OWASP Mobile Top 10 Risk
M3 : Insufficient Transport Layer Protection

Anant Shrivastava

About Me

- Anant Shrivastava (@anantshri)
- http://www.anantshri.info
- Independent Information Security Consultant
- Focus Area's : Web, Mobile, Linux
- Current Project:
  - CodeVigilant (codevigilant.com)
    - An initiative to find flaws in open source software and perform a responsible disclosure. Website currently holds 160+ disclosed vulnerability in various wordpress plugins.
  - Android Tamer (androidtamer.com)
    - Live ISO environment for Android Security Researchers. Used by multiple researchers as well as Trainers across the globe.

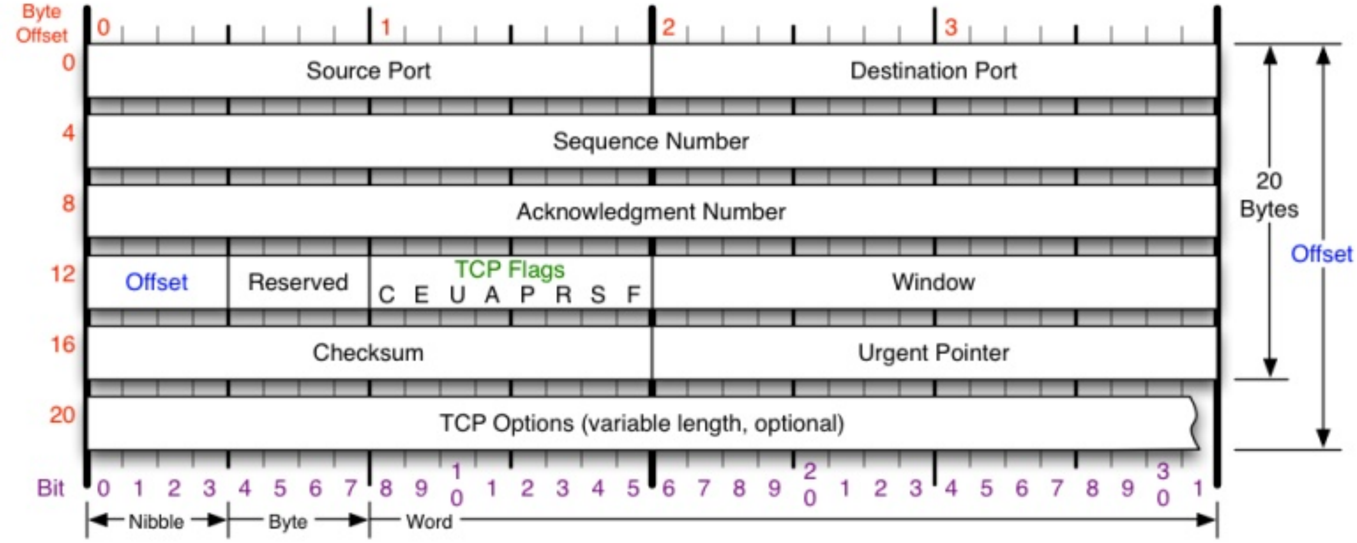
Agenda

- Understand Transport Layer
- Understand Transport Protections
- Understand Complexities/Insecurities in transport layer protection.
- How to Find Insecure or inadequate protections
- How to Prevent it

Transport Layer

- OSI Model Layer 4 (from bottom or top)
- A **transport layer** provides end-to-end or host-to-host communication services for applications within a layered architecture of network components and protocols.
- Protocols in Use : TCP and UDP
- The transport layer is responsible for delivering data to the appropriate application process on the host computers
- Unique Identifier : IP:Port (URI)
- **In short backbone of internet communication**

TCP Headers



TCP Flags

C E U A P R S F

Congestion Window

C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Receiver Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

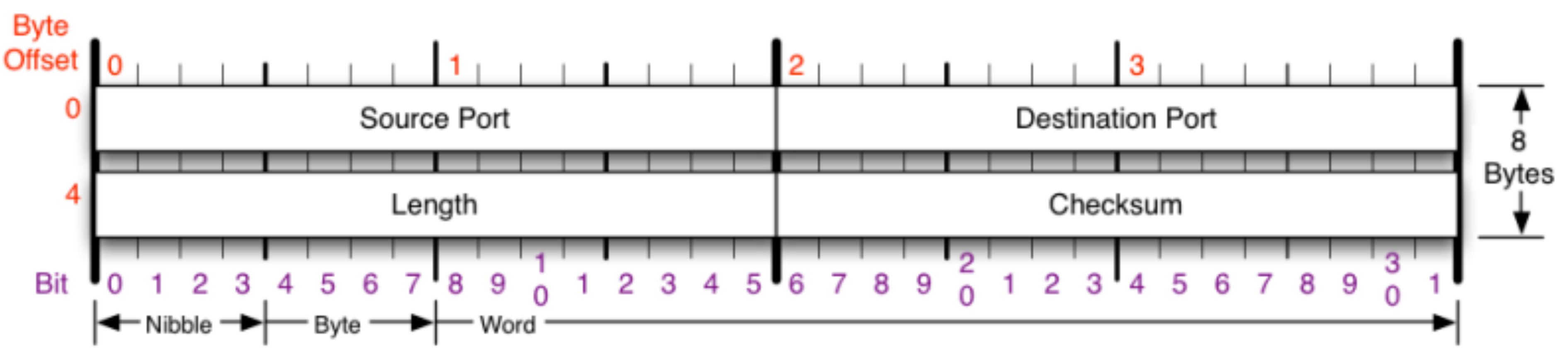
Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

UDP Headers



Checksum

Checksum of entire UDP segment and pseudo header (parts of IP header)

RFC 768

Please refer to RFC 768 for the complete User Datagram Protocol (UDP) Specification.

Transport Layer Protections

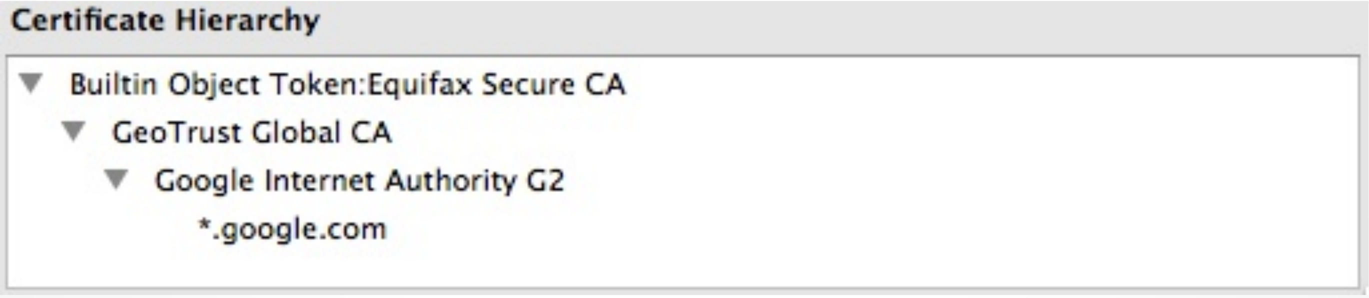
- Commonly known as Transport Layer Security (TLS) or formerly Secure Socket Layer (SSL)
- Latest version in use TLSv1.2
- Commonly found: SSLv2, SSLv3/TLSv1.0,TLSv1.1
- Uses X509 Certificate based asymetric encryption.
- What we generally know as HTTPS

- TLS v1.3 in draft since July 2014.
- first defined in 1999 and last updated in RFC 5246 (August 2008) and RFC 6176 (March 2011).

TLS Certificates

- Issued by a CA (Certification Authority)
- Follows a chain of trust to establish the identity of a website.
- For internal purposes people use self-signed certificate which doesn’t following trusted chain.

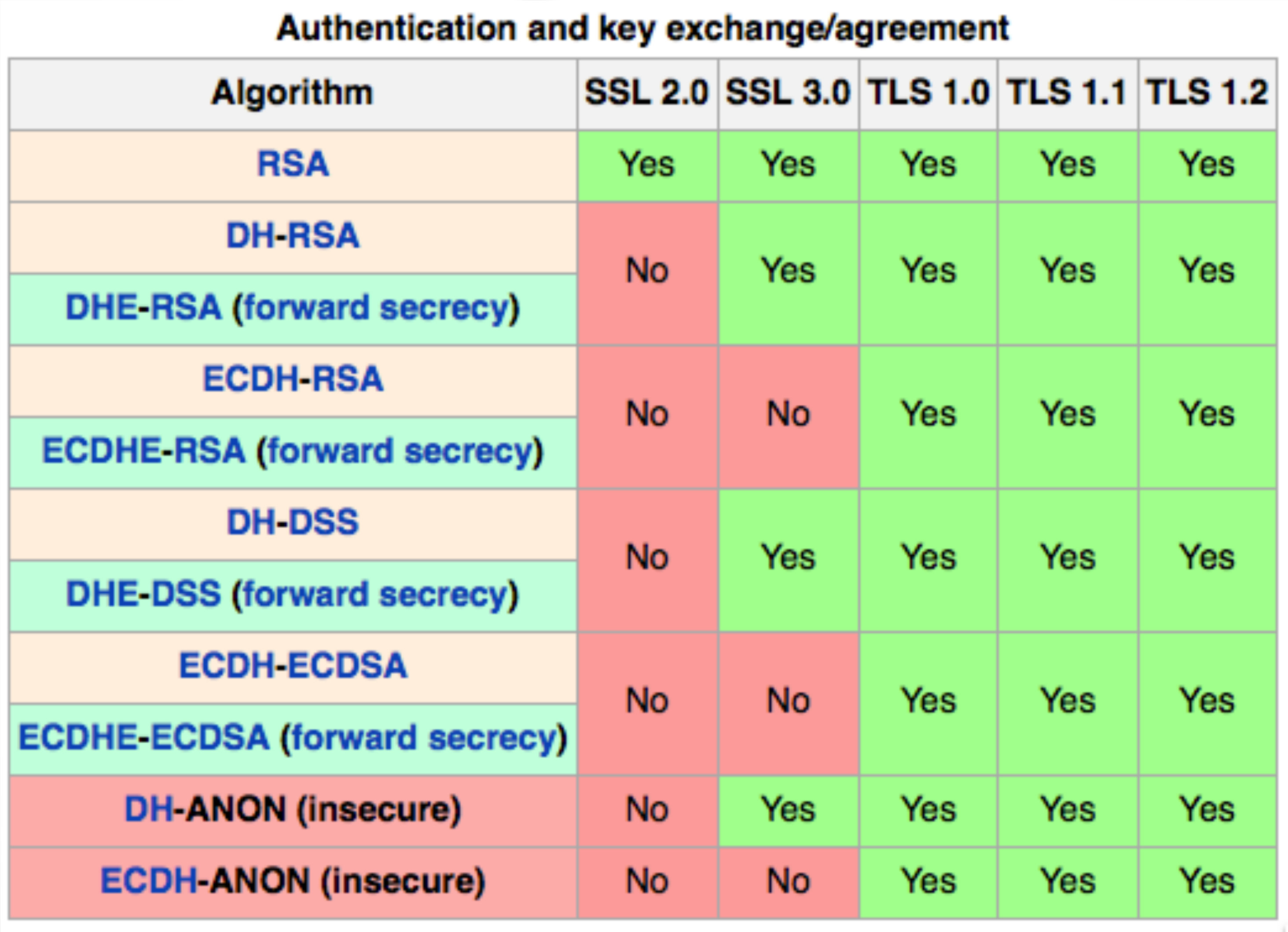
- Example of trusted chain @ google.com



Certificate Hierarchy

- ▼ Builtin Object Token:Equifax Secure CA
  - ▼ GeoTrust Global CA
    - ▼ Google Internet Authority G2
      - *.google.com

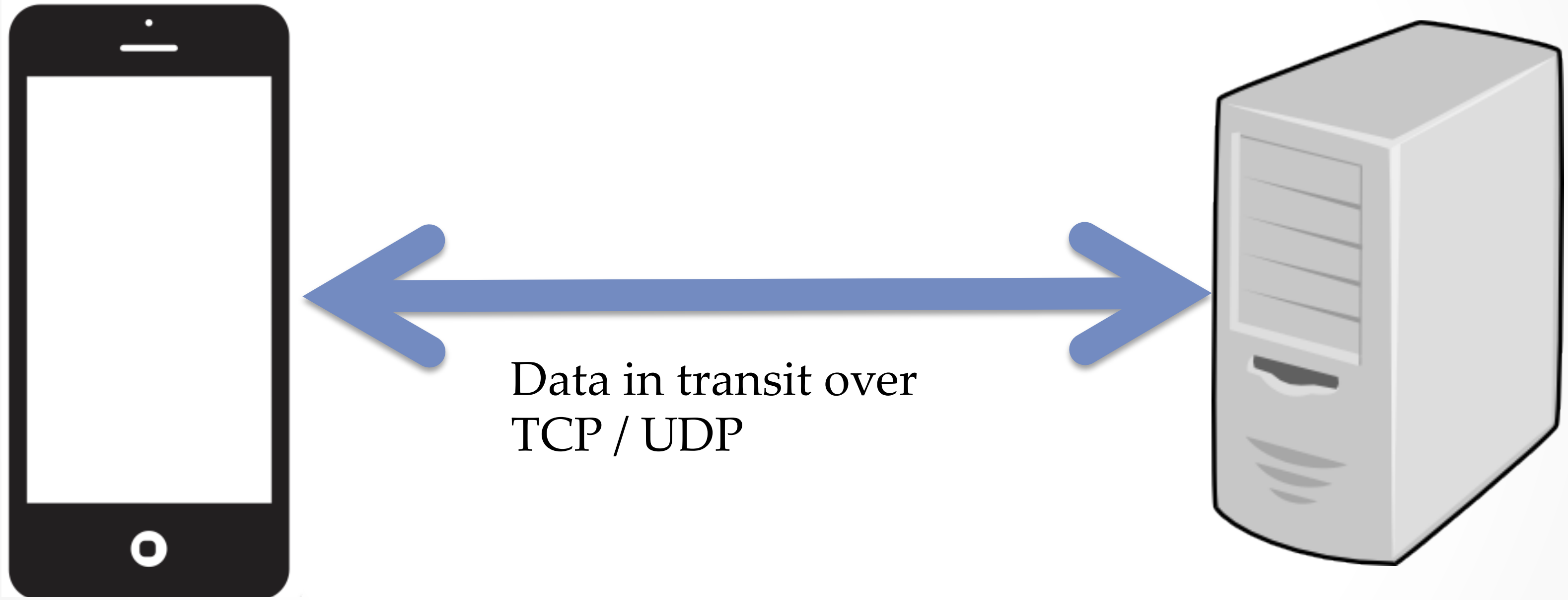
Various Algorithms in use



Authentication and key exchange/agreement

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2
RSA	Yes	Yes	Yes	Yes	Yes
DH-RSA	No	Yes	Yes	Yes	Yes
DHE-RSA (forward secrecy)					
ECDH-RSA	No	No	Yes	Yes	Yes
ECDHE-RSA (forward secrecy)					
DH-DSS	No	Yes	Yes	Yes	Yes
DHE-DSS (forward secrecy)					
ECDH-ECDSA	No	No	Yes	Yes	Yes
ECDHE-ECDSA (forward secrecy)					
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes

Mobile Prospective



Data in transit over
TCP / UDP

Insecure implementations

- Using Known Weak Ciphers / version (SSLv2, RC4, MD5, CBC in SSL3)
- Communication using Self-signed certificate (ignoring warning)
- Securing only specific portion of communication
- Not validating the chain of trust
- Mixxing TLS and non TLS content on Page

SSL Version 2

- SSL version 2 was designed in 1994 by Netscape. Its 20 years old this year.
- Known attacks
  - Identical cryptographic keys are used for message authentication and encryption.
  - SSL 2.0 has a weak MAC construction that uses the MD5 hash function with a secret prefix, making it vulnerable to length extension attacks.
  - SSL 2.0 does not have any protection for the handshake, meaning a man-in-the-middle downgrade attack can go undetected.
  - SSL 2.0 uses the TCP connection close to indicate the end of data. This means that truncation attacks are possible: the attacker simply forges a TCP FIN, leaving the recipient unaware of an illegitimate end of data message (SSL 3.0 fixes this problem by having an explicit closure alert).
  - SSL 2.0 assumes a single service and a fixed domain certificate, which clashes with the standard feature of virtual hosting in Web servers. This means that most websites are practically impaired from using SSL
- Blocked in most modern browsers (IE 6 users anyone?)

Other versions

- SSLv3 (was working good till 2012)
- SSL 3.0 cipher suites have a weaker key derivation process; half of the master key that is established is fully dependent on the MD5 hash function
- **More attacks**
  - Renegotiation attack
  - **BEAST attack**
  - **CRIME and BREACH attacks**
  - **Padding attacks (Lucky 13)**
  - **RC4 Attacks**
  - **Implementation bugs like (Apple SSL, Heartbleed, GNUTLS Fail)**

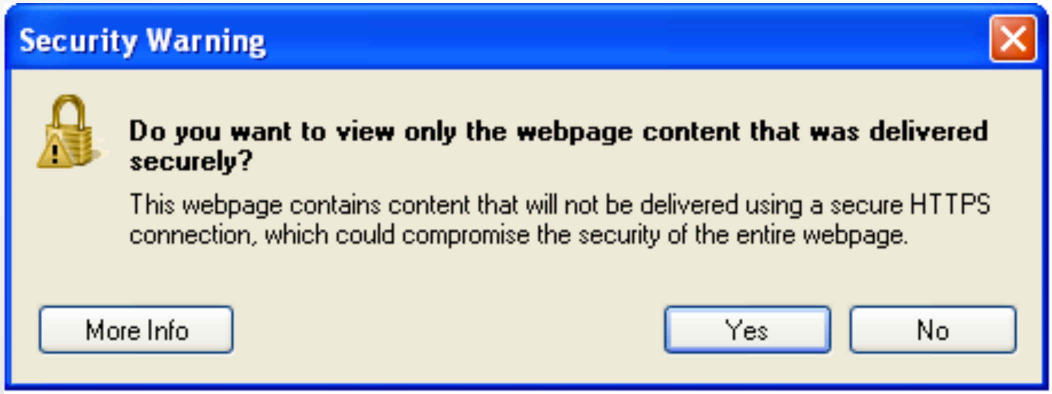
Chain of trust

- Establish chain of trust
- Ensure the connection has exact same chain of trust as official certificate.

- Any certificate in the chain is self-signed, unless it the root.
- Not every intermediate certificate is checked, starting from the original certificate all the way up to the root certificate.
- An intermediate, CA-signed certificate does not have the expected Basic Constraints or other important extensions.
- The root certificate has been compromised or authorized to the wrong party.
- Ref : http://cwe.mitre.org/data/definitions/296.html

Mixing content

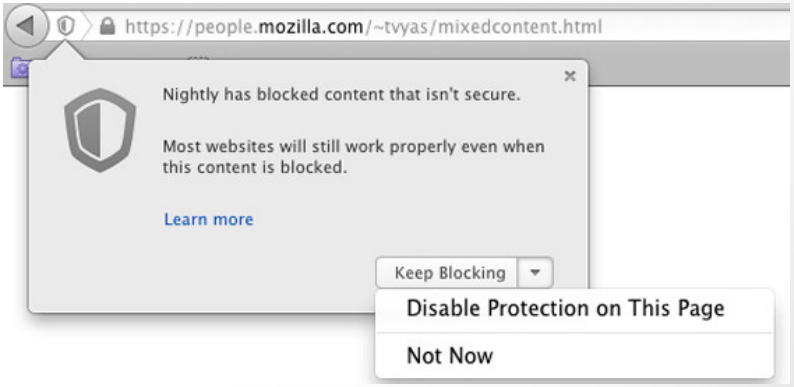
- HTTP and HTTPS content
- HTTP can be cached and read over the wire.
- Analytics and tracking generally use http for quick transaction and hence susceptible.



Security Warning

Do you want to view only the webpage content that was delivered securely?

This webpage contains content that will not be delivered using a secure HTTPS connection, which could compromise the security of the entire webpage.



https://people.mozilla.com/~tvyas/mixedcontent.html

Nightly has blocked content that isn't secure.

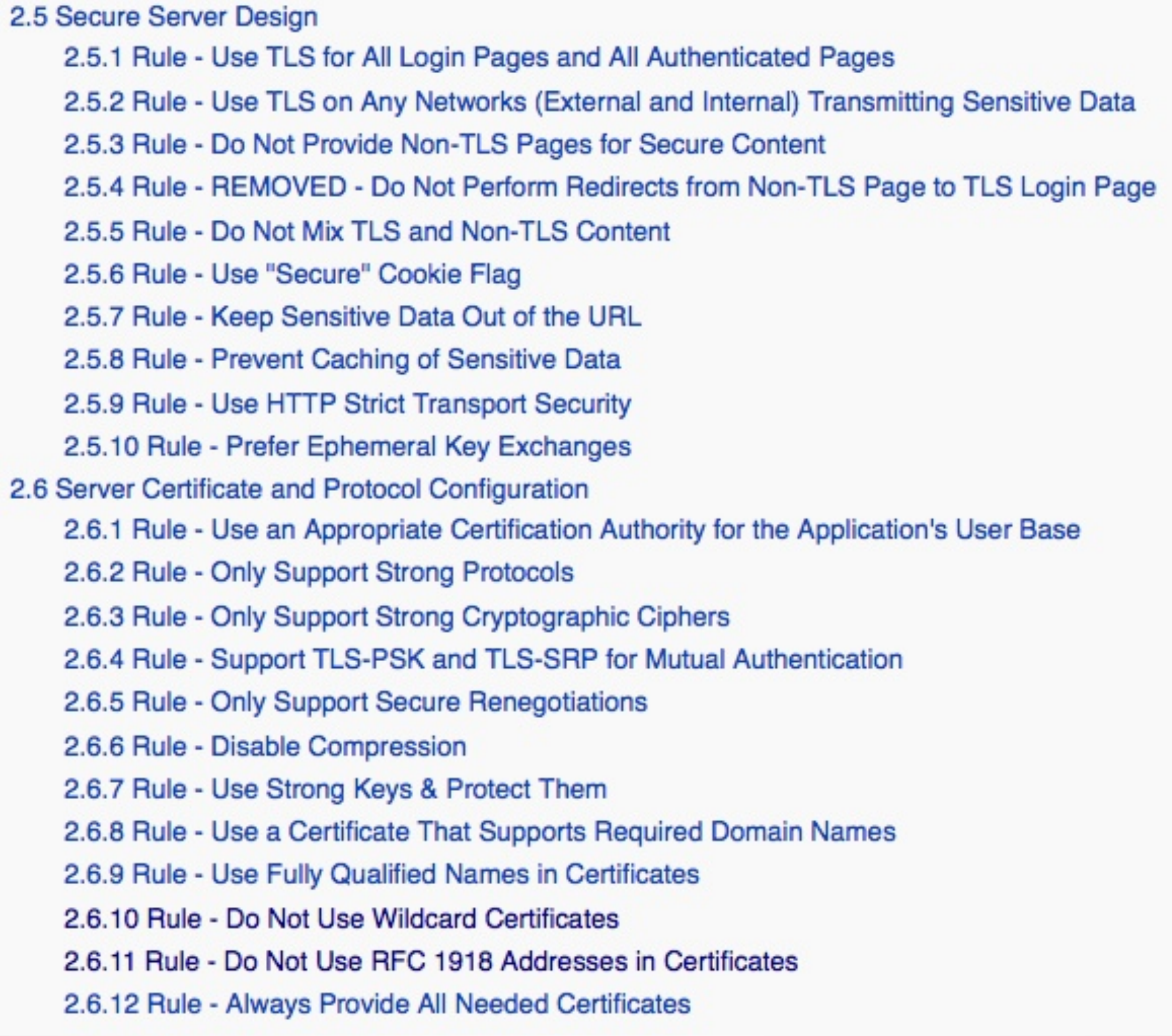
Most websites will still work properly even when this content is blocked.

Detecting SSL issues

- Launch emulator / start device.
- Add proxy settings for burp/zap/ironwasp etc
- Run application and check if traffic interception works and application performs its actions. (Implementation is flawed)
- Identify end points

- End point Implementation flaws : use SSLScan (either original or rbsec/sslscan at github)

Preventions



- 2.5 Secure Server Design
  - 2.5.1 Rule - Use TLS for All Login Pages and All Authenticated Pages
  - 2.5.2 Rule - Use TLS on Any Networks (External and Internal) Transmitting Sensitive Data
  - 2.5.3 Rule - Do Not Provide Non-TLS Pages for Secure Content
  - 2.5.4 Rule - REMOVED - Do Not Perform Redirects from Non-TLS Page to TLS Login Page
  - 2.5.5 Rule - Do Not Mix TLS and Non-TLS Content
  - 2.5.6 Rule - Use "Secure" Cookie Flag
  - 2.5.7 Rule - Keep Sensitive Data Out of the URL
  - 2.5.8 Rule - Prevent Caching of Sensitive Data
  - 2.5.9 Rule - Use HTTP Strict Transport Security
  - 2.5.10 Rule - Prefer Ephemeral Key Exchanges
- 2.6 Server Certificate and Protocol Configuration
  - 2.6.1 Rule - Use an Appropriate Certification Authority for the Application's User Base
  - 2.6.2 Rule - Only Support Strong Protocols
  - 2.6.3 Rule - Only Support Strong Cryptographic Ciphers
  - 2.6.4 Rule - Support TLS-PSK and TLS-SRP for Mutual Authentication
  - 2.6.5 Rule - Only Support Secure Renegotiations
  - 2.6.6 Rule - Disable Compression
  - 2.6.7 Rule - Use Strong Keys & Protect Them
  - 2.6.8 Rule - Use a Certificate That Supports Required Domain Names
  - 2.6.9 Rule - Use Fully Qualified Names in Certificates
  - 2.6.10 Rule - Do Not Use Wildcard Certificates
  - 2.6.11 Rule - Do Not Use RFC 1918 Addresses in Certificates
  - 2.6.12 Rule - Always Provide All Needed Certificates

Preventions

- Assume connection is compromised
- Disable weak ciphers and versions
- Perform entire sensitive communication over TLS
- Never allow connection using Self-signed certificate.
- Use secure versions of tracking/analytics/ad network SDK
- Add a second layer of encryption for sensitive data.
- Follow Rules: https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- Perform certificate validation via Certificate pinning : refer http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/

Preventions

- iOS: For **CFNetwork**, use Secure Transport API to designate trusted client certificates
- iOS: all NSURL calls (or wrappers of NSURL) do not allow self signed or invalid certificates such as the NSURL class method **setAllowsAnyHTTPSCertificate**.
- iOS Cert Pinning : export your certificate, include it in your app bundle, and anchor it to your trust object. Using the NSURL method **connection:willSendRequestForAuthenticationChallenge**: will now accept your cert.
- Android: ensure **org.apache.http.conn.ssl.AllowAllHostnameVerifie**r or **SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER** are not present

References

- https://en.wikipedia.org/wiki/Transport_layer
- https://www.owasp.org/index.php/Mobile_Top_10_2014-M3
- http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf
- TCP UDP Headers : http://nmap.org/book/tcpip-ref.html
- https://commons.wikimedia.org/wiki/File:Mobile_and_desktop_device_templates.svg
- https://openclipart.org/collection/collection-detail/silpstream/8546
- http://www.troyhunt.com/2013/06/understanding-risk-of-mixed-content.html

Question Time