The Secure Socket API: TLS as an Operating System Service

Published in: 27th USENIX Security Symposium, 2018

Summarized by

Sangwon Lim (sangwonlim@snu.ac.kr)

2022-10-31

Contents

Introduction

- Transport Layer Security (TLS) is the most popular security protocol used on the Internet
- Because of the complex security APIs, developers make frequent mistakes
- The authors present the Secure Socket API (SSA), a TLS API for applications designed to work within the confines of the existing standard POSIX socket API already familiar to developers

Introduction

- The Contributions of this paper
  - An analysis of contemporary use of TLS by 410 Linux packages and a qualitative breakdown of OpenSSL’s 504 API endpoints for TLS functionality
  - A description of the Secure Socket API and how it fits within the existing POSIX socket API, with descriptions of the relevant functions, constants, and administrator controls
  - A description of and source code for a prototype implementation of the Secure Socket API
  - A description of and source code for a tool that dynamically ports existing OpenSSL-using applications to use the SSA without requiring modification

Motivation

- TLS use by applications is mired by complicated APIs and developer mistakes
  - ➔ Developers lack a common, usable security API

- A related problem is that the reliance on application developers to implement security inhibits the control administrators have over their own machines
  - ➔ Administrators lack control over secure connections

Secure Socket API (SSA) Design Goals

- Both easy to use for developers and grants a high degree of control to system administrators
  1. Enable developers to use TLS through the existing set of functions provided by the POSIX socket API
  2. Support direct administrator control over the parameters and settings for TLS connections made by the SSA
  3. Export a minimal set of TLS options to applications that allow general TLS use and drastically reduce the amount of TLS functions in contemporary TLS APIs
  4. Facilitate the adoption of the SSA by other programming languages

OpenSSL Analysis

- The authors explore what functionality should be present in the SSA and how to distill the 504 TLS-related OpenSSL symbols to the handful provided by the POSIX socket interface

- They collected the source code for all standard Ubuntu repository software packages that directly depend on libssl
  - ✓410 packages using C/C++

- Immediately they found that 170 of the 504 API symbols are not used by any application in their analysis

OpenSSL Analysis

- Breakdown of OpenSSL's libssl symbols

Category	Symbols	Uses
TLS Functionality		
Version selection	29	1306
Cipher suite selection	39	1467
Extension management	68	597
Certificate/Key management	73	2083
Certificate/Key validation	51	3164
Session management	61	1155
Configuration	19	1337
Other		
Allocation	33	6087
Connection management	41	5228
Miscellaneous	64	1468
Instrumentation	26	232

The Secure Socket API (SSA)

- Under the Secure Socket API, all TLS functionality is built directly into the POSIX socket API

- Administrators set global policy (and for individual applications)
  - ✓TLS Version, Cipher Suites, Certificate Validation, Enabled Extensions, Session Caching, Default Paths for keys and certificates
  - ✓Developers can increase security, but cannot decrease it

The Secure Socket API (SSA)

POSIX Function	General Behavior	Behavior under IPPROTO_TLS
`socket`	Create an endpoint for communication utilizing the given protocol family, type, and optionally a specific protocol.	Create an endpoint for TLS communication, which utilizes TCP for its transport protocol if the `type` parameter is `SOCK_STREAM` and uses DTLS over UDP if `type` is `SOCK_DGRAM`.
`connect`	Connect the socket to the address specified by the `addr` parameter for stream protocols, or indicate a destination address for subsequent transmissions for datagram protocols.	Perform a connection for the underlying transport protocol if applicable (e.g., TCP handshake), and perform the TLS handshake (client-side) with the specified remote address. Certificate and hostname validation is performed according to administrator and as optionally specified by the application via `setsockopt`.
`bind`	Bind the socket to a given local address.	No TLS-specific behavior.
`listen`	Mark a connection-based socket (e.g., `SOCK_STREAM`) as a passive socket to be used for accepting incoming connections.	No TLS-specific behavior.
`accept`	Retrieve connection request from the pending connections of a listening socket and create a new socket descriptor for interactions with the remote endpoint.	Retrieve a connection request from the pending connections, perform the TLS handshake (server-side) with the remote endpoint, and create a new descriptor for interactions with the remote endpoint.

```c
/* Use hostname address family */
struct sockaddr_host addr;
addr.sin_family = AF_HOSTNAME;
strcpy(addr.sin_addr.name, "www.example.com");
addr.sin_port = htons(443);

/* Request a TLS socket (instead of TCP) */
fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TLS);
/* TLS Handshake (verification done for us) */
connect(fd, &addr, sizeof(addr));

/* Hardcoded HTTP request */
char http_request[] = "GET / HTTP/1.1\r\n..."
char http_response[2048];
memset(http_response, 0, 2048);
/* Send HTTP request encrypted with TLS */
send(fd,http_request,sizeof(http_request)-1,0);
/* Receive decrypted response */
recv(fd, http_response, 2047, 0);
/* Shutdown TLS connection and socket */
close(fd);
/* Print response */
printf("Received:\n%s", http_response);
return 0;
```

< Client example >

The Secure Socket API (SSA)

POSIX Function	General Behavior	Behavior under IPPROTO_TLS
`send`, `sendto`, etc.	Transmit data to a remote endpoint.	Encrypt and transmit data to a remote endpoint.
`recv`, `recvfrom`, etc.	Receive data from a remote endpoint.	Receive and decrypt data from a remote endpoint.
`shutdown`	Perform full or partial tear-down of connection, based on the `how` parameter.	Send a TLS close notify.
`close`	Close a socket, perform connection tear-down if there are no remaining references to socket.	Close a socket, send a TLS close notify, and tear-down connection, if applicable.
`select`, `poll`, etc.	Wait for one or more descriptors to become ready for I/O operations.	No TLS-specific behavior.
`setsockopt`	Manipulate options associated with a socket, assigning values to specific options for multiple protocol levels of the OSI stack.	Manipulate TLS specific options when the `level` parameter is `IPPROTO_TLS`, such as specifying a certificate or private key to associate with the socket. Other `level` values interact with the socket according to their existing semantics.
`getsockopt`	Retrieve a value associated with an option from a socket, specified by the `level` and `option_name` parameters.	For a `level` value of `IPPROTO_TLS`, retrieve TLS-specific option values. Other `level` values interact with the socket according to their existing semantics.

```c
/* Use standard IPv4 address */
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
/* We want to listen on port 443 */
addr.sin_port = htons(443);

/* Request a TLS socket (instead of TCP) */
fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TLS);
/* Bind to local address and port */
bind(fd, &addr, sizeof(addr));
/* Assign certificate chain */
setsockopt(fd, IPPROTO_TLS,
           TLS_CERTIFICATE_CHAIN,
           CERT_FILE, sizeof(CERT_FILE));
/* Assign private key */
setsockopt(fd, IPPROTO_TLS, TLS_PRIVATE_KEY,
           KEY_FILE, sizeof(KEY_FILE));
listen(fd, SOMAXCONN);

while (1) {
  struct sockaddr_storage addr;
  socklen_t addr_len = sizeof(addr);
  /* Accept new client and do TLS handshake
  using cert and keys provided */
  int c_fd = accept(fd, &addr, &addr_len);
  /* Receive decrypted request */
  recv(c_fd, request, BUFFER_SIZE, 0);
  handle_req(request, response);
  /* Send encrypted response */
  send(c_fd, response, BUFFER_SIZE, 0);
  close(c_fd);
}
```

< Server example >

The Secure Socket API (SSA)

- Sample of socket options at the IPPROTO_TLS level
  - TLS_REMOTE_HOSTNAME
  - TLS_HOSTNAME
  - TLS_CERTIFICATE_CHAIN
  - TLS_PRIVATE KEY
  - TLS_TRUSTED_PEER_CERTIFICATES
  - TLS_SESSION TTL
  - TLS_DISABLE_CIPHER
  - TLS_PEER_IDENTITY

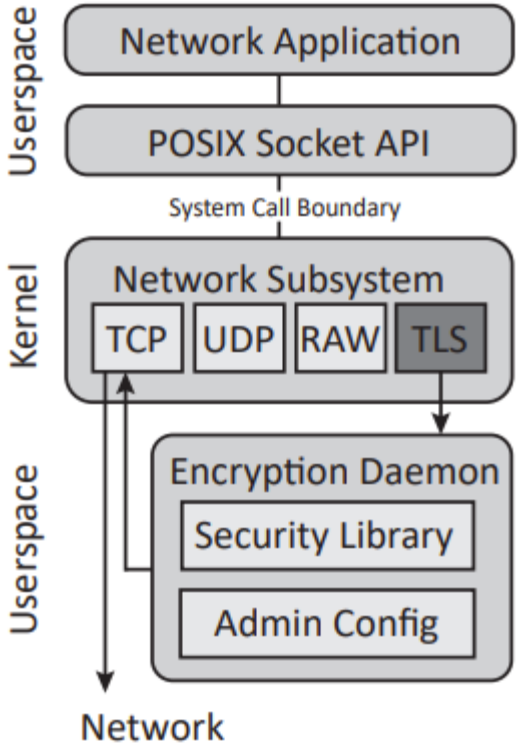
The Secure Socket API (SSA)

- Summary of code changes required to port a sample of applications to use the SSA

Program	LOC Modified	LOC removed	Familiar with code	Time Taken
wget	15	1,020	No	5 Hrs.
lighttpd	8	2,063	No	5 Hrs.
ws-event	5	0	Yes	5 Min.
netcat	5	0	No	10 Min.

Implementation

- Data flow for SSA usage by network applications



- The application shown is using the TLS (which uses TCP internally for connection-based SOCK STREAM sockets)
- TLS: a loadable Linux kernel module
- Encryption daemon: security library role

Implementation

- Coercing Existing Applications
  - The authors explored the ability to dynamically coerce TLS applications using security libraries to use the SSA instead
  - They supply replacement OpenSSL functions through a shared library for dynamically linked applications to override normal behavior
  - In the experimentation with this tool, they successfully forced wget, irssi, curl, and lighttpd to use the SSA for TLS dynamically, bringing the TLS behavior of these applications under admin control.

Discussion

- Benefits
  - Because the SSA design moves all TLS functionality to a centralized service, administrators gain the ability to configure TLS behavior on a system-wide level, and tailor settings of individual applications to their specific needs
  - By implementing the SSA with a kernel module, developers who wish to use it do not have to link with any additional userspace libraries
- Alternative Implementation
  - User space only: wrapping the native socket API
  - Kernel only: some performance gains in TLS are also possible

Conclusion

- The authors explored TLS library simplification and furthering administrator control through the POSIX socket API
  - ✓Developer’s typical mistakes → simple API & admin control
- They showed that existing code could be changed to use SSA without much effort
- The POSIX socket API is a natural fit for a TLS API