SSL Splitting

Christopher Lesniewski-Laas and M. Frans Kaashoek
{ctl,kaashoek}@mit.edu

MIT LCS
Bandwidth Offloading

GET /tux.png

Server
mypenguin.org

Mirror
mirrors.kernel.org

Client

Client

(DSL)

(OC12)
Bandwidth Offloading

Server
mypenguin.org

(Mirror)
mirrors.kernel.org

(Mirror)
daemonporn.com

‘GET /tux.png’
Secure Bandwidth Offloading

\[ \sigma = \text{Sign(tux.png)} \]

\[
\begin{align*}
\text{Server} & \quad \text{mypenguin.org} \\
\text{Mirror} & \quad \text{mirrors.kernel.org} \\
\text{Mirror} & \quad \text{daemonporn.com}
\end{align*}
\]

\[
\begin{align*}
\text{Client} & \quad \text{Client} \\
\text{Client} & \quad \text{Client}
\end{align*}
\]

\[ \text{\textquote{GET /tux.png}} \]
Secure Bandwidth Offloading

\[ \sigma = \text{Sign}(\text{tux.png}) \]

Server
\[ \text{mypenguin.org} \]

Mirror
\[ \text{mirrors.kernel.org} \]

Mirror
\[ \text{daemonporn.com} \]

Client
\[ \text{GET /tux.png} \]

(OC12)

(OC12)

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]
Existing Solutions Aren’t Practica

- Force users to install specialized browser
  - Ex: S-HTTP, SFSRO, BitTorrent, RPM+PGP
- Operates at the channel level, not file level
  - Ex: SSL
SSL's Authentication Layer

Client

Hello
Certificate
Negotiate shared secret
Done
(knows shared secret k)

Server

Handshak
Request
File transf
(knows shared secret k)

GET /tux.png

X = MAC_k( )

X’ = MAC_k( )
Check: X = X’?

X = MAC_k( )

When All You Have Is A Hammer...

Client

\[ X = MAC_k(\text{Client}) \]

\[ X' = MAC_k(\text{Client}) \]

Check: \( X = X' ? \)

Server

\[ X = MAC_k(\text{Server}) \]
SSL Splitting

Client

Proxy

Server

\[ X = MAC_k(\text{tux.png(1/2)}) \]

\[ 'tux.png(1/2)' \]

\[ X = \text{Cache('tux.png(1/2)')} \]

\[ X' = MAC_k(\text{tux.png(1/2)}) \]

Check: \( X = X' \)?
SSL Splitting

1. Connect

- Server
- Proxy
- Client

Connect
SSL Splitting

1. Connect

Server

Proxy

Client
SSL Splitting

1. Connect
2. Handshake

Server (knows k)

Proxy (cannot learn k)

Negotiate shared key k

Client (knows k)
SSL Splitting

1. Connect
2. Handshake
3. Request

GET /tux.png
SSL Splitting: Cache Hit

1. Connect
2. Handshake
3. Request
4. Stub record

Server

ID=SHA-1(tux.png), X=MAC_k(tux.png)

Proxy

ID

Cache

Client
SSL Splitting: Cache Hit

1. Connect
2. Handshake
3. Request
4. Stub record
5. Spliced record

Proxy

Client

Server

Cache

Check MAC X

, X
SSL Splitting: Cache Miss

Proxy

ID=SHA-1(tux.png), X=MAC_k(tux.png)

Server

Cache

Client

miss!
SSL Splitting: Cache Miss

Server

Proxy

Cache

Client

Get(ID)

ID

miss!

ID

USENIX Security 2003 – p. 1
SSL Splitting: Cache Miss

Server

Get(ID)

Proxy

Cache

ID

miss!

Client
SSL Splitting: Cache Miss

- Server
- Proxy
- Client
- Cache

Get(ID)
Insert
Check MAC X

SSL Splitting: Cache Miss

- Get(ID)
- Insert
- Check MAC X
Caveats

- No end-to-end confidentiality
- Only distributes bandwidth load, not CPU
Implementation

- Server
  - Unmodified Apache
  - Modified OpenSSL library
- Proxy: Perl and C
  - Splicing is not a cryptographic operation
- Client: Netscape, IE, w3m...
Performance Questions

• How much data do we send over the server-proxy link?
• How does overhead vary with file size?
• How much overhead with realistic file size distributions?
Experiments

- Client replayed prerecorded request patterns
- Measured bytes over server interfaces
- Key performance metric is "rate" $r$:

\[
\begin{align*}
    r &= \frac{\text{wire bytes sent by server}}{\text{total size of files received by clients}} \\
    \text{ - Smaller is better} \\
    \text{ - If no caching, } r &= 1 + \% \text{ overhead}
\end{align*}
\]
Experimental Setup

- Server: 160 kbps upstream, 500 MHz AMD
  - CPU could push $\approx 4$ Mbps using HTTPS
- Client: 100 Mbps LAN, 1.2 GHz Athlon
- Proxy: 100 Mbps LAN, 700 MHz P3
Single File Microbenchmark
Some Apache Quirks

Apache puts HTTP headers into separate record

Apache bug: record size halved

Rate

File size (bytes)
Understanding Single File Results

- Model: \( r = f(\text{file size}) \)
- Constant 1.5 KB overhead per file
- Uncached: 5% overhead per byte
- Cached: 62 bytes sent per 16 KB record
  - 8 KB records for files \( \geq 4 \) MB
Real Workloads

- Do real access patterns benefit from SSL splitting?
- 7-month web traces taken from www.lcs.mit.edu and amsterdam.lcs.mit.edu
How The Simulator Works

- Input: list of file requests and sizes
- Use microbenchmark results to predict number of bytes sent by server
- Infinite cache
Simulation Accuracy

- 2 hours, 10 MB transferred, 4.43 MB of files
Long-Term Savings $\approx 83\%$

- 7 months, 109 GB transferred, 10.6 GB of files
Summary

- SSL Splitting does not:
  - Provide confidentiality
  - Reduce server CPU load

- SSL Splitting does:
  - Reduce server bandwidth use by 25–90%
  - Guarantee end-to-end data integrity
  - Work with normal Web browsers!

- You might use it if: you’re a Web site admin and you’re not sure you trust your mirrors.
Availability

http://pdos.lcs.mit.edu/barnraising/