이승용 - bongbongco@gmail.com

FREAK 취약점 진단

cve-2015-0204

1. 목적

FREAK 취약점의 이해와 진단 스크립트 제작

2. 개요

OpenSSL 취약점 중 취약점 발생원인을 이해하기 쉬운 FREAK 취약점을 통해서 SSL 취약점 발생 원인에 대해서 알아보고 간단한 진단 스크립트를 제작하고자 합니다.

3. 설명

3.1 소개

FREAK 취약점은 SSL서버가 수출용 RSA를 허용하도록 다운드레이드 시킨 후, 무작위 대입 공격을 통해서 RSA키를 얻어 중간자 공격을 수행하는 취약점입니다.

※ 수출용 RSA란.
'Netscape' 에서 처음 SSL을 개발할 때, 미국 정부의 암호화 시스템 해외 수출이 제한되어 있었습니다. 따라서 비교적 약한 암호화 키를 적용한 수출용 암호 시스템을 만들어야 했고, RSA 암호의 경우 키의 길이를 512비트로 제한했습니다.

3.2 취약한 버전

OpenSSL 0.9.8zd 이전 버전
OpenSSL1.0.0 ~ OpenSSL1.0.0p 이전 버전
OpenSSL1.0.1 ~ OpenSSL1.0.1k 이전 버전
이 밖에 취약한 OpenSSL을 사용한 구글 , 애플 등의 제품

3.3 취약점 패치 방법

OpenSSL 0.9.8 사용자는 OpenSSL 0.9.8zd로 업그레이드
OpenSSL1.0.0 사용자는 OpenSSL1.0.0p로 업그레이드
OpenSSL1.0.1 사용자는 OpenSSL1.0.1k로 업그레이드

3.4 “수출용” RSA를 이용한 MITM 공격 원리

1. 클라이언트의 Hello 메시지에, ‘표준 RSA’ 암호스위트(ciphersuites)를 요청한다.
2. MITM 공격자는 이 메시지를 ‘수출용 RSA’를 요청하는 것으로 변조한다.
3. 서버는 Long-term key로 서명 된 512비트의 수출용 RSA키로 응답한다.
4. OpenSSL과 SecureTransport의 버그로 인하여, 클라이언트는 이 약한 키를 받아들인다.
5. 공격자는 이에 상응하는 RSA 복호화 키를 복구한다.
6. 클라이언트가 ‘pre-master secret’을 암호화하여 서버로 보내면, 공격자는 TLS ‘master secret’을 복구하기 위해 이를 복호화한다.
7. 여기서부터, 공격자는 플레인 텍스트(plain text)를 보게 되며, 어느 것이든 주입할 수 있게 된다.

공격자가 일단 RSA의 복호화 키 복구에 성공한다면, 공격받은 해당 서버가 재시작 되기 전 까지 MITM 공격은 유효하다.

3.5 진단 스크립트

OpenSSL을 이용한 진단

OpenSSL을 이용하면 쉽게 진단을 수행할 수 있습니다. 기본적으로 Linux 나 Mac OS의 경우 OpenSSL 이 설치되어 있어 별도의 설치과정이 필요하지 않습니다.

※ OpenSSL 설치 방법

Windows : https://code.google.com/archive/p/openssl-for-windows/downloads

Ubuntu : apt-get install openssl

CentOS : yum install openssl

MacOS : brew install openssl

- **취약한 서버**

```
Seungyongui-MBP:~ seungyonglee$ openssl s_client -connect [redacted] -cipher EXPORT
CONNECTED(00000003)
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 273 bytes and written 199 bytes
---
New, TLSv1/SSLv3, Cipher is EXP-ADH-DES-CBC-SHA
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1
    Cipher    : EXP-ADH-DES-CBC-SHA
    Session-ID: 57B8290791A67981AAA7C3BE18B5FC1B688C3E2EF77BAF27F46B99ED6783F35E
    Session-ID-ctx:
    Master-Key: 826C5CFE1718931583761538347 1A4E9983B435734BFEAB82218C7915EE37999A73A6A1FD3B93AF
9A78FF5E53C38AC4C
    Key-Arg   : None
    Start Time: 1471686919
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

• **취약하지 않은 서버**

```
Seungyongui-MacBook-Pro:~ seungyonglee$ openssl s_client -connect [redacted] -cipher EXPORT ]
CONNECTED(00000003)
13798:error:14077410:SSL routines:SSL23_GET_SERVER_HELLO:sslv3 alert handshake failure:/BuildRoo
t/Library/Caches/com.apple.xbs/Sources/OpenSSL098/OpenSSL098-59.60.1/src/ssl/s23_clnt.c:593:
```

스크립트 파일은 SSLChecker.py와 Freak.py로 구성됩니다. 추후 다른 SSL 취약점 진단 모듈 추가를 위해서 분리하여 구성하였습니다. 스크립트 사용 대신 'openssl s_client -connect [IP]:[PORT] -cipher EXPORT' 명령어를 직접 입력하셔도 쉽게 진단 결과를 확인할 수 있습니다. s_client 는 SSL 통신 가능한 HOST를 분석하기 위한 명령어 입니다. 옵션 중 '-cipher' 는 서버와 클라이언트 간에 이용할 암호화 목록을 지정하는 것으로 수출용 암호를 선택하여 수출용 암호를 지원하는 지 확인할 수 있습니다.

- 리눅스 환경에서 제거되어야 할 암호화 방식:
  - EXP-DES-CBC-SHA
  - EXP-RC2-CBC-MD5
  - EXP-RC4-MD5
  - EXP-EDH-RSA-DES-CBC-SHA
  - EXP-EDH-DSS-DES-CBC-SHA
  - EXP-ADH-DES-CBC-SHA
  - EXP-ADH-RC4-MD5

- 윈도우 환경에서 제거되어야 할 암호화 방식:
  - SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA
  - SSL_RSA_EXPORT1024_WITH_RC4_56_SHA
  - SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
  - SSL_RSA_EXPORT_WITH_RC4_40_MD5
  - TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
  - TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
  - TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
  - TLS_RSA_EXPORT_WITH_RC4_40_MD5

Freak.py

```
#!/usr/bin/env python
#_*_ coding: utf-8 _*_
import sys
import subprocess
from socket import *


class FreakCheckTarget:
    def __init__(self):
```

```
        self.ip = ''
        self.port = ''

    def set(self, targetInfo):
        self.ip, self.port = targetInfo.split(':')

class FreakChecker:
    def __init__(self):
        self.judge = ''
        self.platform = sys.platform

    def run(self, ip, port):
        try:
            if 'win32' in self.platform:
                return 'Please Using linux'
            else:
                self.judge = subprocess.Popen(['openssl', 's_client', '-connect', ip+':'+str(port),'-cipher','EXPORT'],
stderr=subprocess.STDOUT, stdout=subprocess.PIPE)
            self.judge = self.judge.communicate()[0]
            return self.judge
        except:
            return 'Error Freak Check'


def FreakCheck(targetInfo):
    freakCheckTarget = FreakCheckTarget()
    freakChecker = FreakChecker()

    freakCheckTarget.set(targetInfo)

    checkResult = freakChecker.run(freakCheckTarget.ip, freakCheckTarget.port)

    if 'Please Using linux' in checkResult:
        print 'Not Support Windows'
    elif 'Error Freak Check' in checkResult:
        print '[Error] Failed Freak check'
    elif 'Cipher is EXP' in checkResult:
        print '수출용 RSA를 지원하고 있습니다.'
    else:
        print '수출용 RSA를 지원하고 있지 않습니다.'
```

스크립트의 주된 기능은 'openssl s_client -connect [IP]:[PORT] -cipher EXPORT' 명령어 수행과 수행 결과에서 'Cipher is EXP' 문구 포함 여부를 확인하여 수출용 RSA를 지원하는 지를 확인하는 것 입니다.

Scapy-ssl_tls 를 이용한 진단

Python의 Scapy API는 손 쉽게 네트워크 패킷 조작을 할 수 있도록 도와줍니다. 추가로 Scapy-ssl_tls를 이용하게 되면 암호화 통신까지 간단히 수행할 수 있는 데, 이를 이용하여 진단을 수행하도록 하겠습니다. 수행 방법은 소켓 통신을 이용하여 서버와 연결 후, Scapy-ssl_tls API 로 취약한 암호화 기법으로 구성한 패킷을 생성하여 해당 패킷에 대한 응답 여부로 (취약한 암호화 기법을 지원하는 지 여부) 취약함을 판단합니다. 아래 이미지는 RSA_EXPORT_WITH_RC4_40_MD5 암호화 기법을 이용하여 연결을 시도하는 것 입니다.

```
###[ SSL/TLS ]###
  \records   \
   |###[ TLS Record ]###
   |  content_type= handshake
   |  version   = TLS_1_1
   |  length    = 0x2d
   |###[ TLS Handshake ]###
   |     type      = client_hello
   |     length    = 0x29
   |###[ TLS Client Hello ]###
   |        version   = TLS_1_1
   |        gmt_unix_time= 1471886284
   |        random_bytes= '=\x99\x11\xccA\xf9=\xde\xae\x03\xd5\x9b\xd9\x1e\xe0\x17g\xd8\x05\\W\xa1{j\xcb!k\xad'
   |        session_id_length= 0x0
   |        session_id= ''
   |        cipher_suites_length= 0x2
   |        cipher_suites= ['RSA_EXPORT_WITH_RC4_40_MD5']
   |        compression_methods_length= 0x1
   |        compression_methods= ['NULL']
   |        \extensions\
```

Freak_Scapy.py

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
from scapy import *
from scapy_ssl_tls.ssl_tls import *
from scapy_ssl_tls.ssl_tls_crypto import *
import socket

BUFFER_SIZE = 1024
TLS_EXPORT_CIPHERS = [
    TLSCipherSuite.RSA_EXPORT_WITH_RC4_40_MD5
   ,TLSCipherSuite.RSA_EXPORT_WITH_RC2_CBC_40_MD5
   ,TLSCipherSuite.RSA_EXPORT_WITH_DES40_CBC_SHA
   ,TLSCipherSuite.RSA_EXPORT1024_WITH_RC4_56_MD5
   ,TLSCipherSuite.RSA_EXPORT1024_WITH_RC2_CBC_56_MD5
   ,TLSCipherSuite.RSA_EXPORT1024_WITH_DES_CBC_SHA
   ,TLSCipherSuite.RSA_EXPORT1024_WITH_RC4_56_SHA
   ]

class ScapyFreakCheckTarget:
    def __init__(self, targetInfo):
        self.ip, self.port = targetInfo.split(':')
        self.target = (self.ip, int(self.port))

class ScapyFreakChecker:
    def __init__(self):
        self.checker = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.message = ''
        self.response = ''

    def Connect(self,destination):
        self.checker.connect(destination)

    def Send(self):
        self.checker.sendall(str(self.message))
        self.response = self.checker.recv(BUFFER_SIZE*8)

    def CommunicationView(self):
        self.message.show()
        SSL(str(self.message)).show()
        print 'Payload'
        print 'received, %d -- %s' %(len(self.response), repr(self.response))
```

```
        SSL(str(repr(self.response))).show()

    def Close(self):
        self.checker.close()


def ScapyFreakCheck(targetInfo):
    scapyFreakCheckTarget = ScapyFreakCheckTarget(targetInfo)
    scapyFreakChecker = ScapyFreakChecker()


    scapyFreakChecker.Connect(scapyFreakCheckTarget.target)

    for tlsExportCipher in TLS_EXPORT_CIPHERS:
        scapyFreakChecker.message = TLSRecord(version=TLSVersion.TLS_1_1)/TLSHandshake()/
TLSClientHello(version=TLSVersion.TLS_1_1, cipher_suites=tlsExportCipher)
        scapyFreakChecker.Send()
        #scapyFreakChecker.CommunicationView()
        print '\n=====================================================\n'

        if not len(scapyFreakChecker.response) < 10:
            print 'Server is Supported ' + TLS_CIPHER_SUITES.get(tlsExportCipher)
        else:
            print 'Server is not Supported ' + TLS_CIPHER_SUITES.get(tlsExportCipher)
            print '\n=====================================================\n'

    scapyFreakChecker.Close()
```

반환되는 메시지의 길이를 확인하여 해당 암호화 기법의 지원 여부를 확인하였습니다.

4. 결론

서버에 Freak 취약점의 존재 유무를 확인하기 위해서는 수출용 암호화 기법을 지원하는 지 확인하는 것이 핵심이었습니다. 단지 암호화 통신을 하고 있는 것만으로는 부족하고 암호화 기법 종류에 대한 이해와 정리가 필요하겠다는 생각이 듭니다. 아직은 많이 부족하지만 추후 업무에 활용할 수 있도록 Openssl 에서 발생한 대표적인 취약점들의 진단 기능을 추가하고 사용하기 편하게 다듬어 '모의해킹 도구' 게시판을 통해 공유하도록 하겠습니다.