The Secure Socket API

TLS as an Operating System Service

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()
SSL_CTX_set_cert_verify_callback()

x509_verify_cert()
317 Lines
can we do better?
can we use the POSIX socket API?
can we use the POSIX socket API?

```c
int socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```
can we use the POSIX socket API?

```c
int socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TLS);
```
the Secure Socket API (SSA)
the Secure Socket API (SSA)

```
connect      TLS handshake
send/sendto/etc. encrypt and send
recv/recvfrom/etc. decrypt and read
close        TLS close notify
```
TLS via the POSIX socket API
Userspace Encryption Daemon

User-space

Encryption Daemon
- Security Library
- Admin Config

Administrative Configuration

Network Application
- POSIX Socket API

Kernelspace

Network Subsystem
- TCP
- UDP
- Raw
- TLS

Internet
**TLS API reduction**

<table>
<thead>
<tr>
<th>OpenSSL</th>
<th>Secure Socket API</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_CTX_new</td>
<td>socket</td>
</tr>
<tr>
<td>SSL_CTX_set_verify</td>
<td>bind</td>
</tr>
<tr>
<td>SSL_new</td>
<td>listen</td>
</tr>
<tr>
<td>SSL_set_fd</td>
<td>connect</td>
</tr>
<tr>
<td>TLS_method</td>
<td>setsockopt</td>
</tr>
<tr>
<td>SSL_exts_set_hostname</td>
<td>getsockopt</td>
</tr>
<tr>
<td>SSL_do_handshake</td>
<td>close</td>
</tr>
<tr>
<td>SSL_set_verify_callback</td>
<td>recv/recvfrom/recvmsg</td>
</tr>
<tr>
<td>SSL_getpeer_certificate</td>
<td>send/sendto/sendmsg</td>
</tr>
<tr>
<td>And 495 more...</td>
<td>getaddrinfo</td>
</tr>
</tbody>
</table>

Symbol Count: 504 → 14
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\n\n", http_response);
    return 0;
}
reconnaissance

<table>
<thead>
<tr>
<th>Features</th>
<th>Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>version selection</td>
<td>29</td>
</tr>
<tr>
<td>cipher suite selection</td>
<td>39</td>
</tr>
<tr>
<td>extension management</td>
<td>68</td>
</tr>
<tr>
<td>certificate/key management</td>
<td>73</td>
</tr>
<tr>
<td>certificate/key validation</td>
<td>51</td>
</tr>
<tr>
<td>session management</td>
<td>61</td>
</tr>
<tr>
<td>configuration</td>
<td>19</td>
</tr>
<tr>
<td>allocation</td>
<td>33</td>
</tr>
<tr>
<td>connection management</td>
<td>41</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>64</td>
</tr>
<tr>
<td>instrumentation</td>
<td>26</td>
</tr>
</tbody>
</table>

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
administrator options

• global configuration file assigns TLS defaults

• per-application profiles can further customize settings

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

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

• thanks to our sponsors
Image Attributions

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