Secure Socket Layer Health Assessment

Mick Pouw, Eric van den Haak

February 5, 2014

Background

- Tilburg University
- Lots of SSL/TLS services
- No quick SSL service checking (Manually)
- Existing tools lack possibility of integrating in existing monitoring software or lack in rating
- What about a new tool?

How can we determine SSL "health" of a server side implementation?

- How can we determine a "bad" SSL implementation?
- What mistakes are commonly made by server administrators regarding implementing SSL?
- How can we classify these mistakes?
- How can we develop a tool that automates checking the SSL "health" of a server side implementation?

Implementing SSL, the right way

- Certificates
- Protocols
- Server settings

Certificates

- Subject
- Validity
- (Chain of) Trust
- Hash algorithm
- Debian weak key
- Revocation

Protocols

- SSLv2 **must** be disabled
- SSLv3 should be disabled, backwards compatibility
- TLSv1.0 should be enabled
- TLSv1.1 should be enabled
- TLSv1.2 should be enabled

Server Settings

- Compression (Crime)
- RC4 (Randomness)
- MD5 (Collision)
- Strong key size (Brute force)
- Perfect forward Secrecy (Future decryption)

Common mistakes

Test	Percentage passed
Signature hash algorithm	100%
Certificate (chain) trusted	100%
Certificate is valid	100%
No Debian weak keys	100%
Subject name matches	91%
Compression disabled	100%
Cipher suites do not contain MD5	57%
Perfect forward secrecy available	46%
Cipher suites do not contain RC4	17%
Key length at least 128bits	89%
SSLv2 disabled	94%
SSLv3 disabled	3%
TLSv1.0 enabled	97%
TLSv1.1 enabled	63%
TLSv1.2 enabled	63%

Determining a test

- Weight ($0 <= weight <= 100$)
- Required (Show-stopper)

Example test

Name	Example
Proposition	Requirement in order to pass the test
Weight	50
Required	No

Formulas

$$\{requiredtests\} \subset \{passedtests\} \tag{1}$$

The set of all required tests has to be a subset of all passed tests.

$$100 * \frac{\sum_{i=1}^{N} p_i}{\sum_{j=1}^{M} t_j} \tag{2}$$

Where p is a set of all weights of the passed tests and t is a set of all weights of all performed tests.

Classification

Description	Weight	Required
Signature hash algorithm	80	No
Certificate (chain) trusted	0	Yes
Certificate is valid	0	Yes
No Debian weak keys	100	No
Subject name matches	0	Yes
Compression disabled	50	No
Cipher suites do not contain MD5	50	No
Perfect forward secrecy available	50	No
Cipher suites do not contain RC4	80	No
Key length at least 128bits	80	No
SSLv2 disabled	100	No
SSLv3 disabled	30	No
TLSv1.0 enabled	75	No
TLSv1.1 enabled	100	No
TLSv1.2 enabled	100	No

Proof of Concept

- Python
- Used software
  - SSLyze
  - OpenSSL
  - Curl
- Modular framework
  - Tests
  - Output

Running the tool!

- Entire Tilburg University IPv4 space
- SURFnet IDP page hosts

Score	SURFconext	UvT
$<$ 40%	5	27
40-50%	8	1
50-60%	82	64
60-70%	9	6
70-80 %	13	1
$>$ 80 %	20	32

Conclusions

- Found a new way of determining SSL "Health"
- Developed a proof of concept that assess SSL services

Future work

- Start TLS
- Server Name Indication (SNI) for HTTPS
- Improve framework's dependencies

Demo

Questions?