

((بسم الله الرحمن الرحيم))

آشنایی با حملات سرریز بافر

ارائه دهنده:

مهسا ناصری کریمی

((معرفی حملات نرم افزاری))

هر برنامه کامپیوتری که برای اجرا در محیط شبکه طراحی و پیاده سازی می گردد. می بایست توجه خاصی به مقوله امنیت داشته باشد. چهار نوع حمله وجود دارد که در زیر هر یک را توضیح خواهیم داد:

الف. وقفه:

زمانیکه سیستم از بین می رود یا غیر قابل دسترسی می شود. حمله روی دسترسی انجام می شود. مثال ها شامل از دست رفتن قسمتی از سخت افزار مثل هارد , حذف خط ارتباط یا غیرفعالسازی فایل مدیریت سیستم می شود.

ب. بریدگی:

دسترسی غیرمجاز است که حمله بحرمانگی سیستم صورت می گیرد. گروه غیرمجاز میتواند یک شخص , یک برنامه یا یک کامپیوتر باشد.

ج. دستکاری:

دسترسی غیرمجاز در این مرحله همراه با مداخله صورت می گیرد. مثال ها شامل تغییر ارزش در یک فایل اطلاعات , تغییر برنامه به نحوی که به صورت دیگری عمل کند و تغییرات در محتوی پیام در حال انتقال در شبکه است.

د. تقلید و جعل:

افراد غیرمجاز چیزهای جعلی را وارد سیستم می کنند. این حمله روی صحت و سندیت صورت می گیرد. مانند وارد کردن پیام های جعلی در شبکه یا اضافه کردن رکوردهایی به فایل.

((حمله سرریز بافر))

حمله سرریز پشته یکی از اولین و خطرناک ترین حملات به برنامه های کاربردی می باشد. اولین مورد برای حمله به یک سیستم هدف گرفتن پشته سیستم عامل است. سرریز پشته می تواند منجر به مختل شدن یک برنامه کاربردی یا سیستم عامل شود. گونه بسیار پیچیده و خطرناکی از حمله به پشته آن است , که نفوذگر بتواند پس از سرریز شدن , کنترل اجرای آن برنامه یا پروسه را در اختیار گیرد. امروزه حمله سرریز پشته یکی از یکی از روش های مهلک و رایج حمله علیه ماشین های آسیب پذیر محسوب می شود. هر برنامه یا پروسه که در بخشی از کد خود از پشته یا بافر استفاده کند , ممکن است در اثر سرریز شدن به ناگاه مختل شود. نفوذ پروسه هایی را که به نحوی از پشته یا بافر استفاده کرده اند ولی تمهیدی نیندیشیده اند , کشف می کند و آنها را مورد هدف قرار می دهد. متأسفانه برخی از سرویس دهنده های حساس شبکه قبل از عرضه به دقت آزمایش نشده اند و اگر یک رشته خاص و طولانی برای آنها ارسال شود , به ناگاه در هم می شکنند. زیرا سرریز شدن بافر در آنها منجر به نقض حریم حافظه شده و رون اجرای آنها توسط سیستم عامل متوقف می شود. گونه بسیار پیچیده و خطرناکی

از حمله به پشته آن است که نفوذگر بتواند پشته را به نحوی سرریز کند که پس از سرریز شدن ، کنترل اجرای آن برنامه یا پروسه را در اختیار بگیرد.

((پشته چیست؟))

پشته یک نوع ساختمان داده ایست که آیت‌های داده در آن ذخیره میشوند. دسترسی به آیت‌ها مبتنی بر روش خروج به عکس ورود است. یعنی در هر لحظه میتوان به آخرین عنصر ذخیره شده در پشته دسترسی داشت. برای دسترسی به عناصر زیرین پشته باید ، ابتدا باید عناصر روی آن از پشته خارج شوند. نفوذگر میتواند به صورت عمودی در فضای بافر در نظر گرفته شده برروی پشته (که به منظور ذخیره متغیرهای محلی ایجاد شده است) داده های به حجم بزرگتر از ظرفیت آنها قرار دهد. به گونه ای که این حریم فضا شکسته شده و فضای زیرین این ناحیه با داده پر شود. چون این فضا شامل اطلاعات حیاتی برای برگشت از تابع است ، لذا سرنوشت ادامه روند اجرا مشخص نخواهد بود. (آغاز بحران فروپاشی) در سیستم عامل یونیکس خطرناک ترین کد اجرایی که نفوذگر میتواند از آن استفاده کند ، قطعا کدی است که برنامه پوسته فرمان را فراخوانی و اجرا می کند.

((سازماندهی و آرایش بافر پیش از سرریز شدن))

سوآلی که ممکن است ذهن برنامه نویسان حرفه ای را به خود مشغول کند آنست که با فرض سرریز شدن پشته و شکستن حریم فضای حافظه ، چگونه میتوان آدرس برگشت و همچنین کدهای اجرا را در این فضا سازماندهی و تنظیم کرد تا پس از اجرای دستور

((برگشت)) به دلیل تغییر در محتوای اشاره گر بازگشت کنترل اجرا به قطعه کد موردنظر که آن هم در پشته واقع است , شود؟

برای حل این مسئله نفوذگر یک محل تقریبی از حافظه را حدس میزند و اشاره گر برگشت را به آنجا تنظیم میکند. سپس برای آنکه احتمال خطا را در حدسش کمتر کند تعداد زیادی کد **NOP** قبل از کد اجرایی مورد نظرش اضافه می نماید. بدین ترتیب اگر مقدار تنظیم شده در اشاره گر بازگشت دارای اندکی خطای محاسبه باشد , باز هم مشکلی پیش نمی آید. در حقیقت کنترل اجرا به وسط دریایی از دستورات **NOP** که کاملاً بی ضرر و بی خاصیت هستند فرود می آید و منجر به فروپاشی و توقف برنامه نمیشود. هرگاه پس از اجرای دستور بازگشت کنترل اجرا به یک نقطه از فضایی که با **NOP** پر شده است منتقل شود , در نهایت قطعه کدی که پس از **NOP** ها قرار گرفته , اجرا خواهد شد.



شکل: سقوط کنترل اجرا در دریایی از دستورالعملهای NOP

((حملات بعد از در هم شکستن پشته))

شاید بپرسید در یک نقطه آسیب پذیر که در نقطه ای از آن پشته سرریز خواهد شد , نفوذگر به چه نحو پشته را سرریز می نماید و چه نوع کد دستورالعملی را اجرا می نماید تا کنترل سیستم را در اختیار بگیرد؟

اگر پروسه ای که مورد حمله قرار گرفته در مورد **super-user** اجرا شده باشد , آنگاه دست نفوذگر برای اجرا هر قطعه کد یا پروسه باز است. اگر پروسه ای که در اثر سرریز شدن پشته از کنترل خارج شده است در مورد کاربر باشد آنگاه نفوذگر قطعه کدهای محدودتری را اجرا میکند.

((سه تکنیک اساسی برای در اختیار گرفتن یک برنامه))

الف. ایجاد رخنه عبور با استفاده از پروسه **inetd**:

در سیستم عامل یونیکس پروسه **inetd** به تمام پورت ها گوش میدهد و به ازای دریافت تقاضا برای برقراری ارتباط با یک شماره پورت , پروسه متناظر با آن را اجرا می نماید. هرگاه نفوذگر نقطه ضعفی را در یک پروسه کشف کند, تلاش خواهد کرد تا پشته آن پروسه را به گونه ای سرریز کند تا فرمانی به صورت زیر اجرا شود:

```
/bin/sh-c"echo 12345 stream tcp nowait root/bin/sh -i">>/etc/inetd.conf;killall-HUP inetd
```

برای اجرا شدن این فرمان کافی است که این رشته به عنوان آرگومان ورودی به تابع سیستمی **execve** ارسال و اجرا شود. یعنی : قطعه کد اجرایی که پس از سرریز شدن هسته باید اجرا شود کدهای اجرایی لازم برای فراخوانی تابع **execve** با آرگومان فوق است. این قطعه کد کوچک ولی بسیار خطرناک است.

ب. ایجاد رخنه با استفاده از پروسه **NetCat** و **TFTP**:

مکانیزم اجرای پشته فرمان و دستکاری در فایل **/etc/inetd.conf** فقط در محیط های سازگار یونیکس کاربرد دارد. یعنی فقط زمانی از این تکنیک استفاده میشود که پروسه ای که در اثر فروپاشی پشته , کنترل از آن خارج شده در محیط یونیکس اجرا شده باشد. این روش در محیط هایی که در آنها از سیستم عامل ویندوز استفاده میشود قابل اجرا نیست. برای چنین محیطی **TFTP** می تواند به عنوان یک رخنه عبور همانند پروسه **inetd** نقش موثر و خطرناکی را ایفا کند.

TFTP یک سرویس دهنده ساده و کوچک برای انتقال فایل است که از پروتکل **UDP** برای انتقال داده های فایل بهره می گیرد. اگر چه قابلیت های **TFTP** در مقایسه با **FTP** بسیار ضعیف و ناچیز است ولی سادگی ، سرعت و کوچکی حجم آن باعث شده تا هنوز در سیستم عامل ویندوز و برخی گونه های یونیکس ، از آن به عنوان یک سرویس دهنده جانبی حمایت شود. حتی برخی از مسیرهایها از **TFTP** برای انتقال فایل های پیکربندی و بوت شدن خود استفاده می نمایند.

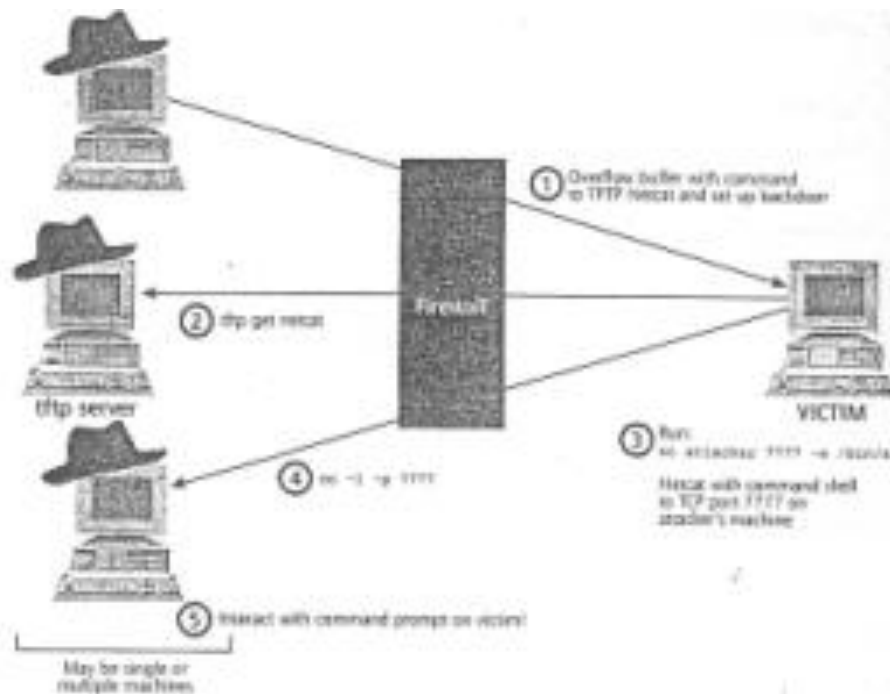
کدهای اجرایی سرویس دهنده **TFTP** کم حجم و کوچک است و نفوذگر میتواند این کدها را بدرون پشته یک برنامه آسیب پذیر بفرستد بگونه ای که پس از سرریز شدن پشته ، سرویس دهنده **TFTP** اجرا گردد.

راه دیگر اجرای **TFTP** آن است که کدهای اجرایی تابع **execve** بگونه ای تنظیم و ارسال شود که پس از سرریز شدن پشته برنامه **TFTP** اجرا شود. پس آنکه **TFTP** روی ماشین هدف اجرا شد ، با استفاده از آن ابزار قدرتمند و کوچک **NatCat** از ماشین نفوذگر بر روی ماشین قربانی منتقل شده و اجرا می شود.

امروزه روش **TFTP** پس از فروپاشی پشته ، انتقال **NatCat** و اجرای آن ، یکی از حملات متداول و خطرناک محسوب می شود. نفوذ گر قبل از شروع حمله ، سرویس دهنده **TFTP** را روی ماشین خودش اجرا کرده و فایل اجرایی **NatCat** را در اختیار آن قرار می دهد تا پس از آنکه حمله شروع شد و کنترل اجرای ماشین هدف به دست پروسه **TFTP** افتاد به سرعت فایل برنامه **NatCat** منتقل شده و نصب گردد.

برنامه **NatCat** کمتر از **100HB** حجم دارد.

اکنون در زیر روش **TFTP** را با استفاده از شکل توضیح می دهیم:



مرحله 1:

در این مرحله نفوذگر به یک پروسه آسیب پذیر حمله کرده و آن را به گونه ای سرریز می کند تا برنامه سرویس دهنده **TFTP** اجرا گردد.

مرحله 2:

برنامه **TFTP** روی ماشین قربانی اجرا شده و از ماشین نفوذگر یک نسخه از برنامه **NatCat** را دریافت و نصب می کند.

مرحله 3:

ماشین قربانی **NatCat** را به نحوی اجرا می کند تا در هنگام برقراری ارتباط با آن برنامه پوسته فرمان فعال شود. اجرای **NatCat** روی ماشین قربانی به این صورت است که ارتباط **TCP** از درون شبکه با ماشین نفوذگر برقرار خواهد شد.

مرحله 4:

نفوذگر روی ماشینش نیز یک نسخه از **NatCat** را ((در مورد شنود)) اجرا کرده و منتظر برقراری ارتباط از ماشین قربانی می ماند.

مرحله 5:

پس از برقراری ارتباط نفوذگر از طریق **NatCat** بروی ماشین قربانی نفوذ دارد و خود می داند که با آن ماشین چه کند.

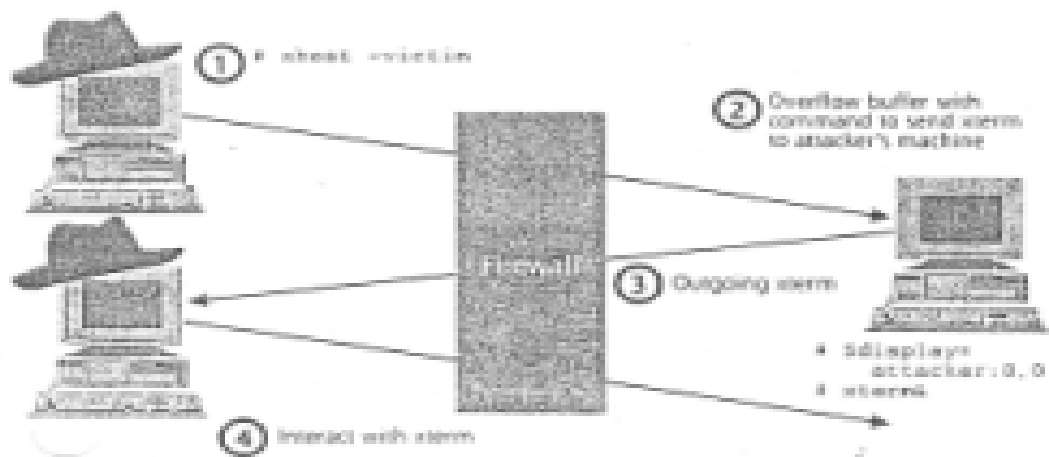
خطرناک ترین روش فوق آن است که ارتباط **NatCat** از درون شبکه با نفوذگر برقرار می شود و چون دیوارهای آتش معمولاً فقط ارتباطات **TCP** از بیرون به درون را مسدود می کنند لذا این روش احتمال موفقیت بسیار زیادی خواهد داشت. یکی دیگر از ویژگی های این روش آن است که فایل پیکربندی **inetd.conf** به هیچ وجه تغییر نخواهد کرد و با بررسی فایل های پیکربندی نمی توان چنین نفوذی را کشف و گزارش کرد.

ج. ایجاد رخنه عبور توسط **(X-Terminal)Xterm**:

یکی دیگر از روش های رخنه عبور در ماشین قربانی پس از سرریز شدن پشته استفاده از سیستم **X Window** که به اختصار سرویس **X**

نامیده می شود. سرویس **X TFTP** یک برنامه گرافیکی و مبتنی بر پنجره است که در اغلب محیط های یونیکس و برخی از بسته های جانبی **Windows NT** از آن حمایت می شود. **X Window** برنامه ای برای ورود به یک سیستم از راه دور و اجرای برنامه روی آن سیستم می باشد. در حقیقت با برنامه **X** می توان از طریق یک پنجره گرافیکی، مبتنی بر منو و آیکون با ماشین راه دور محاوره داشت. در صورتی که نفوذگر بتواند از راه دور و توسط **X Window** به یک سیستم وارد شود، کنترل سیستم در اختیار او خواهد بود. تلاش او بر آن خواهد بود که پس از سرریز شدن پشته در یک پروسه آسیب پذیر و در اختیار گرفتن کنترل آن برنامه **X Window** را روی ماشین قربانی فعال و اجرا کند.

اکنون در زیر روش **(X-Terminal)Xterm** را با استفاده از شکل توضیح می دهیم:



مرحله 1:

نفوذگر ماشین خود را به گونه ای تنظیم می کند تا تمام ارتباطات **X Window** را بپذیرد و یک نشست **X** بین ماشین او و ماشین قربانی برقرار شود.

مرحله 2:

نفوذگر یک پروسه آسیب پذیر را روی ماشین هدف سرریز کرده و تلاش می کند تا پوسته فرمان را روی آن ماشین اجرا نماید.

مرحله 3:

پوسته فرمان اجراشده بر روی ماشین قربانی به گونه ای تنظیم شده تا برنامه **X Window** را اجرا نموده و آن را وادار به برقراری یک نشست با ماشین نفوذگر نماید.

مرحله 4:

نفوذگر که از قبل منتظر چنین نشستی بود از طریق برنامه **X** هر فرمانی را که علاقه داشته باشد تایپ و ارسال می نماید. این فرامین بر روی ماشین قربانی اجرا خواهد شد.

((نکته))

دیوارهای آتش به طور معمول نشست های **X** از بیرون شبکه به درون را مسدود می کنند ولی اکثر آنها نشست های **X** از درون به بیرون را مجاز می دانند. یعنی به کاربران درون شبکه اجازه می دهند که توسط برنامه **X** با ماشین راه دور نشست **X Terminal** داشته باشند. همانند **TFTP** ویژگی این حمله آن است که نشست **X**

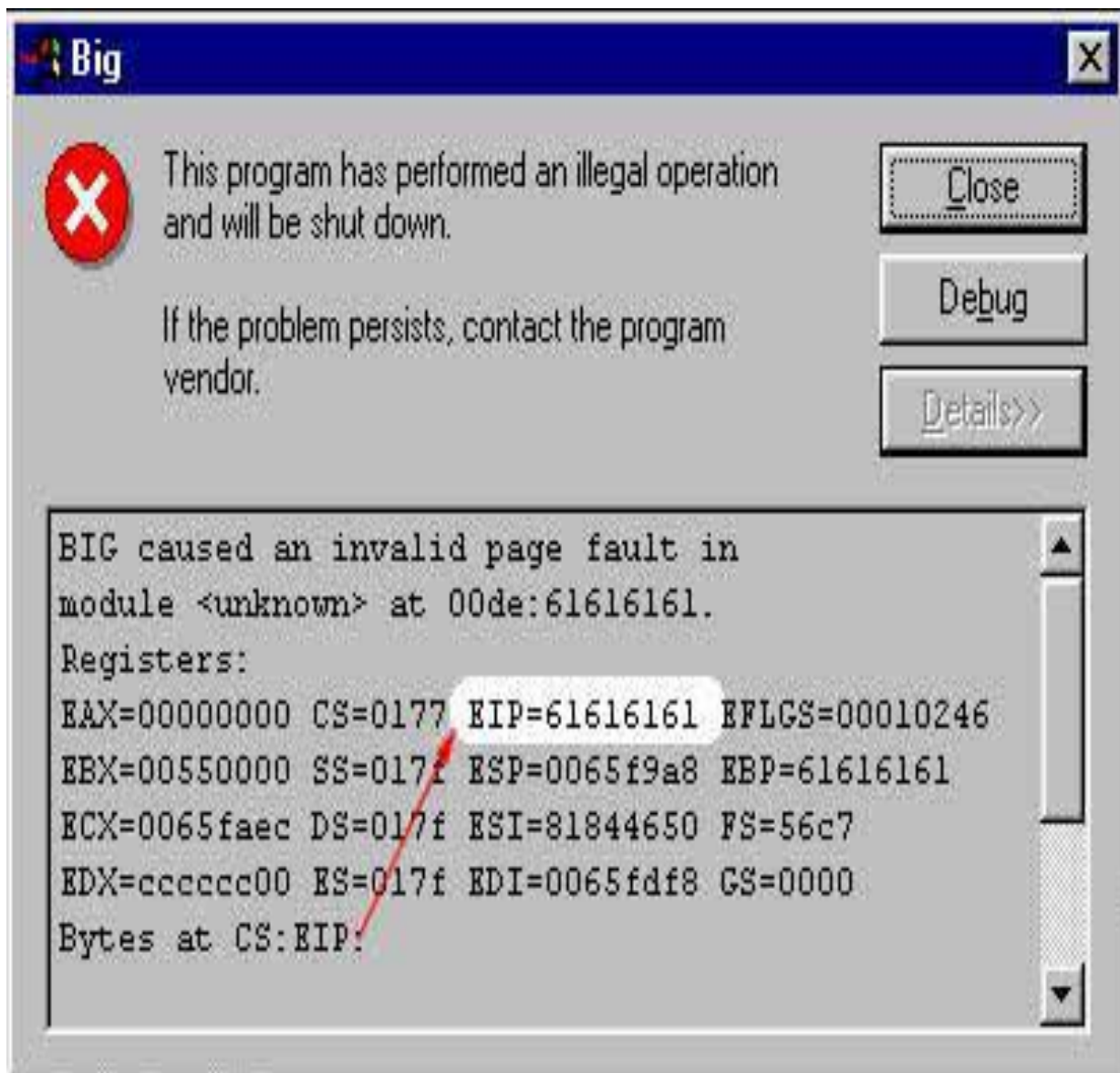
از سوی ماشین قربانی با ماشین نفوذگر برقرار می شود و بدن ترتیب احتمال موفقیت آن بسیار خواهد بود. ویژگی دیگر این حمله آن است که هیچ تغییری در فایل‌های پیکربندی سیستم قربانی داده نمی شود. لذا نمی توان با بررسی فایل‌های پیکربندی وقوع حمله را کشف کرد.

اکنون در زیر نمونه کدهای را به زبان C بررسی می کنیم:

```
-----  
-----  
/* 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 ورودی از صفحه کلید را در آرایه `input_buff` قرار میدهد سپس هنگامی که تابع `insecure_buff` با فرستادن `input_buff` فراخوانی شود , این تابع مقدار موجود در `input_buff` را در `insecure_buff` کپی خواهد کرد. نکته اصلی کوچکتر بودن اندازه `insecure_buff` از `input_buff` است. بطوریکه اگر `input_buff` بیشتر از 100 کاراکتر را در خود جای داده باشد , `insecure_buff` سرریز خواهد شد.

حال برنامه را اجرا کرده و بیش از 100 کاراکتر 'a' را به عنوان ورودی به برنامه بدهید. پس از زدن کلید `Enter` پیغام خطایی شبیه مورد زیر دریافت خواهید کرد:



اکنون قطعه کدی را به زبان **C++** مورد بررسی قرار می دهیم:

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <stdio.h>
```

```
int Hijack()
```

```

{
    cout << "This function should be hijacked!";
    getchar();
    return 0;
}
int Function1()
{
    char var1[15];
    cout << "_____1234567890abcd" << endl;
    cout << "Enter Var1:";
    cin >> var1;
    cout << var1 << endl;
    return 0;
}
int main()
{
    Function1();
    getchar();
    return 0;
}

```

پس از اجرای قطعه کد بالا حالتی مانند شکل زیر رخ میدهد:

```
C:\D:\temp\Debug\vuln.exe
```

```
1234567890abcd
Enter Var1:1234567890123456AAAAABCDE
1234567890123456AAAAABCDE
```



((چگونه از سرریز بافر جلوگیری کنیم؟))

حملات سرریز بافر از ضعف حاصل از عدم تست اندازه داده ورودی استفاده می کنند. اگر برنامه نویس `big.exe` قبل از دستور `strcpy(insecure_buff, big);` اندازه ورودی `big` را چک می کرد یا بجای تابع `strcpy` از `strncpy` به صورت `strncpy(insecure_buff, big, 100);` استفاده می کرد , مشکل سرریز بافر بروز نمی کرد. لذا مهمترین اصل در برنامه نویسی یک سرویس یا برنامه مقاوم در برابر سرریز بافرها چک کردن اندازه تمام ورودی ها به برنامه قبل از انجام هر کاری روی داده هاست. برای برنامه نویسان محیط `linux/unix` کتابخانه هایی مانند `StackGuard` وجود دارند که برنامه نویس می تواند با لینک کردن این کتابخانه ها به نرم افزارش , جلوی بسیاری از سرریزها را بگیرد.

اما در اکثر موارد , به کد سورس برنامه یا سرویس نصب شده روی سیستم خود دسترسی نداریم و قادر به تشخیص ضعف های احتمالی از طریق بررسی سورس برنامه نیستیم. هم چنین ابتکار از توان افراد غیر متخصص

خارج می باشد. چاره ای که در بعضی سیستم های عامل مانند **Sun/OS** و **Linux** اندیشیده شده است , ممانعت از اجرای کد در محیط پشته است. هم چنین این روش به صورت محدودی روی سیستم های ویندوز پیاده شده است. اما بهترین و راحت ترین روش برای عموم نصب تمام **Patch** ها و **Fix** های ارائه شده توسط تولید کننده نرم افزار است که در **99%** مواقع موثر خواهد بود.

((منابع))

www.webappsec.org

www.wikipedia.com

www.sysinternals.com

www.grsecurity.net

*گروه امنیت دانشگاه یزد