APRIL 23, 2017

TUTORIALS & EXPLOITS

LEAVE A COMMENT

TLS & SSL Penetration Testing

The definitive guide for SSL / TLS security assessments while performing penetration testing & **network security audits**.

Introduction

This article documents the process of using semi automated tools to perform SSL & TLS security assessments and how to validate the tool findings using manual testing methods. The aim is to optimise the TLS & SSL security testing process helping consultants spend less time on TLS / SSL while performing penetration testing. engagements.

What is TLS & SSL?

> Secure Socket Layer (SSL) & Transport Layer Security (TLS) encryption are used to secure Internet and network traffic by providing communication security (encryption in transit) and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

Therefore, TLS security configuration is important and time should be spent learning and identifying the common vulnerabilities and security misconfigurations.

TLS / SSL Security Testing Tools

testssl.sh

testssl.sh is our preferred tool for testing, it covers all the required tests for TLS & SSL assessments and is regularly updated.

Installation

You can install the latest version of tesetssl.sh by performing a git clone of their repository:

```
git clone https://github.com/drwetter/testssl.sh.git
```

testssl.sh Examples

There are many testing options that can be used with testssl.sh and the options you should use will depend greatly on your testing requirements. Below are some useful examples, for an overview of testssl.sh command line options. run `./testssl.sh` with no other options.

Test Everything on a Single Host and Output to console

```
./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U TARGET-HOST
```

Test Everything on a Single Host and Output to HTML

```
./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U TARGET-HOST | aha > OUTPUT-FILE.html
```

Test all hosts on a Subnet and Output to HTML

```
./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U 192.168.1.0/24 | aha > OUTPUT-FILE.html
```

Same as above, but only enumerate each servers supported ciphers:

```
./testssl.sh -E 192.168.1.0/24 | aha > OUTPUT-FILE.html
```

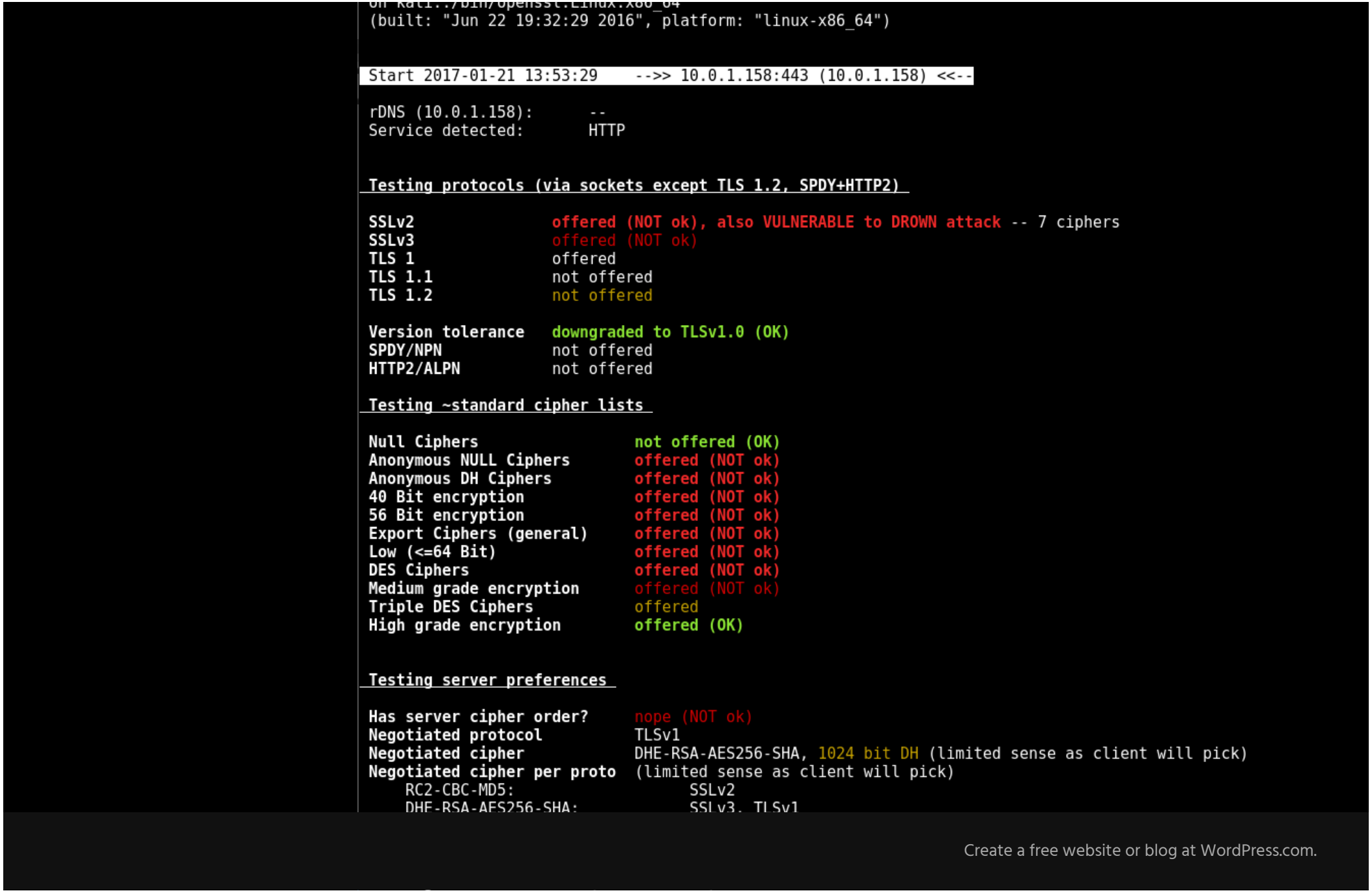
Screenshots of Console Output

```
root@kali:~/Tools/ssl-testing/testssl.sh# ./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U 10.0.1.158

###########################################################
    testssl.sh       2.8rc1 from https://testssl.sh/dev/
    (424cf23 2016-08-09 10:35:58 -- 1.531)

      This program is free software. Distribution and
             modification under GPLv2 permitted.
      USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

       Please file bugs @ https://testssl.sh/bugs/
```



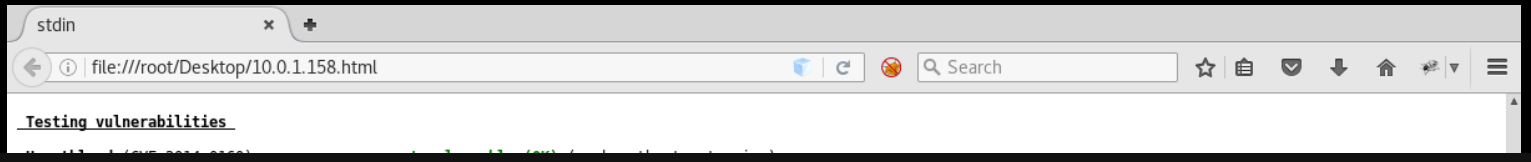
```
TLS extensions (standard)    (none)
Session Tickets RFC 5077     (none)
SSL Session ID support       yes
TLS clock skew               +43580 sec from localtime
Signature Algorithm          SHA1 with RSA
Server key size              RSA 1024 bits
Fingerprint / Serial         SHA1 A111B17B7853A34132FC189A18535031735464B9 / 5A4F
                             SHA256 9BD472E69A83BBA42053F7AD94477EFE522986A1B352B4F5D3E039AFAF6753AB

Common Name (CN)             "localhost.localdomain"
subjectAltName (SAN)         --
Issuer                       self-signed (NOT ok)
Trust (hostname)             certificate does not match supplied URI
Chain of trust               NOT ok (self signed)
EV cert (experimental)       no
Certificate Expiration       364 >= 60 days (2017-01-21 12:20 --> 2018-01-21 12:20 -0500)
# of certificates provided   1
Certificate Revocation List  --
OCSP URI                     --
OCSP stapling                --


 Testing HTTP header response @ "/"

HTTP Status Code             403 Forbidden
HTTP clock skew              +43581 sec from localtime
Strict Transport Security    --
Public Key Pinning           --
Server banner                Apache/2.2.3 (CentOS)
Application banner           --
Cookie(s)                    (none issued at "/")
Security headers             --
Reverse Proxy banner         --
```

Screenshots of HTML Output



```
POODLE, SSL (CVE-2014-3566)             VULNERABLE (NOT ok), uses SSLv3+CBC (check TLS_FALLBACK_SCSV mitigation below)
TLS_FALLBACK_SCSV (RFC 7507), experim.  Downgrade attack prevention NOT supported
FREAK (CVE-2015-0204)                   VULNERABLE (NOT ok), uses EXPORT RSA ciphers
DROWN (2016-0800, CVE-2016-0703), exper. vulnerable (NOT ok), SSLv2 offered with 7 ciphers
LOGJAM (CVE-2015-4000), experimental    VULNERABLE (NOT ok), uses DHE EXPORT ciphers, common primes not checked.
BEAST (CVE-2011-3389)                   SSL3: EXP-RC2-CBC-MD5 EXP-DES-CBC-SHA
                                            DES-CBC-SHA DES-CBC3-SHA EXP-EDH-RSA-DES-CBC-SHA
                                            EDH-RSA-DES-CBC-SHA EDH-RSA-DES-CBC3-SHA EXP-ADH-DES-CBC-SHA
                                            ADH-DES-CBC-SHA ADH-DES-CBC3-SHA AES128-SHA
                                            DHE-RSA-AES128-SHA ADH-AES128-SHA AES256-SHA DHE-RSA-AES256-SHA ADH-AES256-SHA EXP1024-DES-CBC-SHA EXP-RC2-CBC-MD5
                                        TLS1: EXP-RC2-CBC-MD5 EXP-DES-CBC-SHA
                                            DES-CBC-SHA DES-CBC3-SHA EXP-EDH-RSA-DES-CBC-SHA
                                            EDH-RSA-DES-CBC-SHA EDH-RSA-DES-CBC3-SHA EXP-ADH-DES-CBC-SHA
                                            ADH-DES-CBC-SHA ADH-DES-CBC3-SHA AES128-SHA
                                            DHE-RSA-AES128-SHA ADH-AES128-SHA AES256-SHA DHE-RSA-AES256-SHA ADH-AES256-SHA EXP1024-DES-CBC-SHA EXP-RC2-CBC-MD5
                                        VULNERABLE -- and no higher protocols as mitigation supported
RC4 (CVE-2013-2566, CVE-2015-2808)      VULNERABLE (NOT ok): ADH-RC4-MD5 RC4-SHA RC4-MD5 RC4-MD5 RC4-64-MD5 EXP1024-RC4-SHA EXP-ADH-RC4-MD5 EXP-RC4-MD5 EXP-RC4-MD5


 Testing all 183 locally available ciphers against the server, ordered by encryption strength

Hexcode  Cipher Suite Name (OpenSSL)       KeyExch.  Encryption Bits     Cipher Suite Name (RFC)
-----------------------------------------------------------------------------------------------------------------
 x39     DHE-RSA-AES256-SHA                DH 1024   AES        256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 x3a     ADH-AES256-SHA                    DH 1024   AES        256      TLS_DH_anon_WITH_AES_256_CBC_SHA
 x35     AES256-SHA                        RSA       AES        256      TLS_RSA_WITH_AES_256_CBC_SHA
 x33     DHE-RSA-AES128-SHA                DH 1024   AES        128      TLS_DHE_RSA_WITH_AES_128_CBC_SHA
 x34     ADH-AES128-SHA                    DH 1024   AES        128      TLS_DH_anon_WITH_AES_128_CBC_SHA
 x2f     AES128-SHA                        RSA       AES        128      TLS_RSA_WITH_AES_128_CBC_SHA
 x030080 RC2-CBC-MD5                       RSA       RC2        128      SSL_CK_RC2_128_CBC_WITH_MD5
 x18     ADH-RC4-MD5                       DH 1024   RC4        128      TLS_DH_anon_WITH_RC4_128_MD5
 x05     RC4-SHA                           RSA       RC4        128      TLS_RSA_WITH_RC4_128_SHA
 x04     RC4-MD5                           RSA       RC4        128      TLS_RSA_WITH_RC4_128_MD5
 x010080 RC4-MD5                           RSA       RC4        128      SSL_CK_RC4_128_WITH_MD5
 x16     EDH-RSA-DES-CBC3-SHA              DH 1024   3DES       168      TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
 x1b     ADH-DES-CBC3-SHA                  DH 1024   3DES       168      TLS_DH_anon_WITH_3DES_EDE_CBC_SHA
 x0a     DES-CBC3-SHA                      RSA       3DES       168      TLS_RSA_WITH_3DES_EDE_CBC_SHA
 x0700c0 DES-CBC3-MD5                      RSA       3DES       168      SSL_CK_DES_192_EDE3_CBC_WITH_MD5
 x080080 RC4-64-MD5                        RSA       RC4        64       SSL_CK_RC4_64_WITH_MD5
 x15     EDH-RSA-DES-CBC-SHA               DH 1024   DES        56       TLS_DHE_RSA_WITH_DES_CBC_SHA
 x1a     ADH-DES-CBC-SHA                   DH 1024   DES        56       TLS_DH_anon_WITH_DES_CBC_SHA
 x62     EXP1024-DES-CBC-SHA               RSA(1024) DES        56,exp   TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
 x09     DES-CBC-SHA                       RSA       DES        56       TLS_RSA_WITH_DES_CBC_SHA
 x61     EXP1024-RC2-CBC-MD5               RSA(1024) RC2        56,exp   TLS_RSA_EXPORT1024_WITH_RC2_56_MD5
 x060040 DES-CBC-MD5                       RSA       DES        56       SSL_CK_DES_64_CBC_WITH_MD5
 x64     EXP1024-RC4-SHA                   RSA(1024) RC4        56,exp   TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
 x60     EXP1024-RC4-MD5                   RSA(1024) RC4        56,exp   TLS_RSA_EXPORT1024_WITH_RC4_56_MD5
 x14     EXP-EDH-RSA-DES-CBC-SHA           DH(512)   DES        40,exp   TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
 x19     EXP-ADH-DES-CBC-SHA               DH(512)   DES        40,exp   TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
 x08     EXP-DES-CBC-SHA                   RSA(512)  DES        40,exp   TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
 x06     EXP-RC2-CBC-MD5                   RSA(512)  RC2        40,exp   TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
 x040080 EXP-RC2-CBC-MD5                   RSA(512)  RC2        40,exp   SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
 x17     EXP-ADH-RC4-MD5                   DH(512)   RC4        40,exp   TLS_DH_anon_EXPORT_WITH_RC4_40_MD5
 x03     EXP-RC4-MD5                       RSA(512)  RC4        40,exp   TLS_RSA_EXPORT_WITH_RC4_40_MD5
 x020080 EXP-RC4-MD5                       RSA(512)  RC4        40,exp   SSL_CK_RC4_128_EXPORT40_WITH_MD5

 Testing all locally available ciphers per protocol against the server, ordered by encryption strength

Hexcode  Cipher Suite Name (OpenSSL)       KeyExch.  Encryption Bits     Cipher Suite Name (RFC)
-----------------------------------------------------------------------------------------------------------------
SSLv2
 x030080 RC2-CBC-MD5                       RSA       RC2        128      SSL_CK_RC2_128_CBC_WITH_MD5
 x010080 RC4-MD5                           RSA       RC4        128      SSL_CK_RC4_128_WITH_MD5
 x0700c0 DES-CBC3-MD5                      RSA       3DES       168      SSL_CK_DES_192_EDE3_CBC_WITH_MD5
 x080080 RC4-64-MD5                        RSA       RC4        64       SSL_CK_RC4_64_WITH_MD5
 x060040 DES-CBC-MD5                       RSA       DES        56       SSL_CK_DES_64_CBC_WITH_MD5
 x040080 EXP-RC2-CBC-MD5                   RSA(512)  RC2        40,exp   SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
 x020080 EXP-RC4-MD5                       RSA(512)  RC4        40,exp   SSL_CK_RC4_128_EXPORT40_WITH_MD5
SSLv3
 x39     DHE-RSA-AES256-SHA                DH 1024   AES        256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 x3a     ADH-AES256-SHA                    DH 1024   AES        256      TLS_DH_anon_WITH_AES_256_CBC_SHA
```

TLS & SSL Vulnerability Testing

Common TLS & SSL Vulnerabilities are listed below in discovered CVE date order, each vulnerability has a definition along with automated and manual (where possible) test instructions.

SWEET32 (CVE-2016-2183)

Definition

Legacy block ciphers that use a 64-bit block size such as Triple-DES (3DES) are vulnerable to a practical collision attack when used in CBC mode of operation. When CBC mode of operation is used a simple birthday attack can be used identify 64-Bit block cipher collisions. When a collision occurs it means the input is the same as the output, making it possible to perform a BEAST style attack to exfiltrate encrypted data.

The authors Karthik Bhargavan and Gaetan Leurent were able to run JavaScript in a web browser (acts as the MiTM) and send a large enough amount of data to cause a collision, then use this information to recover sessions cookies.

Testing for SWEET32

Identify if the server offers Triple-DES ciphers, if the server supports Triple-DES it's vulnerable to SWEET32.

SWEET32 testssl.sh

Identify weak ciphers using testssl.sh:

```
./testssl.sh --ciphers TARGET
```

If the output shows Tripe DES Ciphers, like in the screenshot below, the target server is vulnerable to SWEET32:

```
 Testing ~standard cipher lists

Null Ciphers                    not offered (OK)
Anonymous NULL Ciphers          not offered (OK)
Anonymous DH Ciphers            not offered (OK)
40 Bit encryption               not offered (OK)
56 Bit encryption               not offered (OK)
Export Ciphers (general)        not offered (OK)
Low (<=64 Bit)                  not offered (OK)
DES Ciphers                     not offered (OK)
Medium grade encryption         not offered (OK)
Triple DES Ciphers              not offered (OK)
High grade encryption           offered (OK)
```

Additionally you can enumerate all ciphers offered by a server for each protocol using:

```
./testssl.sh -E TARGET
```

Any ciphers using 3DES are vulnerable to SWEET32.

```
 Testing all locally available ciphers per protocol against the server, ordered by encryption strength

Hexcode  Cipher Suite Name (OpenSSL)       KeyExch.  Encryption Bits     Cipher Suite Name (RFC)
------------------------------------------------------------------------------------------------------------
SSLv2
SSLv3
TLS 1
 xc014   ECDHE-RSA-AES256-SHA             ECDH 256   AES       256      TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
 x39     DHE-RSA-AES256-SHA               DH 2048    AES       256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 x35     AES256-SHA                       RSA        AES       256      TLS_RSA_WITH_AES_256_CBC_SHA
 xc013   ECDHE-RSA-AES128-SHA             ECDH 256   AES       128      TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
 x33     DHE-RSA-AES128-SHA               DH 2048    AES       128      TLS_DHE_RSA_WITH_AES_128_CBC_SHA
 x2f     AES128-SHA                       RSA        AES       128      TLS_RSA_WITH_AES_128_CBC_SHA
 xc012   ECDHE-RSA-DES-CBC3-SHA           ECDH 256   3DES      168      TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
 x16     EDH-RSA-DES-CBC3-SHA             DH 2048    3DES      168      TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
 x0a     DES-CBC3-SHA                     RSA        3DES      168      TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS 1.1
 xc014   ECDHE-RSA-AES256-SHA             ECDH 256   AES       256      TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
 x39     DHE-RSA-AES256-SHA               DH 2048    AES       256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 x35     AES256-SHA                       RSA        AES       256      TLS_RSA_WITH_AES_256_CBC_SHA
 xc013   ECDHE-RSA-AES128-SHA             ECDH 256   AES       128      TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
 x33     DHE-RSA-AES128-SHA               DH 2048    AES       128      TLS_DHE_RSA_WITH_AES_128_CBC_SHA
 x2f     AES128-SHA                       RSA        AES       128      TLS_RSA_WITH_AES_128_CBC_SHA
 xc012   ECDHE-RSA-DES-CBC3-SHA           ECDH 256   3DES      168      TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
 x16     EDH-RSA-DES-CBC3-SHA             DH 2048    3DES      168      TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
 x0a     DES-CBC3-SHA                     RSA        3DES      168      TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS 1.2
 xc030   ECDHE-RSA-AES256-GCM-SHA384      ECDH 256   AESGCM    256      TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
 xc028   ECDHE-RSA-AES256-SHA384          ECDH 256   AES       256      TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
 xc014   ECDHE-RSA-AES256-SHA             ECDH 256   AES       256      TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
 x9f     DHE-RSA-AES256-GCM-SHA384        DH 2048    AESGCM    256      TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
 x6b     DHE-RSA-AES256-SHA256            DH 2048    AES       256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
 x39     DHE-RSA-AES256-SHA               DH 2048    AES       256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 x9d     AES256-GCM-SHA384                RSA        AESGCM    256      TLS_RSA_WITH_AES_256_GCM_SHA384
 x3d     AES256-SHA256                    RSA        AES       256      TLS_RSA_WITH_AES_256_CBC_SHA256
 x35     AES256-SHA                       RSA        AES       256      TLS_RSA_WITH_AES_256_CBC_SHA
 xc02f   ECDHE-RSA-AES128-GCM-SHA256      ECDH 256   AESGCM    128      TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
 xc027   ECDHE-RSA-AES128-SHA256          ECDH 256   AES       128      TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
 xc013   ECDHE-RSA-AES128-SHA             ECDH 256   AES       128      TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
 x9e     DHE-RSA-AES128-GCM-SHA256        DH 2048    AESGCM    128      TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
 x67     DHE-RSA-AES128-SHA256            DH 2048    AES       128      TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
 x33     DHE-RSA-AES128-SHA               DH 2048    AES       128      TLS_DHE_RSA_WITH_AES_128_CBC_SHA
 x9c     AES128-GCM-SHA256                RSA        AESGCM    128      TLS_RSA_WITH_AES_128_GCM_SHA256
 x3c     AES128-SHA256                    RSA        AES       128      TLS_RSA_WITH_AES_128_CBC_SHA256
 x2f     AES128-SHA                       RSA        AES       128      TLS_RSA_WITH_AES_128_CBC_SHA
```

SWEET32 Nmap Testing

Nmap can also be used to enumerate the ciphers of a server, the NSE plugin will also notify if any 64-Bit block ciphers are available.

```
nmap --script=ssl-enum-ciphers -p443 TARGET
```

If you see the following in the output, 64-Bit block ciphers were discovered

```
warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
```

Manually Testing for SWEET32

Use the Nmap enum-ciphers NSE script documented above to enumerate the ciphers

---

DROWN (CVE-2016-0800)

DROWN Definition

DROWN (Decrypting RSA using Obsolete and Weakened eNcryption), The most general variant of the DROWN attack

export-grade cryptography that was introduced to comply with 1990s-era U.S. government restrictions (EXPORT grade encryption is detailed in the FREAK vulnerability description below).

Testing for DROWN

testssl.sh DROWN Testing

```
./testssl.sh -D TARGET
```

Nmap DROWN Testing

```
nmap -p 443 -sV --script=sslv2-drown
```

FREAK (CVE-2015-0204)

FREAK Definition

FREAK (Factoring RSA Export Keys), exploits a cryptographic weakness within TLS / SSL that was originally introduced by the US government decades earlier. The idea behind the RSA_EXPORT keys was to allow exports to contain encryption that could not be be broken by average computing resources but could be broken by the NSA.

The FREAK attack performs a downgrade attack (forces a server to use a weaker cipher), when combined with a Man-in-The-Middle (MiTM) type attack, this allows an attacker to capture data and break the decryption of the weak keys.

Automated testing for the FREAK Attack

testssl.sh FREAK Attack Testing

```
./testssl -F TARGET
```

Manual testing for the FREAK Attack

Manually enumerate the servers ciphers using either `./testssl.sh -E TARGET` or `nmap -p 443 --script=ssl-enum-ciphers TARGET`, ensure none of the following ciphers supported by the server contain: `EXPORT`.

Example:

Testing all locally available ciphers per protocol against the server, ordered by encryption strength

Hexcode	Cipher Suite Name (OpenSSL)	KeyExch.	Encryption	Bits	Cipher Suite Name (RFC)
SSLv2					
x030080	RC2-CBC-MD5	RSA	RC2	128	SSL_CK_RC2_128_CBC_WITH_MD5
x010080	RC4-MD5	RSA	RC4	128	SSL_CK_RC4_128_WITH_MD5
x0700c0	DES-CBC3-MD5	RSA	3DES	168	SSL_CK_DES_192_EDE3_CBC_WITH_MD5
x080080	RC4-64-MD5	RSA	RC4	64	SSL_CK_RC4_64_WITH_MD5
x060040	DES-CBC-MD5	RSA	DES	56	SSL_CK_DES_64_CBC_WITH_MD5
x040080	EXP-RC2-CBC-MD5	RSA(512)	RC2	40,exp	SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
x020080	EXP-RC4-MD5	RSA(512)	RC4	40,exp	SSL_CK_RC4_128_EXPORT40_WITH_MD5
SSLv3					
x39	DHE-RSA-AES256-SHA	DH 1024	AES	256	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
x3a	ADH-AES256-SHA	DH 1024	AES	256	TLS_DH_anon_WITH_AES_256_CBC_SHA
x35	AES256-SHA	RSA	AES	256	TLS_RSA_WITH_AES_256_CBC_SHA
x33	DHE-RSA-AES128-SHA	DH 1024	AES	128	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
x34	ADH-AES128-SHA	DH 1024	AES	128	TLS_DH_anon_WITH_AES_128_CBC_SHA
x2f	AES128-SHA	RSA	AES	128	TLS_RSA_WITH_AES_128_CBC_SHA
x18	ADH-RC4-MD5	DH 1024	RC4	128	TLS_DH_anon_WITH_RC4_128_MD5
x05	RC4-SHA	RSA	RC4	128	TLS_RSA_WITH_RC4_128_SHA
x04	RC4-MD5	RSA	RC4	128	TLS_RSA_WITH_RC4_128_MD5
x16	EDH-RSA-DES-CBC3-SHA	DH 1024	3DES	168	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
x1b	ADH-DES-CBC3-SHA	DH 1024	3DES	168	TLS_DH_anon_WITH_3DES_EDE_CBC_SHA
x0a	DES-CBC3-SHA	RSA	3DES	168	TLS_RSA_WITH_3DES_EDE_CBC_SHA
x15	EDH-RSA-DES-CBC-SHA	DH 1024	DES	56	TLS_DHE_RSA_WITH_DES_CBC_SHA
x1a	ADH-DES-CBC-SHA	DH 1024	DES	56	TLS_DH_anon_WITH_DES_CBC_SHA
x62	EXP1024-DES-CBC-SHA	RSA(1024)	DES	56,exp	TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
x09	DES-CBC-SHA	RSA	DES	56	TLS_RSA_WITH_DES_CBC_SHA
x61	EXP1024-RC2-CBC-MD5	RSA(1024)	RC2	56,exp	TLS_RSA_EXPORT1024_WITH_RC2_56_MD5
x64	EXP1024-RC4-SHA	RSA(1024)	RC4	56,exp	TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
x60	EXP1024-RC4-MD5	RSA(1024)	RC4	56,exp	TLS_RSA_EXPORT1024_WITH_RC4_56_MD5
x14	EXP-EDH-RSA-DES-CBC-SHA	DH(512)	DES	40,exp	TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
x19	EXP-ADH-DES-CBC-SHA	DH(512)	DES	40,exp	TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
x08	EXP-DES-CBC-SHA	RSA(512)	DES	40,exp	TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
x06	EXP-RC2-CBC-MD5	RSA(512)	RC2	40,exp	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
x17	EXP-ADH-RC4-MD5	DH(512)	RC4	40,exp	TLS_DH_anon_EXPORT_WITH_RC4_40_MD5
x03	EXP-RC4-MD5	RSA(512)	RC4	40,exp	TLS_RSA_EXPORT_WITH_RC4_40_MD5

Logjam (CVE-2015-4000)

vulnerability allows a Man-in-The-Middle (MiTM) attacker to perform a downgrade attack and use the Diffie-Hellman export ciphers (DHE_EXPORT).

Automated testing for Logjam

testssl.sh test for Logjam

```
./testssl.sh -J TARGET
```

Manually testing for Logjam

Disable EXPORT ciphers, Instructions are the same as the FREAK attack, documented above.

Manually enumerate the ciphers suites offered by the server, using either `./testssl.sh -E TARGET` or `nmap -p 443 --script=ssl-enum-ciphers TARGET`.

Heartbleed (CVE-2014-0160)

Heartbleed Definition

A flaw was found in the way OpenSSL handled TLS and DTLS Heartbeat Extension packets that allows an attacker to disclose information from encrypted TLS / DTLS data. A malicious client could send a specially crafted TLS or DTLS Heartbeat packet to disclose a limited portion of memory per request from a connected client or server.

The disclosed portions of memory could include sensitive information such as private keys (used by service providers to encrypt data), names, usernames and passwords of actual users. Allowing attackers to potentially eavesdrop on communications, impersonate users and services and steal data.

Automated testing for Heartbleed

```
# ./testssl.sh -B 10.0.1.159

###########################################################
    testssl.sh       2.8rc1 from https://testssl.sh/dev/
    (424cf23 2016-08-09 10:35:58 -- 1.531)

      This program is free software. Distribution and
             modification under GPLv2 permitted.
      USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

       Please file bugs @ https://testssl.sh/bugs/

###########################################################

 Using "OpenSSL 1.0.2-chacha (1.0.2i-dev)" [~183 ciphers]
 on kali:./bin/openssl.Linux.x86_64
 (built: "Jun 22 19:32:29 2016", platform: "linux-x86_64")


 Start 2017-01-20 22:43:38    -->> 10.0.1.159:443 (10.0.1.159) <> 10.0.1.159:443 (10.0.1.159)
```

Testing for Heartbleed using Nmap

```
# nmap -p 443 --script ssl-heartbleed --script-args vulns.showall 10.0.1.159

Starting Nmap 7.25BETA2 ( https://nmap.org ) at 2017-01-20 22:45 EST
Nmap scan report for 10.0.1.159
```

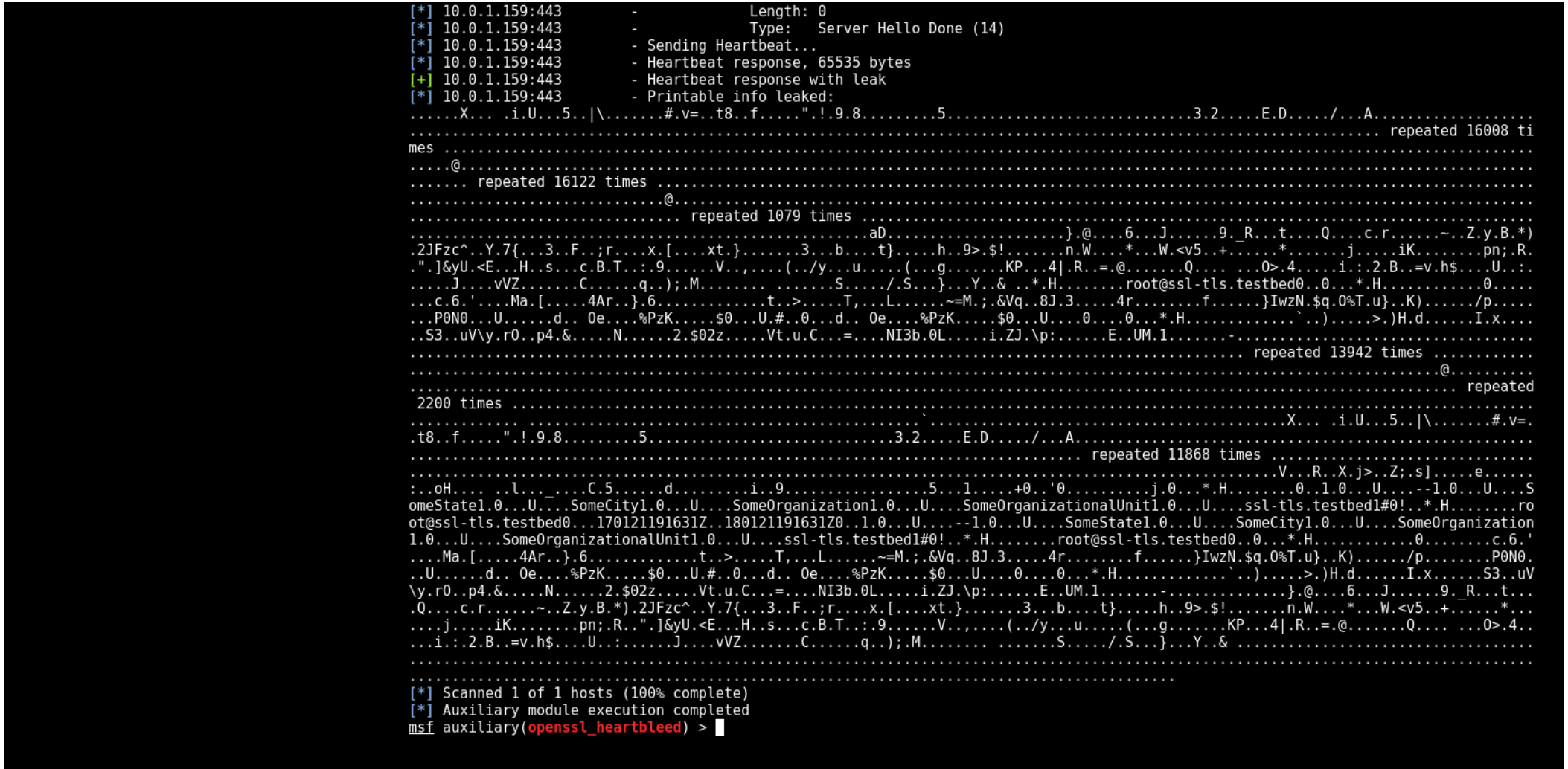
```
| ssl-heartbleed:
|   VULNERABLE:
|   The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic softwar
|     State: VULNERABLE
|     Risk factor: High
|       OpenSSL versions 1.0.1 and 1.0.2-beta releases (including 1.0.1f and 1.0.2-beta1) of O
|
|     References:
|       http://www.openssl.org/news/secadv_20140407.txt
|       http://cvedetails.com/cve/2014-0160/
|_      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160
MAC Address: 00:0C:29:35:3D:E8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.36 seconds
```

Manually testing for Heartbleed

Use Metasploit to validate the existed of Heartbleed, using the verbose setting will show the Heartbleed exposed memory leak.

```
msf> use auxiliary/scanner/ssl/openssl_heartbleed
msf> set rhosts TARGET-ADDRESS
msf> set verbose true
msf> run
```



POODLE SSLv3 (CVE-2014-3566)

POODLE Definition

The POODLE attack (Padding Oracle On Downgraded Legacy Encryption) was discovered by Google's Security Team on October 14, 2014. The vulnerability exploits the way SSLv3 handles padded bytes when used with (cipher block chaining) CBC mode of operation.

The flaw allows a Man-in-The-Middle (MiTM) attacker to decrypt a selected byte of a cipher text in as few as 256 SSLv3 connections, if they are able to force a victim application to repeatedly send the same data over newly created SSL 3.0 connections.

Automated Testing For POODLE

Testing for POODLE using Nmap

```
# nmap -p 443 --script ssl-poodle --script-args vulns.showall 10.0.1.159

Starting Nmap 7.25BETA2 ( https://nmap.org ) at 2017-01-20 22:50 EST
Nmap scan report for 10.0.1.159
Host is up (0.00037s latency).
PORT    STATE SERVICE
443/tcp open  https
| ssl-poodle:
|   VULNERABLE:
|   SSL POODLE information leak
|     State: VULNERABLE
|     IDs:  CVE:CVE-2014-3566  OSVDB:113251
|           The SSL protocol 3.0, as used in OpenSSL through 1.0.1i and other
|           products, uses nondeterministic CBC padding, which makes it easier
|           for man-in-the-middle attackers to obtain cleartext data via a
|           padding-oracle attack, aka the "POODLE" issue.
```

```
|     References:
|       http://osvdb.org/113251
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566
|       https://www.imperialviolet.org/2014/10/14/poodle.html
|_      https://www.openssl.org/~bodo/ssl-poodle.pdf
MAC Address: 00:0C:29:35:3D:E8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.35 seconds
```

Testing for POODLE using testssl.sh

```
./testssl.sh -O 10.0.1.159

###########################################################
    testssl.sh       2.8rc1 from https://testssl.sh/dev/
    (424cf23 2016-08-09 10:35:58 -- 1.531)

      This program is free software. Distribution and
             modification under GPLv2 permitted.
      USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

       Please file bugs @ https://testssl.sh/bugs/

###########################################################

 Using "OpenSSL 1.0.2-chacha (1.0.2i-dev)" [~183 ciphers]
 on kali:./bin/openssl.Linux.x86_64
 (built: "Jun 22 19:32:29 2016", platform: "linux-x86_64")
```

Manual Testing for POODLE

The version of openssl that ships with Kali no longer support SSLv3. Use the binary within `testssl.sh/bin/openssl.Linux.x86_64` to perform manual SSLv3 testing.

```
./openssl.Linux.x86_64 s_client -ssl3 -connect 10.0.1.159:443
```

If the handshake completes, the server is vulnerable to POODLE.

Example output for a server that is vulnerable to POODLE (certificate snipped out of the response):

```
---
No client certificate CA names sent
Server Temp Key: DH, 1024 bits
---
SSL handshake has read 1398 bytes and written 373 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : SSLv3
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID: 0450660185C7B2623CB2145A1C6655BDD8CC281F882C3B9E0ED35E88360639BA
```

```
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1484971085
    Timeout   : 7200 (sec)
    Verify return code: 9 (certificate is not yet valid)
---
```

If the server is not vulnerable to POODLE the handshake will fail, giving an error such as:

```
CONNECTED(00000003)
28395584:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert handshake failure:s3_pkt.c:14
28395584:error:1409E0E5:SSL routines:ssl3_write_bytes:ssl handshake failure:s3_pkt.c:656:
---
no peer certificate available
---
```

CCS Injection Vulnerability (CVE-2014-0224)

CCS Injection Definition

A weakness exists within certain versions of OpenSSL that allows clients and servers to be forced, via a specially crafted handshake packet to use weak keying material for communication. Allowing A Man-in-The-Middle attacker to

Affected OpenSSL Versions:

- OpenSSL before 0.9.8za
- OpenSSL 1.0.0 before 1.0.0m
- OpenSSL 1.0.1 before 1.0.1

Source: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0224

Automated Testing for CCS Injection

testssl.sh CCS Injection Testing

```
./testssl.sh -I TARGET
```

Nmap CCS Injection Testing

```
nmap -p 443 --script=ssl-ccs-injection TARGET
```

POODLE TLS (CVE-2014-8730)

POODLE TLS Definition

Due to TLS padding being a subset of SSLv3's, it's possible to re-purpose the POODLE attack against TLS. TLS is very strict about how its padding is formatted, however some TLS implementations do not perform the check for padding structure after decryption. Implementations that do not are vulnerable to the POODLE attack even with TLS.

Source: https://blog.qualys.com/ssllabs/2014/12/08/poodle-bites-tls

Testing for POODLE TLS

testssl.sh POODLE TLS Testing

Same as testssl.sh instructions above for POODLE SSL

Nmap POODLE TLS Testing

Same as Nmap instructions above for POODLE SSL

BREACH (CVE-2013-3587)

BREACH Attack Definition

BREACH stands for Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext. Similar to CRIME breach exploits a vulnerability within HTTP compression, allowing an attacker to identify if text exists within a page.

Basic example of how the BREACH attack works

When text repetition occurs on a page, deflate will remove repetitive terms helping reduce the size of a page. This can be used to identify existing page content, an example below on a web application that shows the username of the currently logged in user reflected within the page:

1. Enter a username you think will not exist into a search parameter
2. Note the size of the returned page
3. Send another search request for a username you think exists
4. Note the size of the returned page, if the username matched the logged in user the page size will be reduced by deflate

No traffic is actually "decrypted" by analysing the size of the responses it's possible to predict the text

In order for the BREACH attack to successfully exfiltrate data there must be a mechanism to reflect user input within the rendered page and the server must support HTTP compression.

Automated Testing for BREACH

Testing for BREACH using testssl.sh

```
./testssl.sh -T TARGET
```

Manual Testing for BREACH

Enter the following to identify if the server uses compression:

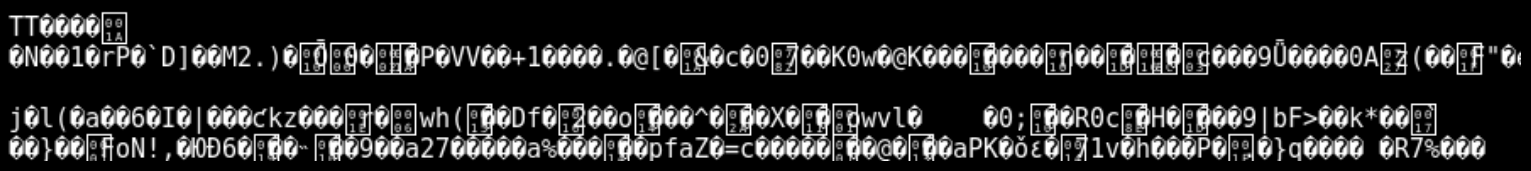
```
GET / HTTP/1.1
Host: TARGET
Accept-Encoding: compress, gzip
```

If the server returns garbled meta characters like in the screenshot below, the server supports compression and is vulnerable to BREACH:

If the target web server does not return compressed data output it is not vulnerable to BREACH and compression is disabled.

```
    Start Time: 1484979069
    Timeout   : 300 (sec)
    Verify return code: 9 (certificate is not yet valid)
---
GET / HTTP/1.1
Host: 10.0.1.158
Accept-Encoding: compress, gzip

HTTP/1.1 403 Forbidden
Date: Sat, 21 Jan 2017 23:43:14 GMT
Server: Apache/2.2.3 (CentOS)
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 2022
Connection: close
Content-Type: text/html; charset=UTF-8
```



RC4 CVE-2013-2566

The RC4 algorithm used in the TLS protocol and SSL protocol, has many single-byte biases. Making it easier for remote attackers to conduct plaintext-recovery attacks via statistical analysis of the ciphertext in a large number of sessions that use the same plaintext.

Automated RC4 Testing

testssl.sh RC4 Testing

```
./testssl.sh -4 TARGET
```

Manual RC4 Testing

Manually enumerate the servers ciphers using either `./testssl.sh -E TARGET` or `nmap -p 443 --script=ssl-enum-ciphers TARGET`, ensure none of the following ciphers supported by the server use RC4.

CRIME (CVE-2012-4929)

CRIME Attack Definition

CRIME is a TLS 1.2 vulnerability that allows Man-In-the-Middle (MiTM) attackers to identify encrypted data and (potentially) perform session hijacking. An attacker can identify encrypted data by examining the size of the ciphertext while introducing multiple payloads from the browser, when a character is matched within the header it's size will differ allowing an attacker to workout session cookies. No data is actually decrypted using the CRIME attack, a weakness within the way TLS 1.2 handles compression allows an attacker to identify when the character exists within the header by comparing the returned size.

Automated Testing for CRIME

Testing for CRIME with testssl.sh

```
./testssl.sh -C TARGET
```

Manually Testing for CRIME

The openssl client used by Kali appears to no longer allow TLS 1.2 compression. If you test with this version of openssl, the response will always be "Compression: NONE" even if the target server had TLS 1.2 compression enabled. Use the openssl.Linux.x86_64 binary that ships with testssl.sh to overcome this issue.

```
./bin/openssl.Linux.x86_64 s_client -connect 10.0.1.158:443
```

Example Output – Not Vulnerable to CRIME

If server is NOT vulnerable to CRIME. “Compression: None” indicates compression is disabled on the server and it is not vulnerable to CRIME.

```
Compression: NONE
```

Example Output – Vulnerable to CRIME

If the server IS vulnerable to CRIME:

```
Compression: zlib compression
```

Renegotiation (CVE-2009-3555)

TLS/SSL Renegotiation Vulnerability Definition

In 2009 a vulnerability was discovered that exploits the TLS & SSL protocols renegotiation process, allowing an attacker to insert data into the start of the session and compromise it’s integrity.

Conditions

- Server must not support secure renegotiation
- Server must allow client side renegotiation

Automated Testing for Renegotiation

Tests for both secure renegotiation and client side renegotiation.

```
# ./testssl.sh -R 10.0.1.159

###########################################################
    testssl.sh       2.8rc1 from https://testssl.sh/dev/
    (424cf23 2016-08-09 10:35:58 -- 1.531)

      This program is free software. Distribution and
             modification under GPLv2 permitted.
      USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

       Please file bugs @ https://testssl.sh/bugs/

###########################################################

 Using "OpenSSL 1.0.2-chacha (1.0.2i-dev)" [~183 ciphers]
 on kali:./bin/openssl.Linux.x86_64
 (built: "Jun 22 19:32:29 2016", platform: "linux-x86_64")


 Start 2017-01-21 00:04:30    -->> 10.0.1.159:443 (10.0.1.159) <> 10.0.1.159:443 (10.0.1.159)
```

Testing for Secure Renegotiation

```
openssl s_client -connect TARGET:443
```

Example when Secure Renegotiation is not enabled:

```
---
SSL handshake has read 1606 bytes and written 503 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
```

Example when Secure Renegotiation is enabled:

```
---
SSL handshake has read 1560 bytes and written 495 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-GCM-SHA384
Server public key is 1024 bit
Secure Renegotiation IS supported
Compression: NONE
```

```
No ALPN negotiated
SSL-Session:
```

Testing for Client Initiated Secure Renegotiation

```
openssl s_client -connect TARGET:443
```

Type:

```
HEAD / HTTP/1.1
R
```

Hit return, if you see the response:

```
HEAD / HTTP/1.1
R
RENEGOTIATING
```

The server allows client renegotiation.

If you see the response:

```
RENEGOTIATING
139681067286040:error:1409E0E5:SSL routines:ssl3_write_bytes:ssl handshake failure:s3_pkt.c:65
```

The server does not support client renegotiation.

TLS & SSL Certificates

The server certificate while not required for encryption should be assessed for configuration errors and weak cryptographic signing, below is a check list for certificate checking:

Pull the target servers certificate using:

```
openssl s_client -connect TARGET:443 | openssl x509 -noout -text
```

Certificate Cheat Sheet

Name	Description
Identify Certificate Issuer	Insure the certificate authority (CA) is from a trusted source, self signed certificates should not be used as they allow for man in the middle attacks (unless they are internal and signed against an internal CA).
Signature	The algorithm used to insure the certificates integrity, you should insure this is cryptographically

Name	Description
Public Key	The key length should be long enough to insure it cannot be compromised, the minimum should be 2048 bit.
Not Before	Certificate start date.
Not After	Certificate end date.
Subject **Subject Alternative Name**	**Subject** should list the DNS name the certificate relates too, if this is incorrect browsers will throw an error. **Subject Alternative Name** should list DNS names for wildcard certificates, all DNS names for this certificate should be listed.

HTTP Security Headers

What are HTTP Security Headers?

HTTP Security Headers, if configured correctly can provide additional security features for your domain. Below is an overview of the primary HTTP security headers:

HTTP Security Headers can be examined using Burp Suite, Curl or testssl.sh (any many other tools).

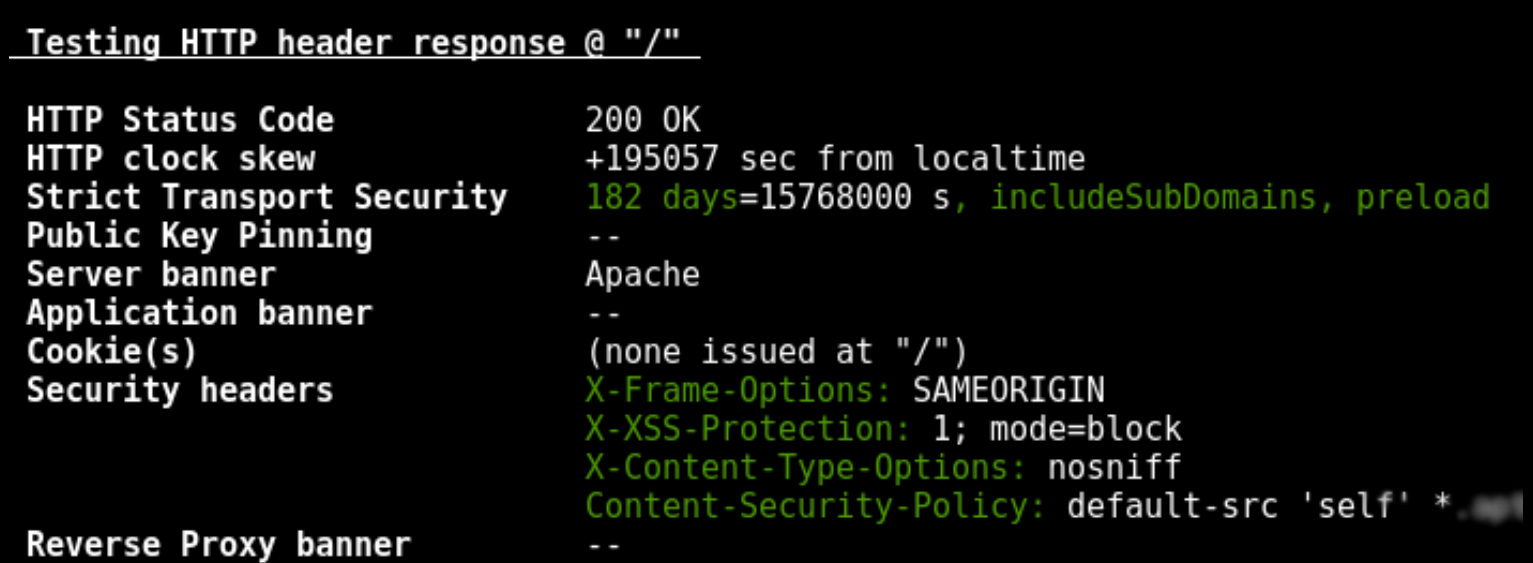
```
curl -s -D - TARGET -o /dev/null
```

Example Output:

```
HTTP/1.1 301 Moved Permanently
Date: Mon, 23 Jan 2017 16:15:51 GMT
Server: Apache
Content-Security-Policy: default-src 'self' *.target
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Location: TARGET
Cache-Control: max-age=3600
Expires: Mon, 23 Jan 2017 17:15:51 GMT
Vary: Accept-Encoding
Content-Length: 233
Content-Type: text/html; charset=iso-8859-1
```

Examining HTTP Security Headers with testssl.sh

```
./testssl.sh -H
```



HTTP Security Header Cheat Sheet

Conclusion

We hope this article has clarified the testing process for some of the more common TLS & SSL vulnerabilities. If we missed anything or you have any suggestions or techniques we would love to hear from you in the comment section below.

Name	Definition

Name	Definition
Strict-Transport-Security (HSTS)	**HTTP Strict Transport Security** (HSTS) security header forces web browsers or clients to communicate with servers only through HTTPS connections. HSTS ensures the connection only uses HTTPS and prevents Man-in-The-Middle (MiTM) attacks, downgrade attacks and cookie-hijacking. HSTS is Trust on First Use (TOFU), meaning it must send at least one insecure connection over HTTP to the host to transfer the secure header. The HSTS preload list is an effort to provide browsers with a list of sites that support HSTS to avoid the initial insecure connection.
Content-Security-Policy (CSP)	**Content-Security-Policy (CSP)** is a HTTP Security Header that helps mitigate the risk of certain types of data injection attacks such as XSS (Cross-site-scripting). CSP allows website administrators to eliminate or mitigate XSS by by defining a policy that stipulates what locations browsers should trust and allow script execution from.
X-Frame-Options	The **X-Frame-Options** header prevents website content being rendered within an **, or an . By preventing content being rendered within these elements, sites can use the X-Frame-Options to prevent clickjacking.**
X-XSS-Protection	An older HTTP Security Header that enables XSS protection in Internet Explorer, Chrome and Firefox. Although this functionality is now provided by CSP by blocking inline JS, this header can still provide protection when used with older browsers that don't support CSP.
X-Content-Type-Options	Prevents MIME-sniff type attacks by forcing the MIME type specified by the Content-Type.

Share this:

Twitter Facebook

Like

Be the first to like this.

PREVIOUS

NEXT

Leave a Reply

Enter your comment here...