

TLS as an Operating System Service

Daniel Zappala

from Mark O'Neill's PhD Dissertation and USENIX Security papers

Part 1

TrustBase

Simplified and Centralized Certificate Validation



certificate validation
problems

certificate authorities (CAs)

- generally can sign certificates for any host (Eckersley et al.)
- have been hacked, sometimes repeatedly (Marlinspike)
- can be influenced and operated by governments (Soghoian et al.)
- don't always follow best practices (see CNNIC)



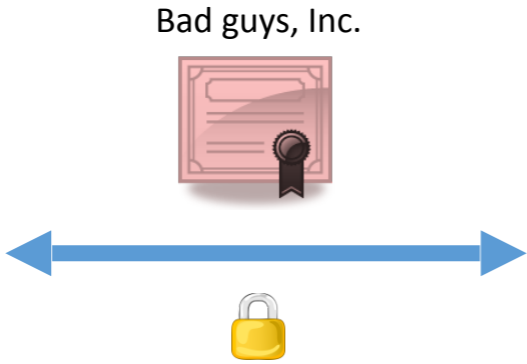
for application developers

- mobile and desktop apps have validation problems
 - Brubaker et al., Georgiev et al., Onwuzurike et al., Fahl et al.
- security libraries are complicated
- security may not be a priority
- security may be a hassle



threat model

client



amazon



alternate and reinforcing strategies

- deal with many of these issues
- have no common platform or API
- have difficulty being adopted



trust decisions are outsourced

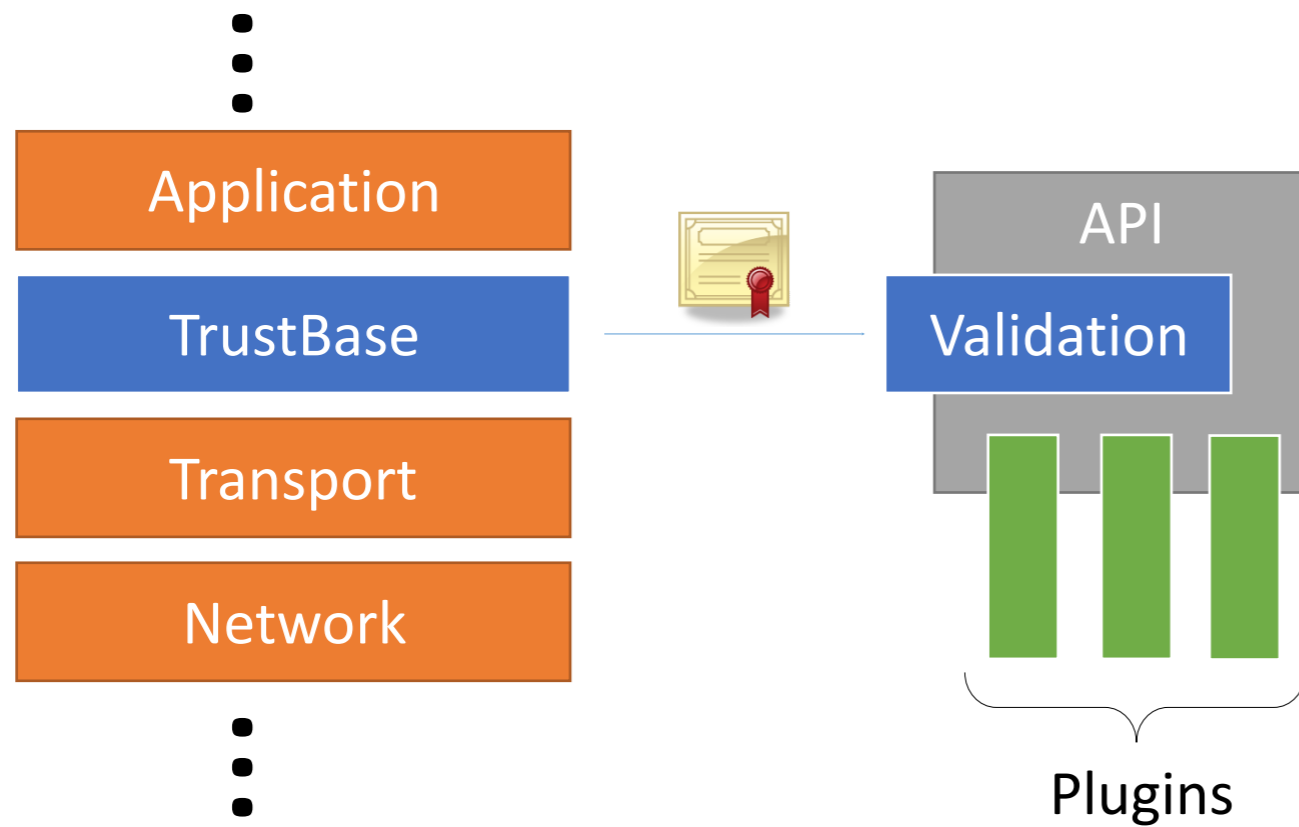
how do we enable admins to
control the trust decisions of
their own machines?

TrustBase

- motivating principles
 - centralize authentication as an OS service
 - empower system admins to dictate how trust decisions are made on their own machines
- design goals
 - secure all existing applications
 - prohibit unprivileged applications from acting against administrator rules
 - provide easy deployment of authentication systems
 - negligible overhead



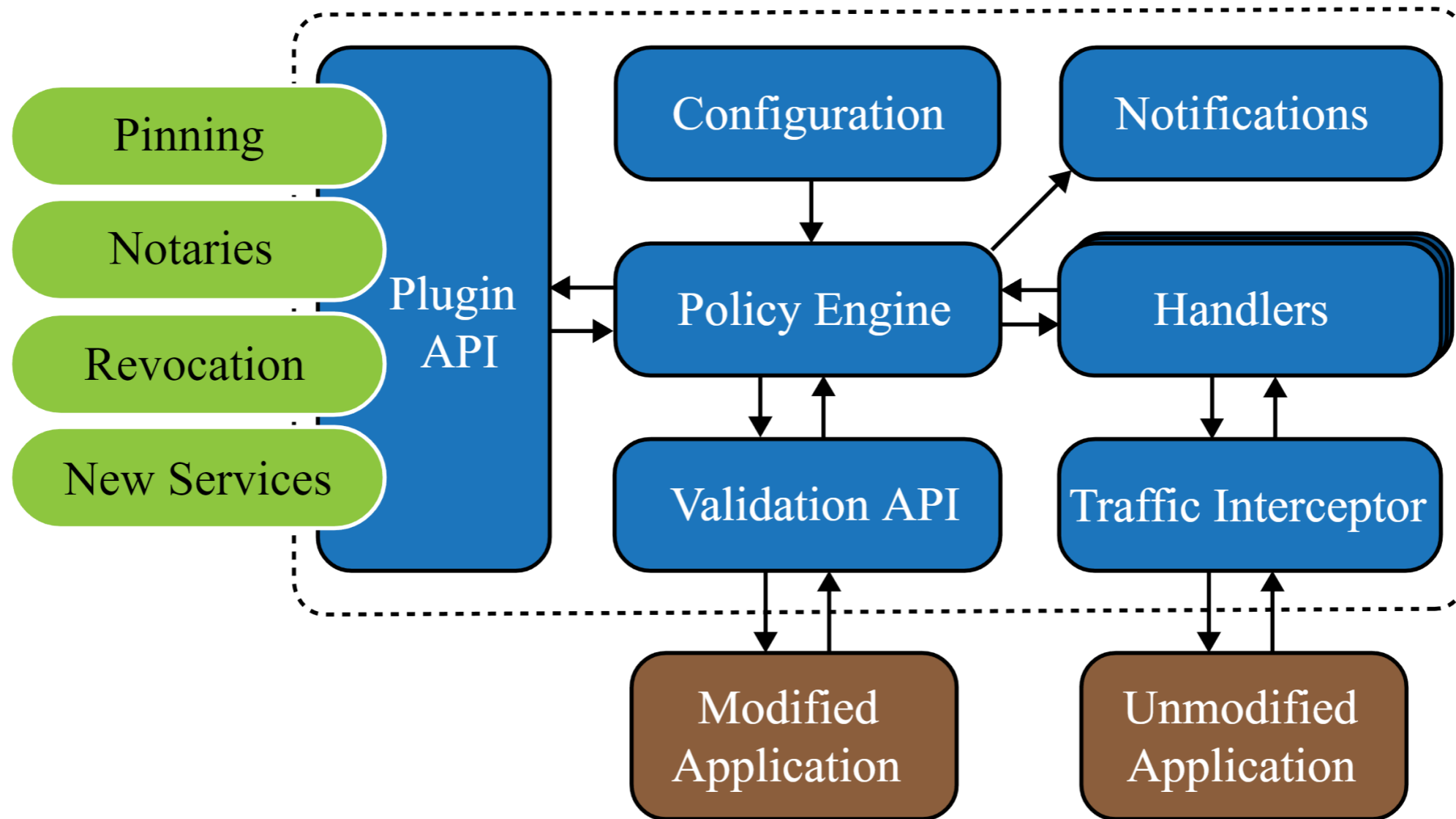
moving trust to the OS



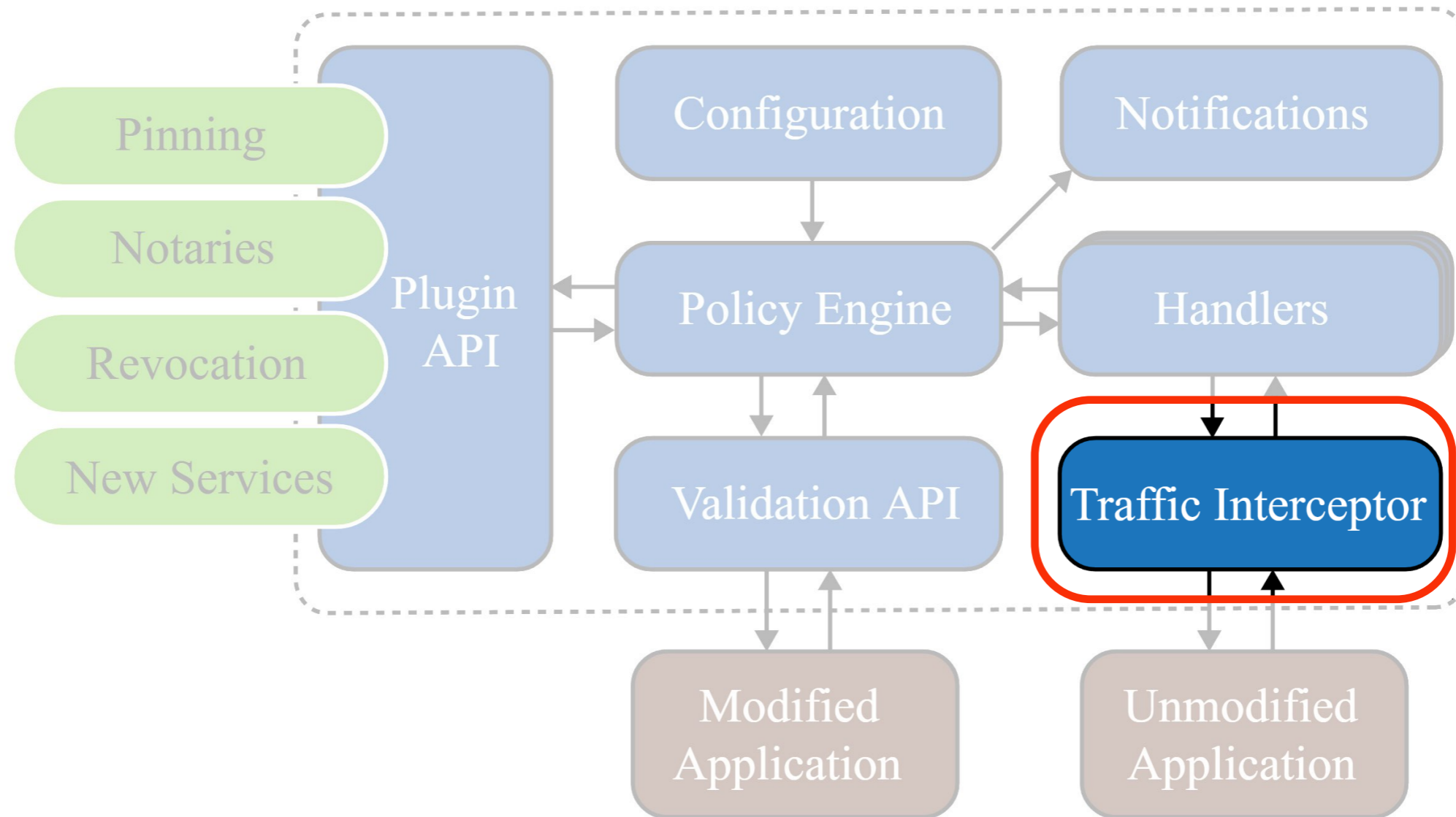
Prototypes for

- Linux
- Android (nonrooted)
- Windows

TrustBase architecture

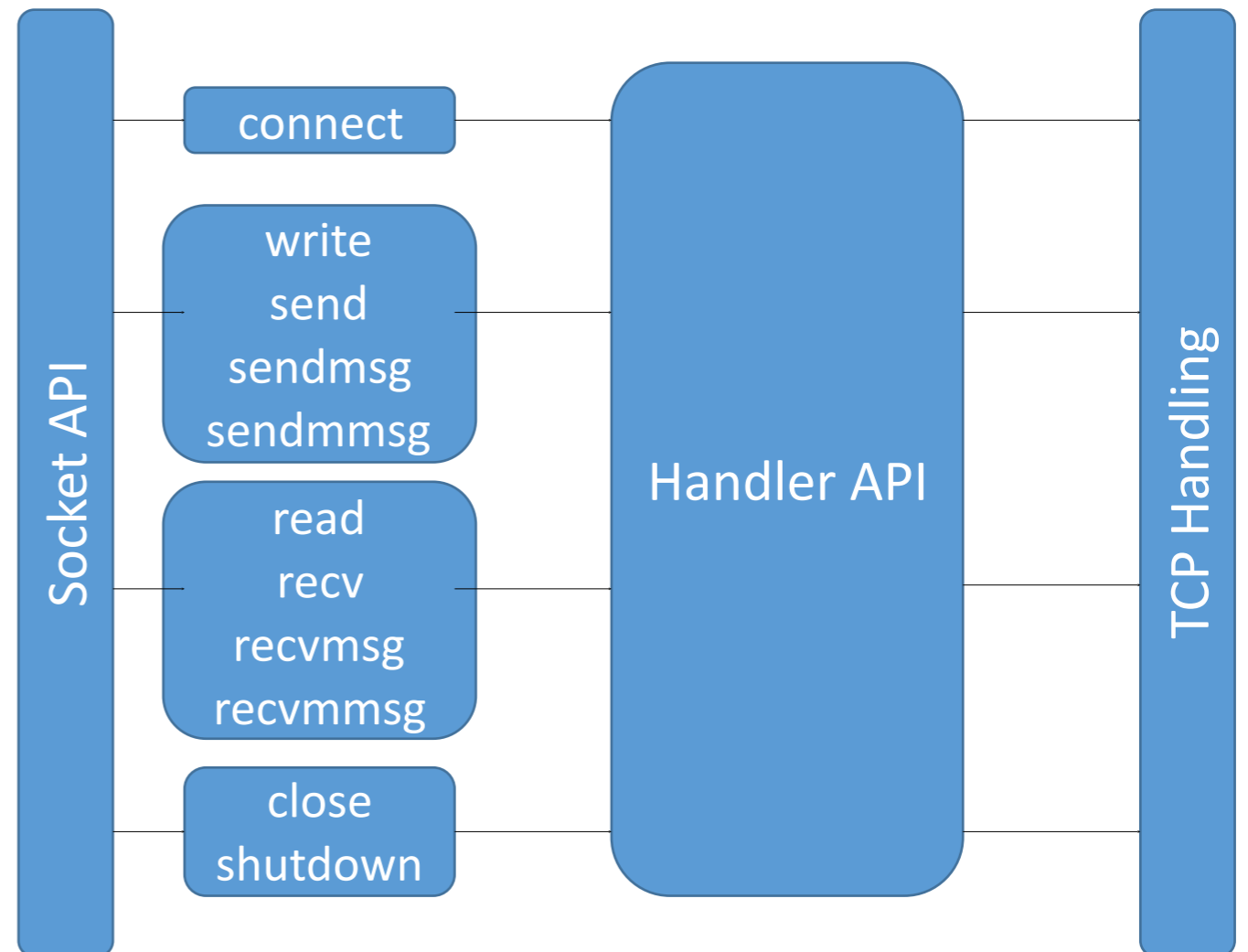


TrustBase architecture

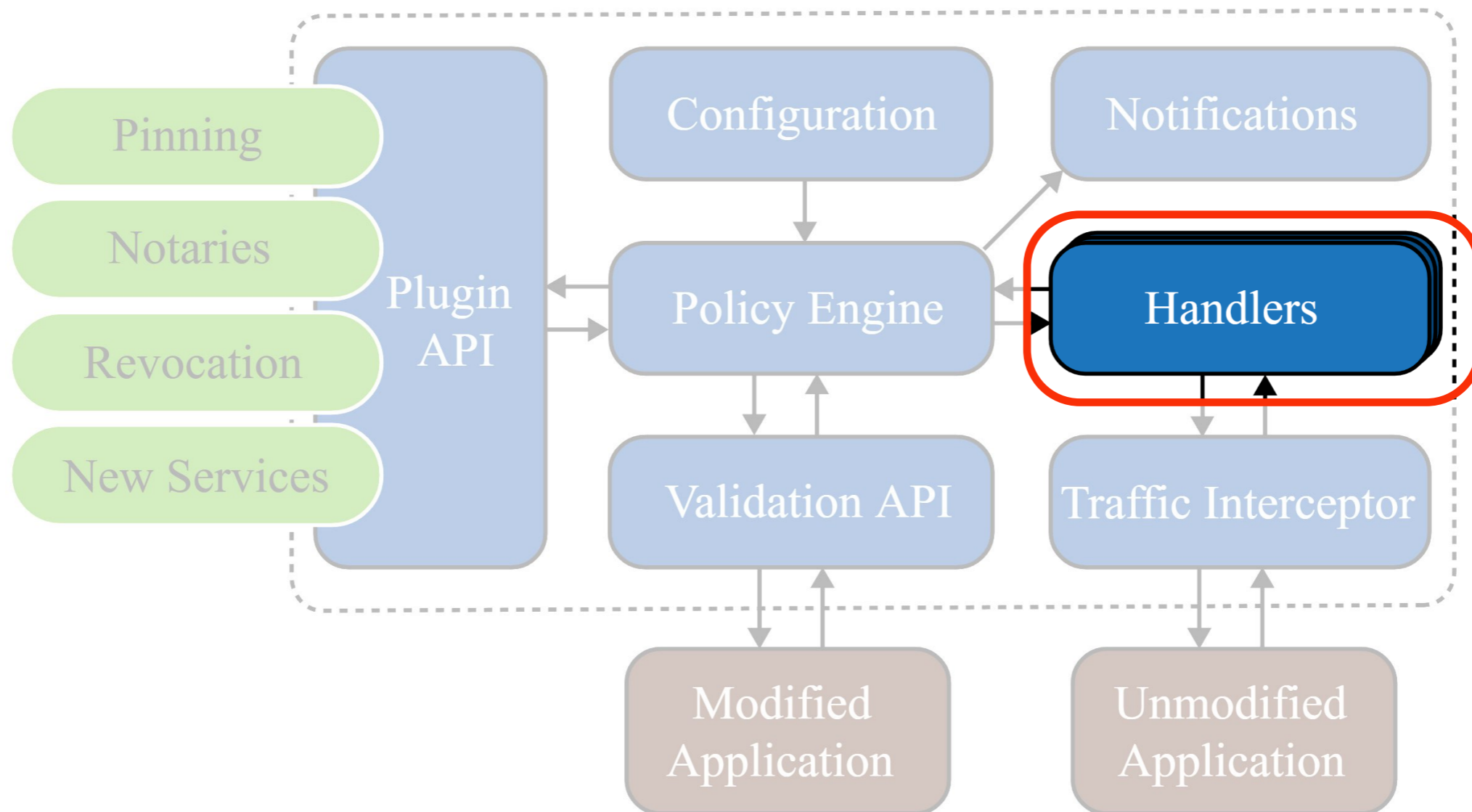


traffic interception (Linux)

- loadable kernel module
- hooks into native transport protocol functionality
- provides generic inspection/modification API



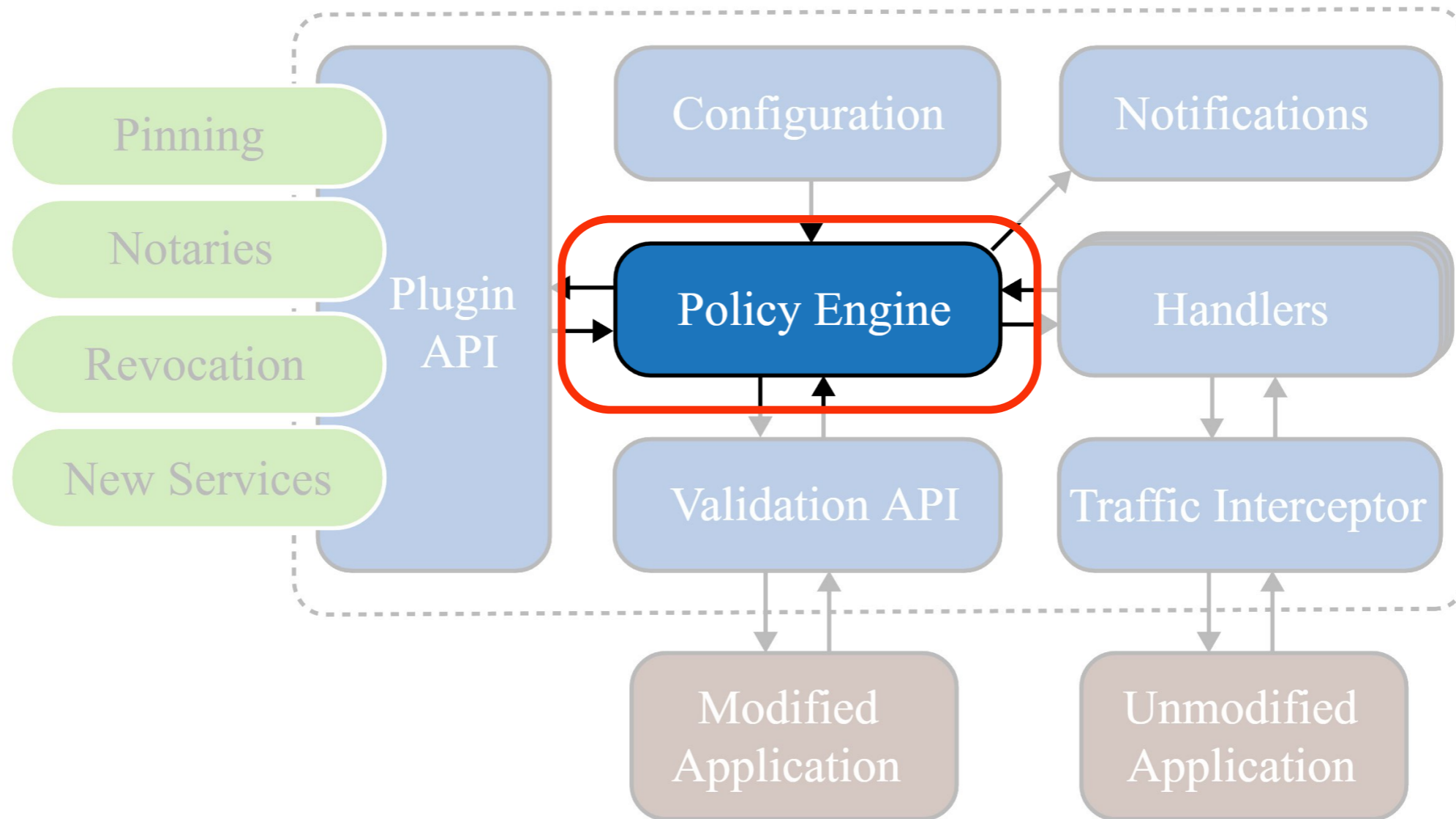
TrustBase architecture



TLS handler

1. monitor traffic for TLS records
2. record handshake messages
3. query policy engine with handshake data
4. receive policy response
 1. block connection if invalid
 2. allow if valid

TrustBase architecture

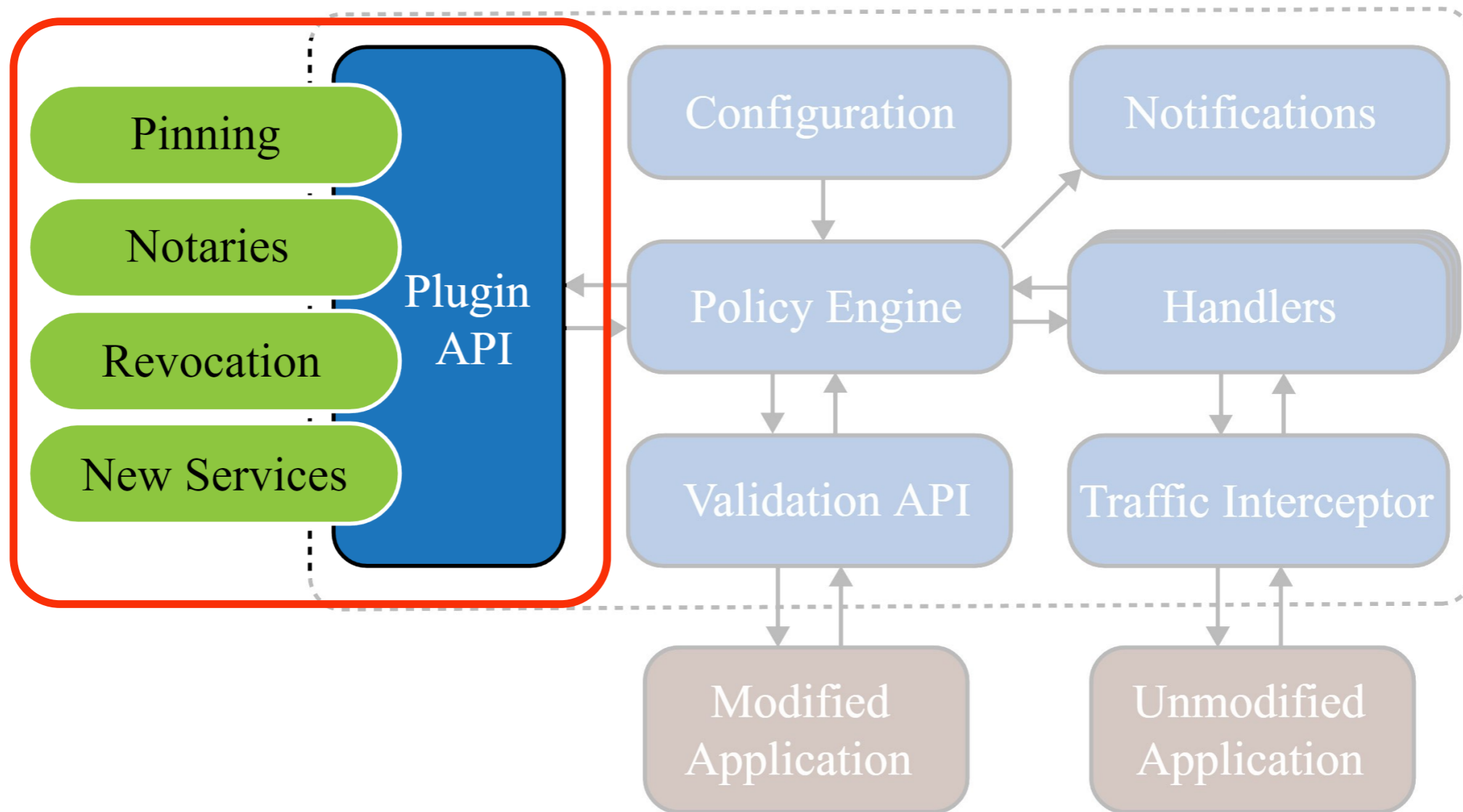


policy engine

- receives queries via Netlink
- implements basic CA validation
- aggregates decisions from plugins
 - necessary
 - voting
- provides native API
 - Linux capabilities



TrustBase architecture



plugins

- API allows synchronous and asynchronous plugins
 - openssl STACK_OF(X509) or ASN.1 DER
- can report back yes/no/abstain/error for each chain
- have access to all handshake info (and more)



addons

- provide additional language support for plugins
- currently have native C and python addons
- API to add additional language support

example plugins and uses

- CA Validation (builtin)
- Certificate Pinning
- OSCP checking
- CRLSet blocking
- DANE
- Notary
- Cipher Suite Auditor

evaluation

centralization and coverage

- bugs are global
- disruption is a DOS
- updates are global
- many eyeballs
- in line with other services

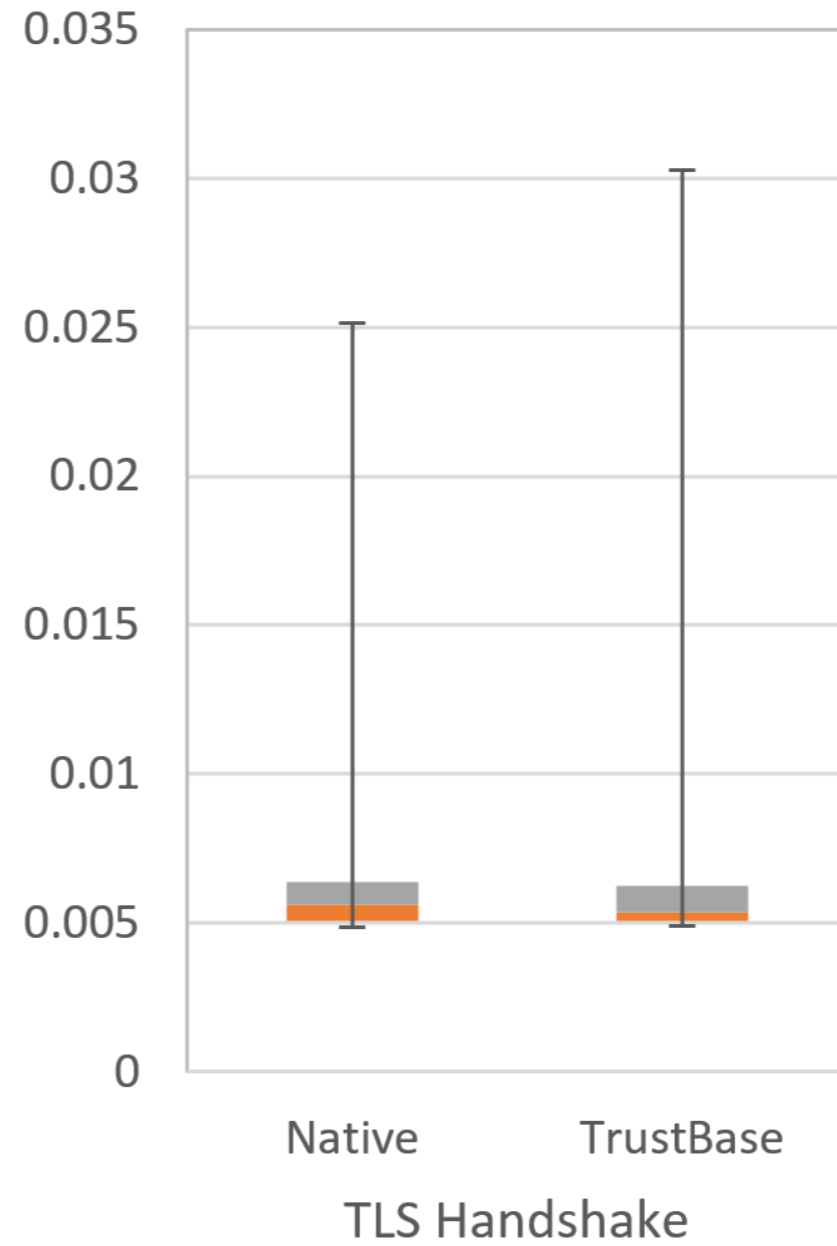
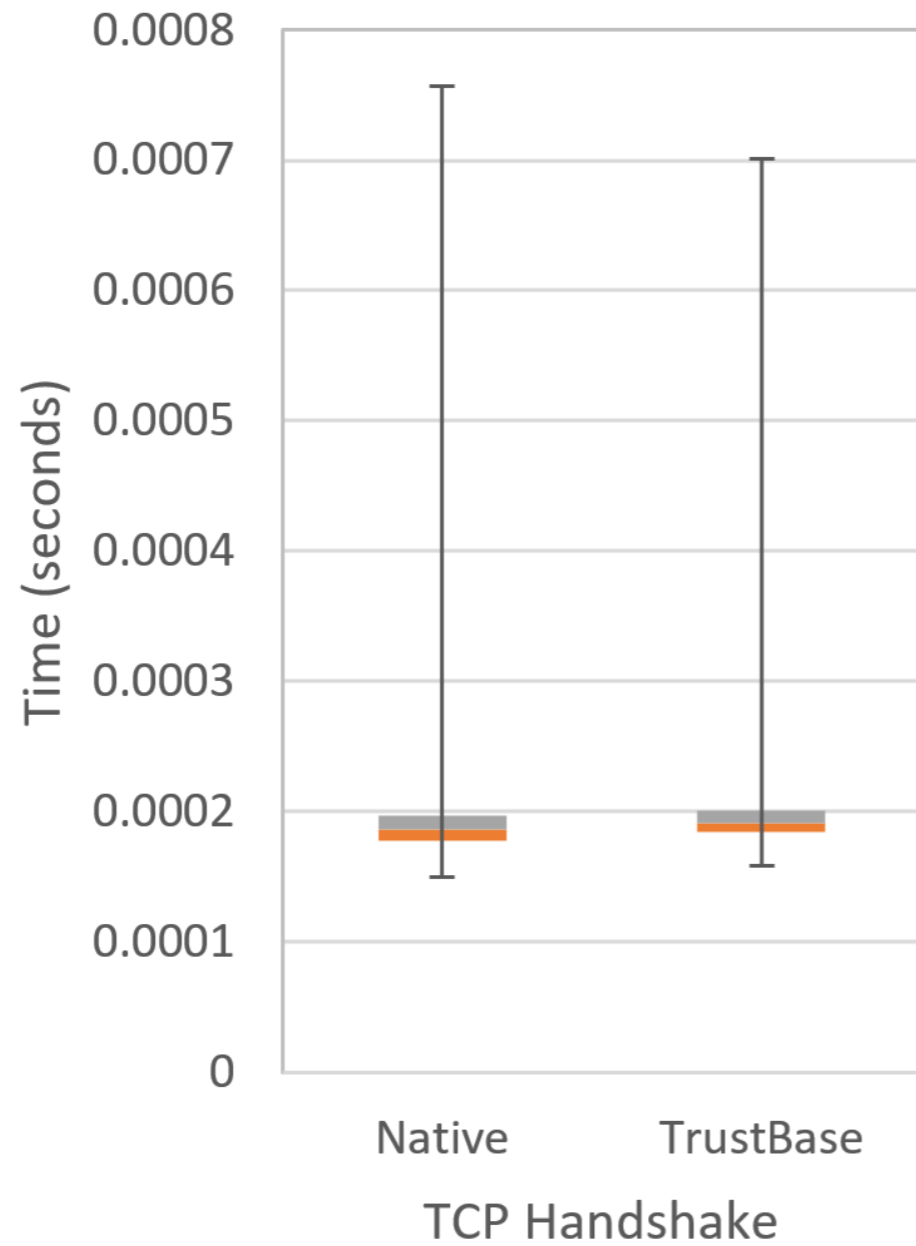
Library	Tool
C++	gnutls-cli
libcurl	curl
libgnutls	sslsca
libssl	openssl s_client
libnss	openssl s_time
JAVA	lynx
SSLConnectionFactory	fetchmail
PERL	firefox
socket::ssl	chrome/chromium
PHP	mpop
fsockopen	w3m
php_curl	ncat
PYTHON	wget
httplib	steam
httplib2	thunderbird
pycurl	kmail
pyOpenSSL	pidgin
python ssl	
urllib, urllib2, urllib3	
requests	

hardening

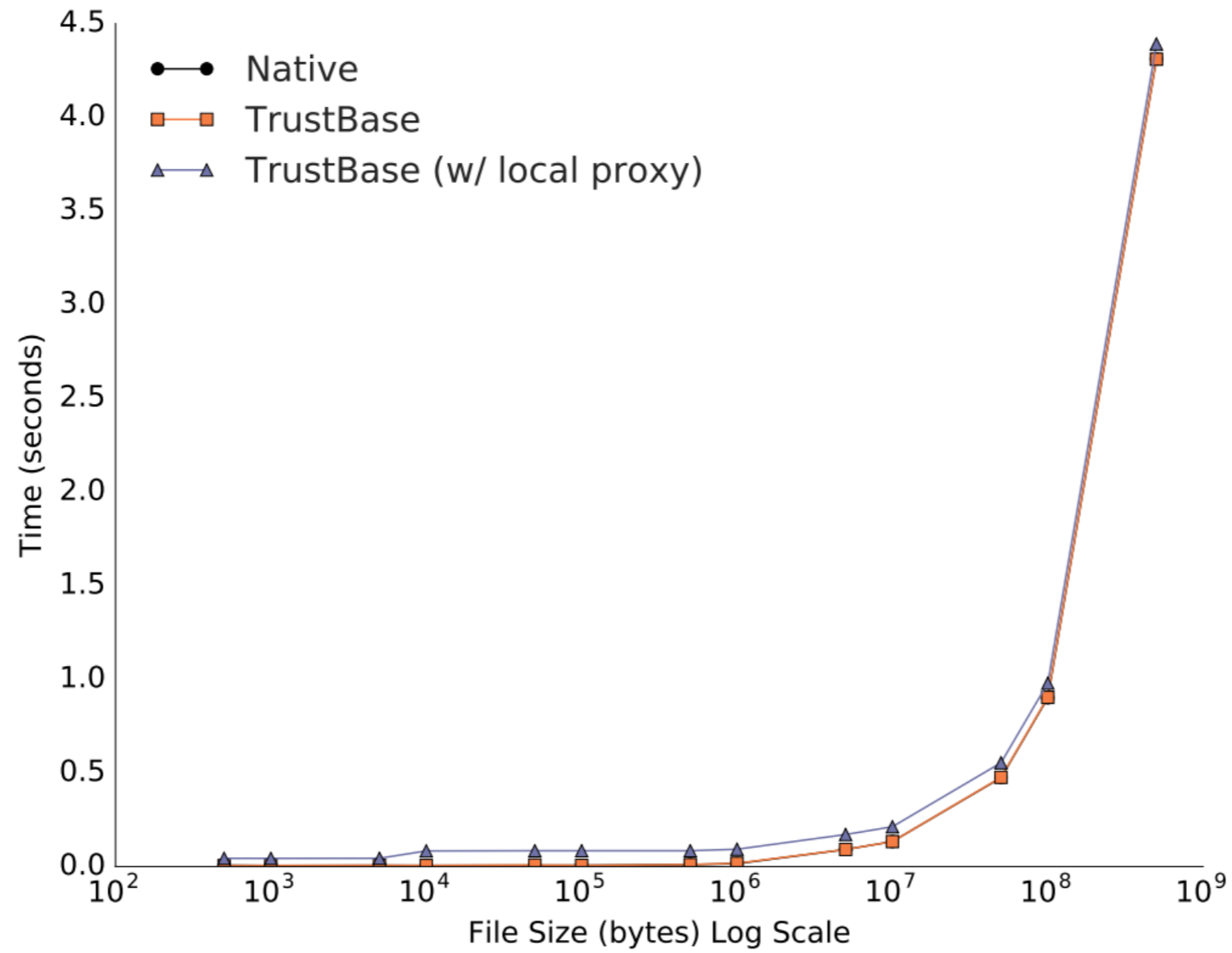


- unprivileged malware cannot unload interception
- `CAP_NET_RAW` is required to use raw sockets (default) and to bypass TrustBase interception
- `CAP_NET_ADMIN` required to receive and respond to queries
- configuration is writable only by privileged users
- daemons run nonroot with only required permissions

performance



performance



trustbase lets **you** trust
who you want
how you want

Part 2
Secure Socket API
Simplified and Centralized TLS API



your apps are vulnerable

Application Errors

We demonstrate that SSL certificate validation is completely broken in many security-critical applications and libraries. Vulnerable software includes Amazon's EC2 Java library and all cloud clients based on it; Amazon's and PayPal's merchant SDKs responsible for transmitting payment details from e-commerce sites to payment gateways; integrated shopping carts such as osCommerce, ZenCart, Ubercart, and PrestaShop; AdMob code used by mobile websites; Chase mobile banking and several other Android apps and libraries; Java Web-services middleware—including Apache Axis, Axis 2, Codehaus XFire, and Pusher library for Android—and all applications employing this middleware. Any SSL connection from any of these programs is insecure against a man-in-the-middle attack.

Georgiev, Martin, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. "The most dangerous code in the world: validating SSL certificates in non-browser software." In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 38-49. ACM, 2012.



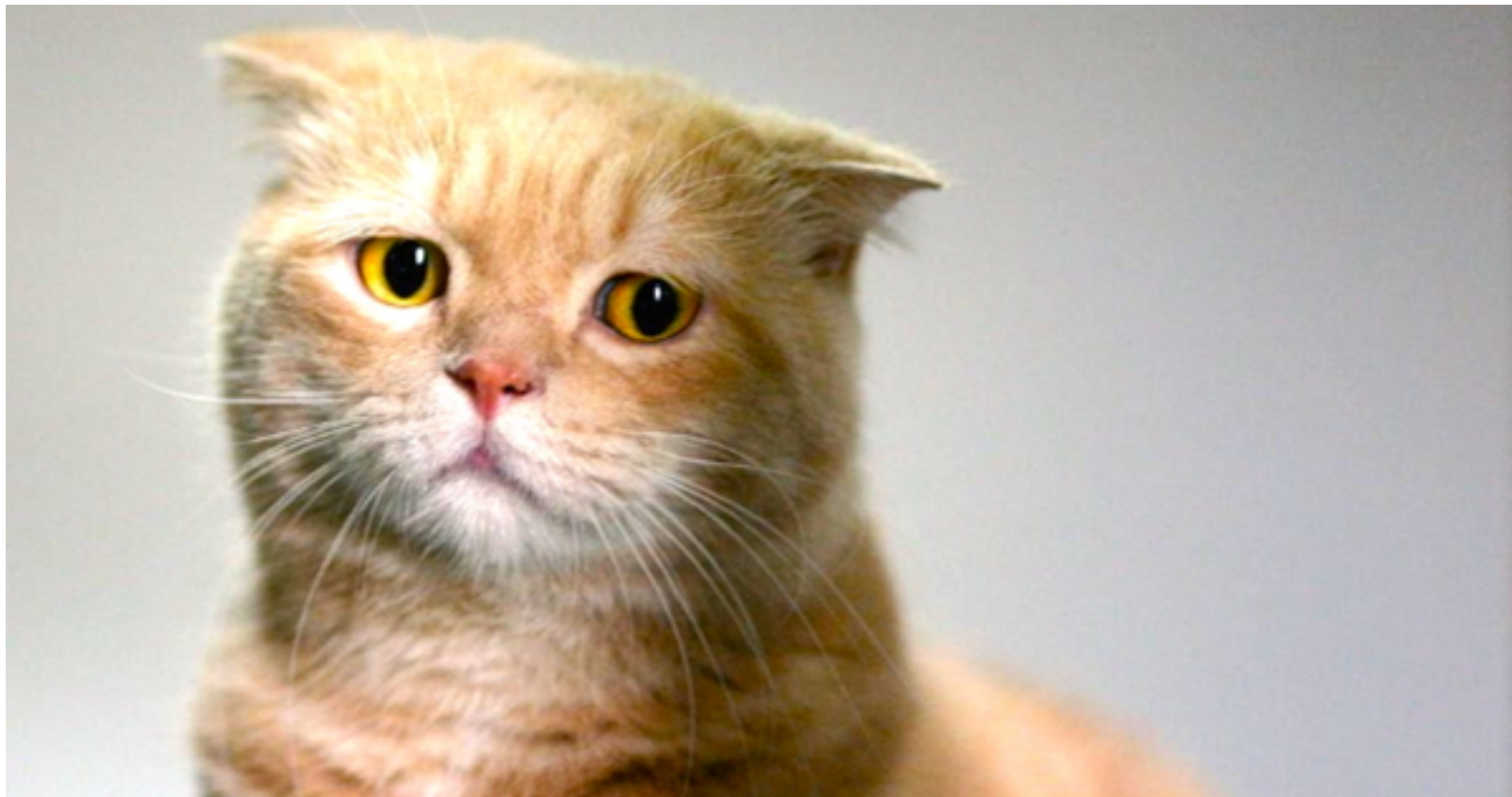
“The root cause of most of these vulnerabilities is the terrible design of the APIs to the underlying SSL libraries”

--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/x509v3.h>
#include <openssl/ssl.h>
#include <string.h>

#include "openssl_hostname_validation.h"

#define HOSTNAME_MAX_SIZE 255
#define CURL_HOST_NOMATCH 0
#define CURL_HOST_MATCH 1

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

#if (OPENSSL_VERSION_NUMBER <= 0x10100000L) || defined(LIBRESSL_VERSION_NUMBER)
#define ASN1_STRING_get0_data ASN1_STRING_data
#endif

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;
    }

    // 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_NOMATCH;

    /* We require at least 2 dots in pattern to avoid too wide wildcard
     * match. */
    wildcard_enabled = 1;
    pattern_label_end = strchr(pattern, '.');
    if (pattern_label_end == NULL || strchr(pattern_label_end, '*') == NULL ||
        pattern_wildcard > pattern_label_end ||
        Curl_raw_equal(pattern, ".*", 0) ||
        wildcard_enabled == 0;

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

    hostname_label_end = strchr(hostname, '.');
    if (hostname_label_end == NULL ||
        !Curl_raw_equal(pattern_label_end, hostname_label_end, 0))
        return CURL_HOST_NOMATCH;

    if (hostname_label_end - hostname < pattern_label_end - pattern_wildcard)
        return CURL_HOST_NOMATCH;

    prefixlen = pattern_wildcard - pattern;
    suffixlen = pattern_label_end - (pattern_wildcard + 1);
    return Curl_raw_equal(pattern_wildcard + 1, hostname_label_end - suffixlen,
        prefixlen) ?
            CURL_HOST_MATCH : CURL_HOST_NOMATCH;
}

int Curl_cert_hostcheck(const char *match_pattern, const char *hostname) {
    if (!match_pattern || !match_pattern[0] || !hostname || !hostname[0])
        return 0;

    if (Curl_raw_equal(hostname, match_pattern)) /* trivial case */
        return 1;

    if (hostmatch(hostname, match_pattern) == CURL_HOST_MATCH)
        return 1;

    return 0;
}

static int validate_hostname(const char *hostname, const X509 *server_cert) {
    HostnameValidationResult result;
    result = matches_common_name(hostname, server_cert);
    if (result == MatchFound)
        return 1;
    if (result == MalformedCertificate)
        return 0;
    if (result == Error)
        return 0;
    return Curl_cert_hostcheck(hostname, server_cert);
}
```

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

/* OpenSSL includes */
#include <openssl/ssl.h>
#include <openssl/x509.h>
#include <openssl/err.h>
#include <openssl/x509v3.h>

/* OpenSSL hostname validation */
#include "openssl_hostname_validation.h"

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

int connect_to_host(char *host, char *service, int protocol) {
    int sock;
    struct sockaddr_in addr;
    struct hostent *he;
    int ret;

    he = getaddrinfo(host, service, &addr, 0, 0, 0);
    if (he == NULL)
        return -1;

    sock = socket(addr->ai_family, addr->ai_socktype, addr->ai_protocol);
    if (sock < 0)
        return -1;

    ret = connect(sock, &addr, sizeof(addr));
    if (ret < 0)
        return -1;

    return sock;
}

int main(int argc, char **argv) {
    char *host = "www.google.com";
    char *service = "http";
    int protocol = IPPROTO_TCP;

    int sock = connect_to_host(host, service, protocol);
    if (sock < 0)
        return -1;

    SSL *ssl = SSL_new(SSL_CTX);
    SSL_set_fd(ssl, sock);
    SSL_set_tlsext_host_name(ssl, host);
    SSL_connect(ssl);

    char query[256];
    sprintf(query, "GET / HTTP/1.1\r\nHost: %s\r\n\r\n", host);
    SSL_write(ssl, query, strlen(query));

    char response[256];
    int query_len = strlen(query);
    int response_len = SSL_read(ssl, response, 256 - query_len);
    printf("Received: %s\n", response);

    close(sock);
    SSL_free(ssl);
    return 0;
}
```

```
Cache-Control: private, max-age=8
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=sVZ8yFpGdcjhhF6xfJ539Uq75b0q7WuBrdNWK10hLkZak3m8iVlUgghV4n18BajscPkcZkeX0_XmdUadCVFKh0M07851-Qp2RmGpWl0zIayvq0L5p7; expires=Fri, 31-Aug-2018 19:41:59 GMT; path=/; domain=.google.com; HttpOnly
Alt-Svc: hq="443"; ma=2592000; quic=51303339; 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
s many special features to help you find exactly what you're looking for." name="description">
<meta content="mood" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Conte
nt-type"><meta content="/images/branding/google/lx/google_standard_color_128dp.png" itemprop
="image"><title>Google</title><script nonce="QWw9Z3w0j4Y78mK4bQw==">(function()mark@localhost
simplesl)
Received:
HTTP/1.1 200 OK
Date: Thu, 01 Mar 2018 19:42:03 GMT
Expires: -1
Cache-Control: private, max-age=8
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=sVZ8yFpGdcjhhF6xfJ539Uq75b0q7WuBrdNWK10hLkZak3m8iVlUgghV4n18BajscPkcZkeX0_XmdUadCVFKh0M07851-Qp2RmGpWl0zIayvq0L5p7; expires=Fri, 31-Aug-2018 19:42:03 GMT; path=/; domain=.google.com; HttpOnly
Alt-Svc: hq="443"; ma=2592000; quic=51303339; 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

7264
<!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
s many special features to help you find exactly what you're looking for." name="description">
<meta content="mood" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Conte
nt-type"><meta content="/images/branding/google/lx/google_standard_color_128dp.png" itemprop
="image"><title>Google</title><script nonce="z037nF5w9/LLWtfnRHw==">(function()mark@localhost
simplesl)
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_socktype = protocol;
    hints.ai_family = AF_UNSPEC; // IP4 or IP6, we don't care
    ret = getaddrinfo(host, service, &hints, &addr_list);
    if (ret != 0) {
        fprintf(stderr, "Failed in getaddrinfo: %s\n", gai_strerror(ret));
        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 < 0) {
            perror("socket");
            continue;
        }
        if (connect(sock, addr_ptr->ai_addr, addr_ptr->ai_addrlen) == -1) {
            perror("connect");
            close(sock);
            continue;
        }
        break;
    }
    freeaddrinfo(addr_list);
    if (addr_ptr == NULL) {
        fprintf(stderr, "Failed to find a suitable address for connection\n");
        exit(EXIT_FAILURE);
    }
    return sock;
}
```

```
static int Curl_raw_equal(const char *first, const char *second) {
    while(*first && *second) {
        if (Curl_raw_toupper(*first) != Curl_raw_toupper(*second))
            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;
        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_NOMATCH;

    /* We require at least 2 dots in pattern to avoid too wide wildcard
     * match. */
    wildcard_enabled = 1;
    pattern_label_end = strchr(pattern, '.');
    if (pattern_label_end == NULL || strchr(pattern_label_end, '*') == NULL ||
        pattern_wildcard > pattern_label_end ||
        Curl_raw_equal(pattern, ".*", 0) ||
        wildcard_enabled == 0;

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

    hostname_label_end = strchr(hostname, '.');
    if (hostname_label_end == NULL ||
        !Curl_raw_equal(pattern_label_end, hostname_label_end, 0))
        return CURL_HOST_NOMATCH;

    if (hostname_label_end - hostname < pattern_label_end - pattern_wildcard)
        return CURL_HOST_NOMATCH;

    prefixlen = pattern_wildcard - pattern;
    suffixlen = pattern_label_end - (pattern_wildcard + 1);
    return Curl_raw_equal(pattern_wildcard + 1, hostname_label_end - suffixlen,
        prefixlen) ?
            CURL_HOST_MATCH : CURL_HOST_NOMATCH;
}

static int validate_hostname(const char *hostname, const X509 *server_cert) {
    HostnameValidationResult result;
    result = matches_common_name(hostname, server_cert);
    if (result == MatchFound)
        return 1;
    if (result == MalformedCertificate)
        return 0;
    if (result == Error)
        return 0;
    return Curl_cert_hostcheck(hostname, server_cert);
}
```

```
the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of 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
 * Please read "everything you need to know about openssl" before
 * attempting to use this. This whitepaper describes the code well
 * and how it should be used.
 * Note from Mark D'Neilly: wildcard matching has been added. A revocation
 * 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
 * server's 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();
ERR_load_BIO_strings();
ERR_load_crypto_strings();
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_set_fd(tls, sock); /* to reference sock in case we need to early return */
if (SSL_connect(tls) != 1) {
    fprintf(stderr, "SSL connect failed\n");
    exit(EXIT_FAILURE);
}

/* set hostname for peer to see hello */
SSL_set_tlsext_host_name(tls, hostname);
/* Assume socket with TLS context */
SSL_set_tlsext_host_name(tls, hostname);

if (SSL_get_peer_certificate(tls) != NULL) {
    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_socktype = protocol;
    hints.ai_family = AF_UNSPEC; // IP4 or IP6, we don't care
    ret = getaddrinfo(host, service, &hints, &addr_list);
    if (ret != 0) {
        fprintf(stderr, "Failed in getaddrinfo: %s\n", gai_strerror(ret));
        exit(EXIT_FAILURE);
    }

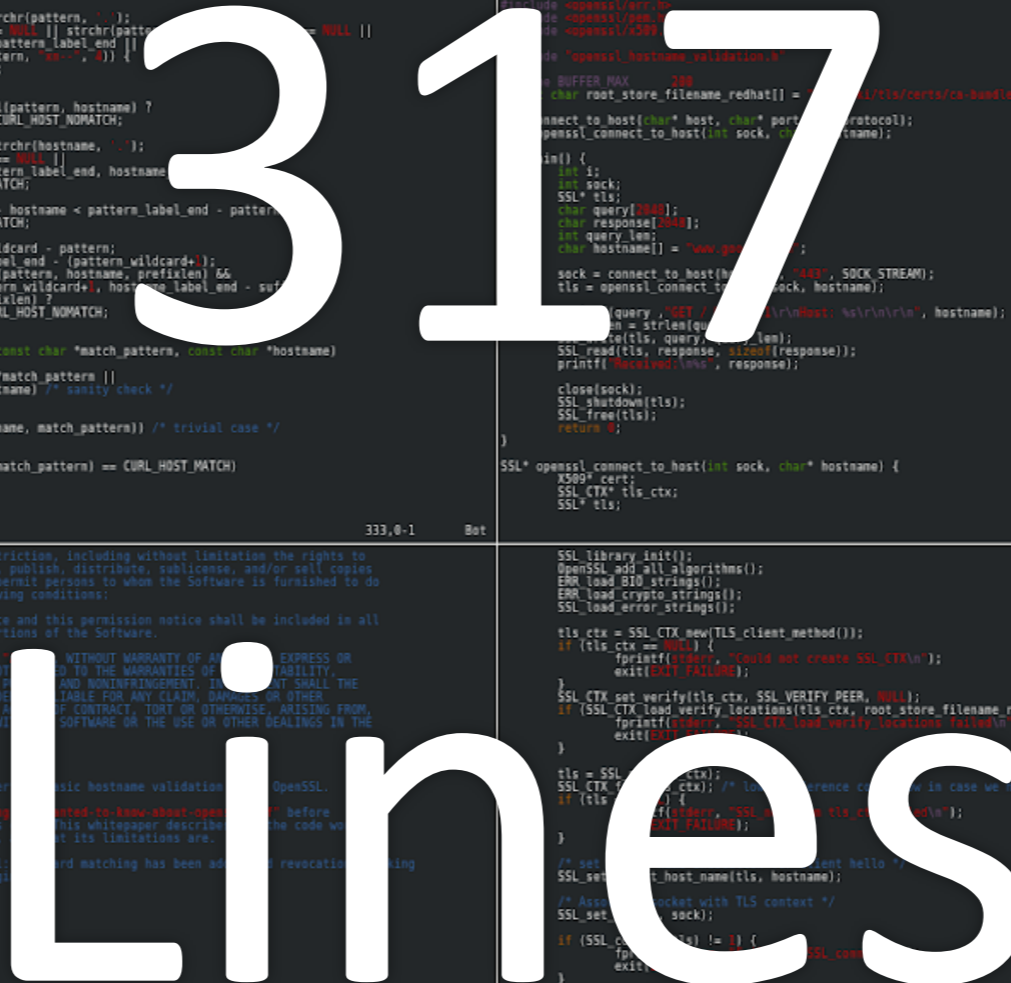
    for (addr_ptr = addr_list; addr_ptr != NULL; addr_ptr = addr_ptr->ai_next) {
        sock = socket(addr_ptr->ai_family, addr_ptr->ai_socktype, addr_ptr->ai_protocol);
        if (sock < 0) {
            perror("socket");
            continue;
        }
        if (connect(sock, addr_ptr->ai_addr, addr_ptr->ai_addrlen) == -1) {
            perror("connect");
            close(sock);
            continue;
        }
        break;
    }
    freeaddrinfo(addr_list);
    if (addr_ptr == NULL) {
        fprintf(stderr, "Failed to find a suitable address for connection\n");
        exit(EXIT_FAILURE);
    }
    return sock;
}
```

```
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_socktype = protocol;
    hints.ai_family = AF_UNSPEC; // IP4 or IP6, we don't care
    ret = getaddrinfo(host, service, &hints, &addr_list);
    if (ret != 0) {
        fprintf(stderr, "Failed in getaddrinfo: %s\n", gai_strerror(ret));
        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 < 0) {
            perror("socket");
            continue;
        }
        if (connect(sock, addr_ptr->ai_addr, addr_ptr->ai_addrlen) == -1) {
            perror("connect");
            close(sock);
            continue;
        }
        break;
    }
    freeaddrinfo(addr_list);
    if (addr_ptr == NULL) {
        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?

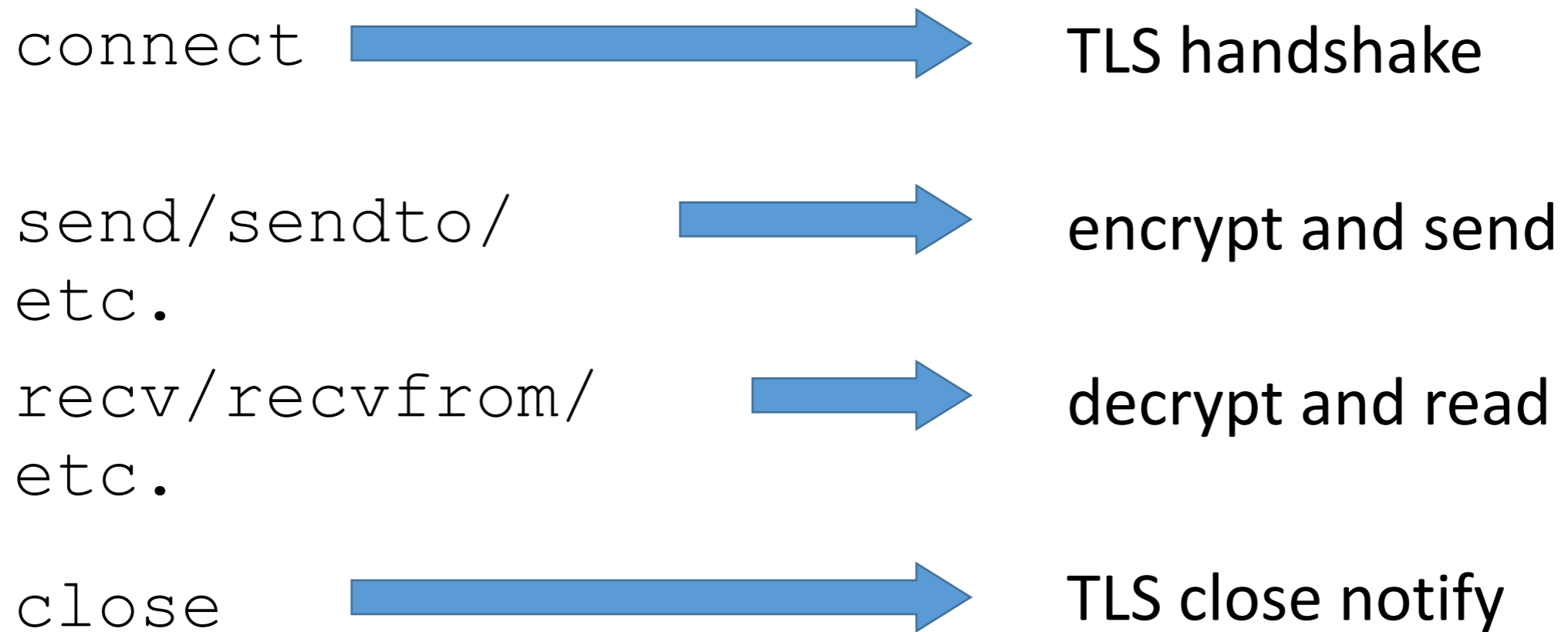
```
int socket = socket(PF_INET, SOCK_STREAM,  
                   IPPROTO_TCP);
```

can we use the POSIX socket API?

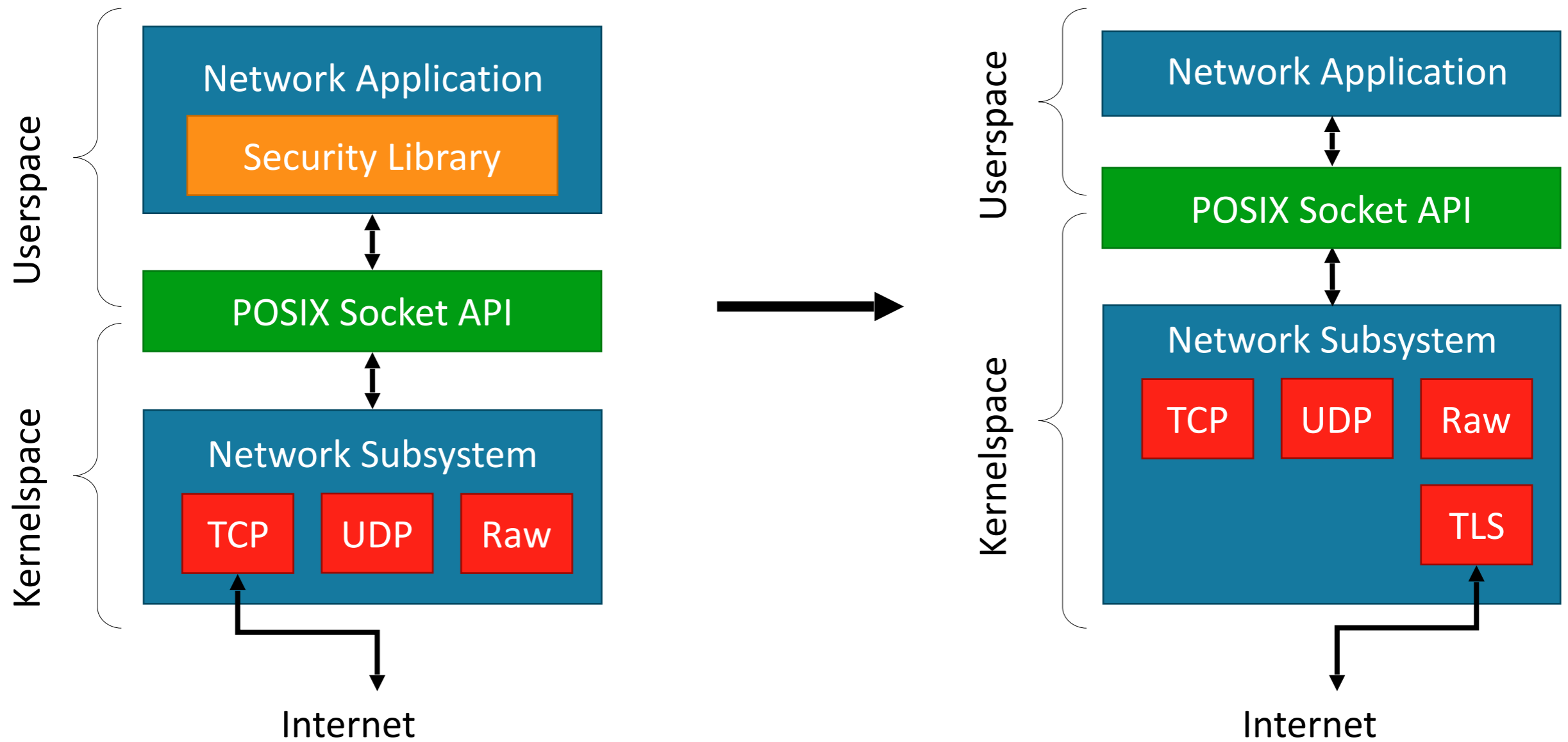
```
int socket = socket(PF_INET, SOCK_STREAM,  
                   IPPROTO_TLS);
```

the Secure Socket API (SSA)

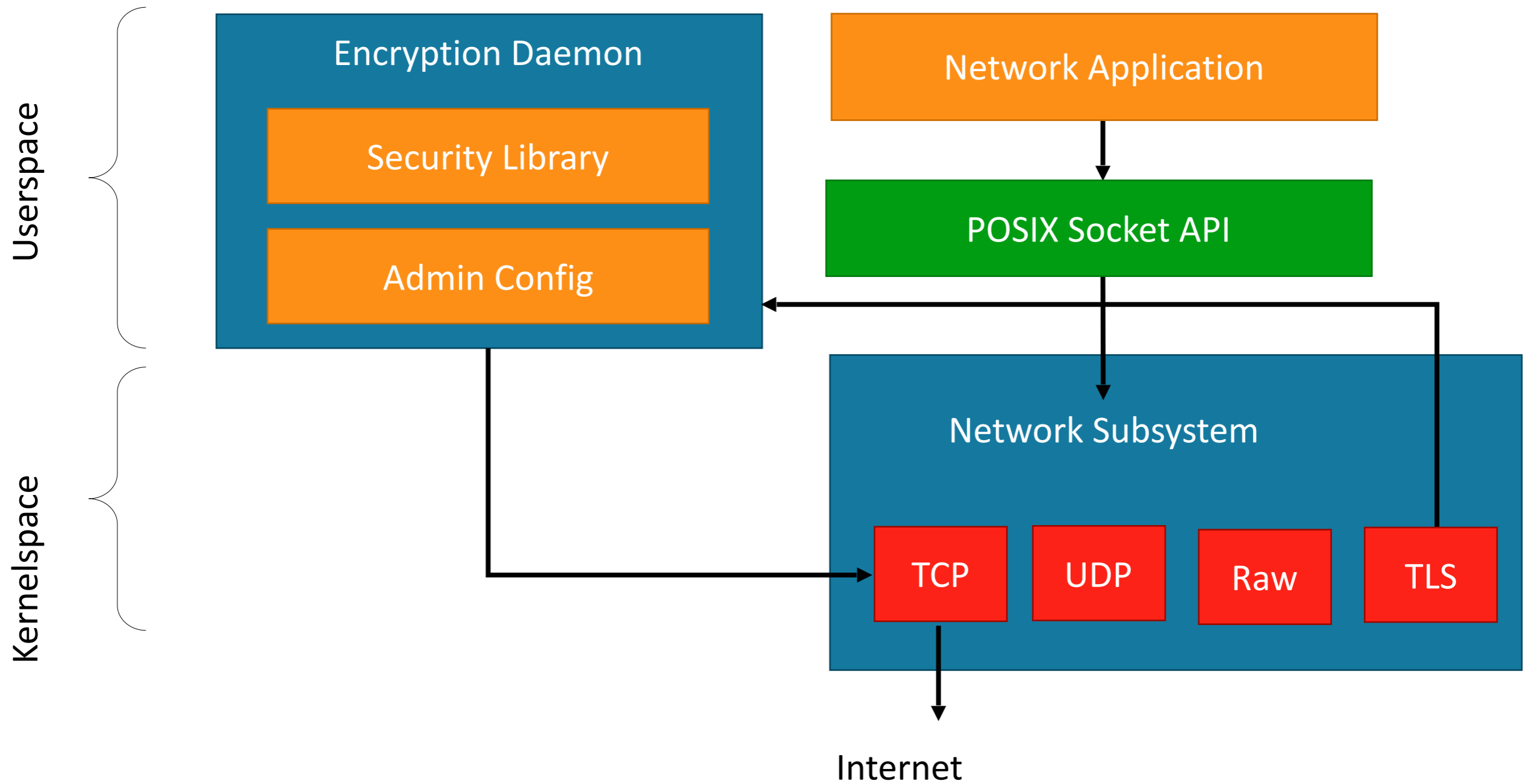
the Secure Socket API (SSA)



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

And 495 more...

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

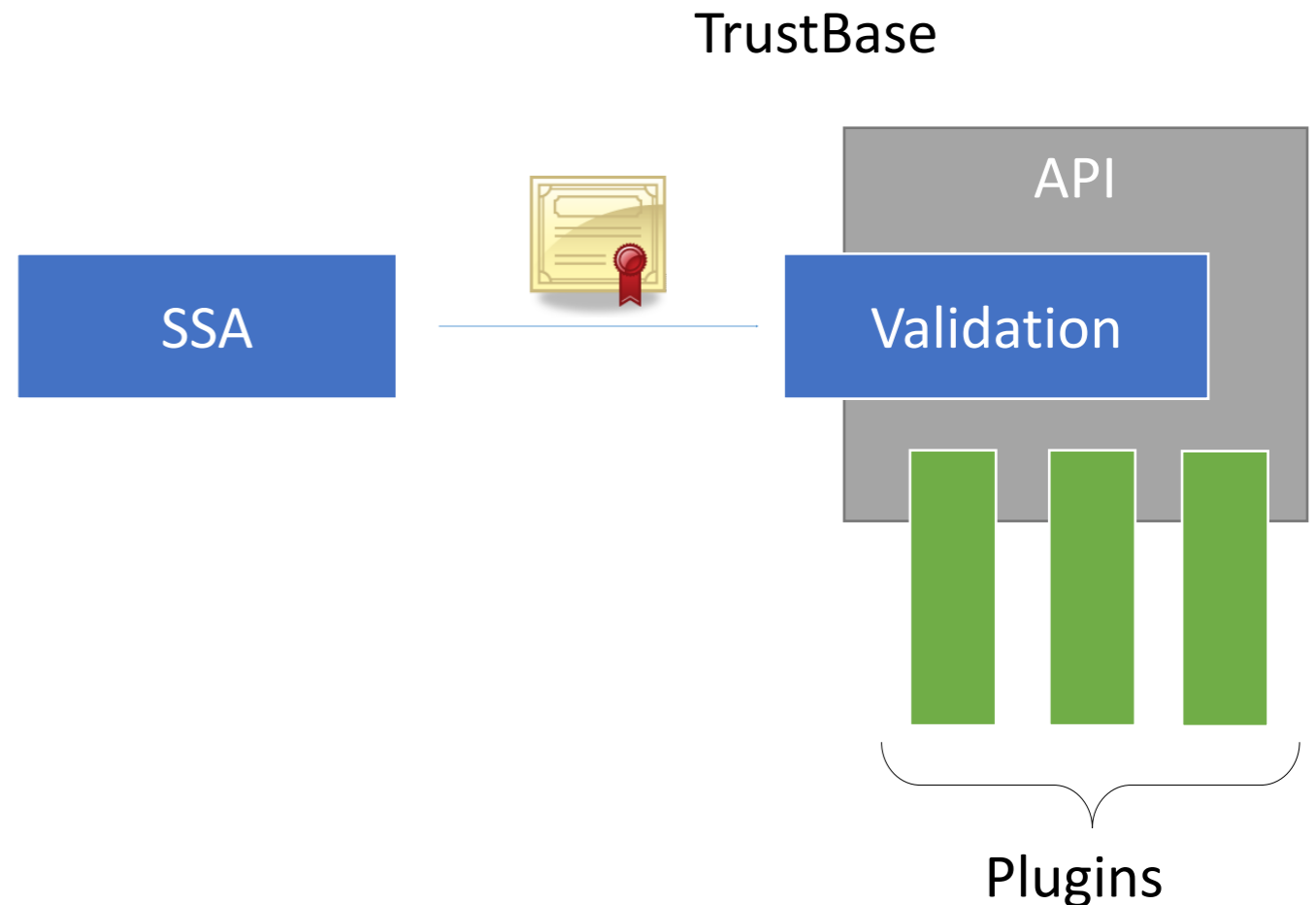
administrator options

- 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

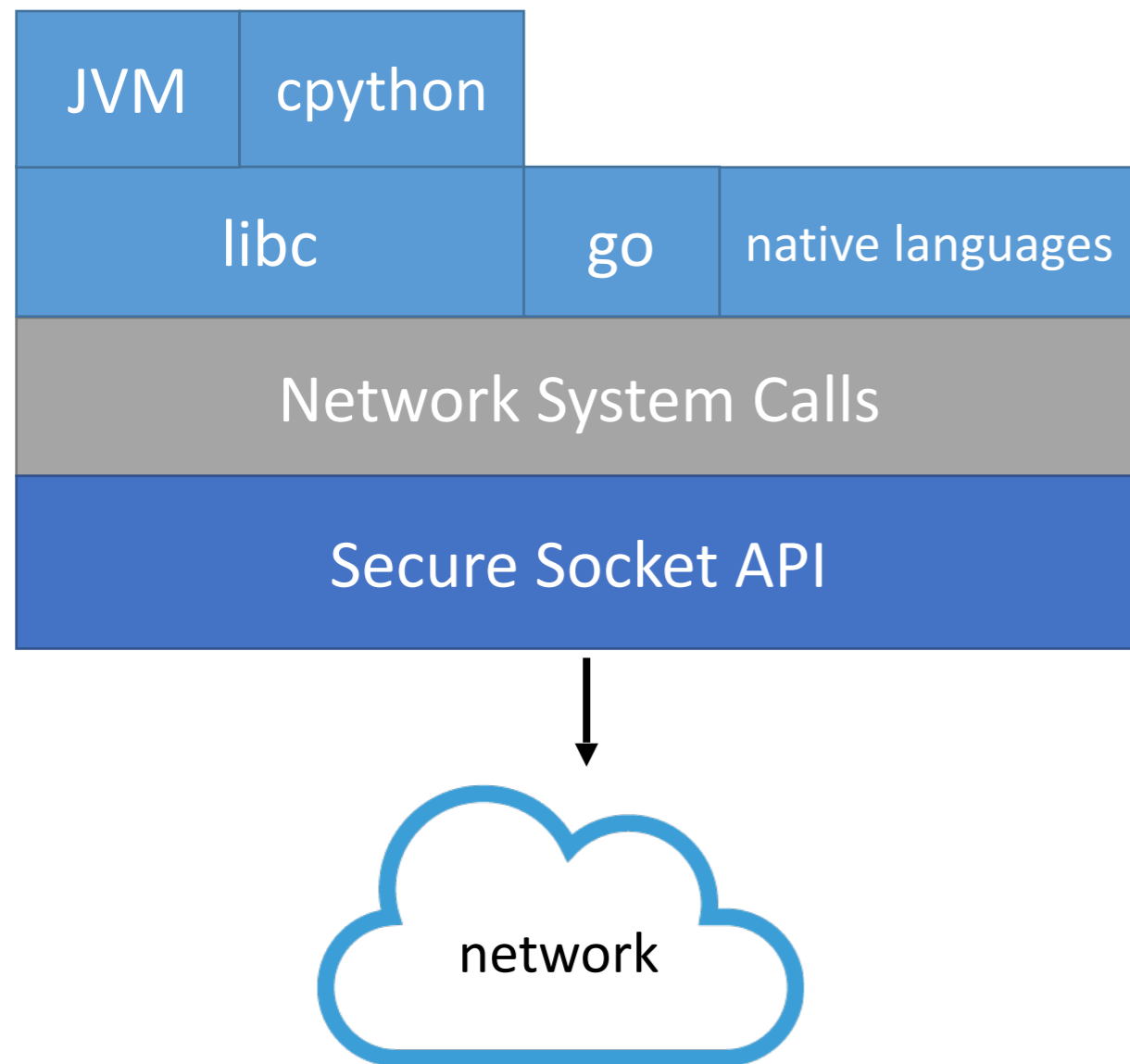
- admin's choice
 - standard validation
 - TrustBase
- TrustBase is an OS service that validates certificates according to admin config
- can enable multiple services (OSCP, CRLsets, custom root stores, Convergence, etc.)



using the SSA

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

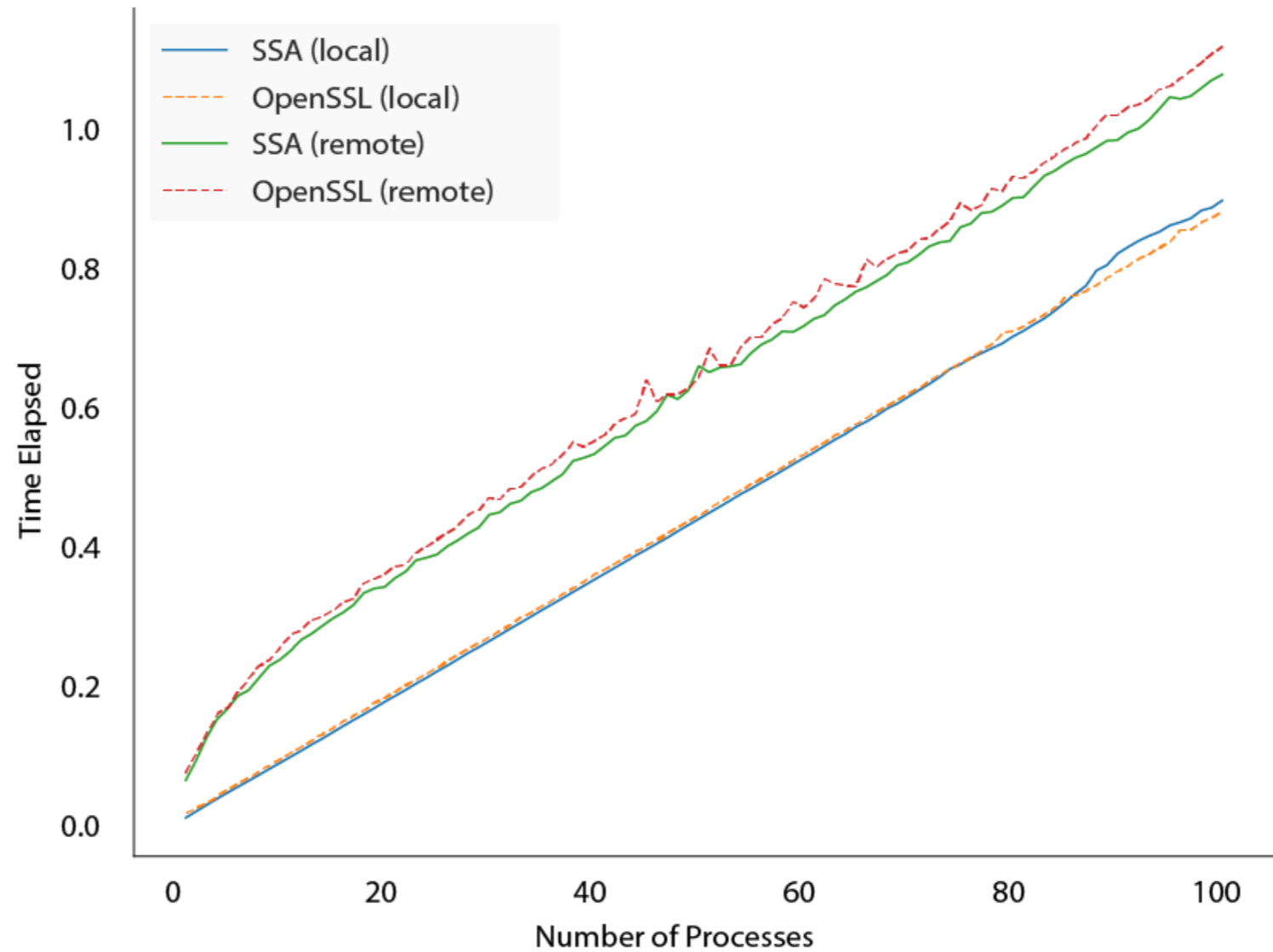
language support



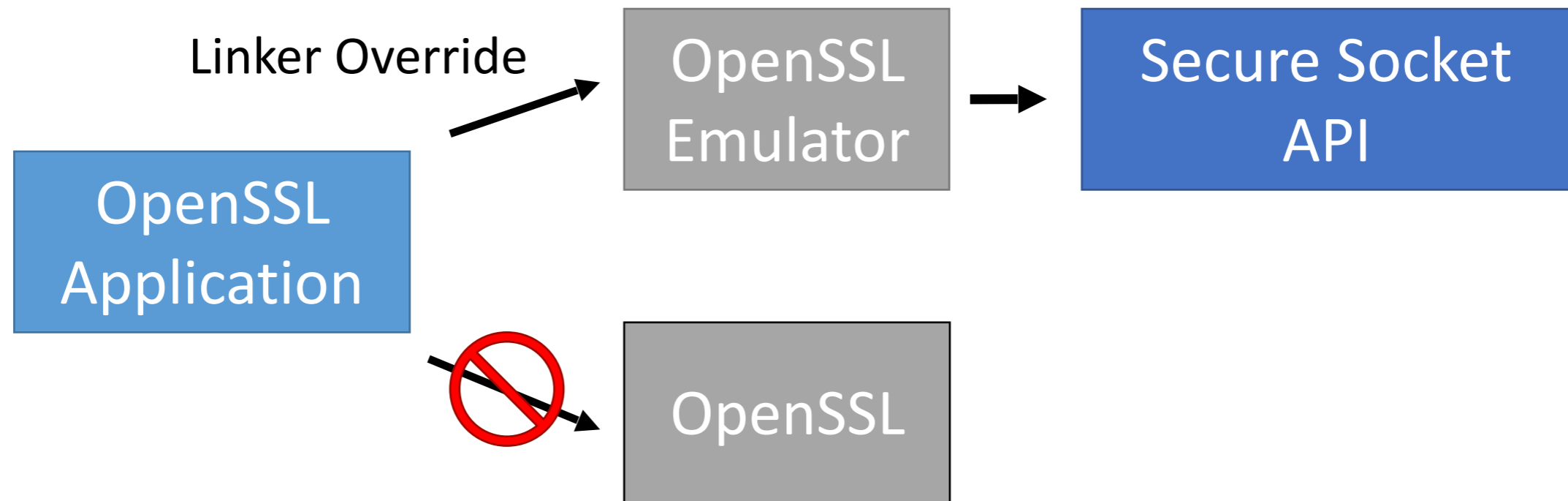
- any language that uses the network uses network system calls (directly or indirectly)
- the SSA is implemented behind the system call layer
- adding SSA support to a language is trivial
 - Go: < 50 lines of code (syscall wrappers)
 - Python: new constants only
 - PHP: new constants only
 - C/C++: new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

Papers

- Mark O'Neill, Scott Heidbrink, Scott Ruoti, Jordan Whitehead, Dan Bunker, Luke Dickinson, Travis Hendershot, Joshua Reynolds, Kent Seamons, and Daniel Zappala, TrustBase: An Architecture to Repair and Strengthen Certificate-based Authentication, *USENIX Security*, August 2017.
- Mark O'Neill, Scott Heidbrink, Jordan Whitehead, Tanner Perdue, Luke Dickinson, Torstein Collett, Nick Bonner, Kent Seamons, and Daniel Zappala, The Secure Socket API: TLS as an Operating System Service, *USENIX Security*, August 2018.

Source Code

- source code: <https://github.com/markoneill/trustbase-linux>,
<https://github.com/markoneill/trustbase-windows>
- kernel module: <https://github.com/markoneill/ssa>
- encryption daemon: <https://github.com/markoneill/ssa-daemon>
- pull requests welcome!
- project website: <https://owntrust.org>
- contact Mark: mto@byu.edu



**Homeland
Security**

- thanks to our sponsors

Next Steps

- Major tech transfer effort
- Libraries for various languages and OSes that will
 - check for kernel support and use that if available
 - check for an encryption daemon running and connect directly if available
 - worst case, do all the OpenSSL work needed for a secure connection
- Either way, developers use a simple POSIX API, and we try to use centralized policy/control where available

