سرریز بافر (Buffer Overflow) چیست ؟

نویسنده بهرنگ فولادی
Behrang@hat-squad.com

1382/8/18

www.hat-squad.com
گروه امنیتی Hat-Squad
(مشاور و ارائه دهنده سرویسهای امنیتی)
Service@hat-squad.com

Copyright © 2003, Hat-Squad Security Group

" درج تمام یا قسمتی از مطالب این مقاله تنها با اجازه نویسنده آن مقدور می باشد "

سرريز بافر (Overflow Buffer) از قديميترين مشكلات امنيتي سيستمهاي كامپيوتري بوده است. در حال حاضر اگر به ضعفهاي امنيتي نرم افزارهاي مختلف كه در سايتهايي مثل SecurityFocus ثبت شده اند ،نگاهي بياندازيد ، متوجه مي شويد كه حداقل 1/3 از اين ضعفها مربوط به Buffer Overflow مي شوند.اين مشكل در تمام سيستم هاي عامل ديده شده است.در اين مقاله با يك مثال ساده ، موضوع Buffer Overflow و چگونگي استفاده نفوذگرها از آن جهت نفوذ به سيستم هاي كامپيوتري بررسي خواهد شد. در پايان روشهايي براي جلوگيري از اين نوع حملات ارائه خواهد شد.برنامه هاي ذكر شده در اين مقاله به زبان C نوشته شده اند و بايد تحت سيستم عامل Windows كامپايل شوند(در اين مقاله تنها سرريز بافر در برنامه هاي Win32 مورد نظر است وبه سيستم هاي عامل ديگر مثل Unix/Linux اشاره نخواهد شد).

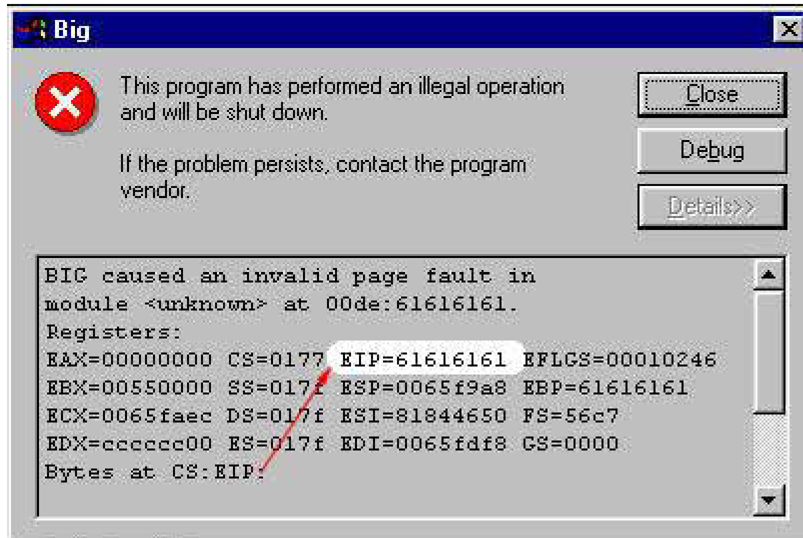
بررسي موضوع را با يك برنامه كوتاه آغاز مي كنيم:

```
/* big.exe */
#include <stdio.h>

int insecure_func (char *big) {

char insecure_buff[100];
strcpy(insecure_buff,big);
return 0;
}

int main (int argc, char *argv[]) {
char input_buff[1024];
gets(input_buff);
insecure_func(input_buff);
return 0;
}
```

برنامه ابتدا رشته ورودي از صفحه كليد را در آرايه input_buff قرار ميدهد سپس هنگامي كه تابع insecure_func با فرستادن input_buff فراخواني شود ، اين تابع مقدار موجود در input_buff رادر insecure_buff كپي خواهد كرد.نكته اصلي كوچكتر بودن اندازه insecure_buff از input_buff است. بطوريكه اگر input_buff بيشتر از 100 كاراكتر را در خود جاي داده باشد ، insecure_buff سرريز (Overflow) خواهد شد.

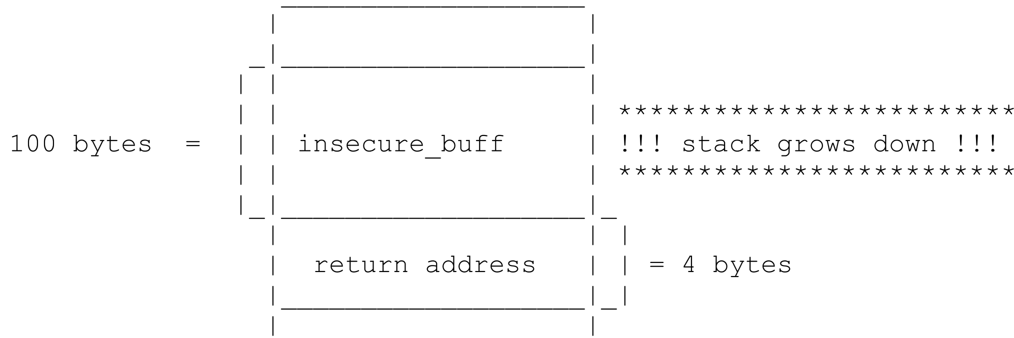
حال برنامه را كامپايل و اجرا كنيد و بيش از 100 كاراكتر 'a' را به عنوان ورودي به برنامه بدهيد. پس از زدن كليد Enter پيغام خطايي شبيه مورد زير دريافت خواهيد كرد:

Copyright © 2003, Hat-Squad Security Group **www.hat-squad.com**



پيغام خطا

ببينيم چه اتفاقي افتاده است: هر گاه يك تابع از درون يك روال ديگر فراخواني ميشود ، سيستم عامل آدرس برگشت به روال فعلي را در محلي از حافظه به نام "پشته" (Stack) قرار داده و كنترل را به روال فراخواني شده مي دهد . بدين ترتيب پس از پايان روال مذكور ، سيستم عامل با بازيابي ادرس برگشت از Stack دوباره اجراي برنامه راه به روال اصلي مي دهد. در مثال بالا آدرس برگشت قبل از وقوع سرريز (پيش از اجراي دستور strcpy) به دستور return 0 اشاره دارد. بنابراين قبل از وقوع سرريز Stack داراي ساختار زير است :



ساختار Stack

پشته (stack) هميشه به سمت آدرسهاي پايين تر رشد مي كند. هنگام شروع اجراي برنامه، سيستم عامل 100 بايت براي بافر insecure_buff كنار مي گذارد. واضح است كه اجرا شدن دستور strcpy در حالتي كه اندازه رشته موجود در big بزرگتر از insecure_buff باشد ، باعث تغيير آدرس برگشت (return address) خواهد شد. اگر كمي دقت كنيد ، مي فهميد كه چرا آدرس برگشت بعد از سرريز x616161610 است. x061 كد اسكي كاراكتر 'a' در مبناي 16 است كه توسط دستور strcpy قرار بوده به بافر insecure_buff كپي شود.

تا اينجا علت و چگونگي بروز overflow stack مشخص شد. اما چگونه اين مشكل براي نفوذ به سيستم مورد استفاده قرار مي گيرد؟ مي دانيم كه تمام برنامه ها و روالهايي كه روي يك سيستم عامل در حال اجرا هستند، در آخرين لايه، چيزي جز كدهاي ماشين نيستند كه پشت سرهم خوانده و اجرا مي شوند. CPU كامپيوتر آدرس حافظه مربوط به دستورالعمل بعدي را در طول اجراي برنامه از رجيستر EIP خوانده و كنترل اجراي برنامه را به آن آدرس منتقل مي كند. حال اگر بتوان آدرس موجود در اين رجيستر را در هر مرحله اي از اجراي برنامه به مقدار ديگري تغيير داد،

Copyright © 2003, Hat-Squad Security Group **www.hat-squad.com**

CPU بدون درنگ اجراي بقيه برنامه را از اين آدرس جديد ادامه خواهد داد. تصور كنيد در محل آدرس جديد كد يك backdoor يا سرويس پنهاني ويا هر نوع كد مخرب ديگري قرار داشته باشد. نتيجه اين خواهد شد كه كامپيوتر اين كد را بجاي كد برنامه اصلي كه مسيرش توسط ما عوض شده، اجرا خواهد نمود و بدين ترتيب نفوذگر خواهد توانست با استفاده از backdoor يا كد محرب اجرا شده، كنترل سيستم مزبور را بدست گيرد. بنابراين نفوذگر براي رسيدن به هدف خود بايد دو مساله را حل كند. **اول يافتن راهي براي ايجاد overflow در سيستم هدف** . براي اينكار نفوذگر ، سرويسها و برنامه هاي در حال اجرا روي سيستم هدف مانند Server Web ، Mail Server، Ftp Server و ... را براي يافتن روشي جهت overflow كردن هر كدام از آنها آزمايش خواهد نمود. بحث يافتن overflow ها از حوصله اين مقاله خارج است و نمي توان روش استانداردي را پيشنهاد كرد و بيشتر روي سعي و خطا استوار است. مرحله دوم **استفاده از برنامه overflow شده براي اجراي كد دلخواه**.

براي نشان دادن روس كار نفوذگر، سعي مي كنيم برنامه با استفاده از برنامه big.exe كد مورد نظر خود را روي سيستم اجرا كنيم. رشته اي را به عنوان ورودي (به جاي 100 كاراكتر a) مي سازيم كه شامل يك كد كوچك دستورات زبان اسمبلي (كه اصطلاحا exploit ناميده مي شود) است كه كار مورد نظر مارا روي سيستم انجام خواهد داد و با تغيير آدرس برگشت، كنترل را به اين كد خودمان مي دهيم. بدين ترتيب به نتيجه مورد نظر خواهيم رسيد.

براي نوشتن كد Exploit احتياج داريم تا بدانيم رشته ورودي ساخته شده توسط ما در چه محلي روي آدرس برگشت ذخيره شده در انتهاي Stack خواهد افتاد.براى دانستن اين موضوع دو راه وجود دارد.روش اول استفاده از يك Disassembler براي يافتن اندازه بافر سرريز شده است. در اين روش مجبوريم به دنبال تابعي بگرديم كه سرريز در آن اتفاق مي افتد. راه دوم انجام آزمون و خطا است ، ابتدا بايد رشته كاراكتري از كد هاي اسكي 32 تا 255 بسازيم ، كد كوچك زير اين كار را برايمان انجام مي دهد:

```
/* ascii.exe */

#include <stdio.h>
void main(void) {
   int i;
   for (i=0;i<256;i++) printf("%c",i);
}
```

اين سوال پيش مي آيد كه چرا كاراكتر ها با كد اسكي بزرگتر از 32 را انتخاب كرديم. توجه داشته باشيد كه رشته اي كه به عنوان ورودي به برنامه هدف مي دهيم نبايد حاوي كد هاي اسكي كاراكترهاي CR(0x0c)،EOF(0x1a)، NULL(0x00) ، LF(0x0a) باشد چون اگر تابع strcpy هنگام كپي كردن رشته كاراكتري به يكي از اين كدها برسد آن را به عنوان انتهاي رشته تلقي خواهد كرد و بقيه رشته كپي نخواهد شد. به همين دليل است كه ما بازه 32 تا 255 را كه شامل هيچكدام از اين كدها نيست انتخاب مي كنيم. حال برنامه را كامپايل كرده ، به صورت زير اجرا مي كنيم :

```
C:\> c:\bof\big.exe | ascii.exe
```


Copyright © 2003, Hat-Squad Security Group **www.hat-squad.com**

اين بار در پيغام خطا رجيستر EIP حاوي آدرس x8b8a89880 است(ترتيب قرارگيري از راست به چپ است). يعني با شروع از محل صدو چهارم بافر (x88-0x20=1040) رشته ورودي روي آدرس برگشت مي افتد. پس در رشته اي كه خواهيم ساخت محلهاي 104 تا 107 حافظه (به طول 4 بايت) بايد حاوي آدرس برگشت به كدي باشد كه مي خواهيم اجرا شود.

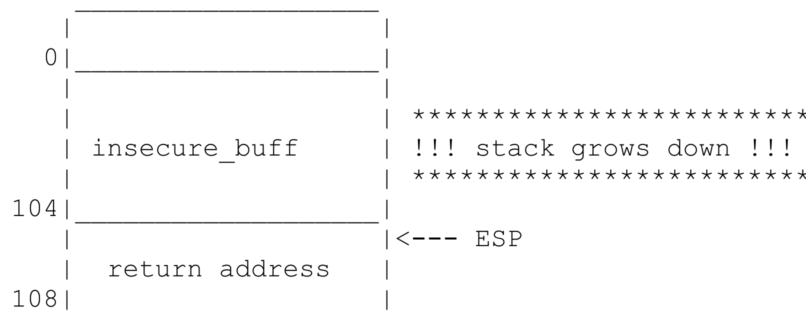
مشكل اول حل شد ، حالا بايد تصميم بگريريم كه چگونه كد Exploit را تشكيل دهيم . در اين باره دو امكان وجود دارد :

1- قرارگيري كد Exploit از ابتداي بافر تا محل 104 بافر

2- قرارگيري كد Exploit از محل 108 بافر به بعد

انتخاب روش اول اندازه كد Exploit مارابه 104 بايت محدود خواهد كرد ، به همين جهت روش دوم را انتخاب مي كنيم و محلهاي حافظه قبل از آدرس صدوچهارم را نيز با كد دستور اسمبلي (No Operand) NOP يعني x900 پر مي كنيم.

مساله آخر تعيين آدرس محل حافظه است كه مي خواهيم به جاي آدرس برگشت واقعي قرار دهيم. ابتدا به بررسي وضعيت رجيسترها و ساختار بافر درست قبل از اجراي دستور RET اسمبلي (توليد شده توسط دستور return 0) مي پردازيم:



همان طور كه مي بينيد ، درست قبل از اجراي دستور RET رجيستر ESP به محل 104 حافظه اشاره خواهد كرد ، پس اگر بتوانيم در اين لحظه يك دستور jmp esp اجرا كنيم ، پردازنده بلافاصله كنترل را به آدرس 4 بايتي قرار داده شده در محلهاي 104 تا 107 مي دهد وما به منظور خود رسيده ايم . ابتدا بايد حافظه سيستم را به دنبال كد دستور jmp esp (0xff0xe4) جستجو كنيم ، آدرس حافظه پيدا شده همان آدرسي است كه بجاي آدرس برگشت واقعي در محلهاي 104 تا 107 بافر قرار مي گيرد. با اين حساب ترتيب اجراي برنامه به صورت زير در خواهد آمد:

RET--> JMP ESP--> Our Exploit Code

تركيب xff0xe40 را مي توان هم در حافظه برنامه big.exe و هم در حافظه مربوط به DLL هاي متصل به آن جستجو كرد. بهترين راه جستجو در DLL هاي متصل به برنامه است (در اينجا يكي دو فصل مربوط به فرمت فايلهاي PE ويندوز را رد مي كنيم و مطالعه آن را به عهده خواننده علاقه مند مي گذاريم). فايلهاي DLL سيستم در ويندوز NT با شروع از آدرس Image Base در حافظه Load مي شوند با كمك برنامه هاي PE Analyser مي توان به آساني اين آدرس را پيدا كرد. در اينجا از برنامه LISTDLLS استفاده مي كنيم كه مي توانيد آن را از سايت sysinternals دريافت كنيد :

Copyright © 2003, Hat-Squad Security Group www.hat-squad.com

```
C:\bof> listdlls big.exe

  .    .     .
  Base        Size       Version           Path
  0x00400000  0x27000                      C:\bof\big.exe
  0x77f60000  0x5c000    4.00.1381.0130    D:\WINNT\System32\ntdll.dll
  0x77f00000  0x5e000    4.00.1381.0133    D:\WINNT\system32\KERNEL32.dll
```

اطلاعات راجع به محل قرارگيري برنامه و DLL هاي آن در حافظه را مي بينيم. مي توانيم در حافظه يا داخل هر كدام از اين 3 فايل نشان داده شده به دنبال كد دستور jmp esp بگرديم . جستجو در حافظه راحتتر است چون نيازي به محاسبه Offset ها بر خلاف داخل فايل نيست. از يك Debugger مثلا SoftICE استفاده مي كنيم و هنگام وقوع سرريز كه كنسول SoftICE ظاهر مي شود ، دستور زير را اجرا مي كنيم :

```
S 1000000 | ffffffff fee4
```

يعني جستجوي حافظه از آدرس x010000000 (اولين آدرسي كه بايت اول مخالف صفر دارد) تا آخرين Offset حافظه يعني xFFFFFFFF0 . نتيجه زير حاصل خواهد شد :

```
Pattern found at 0023:77f327e5    (77f327e5)
```

دستور jmp esp در محل آدرس x77f327e50 پيدا شد اين آدرس حاوي هيچكدام از كدهاي CR(0x0c)، EOF(0x1a)، NULL(0x00) ، LF(0x0a) نيست و براحتي مي توان از آن استفاده كرد.

توجه : آدرس فوق با توجه به ورژن Service Pack نصب شده روي سيستم عامل هدف مقادير متفاوتي خواهد بود ، به اين سبب در مواردي كه Remote Exploit مي نويسيم به طريقي از شماره Service Pack نصب شده روي كامپيوتر هدف اطلاع پيدا كنيم و سپس آدرس درست براي آن Service Pack را استفاده كنيم. روشهاب پيشرفته ديگري براي Exploit نويسي وجود دارند كه اين مشكل را حل مي كنند.

در اين مرحله تمام اطلاعات لازم را در اختيار داريم : 1-محل قرارگيري كد Exploit معلوم شده است 2-آدرس كه بايد بجاي آدرس برگشت واقعي قرار داده شود ، مي دانيم. تنها كار باقيمانده نوشتن برنامه Exploit است كه بر حسب Local يا Remote بودن هدف متفاوت خواهد بود. در اين زمينه مطالب و كدهاي فراواني روي وب موجود هستند كه با پيش زمينه فعلي به آساني مي توانيد از آنها ايده بگيريد.

چگونه از حملات Buffer Overflow چلوگيري كنيم؟

حملات buffer overflow از ضعف حاصل از عدم تست اندازه داده ورودي استفاده مي كنند. اگر برنامه نويس big.exe قبل از دستور strcpy(insecure_buff,big); اندازه ورودي big را چك مي كرد يا بجاي تابع strcpy از strncpy به صورت strncpy(insecure_buff,big,100); استفاده مي كرد، مشكل overflow بروز نمي كرد. **لذا مهمترين اصل در برنامه نويسي يك سرويس يا برنامه مقاوم در برابر overflow ها چك كردن اندازه تمام ورودیها به برنامه قبل از انجام هر كاري روي داده هاست.** براي برنامه نويسان محيط linux/unix كتابخانه هايي مانند StackGuard وجود دارند كه برنامه نويس مي تواند با لينك كردن اين كتابخانه ها به نرم افزارش، جلوي بسياري از overflow ها را بگيرد.

Copyright © 2003, Hat-Squad Security Group www.hat-squad.com

اما در اکثر موارد، به کد سورس برنامه یا سرویس نصب شده روي سیستم خود دسترسي نداریم و قادر به تشخیص ضعفهاي احتمالي از طریق بررسي سورس برنامه نیستیم همچنین اینکار از توان افراد غیر متخصص حارج مي باشد. چاره اي که در بعضي سیستم هاي عامل مانند Sun/OS و Linux اندیشیده شده است، ممانعت از اجراي کد در محیط پشته (Stack) است. همچنین این روش بصورت محدودي روي سیستمهاي ویندوز پیاده شده است. اما بهترین و راحتترین روش براي عموم نصب تمام Patch ها و Fix هاي ارائه شده توسط تولید کننده نرم افزار است که در 99% مواقع موثر خواهد بود.

سوالات و نظرات خود را به آدرس behrang@hat-squad.com ارسال نمایید .

دریافت اطلاعات بیشتر در مورد PE :
http://msdn.microsoft.com/library/en-us/dnwbgen/html/msdn_peeringpe.asp?frame=true
دریافت برنامه LISTDLLS :
http://www.sysinternals.com/

Copyright © 2003, Hat-Squad Security Group **www.hat-squad.com**