به نام خدا

تزریق فرمان

OWASP Attack Category:Command Injection

توضیحات:

تزریق Command حمله ای است که هدف آن اجرای فرمان های دلخواه بر روی سیستم عامل میزبانی است که رو آن یک برنامه ی آسیب پذیر نصب است. تزریق command زمانی ممکن است که یک برنامه، ورودی هایی که کاربر وارد می کند را بدون اعتبارسنجی به خط فرمان یا یک شل سیستمی ارسال کند. این داده شامل فرم ها، کوکی ها، هدر های http و ... می شود. در این نوع از حمله، فرمان های سیستمی که توسط هکر ارسال می شود معمولاً با سطح دسترسی ای که برنامه ی آسیب پذیر دارد، اجرا می شود. تعداد حملات تزریق فرمان به دلیل عدم اعتبارسنجی ورودی ها به صورت مناسب و درست، بسیار گسترده است.

این نوع از حمله با تزریق کد یا همان Code Injection متفاوت است. در حمله ی اینجکت کد، هکر کد خودش را به کد اصلی اضافه و یا جایگزین آن می کند و کد او توسط برنامه اجرا می شود. در تزریق کد، هکر عملکرد پیش فرض برنامه را گسترش می دهد و این کار لزوماً به اجرا شدن فرمان های سیستمی نیست.

نمونه ها:

مثال ۱

کدی که در ادامه مشاهده می کنید دستور cat در یونیکس را اجرا می کند و محتوای فایل ها را به صورت استاندارد نمایش می دهد. این کد قابل اینجکت است:

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

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 system(command);
 return (0);
}
```

در شرایطی که به صورت عادی از آن استفاده کنیم، در خروجی محتوای فایل درخواست شده را مشاهده می کنیم:

```
$ ./catWrapper Story.txt
When last we left our heroes…
```

اما اگر به پایان این دستور یک نقطه ویرگول ; اضافه کنیم و بعد از آن دستور بعدی را اضافه کنیم، برنامه بدون هیچ مشکلی هر دو دستور را انجام می دهد.

```
$ ./catWrapper "Story.txt; ls"
When last we left our heroes…
Story.txt               doubFree.c              nullpointer.c
unstosig.c              www*                    a.out*
format.c                strlen.c                useFree*
catWrapper*             misnull.c               strlength.c             useFree.c
commandinjection.c      nodefault.c             trunc.c                 writeWhatWhere.c
```

اگر سطح دسترسی برنامه ی نوشته شده (catWrapper) بالاتر از میزان استاندارد تنظیم شده باشد، کاربر می تواند دستوراتی که نیاز به مجوزهای بالاتر دارند را نیز اجرا می کند.

مثال ۲

برنامه ی ساده زیر از کاربر نام یک فایل را می گیرد و محتوای آن را به کاربر نشان می دهد. نصب این برنامه با دسترسی root می باشد و دلیل آن این است که این برنامه به منظور ابزاری آموزشی برای مدیران سیستم نوشته شده است تا بدون اینکه بتوانند فایل های سیستمی را دستکاری و سیستم را دچار مشکل کنند، فقط به محتوای آن ها سرکشی کنند.

```
int main(char*argc, char** argv) {
   char cmd[CMD_MAX] = "/usr/bin/cat ";
   strcat(cmd, argv[1]);
   system(cmd);
}
```

با توجه به اینکه برنامه با امتیاز root در حال اجراست، تابع ()system نیز با دسترسی root اجرا می شود. اگر کاربر نام یک فایل را مشخص کند، برنامه آنگونه که از آن انتظار می رود اجرا می شود. اما اگر هکر رشته ای مثل

;rm -rf /

را وارد کند، در فراخوانی ()system، دستور cat به دلیل کم بودن تعداد آرگومان های ورودی با شکست مواجه می شود و سپس به صورت بازگشتی تمام محتوای پارتیشن root را پاک می کند.

مثال ۳

کد زیر قسمتی از یک برنامه با سطح امتیاز بالاست که در آن از متغیر $APPHOME جهت مشخص کردن دایرکتوری نصب برنامه استفاده می شود و سپس اسکریپت نصب کننده ی برنامه را با توجه به آدرس دایرکتوریِ نصب اجرا می کند.

```
...
        char* home=getenv("APPHOME");
        char* cmd=(char*)malloc(strlen(home)+strlen(INITCMD));
        if (cmd) {
                strcpy(cmd,home);
                strcat(cmd,INITCMD);
                execl(cmd, NULL);
        }
        ...
```

مشابه مثال ۲ هکر می تواند کدهای خود را با امتیازی که برای برنامه تعریف شده، اجرا کند. در این مثال، هکر می تواند آدرس ذخیره شده در $APPHOME را به آدرس جایی تغییر دهد که در آن یک فایل آلوده شده ی INITCMD وجود دارد. با توجه به اینکه برنامه آدرس وارد شده توسط کاربر را اعتبارسنجی نمی کند، هکر می تواند برنامه را به گونه ای فریب دهد که یک کد مخرب اجرا شود.

هکر از متغیر مربوط به آدرس، جهت کنترل فرمان هایی که برنامه انجام می دهد، استفاده می کند. بنابراین تاثیر این متغیر به روشنی در این مثال بیان شد. حالا نوبت آن رسیده که ببینیم در صورت تغییر ماهیت و عملکرد یک فرمان توسط هکر چه اتفاقی می افتد.

مثال ۴

کد زیر یک کد CGI است که کاربر به وسیله ی آن پسورد خود را عوض می کند. این کار با اجرای دستور make در دایرکتوری /var/yp صورتی می گیرد. توجه داشته باشید که برای تغییر پسورد کاربر، این برنامه می بایست با دسترسی root نصب شود.

```
system("cd /var/yp && make &> /dev/null");
```

برخلاف مثال قبل، فرمانی که در این مثال آمده به سختی قابل اینجکت است. بنابراین هکر نمی تواند روی آرگومان های ()system کنترلی داشته باشد. با توجه به اینکه برنامه برای دستور make محل مشخص و مطلقی را تعیین نکرده و از قبل نیز متغیری برای آدرس دهی تعیین نشده؛ بنابراین ممکن است هکر بتواند متغیر $PATH که مربوط به آدرس دهی است را به محل ذخیره سازی کدهای باینری بدافزار تغییر دهد و از این طریق بدافزار را با دستور make ایجاد کرده و اسکریپت cgi را از طریق خط فرمان اجرا کند. در صورتی که این برنامه با امتیاز root نصب شود، برنامه ای که هکر آن را make کرده نیز با امتیاز root اجرا می شود.

محیط (environment) نقش موثری را در اجرای دستورات سیستمی در برنامه بازی می کند. توابعی مثل ()system و ()exec از محیط اجرای برنامه ای که آن ها را فراخوانی می کند، استفاده می کنند و بنابراین این فرصت به صورت بالقوه برای هکر ایجاد می شود که روی عملکرد فراخوانی های مربوط به این توابع تاثیر بگذارد.

سایت های زیادی وجود دارد که می گویند تابع Runtime.exec در زبان Java دقیقا مشابه تابع system در زبان C است؛ این موضوع درست نیست. هر دو این امکان را فراهم می کنند که شما یک برنامه/پروسه را اجرا کنید. تابع system در c تمام آرگومان هایش را جهت اجرا به شل /bin/sh پارس (ارسال) می کند. اما Runtime.exec رشته ی ورودی را به آرایه ای از کلمات (word) تقسیم می کند. سپس اولین کلمه از آرایه را به عنوان دستور و مابقی کلمات آن را به عنوان پارامترهای دستور اجرا می کند. تفاوت بین این دو تابع فراتر از عملکرد و نحوه پیاده سازی آن است.. این تابع در جاوا از بسیاری از اعمال مخرب نظیر اجرای دو دستور با هم با استفاده از & یا && یا ا یا ال و ... یا هدایت دوباره ی (redirect) ورودی ها و خروجی ها و ... جلوگیری می کند.) چرا که در صورتی که بخواهیم دو دستور را با همدیگر اجرا کنیم (chaining command) اولین کلمه به عنوان دستور و سایر کلمات به عنوان پارامترهای ورودی آن

دستور در نظر گرفته می شود و بنابراین خطای syntax می دهد و یا آن را به عنوان پارامتری نامعتبر شناسایی می کند.

مثال ۵

کد زیر نمونه ای از آسیب پذیری از نوع تزریق فرمان سیستم عامل روی پلتفرم Unix/Linux است.

- *زبان C*

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

int main(int argc, char **argv)
{
  char command[256];

  if(argc != 2) {
    printf("Error: Please enter a program to time!\n");
    return -1;
  }

  memset(&command, 0, sizeof(command));

  strcat(command, "time ./");
  strcat(command, argv[1]);

  system(command);
  return 0;
}
```

در کد بالا رشته ی ls; cat /etc/shadow را وارد می کنیم. در یونیکس دستورات به وسیله ی نقطه ویرگول از همدیگر جدا می شوند. بنابراین هر دو دستور ls و cat بدون هیچ مشکلی اجرا می شوند.

- *زبان Java*

توضیحاتی که در پاراگراف تفاوت های Runtime.exe با system دادیم.

مثال ۶

قطعه کد زیر به زبان PHP است و این کد شامل آسیب پذیری Command Injection است:

```
<?php
print("Please specify the name of the file to delete");
print("<p>");
```

```
$file=$_GET['filename'];
system("rm $file");
?>
```

نمونه ای از یک درخواست و پاسخ مربوط به یک حمله موفق را در ادامه مشاهده می نمایید:

درخواست:

http://127.0.0.1/delete.php?filename=bob.txt;id

پاسخ:

Please specify the name of the file to delete

uid=33(www-data) gid=33(www-data) groups=33(www-data)

سایر حملات مرتبط:

- تزریق code
- تزریق SQL روش Blind
- تزریق XPath روش Blind
- تزریق LDAP
- پیمایش مسیر یا همان Path Traversal با آدرس دهی نسبی
- پیمایش مسیر یا همان Path Traversal با آدرس دهی مطلق

کنترل ورودی ها:

حالت ایده آل این است که هر برنامه نویس از APIهای موجود در زبان برنامه نویسی استفاده کند. برای مثال در زبان جاوا به جای استفاده از Runtime.exec() برای فرمان 'mail' از API موجود برای آن که همان javax.mail است، استفاده کنید.

در صورتی که برای این کار API آن وجود ندارد، تمام ورودی ها را برای جداسازی کارکترهای خاص و بدخواهانه بررسی کنید. پیاده سازی یک مدل امنیتی از کارکترهای مجاز می تواند مؤثر و مفید باشد. چرا که تعریف کردن کارکترهای مجاز بسیار آسان تر از تعیین کارکترهای غیرمجاز خواهد بود.

منابع:

- CWE-77: Command Injection
- CWE-78: OS Command Injection
- http://blog.php-security.org/archives/76-Holes-in-most-preg_match-filters.html

تاریخ ساخت: July 14, 2014 یا ۲۴ خرداد ۱۳۹۳

تاریخ تحقیق: Aug 3, 2014 یا ۱۲ مرداد ۱۳۹۳

/* تصحیح این مقاله، چه در ترجمه و چه در مباحث علمی ، توسط شما دوستان باعث خوشحالی خواهد بود. لطفا آن را با tamadonEH@gmail.com مطرح نمایید.*/

برای مشاهده لیست مقالات کار شده توسط گروه ما به لینک زیر مراجعه فرمایید

https://github.com/tamadonEH/list/blob/master/list.md