced with the Galois MAC (GMAC), a special case of the AES CMAC algorithm using Galois counter mode. Thus our amendment to list of MAC algorithms is as follows:

MAC Algorithms:

• GMAC -128-96

• AES-128-CMAC-96 [NIST-SP800-38B][FIPS197]

KDF Algorithms:

• KDF\_GMAC\_128-96

• KDF\_AES\_128-96\_CMAC

Note: Given the limitation to 12 byte MAC digests in cases in where two SACK blocks are included in the segment, we then truncate the digest to 96 bits.

1.2 Storing the Keys

Specified in RFC 5925, the master key tuple will contain the source and destination IP addresses, source and destination ports as Connection Identifiers. These include A send and receive ID, which are used to identify the shared keys used for MAC calculations on outgoing segments and on the verification of incoming segments respectively. In addition to these values, RFC 5925 also mandates the inclusion of an option flag. This flag dictates whether non-TCP-AO options should be included in the MAC calculations. The aforementioned values and the respective initial segment sequence numbers (ISN’s) of each participant are then used in the Key Derivation Functions whose respective outputs are each used as unidirectional traffic keys.

Upon the pre-initialization of a connection C, C’s traffic keys are initially set to NULL. Upon C’s establishment its traffic keys are computed and stored locally so that they needn’t be recomputed during C’s lifetime.

The Keys can be stored in a simple array in which the position of the Key in the array reflects its Key ID. For example, a key with the key ID 7 will be located at the 7th position in the Array.  
Note: While processing an incoming segment before we calculate the hash using the key specified by the incoming segment, we must check if this connection is allowed to use that key. If it is allowed to, only then must we calculate the hash. This reduces the computational expense.  
There will be one instance of this structure for every connection. It must be noted that we must maintain a list of each of these instances sorted by Key ID so that binary search may be performed on the list to get the required key in logarithmic (lg n) time.

1.3 Computing the MACs

The MAC is computed over the following fields as specified by RFC 5925.

1> The Sequence Number Extension (SNE)

2> The IP pseudo-header: IP source and destination addresses, protocol number, and segment length, all in network byte order

3> The TCP Header by default including Options. [TCP checksum and Mac zeroed out]. The TCP-AO option is always included in the MAC computation. The interface provides the user with an option to specify if the MAC should be computed over the other options or not.

4> The TCP data, i.e., the payload of the TCP segment[].

Implementation Plan

As of this writing no TCP-AO source code implementation for the Linux Kernel has been published. However, the implementation of the TCP protocol in the Linux Kernel currently includes a TCP-MD5 option that creates a MD5 hash of the TCP segment.[] Since the introduction of TCP-AO deprecates TCP-MD5, RFC 5925 mandates that all TCP-AO implementations be backwards compatible with TCP-MD5. However, this brings about a key caveat, TCP MD5 does not support any security protocol negotiation after the session has been established. As such, the TCP-AO implementation must account for the case where TCP-MD5 is used to establish a connection. To do this, we implement a mutual exclusion rule. That is when TCP-MD5 is used, TCP-AO should not be allowed to run on the same connection.

We know that the TCP header and options must be no larger than 60 bytes[RFC 793]. This leaves 9 bytes for options other than the SACK Permitted, Timestamps and Window scale. In case of non-SYN segments we can continue to use the 16 byte TCP-AO if the only one SACK block is used, but in case of the usage of 2 SACK blocks the size of the TCP-AO MAC has to be reduced further. The truncation to 96 bits does not affect the strength of the MAC algorithms. [\*The paper Eric sent us – Security of Truncated macs\*]. We implement the same structure as suggested by the paper. This basic structure of the option must be added into the process of packet creation in the Kernel files.

As mentioned before, the Linux kernel contains files that specify how a network packet must be processed at both the sender and receivers end. The various files contain functions that create, parse, add to the packets and also decide if a packet should be accepted or not. The changes we have made are to the files responsible for the processing of the TCP headers and options.  
Listed below are the files below are the files that have to be changed:

1. File Name - tcp.h  
   Location - /include/net/tcp.h  
   The purpose of this file is to provide a basic definition of most of the functions involved in the processing of the TCP segments.
2. File Name – tcp\_input.c  
   Location - /net/tcp\_input.c  
   This file contains functions that are used to parse the TCP-AO option and return useful information such as the keyID, rNextKeyId and the MAC sent.
3. File Name – tcp\_output.c  
   Location - /net/tcp\_output.c  
   \*Probably contains the Construction of the packet process. Still working on pinpointing the exact purpose\*
4. File Name – tcp\_IPV4.c  
   Location - /net/tcp\_ipv4.c  
   This file contains functions such as sending resets (RSTs) if the hashes don’t match and so on. This file also contains the sequence numbers which need to be maintained as 64 bits in order to attacks on Long Lived Connections. (Note that the actual sequence number mechanism is not altered but just a copy is maintained to run the MACs over)

1.4 Socket Level Implementation

All the afore-mentioned details run at the Kernel Level and everyday users do not have any way to run these options. They can however use Sockets to interact with the implemented protocols in the Kernel. We must implement changes in the socket files, programming new parameters to allow the user to set the option flag for each connection, specify keys for each connection and even to entirely to enable or disable TCP-AO.

1.5 Verifying successful implementation: [?]

The verification of the results is done in two phases.

In the first phase we test two iterations of the program on the same machine to certify that the TCP Segment is created correctly and the MAC is the same on both the client and the server.

Once we have satisfactorily verified that our implementation works in the first phase, we will test our implementation of TCP-AO on two separate machines. We then use WireShark to capture the messages they exchanged. Upon inspection, we will ensure that segments include the required TCP-AO components in its TCP header options.

We then inject an RST segment from a 3rd party machine and check if the connection is dropped. Ideally the connection should not be dropped as the RST segment is unauthenticated.

References-

[1] MD5 - 2385

[2] RFC 5925

[3] RFC 5926

[4] https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions.

[5] RFC 4543 The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH.