

# The Secure Socket API

TLS as an Operating System Service

**Mark O'Neill**, Scott Heidbrink, Jordan Whitehead, Tanner Perdue, Luke Dickinson,  
Torstein Collett, Nick Bonner, Kent Seamons, and Daniel Zappala



your apps are vulnerable



why?

“The root cause of most of these vulnerabilities is the **terrible design** of the APIs to the underlying SSL libraries”

*--The most dangerous code in the world: validating SSL certificates in non-browser software. Martin Georgiev et al., 2012. ACM CCS.*

using TLS is hard

Symbols in libssl: 504



```
SSL_CTX_set_verify()      x509_verify_cert()  
SSL_CTX_set_cert_verify_callback()
```

```

#include <openssl/x509.h>
#include <openssl/ssl.h>
#include <string.h>
#include <openssl_hostname_validation.h>

#define HOSTNAME_MAX_SIZE 255
#define CURL_HOST_MATCH 1
#define CURL_HOST_MISMATCH 0

int Curl_cert_hostcheck(const char *match_pattern, const char *hostname);

// IPVERS: VERSION NUMBER = 3011000000 // defined(LIBRESSL_VERSION_NUMBER)
#define ASN1_STRING_get0_data ASN1_STRING_data

static HostnameValidationResult matches_common_name(const char *hostname, const X509 *server_cert) {
    int common_name_loc = -1;
    X509_NAME_ENTRY *common_name_entry = NULL;
    ASN1_STRING *common_name_asn1 = NULL;
    char *common_name_str = NULL;

    // Find the position of the CN field in the Subject field of the certificate
    common_name_loc = X509_NAME_get_index_by_NID(X509_get_subject_name(X509 *) server_cert, NID_commonName, -1);
    if (common_name_loc < 0) {
        return Error;
    }

    // Extract the CN field
    common_name_entry = X509_NAME_get_entry(X509_get_subject_name(X509 *) server_cert, common_name_loc);
    if (common_name_entry == NULL) {
        return Error;
    }

    // Convert the CN field to a C string
    common_name_asn1 = X509_NAME_ENTRY_get_data(common_name_entry);
    if (common_name_asn1 == NULL) {
        return Error;
    }

    common_name_str = (char *) ASN1_STRING_data(common_name_asn1);

    // Make sure there isn't an embedded NUL character in the CN
    if ((size_t)ASN1_STRING_length(common_name_asn1) != strlen(common_name_str)) {
        return MalformedCertificate;
    }

    // Compare expected hostname with the CN
    if (Curl_cert_hostcheck(common_name_str, hostname) == CURL_HOST_MATCH) {
        return MatchFound;
    }
}

```

```

static int hostmatch(const char *hostname, const char *pattern)
{
    const char *pattern_label_end, *pattern_wildcard, *hostname_label_end;
    int wildcard_enabled;
    size_t prefixlen, suffixlen;
    pattern_wildcard = strchr(pattern, '*');
    if (pattern_wildcard == NULL) {
        return Curl_raw_equal(pattern, hostname) ?
            CURL_HOST_MATCH : CURL_HOST_MISMATCH;
    }

    // We require at least 2 dots in pattern to avoid too-wide wildcard
    match = 0;

    wildcard_enabled = 0;
    pattern_label_end = strchr(pattern, '.');
    if (pattern_label_end == NULL || strchr(pattern_label_end, '*') == NULL ||
        pattern_wildcard > pattern_label_end ||
        Curl_raw_nequal(pattern, hostname, 0)) {
        wildcard_enabled = 1;
    }

    if (!wildcard_enabled)
        return Curl_raw_equal(pattern, hostname) ?
            CURL_HOST_MATCH : CURL_HOST_MISMATCH;

    hostname_label_end = strchr(hostname, '.');
    if (!hostname_label_end ||
        !Curl_raw_equal(pattern_label_end, hostname_label_end))
        return CURL_HOST_MISMATCH;

    prefixlen = pattern_wildcard - pattern;
    suffixlen = pattern_label_end - (pattern_wildcard + 1);
    return Curl_raw_nequal(pattern, hostname, prefixlen) ||
        Curl_raw_nequal(pattern_wildcard + 1, hostname_label_end - suffixlen,
            suffixlen) ?
            CURL_HOST_MISMATCH :
            CURL_HOST_MATCH;
}

int Curl_cert_hostcheck(const char *match_pattern, const char *hostname)
{
    return hostmatch(hostname, match_pattern) == CURL_HOST_MATCH;
}

```

```

#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <openssl/ssl.h>
#include <openssl/x509.h>
#include <openssl_hostname_validation.h>

#define BUFFER_MAX 256
static char root_store_filename_redhat[] = "/etc/pki/tls/certs/ca-bundle.crt";

int connect_to_host(char *host, char *port, int protocol);
SSL *openssl_connect_to_host(int sock, char *hostname);

int main() {
    int i;
    int sock;
    SSL *tls;
    char query[BUFFER_MAX];
    char response[BUFFER_MAX];
    int query_len;
    char hostname[] = "www.google.com";

    sock = connect_to_host(hostname, "443", SOCK_STREAM);
    tls = openssl_connect_to_host(sock, hostname);

    sprintf(query, "GET /HTTP/1.1/robots.txt/ HTTP/1.1", hostname);
    query_len = strlen(query);
    SSL_write(tls, query, query_len);
    SSL_read(tls, response, sizeof(response));
    printf("Query: %s, response: %s\n", query, response);

    close(sock);
    SSL_shutdown(tls);
    SSL_free(tls);
    return 0;
}

openssl_connect_to_host(int sock, char *hostname,
    X509 *cert, int protocol) {
    SSL *tls;
}

```

```

Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: IP_JAR=2018-03-01-19; expires=Sat, 31-Mar-2018 19:41:59 GMT; path=/; domain=google.com
Set-Cookie: NID=124=afErBe|YBKBIA001-KPkeZ-4WIdrgZIVQV4rELIAQrhhhsZg3EXfYI7IAF0QMMW7LfL3-TW8W3WgJds; expires=Fri, 31-Aug-2018 19:42:03 GMT; path=/; domain=google.com; HttpOnly
Alt-Svc: h2="443"; ma=2592000; quic="51303330; quic=51303338; quic=51303337; quic=51303335; quic=443"; ma=2592000; v="41.39.38.37.35"
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

7271
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta c
content="Search the world's information, including webpages, images, videos and more. Google ha
many special features to help you find exactly what you're looking for." name="description">
<meta content="noop" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Conte
nt-Type"><meta content="/images/branding/google/1x/google_standard_color_128dp.png" itemprop
="image"><title>Google</title><script nonce="4QW9Z2Wj1k47Y8Hk4bQw=">(function()mark@localhost
simplyssl) {
    // SSL get peer certificate(tls);
    cert = SSL_get_peer_certificate(tls);
    if (cert == NULL) {
        fprintf(stderr, "Failed to get peer certificate\n");
        exit(EXIT_FAILURE);
    }

    if (validate_hostname(hostname, cert) != MatchFound) {
        fprintf(stderr, "Failed to validate hostname in certificate\n");
        exit(EXIT_FAILURE);
    }

    return tls;
}

int connect_to_host(char *host, char *service, int protocol) {
    int sock;
    int ret;
    struct addrinfo hints;
    struct addrinfo *addr_ptr;
    struct addrinfo *addr_list;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC; // IP or IPV6 we don't care
    ret = getaddrinfo(host, service, &hints, &addr_list);
    if (ret != 0) {
        fprintf(stderr, "Failed to getaddrinfo\n");
        exit(EXIT_FAILURE);
    }

    for (addr_ptr = addr_list; addr_ptr != NULL; addr_ptr = addr_ptr->ai_next) {
        sock = socket(addr_ptr->ai_family, addr_ptr->ai_socktype, addr_ptr->ai_protocol);
        if (sock == -1) {
            perror("socket");
            continue;
        }
        if (connect(sock, addr_ptr->ai_addr, addr_ptr->ai_addrlen) == -1) {
            perror("connect");
            close(sock);
            continue;
        }
        break;
    }

    fprintf(stderr, "Failed to find a suitable address for connection\n");
    exit(EXIT_FAILURE);
}

return sock;
}

```

# 317 Lines

```

static int Curl_raw_equal(const char *first, const char *second)
{
    while(*first && *second) {
        if (Curl_raw_toupper(*first) != Curl_raw_toupper(*second))
            // get out of the loop as soon as they don't match
            break;
        first++;
        second++;
    }
    return (Curl_raw_toupper(*first) == Curl_raw_toupper(*second));
}

static int Curl_raw_nequal(const char *first, const char *second, size_t max)
{
    while(*first && *second && max) {
        if (Curl_raw_toupper(*first) != Curl_raw_toupper(*second)) {
            break;
        }
        max--;
        first++;
        second++;
    }
    if (0 == max)
        return 1; // they are equal this far
    return Curl_raw_toupper(*first) == Curl_raw_toupper(*second);
}

static int hostmatch(const char *hostname, const char *pattern)
{
    const char *pattern_label_end, *pattern_wildcard, *hostname_label_end;
    int wildcard_enabled;
    size_t prefixlen, suffixlen;
    pattern_wildcard = strchr(pattern, '*');
    if (pattern_wildcard == NULL)
        return Curl_raw_equal(pattern, hostname) ?
            CURL_HOST_MATCH : CURL_HOST_MISMATCH;

    // We require at least 2 dots in pattern to avoid too-wide wildcard
    match = 0;

    wildcard_enabled = 0;
    pattern_label_end = strchr(pattern, '.');
    if (pattern_label_end == NULL || strchr(pattern_label_end, '*') == NULL ||
        pattern_wildcard > pattern_label_end ||
        Curl_raw_nequal(pattern, hostname, 0)) {
        wildcard_enabled = 1;
    }

    if (!wildcard_enabled)
        return Curl_raw_equal(pattern, hostname) ?
            CURL_HOST_MATCH : CURL_HOST_MISMATCH;

    hostname_label_end = strchr(hostname, '.');
}

```

```

the Software is provided "AS-IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

THE SOFTWARE IS PROVIDED "AS-IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

//
// Helper functions to perform basic hostname validation using OpenSSL.
//
// Please read https://github.com/openssl/openssl/blob/master/openssl.pdf before
// attempting to use this code. This whitespace describes how the code works,
// how it should be used, and what its limitations are.
//
// Note from Mark D'Nettil: Wildcard matching has been added and resolution checking
// can be handled by plugins.
//
// Author: Alban Diquet
// License: See LICENSE
//

typedef enum {
    MatchFound,
    MatchNotFound,
    NoSANPresent,
    MalformedCertificate,
    Error
} HostnameValidationResult;

//
// Validates the server's identity by looking for the expected hostname in the
// certificate, as described in RFC 6125. It first tries to find a match
// in the Subject Alternative Name extension. If the extension is not present in
// the certificate, it checks the Common Name instead.
//
// Returns MatchFound if a match was found.
// Returns MatchNotFound if no matches were found.
// Returns MalformedCertificate if any of the hostnames had a NUL character embedded in it.
// Returns Error if there was an error.
//
HostnameValidationResult validate_hostname(const char *hostname, const X509 *server_cert);

```

```

SSL_library_init();
OpenSSL_add_all_algorithms();
SSL_load_error_strings();
SSL_library_init();
OpenSSL_add_all_algorithms();
SSL_load_error_strings();

tls_ctx = SSL_CTX_new(TLS_client_method());
if (tls_ctx == NULL) {
    fprintf(stderr, "Could not create SSL_CTX\n");
    exit(EXIT_FAILURE);
}

SSL_CTX_set_verify(tls_ctx, SSL_VERIFY_PEER, NULL);
if (SSL_CTX_load_verify_locations(tls_ctx, root_store_filename_redhat, NULL) != 0) {
    fprintf(stderr, "SSL_CTX_LOAD_VERIFY_locations failed\n");
    exit(EXIT_FAILURE);
}

tls = SSL_new(tls_ctx);
SSL_CTX_free(tls_ctx); // lower reference count now in case we need to early return
if (tls == NULL) {
    fprintf(stderr, "SSL_new failed\n");
    exit(EXIT_FAILURE);
}

// set server name indication for client hello
SSL_set_tlsext_host_name(tls, hostname);

// Associate socket with TLS context
SSL_set_fd(tls, sock);

if (SSL_connect(tls) != 0) {
    fprintf(stderr, "Failed to SSL connect\n");
    exit(EXIT_FAILURE);
}

cert = SSL_get_peer_certificate(tls);
if (cert == NULL) {
    fprintf(stderr, "Failed to get peer certificate\n");
    exit(EXIT_FAILURE);
}

if (validate_hostname(hostname, cert) != MatchFound) {
    fprintf(stderr, "Failed to validate hostname in certificate\n");
    exit(EXIT_FAILURE);
}

return tls;
}

int connect_to_host(char *host, char *service, int protocol) {
    int sock;
    int ret;
}

```

```

Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: IP_JAR=2018-03-01-19; expires=Sat, 31-Mar-2018 19:42:03 GMT; path=/; domain=google.com
Set-Cookie: NID=124=afErBe|YBKBIA001-KPkeZ-4WIdrgZIVQV4rELIAQrhhhsZg3EXfYI7IAF0QMMW7LfL3-TW8W3WgJds; expires=Fri, 31-Aug-2018 19:42:03 GMT; path=/; domain=google.com; HttpOnly
Alt-Svc: h2="443"; ma=2592000; quic="51303330; quic=51303338; quic=51303337; quic=51303335; quic=443"; ma=2592000; v="41.39.38.37.35"
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

7271
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta c
content="Search the world's information, including webpages, images, videos and more. Google ha
many special features to help you find exactly what you're looking for." name="description">
<meta content="noop" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Conte
nt-Type"><meta content="/images/branding/google/1x/google_standard_color_128dp.png" itemprop
="image"><title>Google</title><script nonce="4QW9Z2Wj1k47Y8Hk4bQw=">(function()mark@localhost
simplyssl) {
    // SSL get peer certificate(tls);
    cert = SSL_get_peer_certificate(tls);
    if (cert == NULL) {
        fprintf(stderr, "Failed to get peer certificate\n");
        exit(EXIT_FAILURE);
    }

    if (validate_hostname(hostname, cert) != MatchFound) {
        fprintf(stderr, "Failed to validate hostname in certificate\n");
        exit(EXIT_FAILURE);
    }

    return tls;
}

int connect_to_host(char *host, char *service, int protocol) {
    int sock;
    int ret;
    struct addrinfo hints;
    struct addrinfo *addr_ptr;
    struct addrinfo *addr_list;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC; // IP or IPV6 we don't care
    ret = getaddrinfo(host, service, &hints, &addr_list);
    if (ret != 0) {
        fprintf(stderr, "Failed to getaddrinfo\n");
        exit(EXIT_FAILURE);
    }

    for (addr_ptr = addr_list; addr_ptr != NULL; addr_ptr = addr_ptr->ai_next) {
        sock = socket(addr_ptr->ai_family, addr_ptr->ai_socktype, addr_ptr->ai_protocol);
        if (sock == -1) {
            perror("socket");
            continue;
        }
        if (connect(sock, addr_ptr->ai_addr, addr_ptr->ai_addrlen) == -1) {
            perror("connect");
            close(sock);
            continue;
        }
        break;
    }

    fprintf(stderr, "Failed to find a suitable address for connection\n");
    exit(EXIT_FAILURE);
}

return sock;
}

```

can we do better?



can we use the POSIX socket API?

can we use the POSIX socket API?

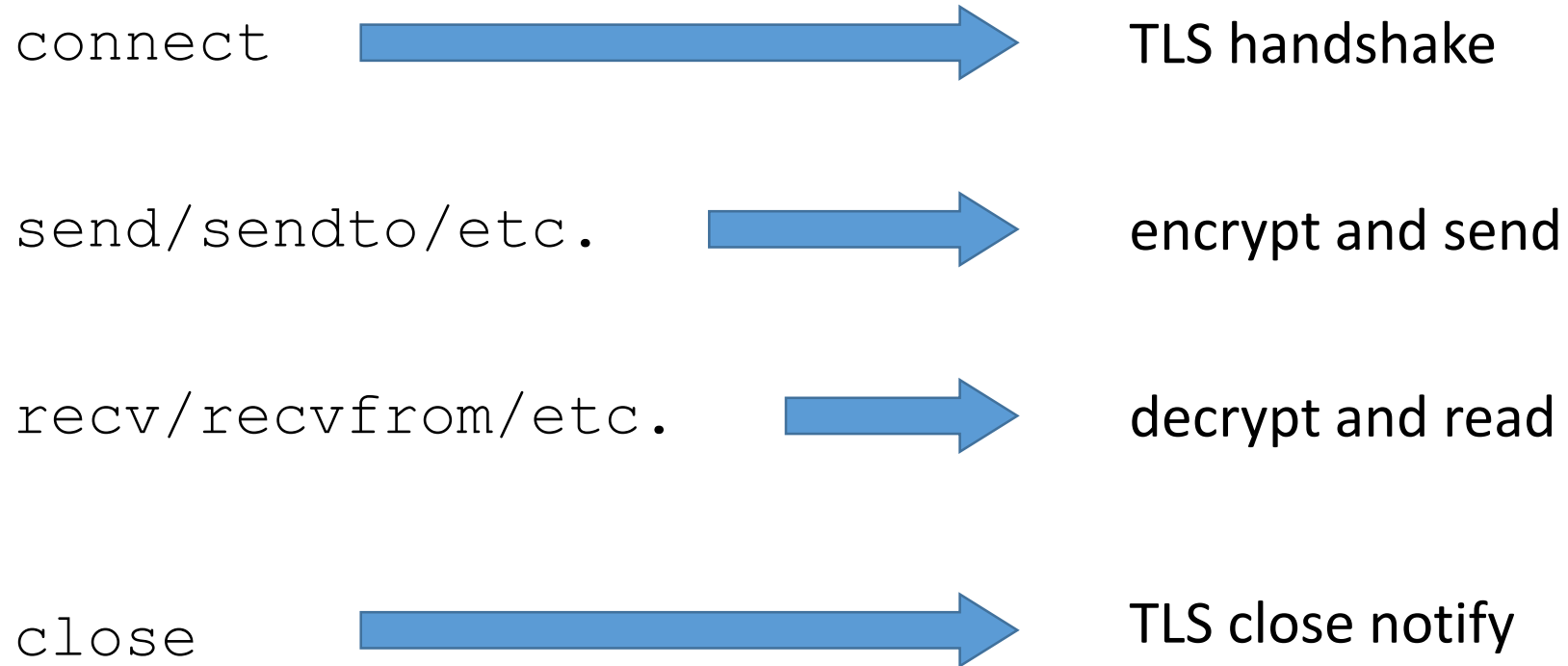
```
int socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

can we use the POSIX socket API?

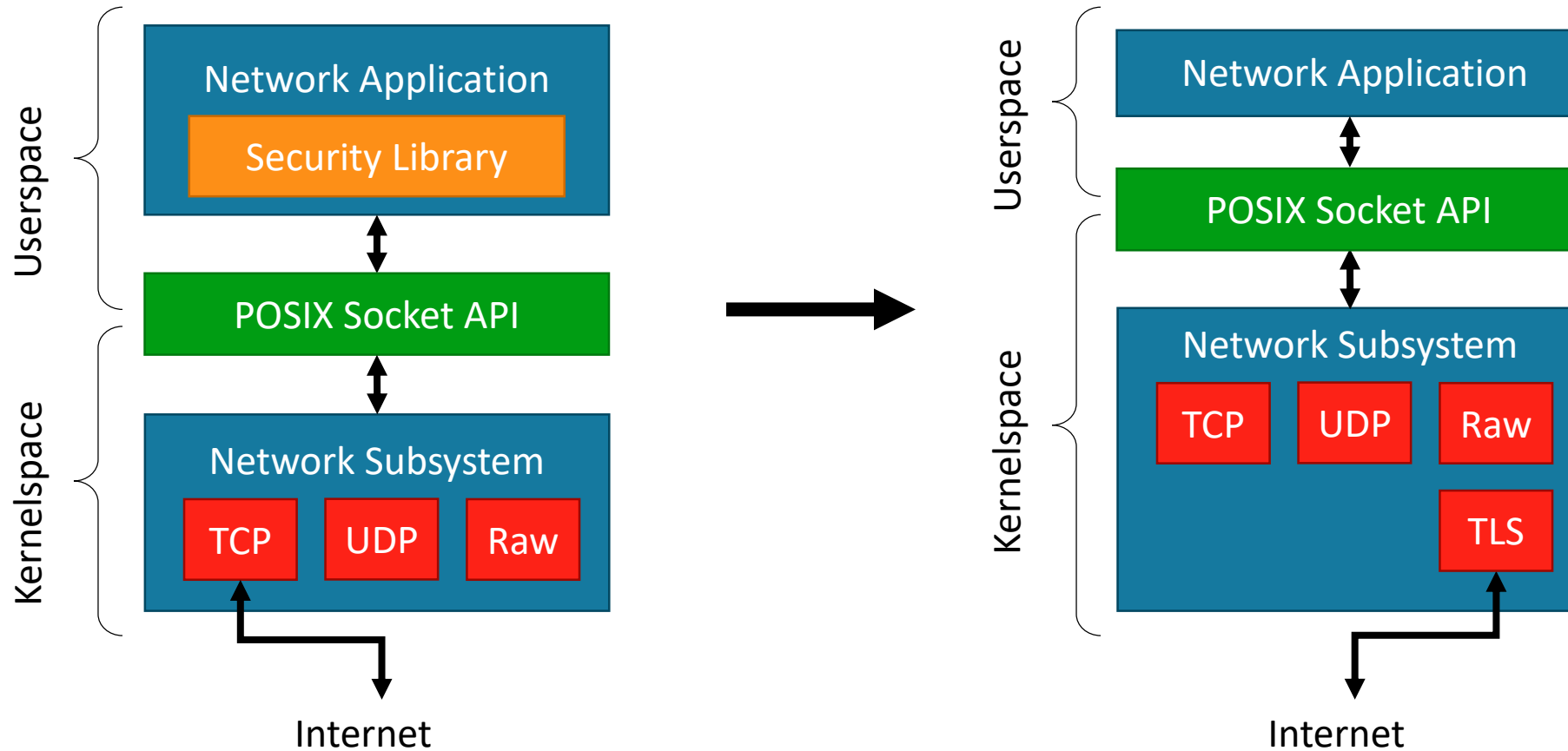
```
int socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TLS);
```

the Secure Socket API (SSA)

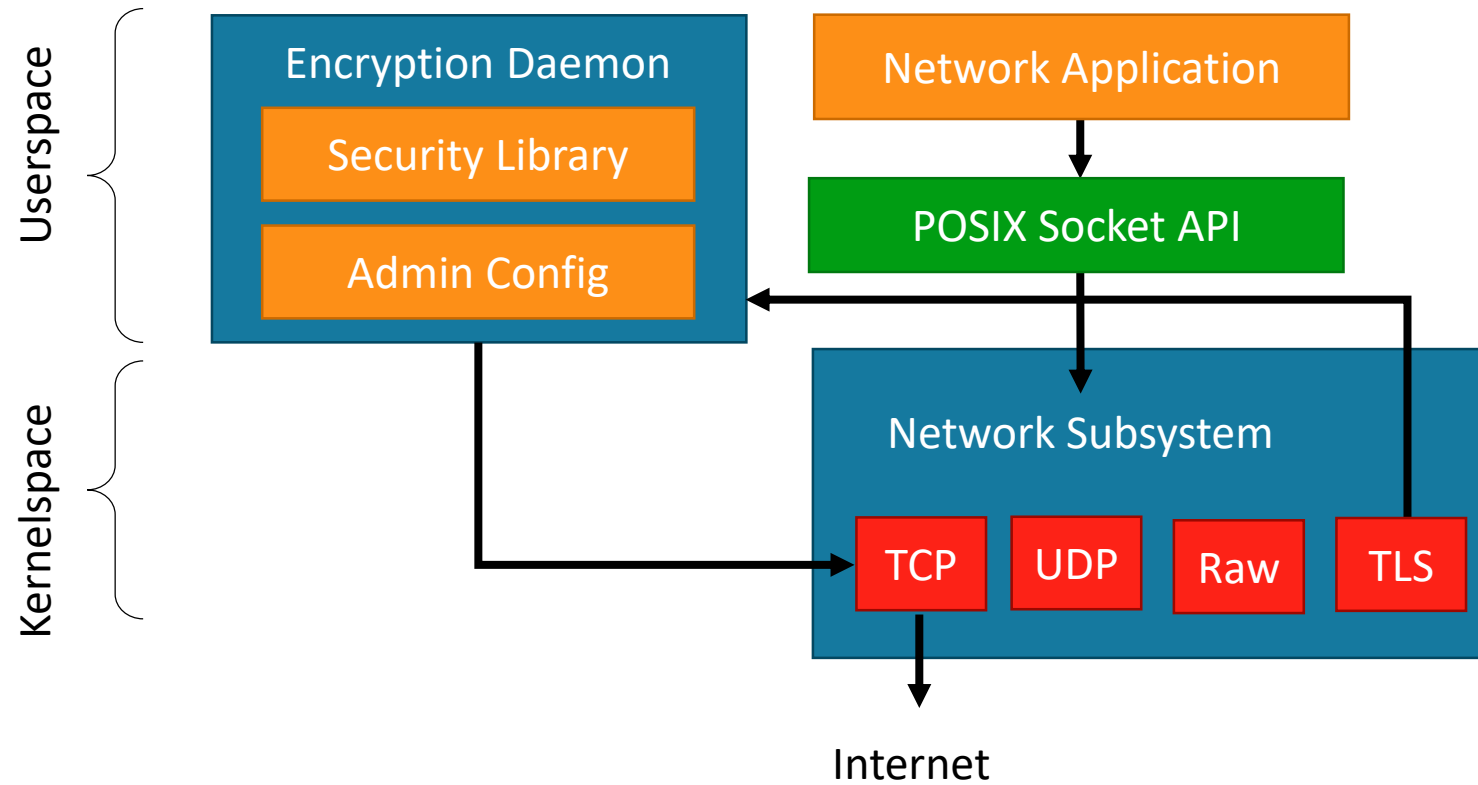
# the Secure Socket API (SSA)



# TLS via the POSIX socket API



# Userspace Encryption Daemon



# TLS API reduction

OpenSSL
SSL_CTX_new
SSL_CTX_set_verify
SSL_new
SSL_set_fd
TLS_method
SSL_exts_set_hostname
SSL_do_handshake
SSL_set_verify_callback
SSL_get_peer_certificate
<b>And 495 more...</b>

Symbol Count  
**504** → **14**

Secure Socket API
socket
bind
listen
connect
setsockopt
getsockopt
close
recv/recvfrom/recvmmsg
send/sendto/sendmsg
getaddrinfo



```
int main() {
    struct sockaddr_host addr;
    addr.sin_family = AF_HOSTNAME;
    strcpy(addr.sin_addr.name, "www.google.com");
    addr.sin_port = htons(443);

    int sock_fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TLS);
    connect(sock_fd, (struct sockaddr*)&addr, sizeof(addr));

    char http_request[] = "GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n";
    char http_response[2048];
    memset(http_response, 0, 2048);
    send(sock_fd, http_request, sizeof(http_request)-1, 0);
    recv(sock_fd, http_response, 2047, 0);
    close(sock_fd);
    printf("Received:\n%s", http_response);
    return 0;
}
```

# reconnaissance

Features	Symbols
version selection	29
cipher suite selection	39
extension management	68
certificate/key management	73
certificate/key validation	51
session management	61
configuration	19
allocation	33
connection management	41
miscellaneous	64
instrumentation	26

analyzed 410 Ubuntu packages that depended on libssl

used developer behavior to guide our design

developer options

**setsockopt**

**getsockopt**

# developer options

```
...  
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));  
...
```

# developer options

**setsockopt**

**getsockopt**

Option
TLS_REMOTE_HOSTNAME
TLS_HOSTNAME
TLS_TRUSTED_PEER_CERTIFICATES
TLS_CERTIFICATE_CHAIN
TLS_PRIVATE_KEY
TLS_ALPN
TLS_SESSION_TTL
TLS_DISABLE_CIPHER
TLS_PEER_IDENTITY
TLS_PEER_CERTIFICATE_CHAIN

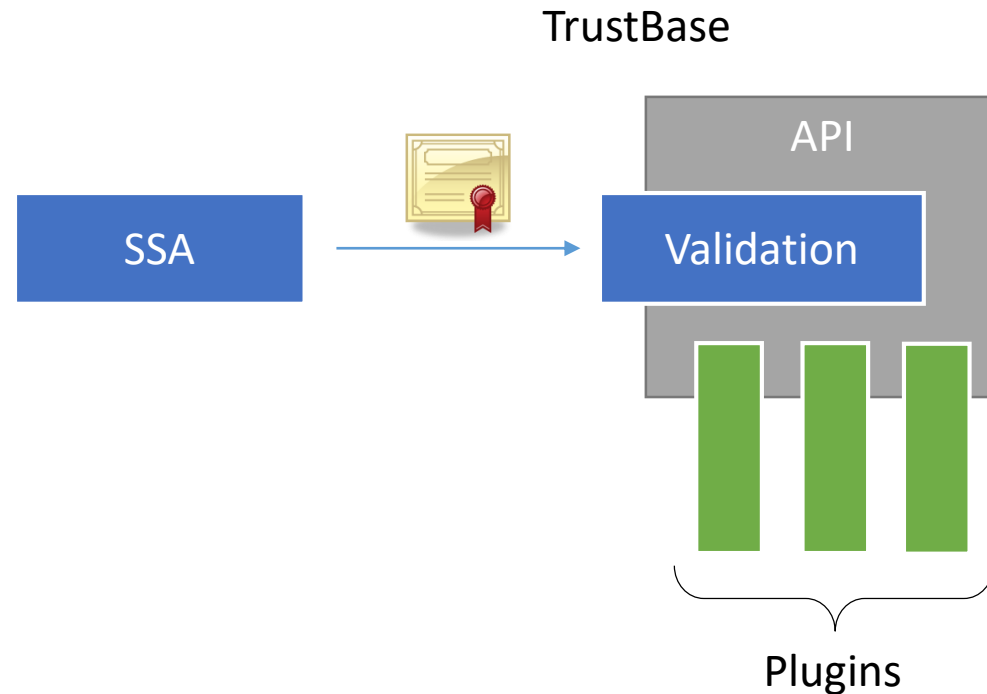
# administrator options

- global configuration file assigns TLS defaults
- per-application profiles can further customize settings

Option	Description
TLS Version	Enabled TLS versions, in order of preference
Cipher Suites	Allowed cipher suites, in order of preference
Certificate Validation	Specified root store for certificate validation, or custom validation engine like TrustBase
Enabled Extensions	Specified TLS extensions to use (e.g., ALPN)
Session Caching	Specified session cache parameters
Default cert/key paths	Specify location of certificates and keys to use when application does not specify

# certificate validation

- admin's choice
  - standard validation
  - TrustBase
- TrustBase is an OS service that validates certificates according to admin config
- can enable multiple services (OSCP, CRLsets, custom root stores, Convergence, etc.)

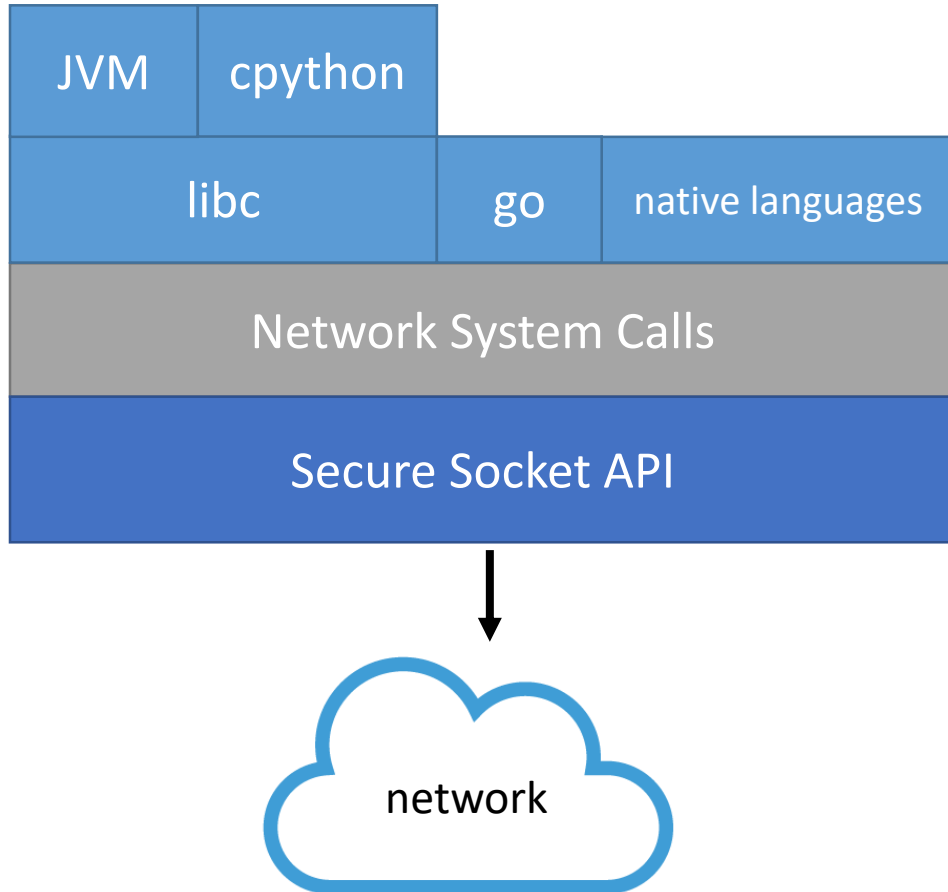


# using the SSA

Application	LOC Modified	LOC Removed	Familiar with Code?	Time Taken
<b>Already using TLS</b>				
wget	15	1,020	No	5 Hrs.
lighttpd	8	2,063	No	5 Hrs.
<b>Not using TLS</b>				
in-house webserver	5	0	Yes	5 Min.
netcat	5	0	No	10 Min.



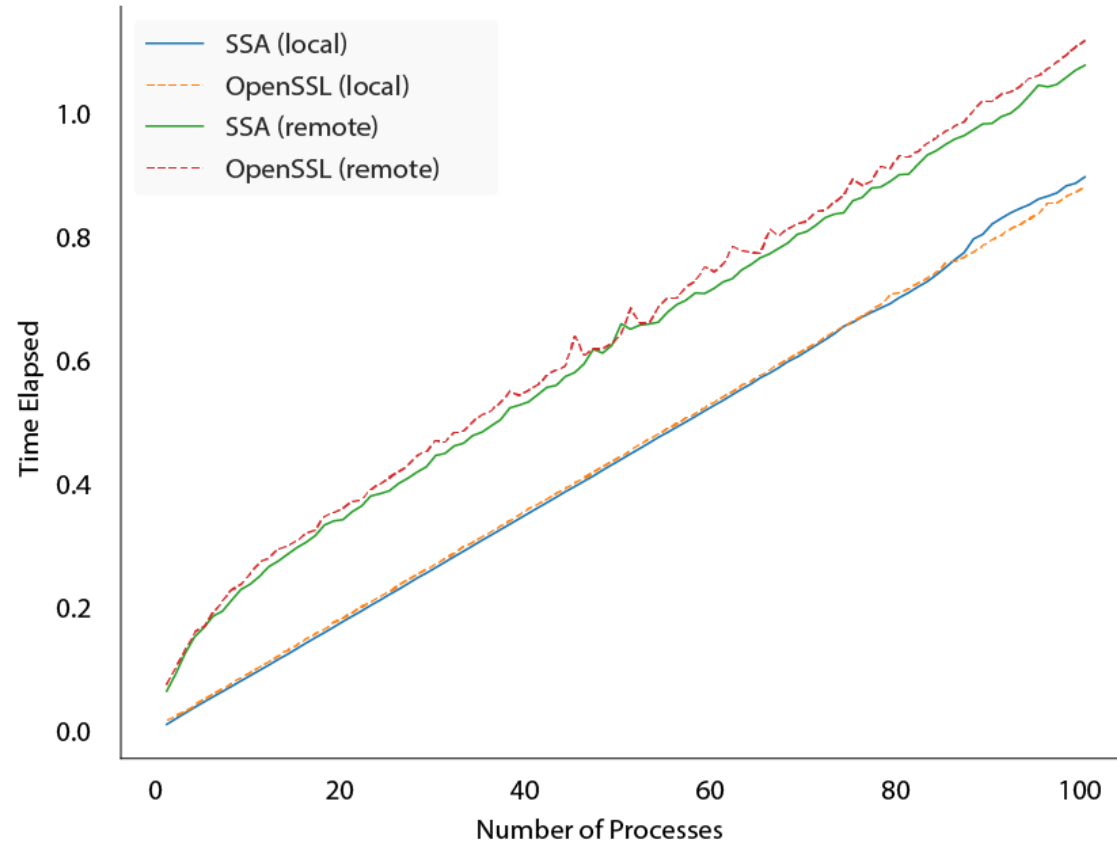
# language support



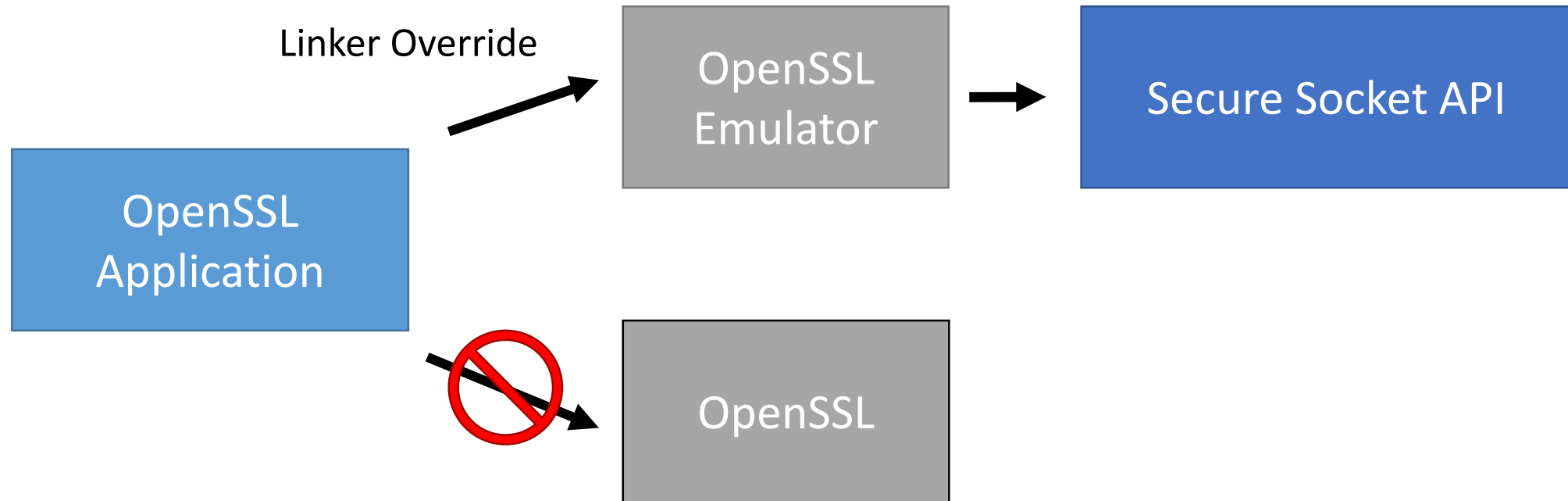
- any language that uses the network uses network system calls (directly or indirectly)
- the SSA is implemented behind the system call layer
- adding SSA support to a language is trivial
  - **Go**: < 50 lines of code (syscall wrappers)
  - **Python**: new constants only
  - **PHP**: new constants only
  - **C/C++**: new constants only

# performance vs OpenSSL

- no discernable time overhead for 0 – 100 concurrent TLS-using processes



# broadening coverage



dynamically ported ncat, wget, lighttpd, irssi

# outcomes

- general benefits
  - TLS through a known API
  - admin control of TLS settings
- implementation benefits
  - easy language support
  - natural privilege separation
  - alternative implementations supported



the Secure Socket API:  
enabling developers to **secure connections**  
using a **known API**  
in ways **you can control**

# Thank You

- kernel module: <https://github.com/markoneill/ssa>
- encryption daemon: <https://github.com/markoneill/ssa-daemon>
- pull requests welcome!
- project website: <https://owntrust.org>
- contact me: [mto@byu.edu](mailto:mto@byu.edu)

- thanks to our sponsors



**Homeland  
Security**

# Image Attributions

- Productivity by Gregor Cresnar from the Noun Project
- confused by Gregor Cresnar from the Noun Project