گزارش امنیتی (مستند مرجع)

عنوان گزارش: ابزارهای اشکال‌زدایی امنیتی در زبان‌های برنامه‌نویسی C/C++

بسمه‌تعالی

چکیده

مجموعه‌ای از کامپایلرها برای اشکال‌زدایی برنامه‌های نوشته‌شده به زبان C و C++ وجود دارند که این زبان‌ها را از نقطه‌نظر نگارش و برخی ایرادات منطقی بررسی می‌کنند، اما این کامپایلرها قادر به بررسی و تحلیل مشکلات امنیتی موجود در کد برنامه نیستند. در کنار کامپایلرهای رایج، ابزارهایی نیز وجود دارند که به برنامه‌نویسان کمک می‌کنند تا پیش از انتشار برنامه، بتوانند مشکلات امنیتی آنها را پیدا و برطرف نمایند. البته، این ابزارها هنوز آن‌چنان پیشرفت نکرده‌اند که تمامی مشکلات امنیتی را بیابند، اما می‌توانند تا حد قابل قبولی از انتشار این حفره‌های امنیتی جلوگیری به‌عمل آورند. در این گزارش، برخی از ابزارهای رایگان و منبع‌باز مذکور، مانند FlawFinder، ITS4، RATS، و Splint مورد تحلیل و بررسی قرار می‌گیرند.

1 مقدمه

ساخت یک سیستم نرم‌افزاری مطمئن و قابل اعتماد، کار بسیار دشواری است. درجه‌ی پیچیدگی خطاها، مساله‌ی بسیار مهمی است و یک خطای به ظاهر ساده می‌تواند عواقب امنیتی جبران‌ناپذیری را به‌همراه داشته‌باشد. به‌عنوان مثال، سرریز بافر، یک کلاس شناخته‌شده‌ی آسیب‌پذیری ناشی از اشتباهات ساده‌ی برنامه‌نویسی است که می‌تواند به یک مهاجم راه دور اجازه دهد تا کد دلخواه خود را روی یک کامپیوتر آسیب‌پذیر اجرا نموده و به این‌طریق، به آن نفوذ نماید.

برای جلوگیری از وقوع این نوع مشکلات، می‌توان با استفاده از ابزار خاصی، کد برنامه را پیش از توسعه‌ی آن مورد بررسی قرار داد و اشکالات احتمالی را برطرف نمود. هزینه‌ی جلوگیری از وقوع برخی از این اشکالات، بسیار کمتر از هزینه‌ی ترمیم آنها در زمان اجرا است.

برخی از مهم‌ترین ابزارهای کنترل امنیتی در زبان برامه‌نویسی C/C++ عبارتند از: FlawFinder، ITS4، RATS، و Splint.

2 ابزار FlawFinder

FlawFinder، یک برنامه‌ی ساده است که کدهای C/C++ را بررسی می‌کند و ضعف‌های امنیتی احتمالی ("نقص‌ها") را بر اساس سطح خطر، مرتب می‌نماید. پیش از انتشار برنامه برای عموم، می‌توان از این نرم‌افزار برای یافتن سریع و حذف برخی از مشکلات امنیتی بالقوه استفاده نمود. این نرم‌افزار، سازگار با CWE[1] است. این برنامه، توسط دیوید اِی ویلر[2]، در سال 2001 طراحی و پیاده‌سازی شده‌است و از زبان‌های برنامه‌نویسی C/C++ پشتیبانی می‌کند. برنامه‌ی فوق به زبان پایتون نوشته شده، در نتیجه، در تمام سیستم‌هایی که از این زبان پشتیبانی می‌کنند (UNIX، Windows، OS/2، Mac، Amiga، و غیره) قابل اجرا است.

FlawFinder، در سیستم های یونیکس (مانند گنو/لینوکس آزمایش‌شده) و ویندوز (با استفاده از Cygwin) کار می کند. این کار نیاز به اجرای پایتون 2 دارد (باید در نسخه‌ی 2.5 یا بالاتر کار کند، اما با نسخه‌ی 2.7 نیز آزمایش شده‌است).

[1] Common Weakness Enumeration (CWE)
[2] David A. Wheeler

1.2 نصب و راه‌اندازی FlawFinder

از آنجاکه FlawFinder به زبان پایتون نوشته شده‌است، نیازی به کامپایل کردن آن نیست. برنامه، شامل یک فایل پایتون است. بنابراین، به نصب نیاز ندارد. تنها کافی است که برنامه را از آرشیو خارج کرده و اجرا کنید.

اگر تاکنون این نرم‌افزار را نصب نکرده‌اید یا نسخه‌ی شما قدیمی است، می‌توانید مستقیما نرم‌افزار FlawFinder را دانلود و نصب نمایید. نسخه‌ی فعلی، FlawFinder 1.31 است. اگر می‌خواهید تغییرات آن را مشاهده کنید، بخش ChangeLog آن را ببینید. حتی می‌توانید کد برنامه‌ی FlawFinder را ببینید. فرض بر آن است که شما از یک سیستم یونیکس، مانند یک سیستم مبتنی بر لینوکس، استفاده می‌کنید. اگر از ویندوز استفاده می‌کنید، پیشنهاد می‌کنیم که ابتدا Cygwin و سپس FlawFinder را روی آن نصب نمایید. با این‌حال، گزارش شده‌است که FlawFinder مستقیماً نیز در ویندوز، کار می‌کند.

ابتدا آن را دانلود کنید. می‌توانید آخرین نسخه‌ی منتشر شده از FlawFinder را در قالب یک فایل tar.gz ، از آدرس https://www.dwheeler.com/flawfinder/flawfinder-1.31.tar.gz دریافت نمایید. همچنین، می‌توانید با وارد شدن به صفحه‌ی FlawFinder SourceForge، و از بخش Files آن، FlawFinder را دانلود کنید. برای ارسال نظر درباره‌ی این نرم‌افزار، گزارش خرابی یا اشکال در آن، و یا مشاهده‌ی آخرین نسخه‌ی این نرم‌افزار، می‌توانید از سایت SourceForge استفاده نمایید.

در سیستم‌های یونیکس، این نرم‌افزار را به‌طور عادی اجرا کنید. ابتدا فایل را از حالت فشرده خارج نمایید آن را برای اجرا، در root قرار دهید.

```
tar xvzf flawfinder-*.tar.gz
cd flawfinder-*
```

اگر در root هستید، sudo را حذف نمایید.

```
sudo make prefix=/usr install
```

می‌توانید این پیش‌فرض‌ها را با استفاده از قراردادهای استاندارد GNU لغو کنید. اگر دستور "*prefix=/usr*" را حذف کنید، آن را در پوشه‌ی پیش‌فرض "user/local/" ذخیره می‌کند. شما می‌توانید bindir و mandir را برای تعیین مکان‌های دلخواه خود تنظیم نمایید.

```
sudo make PYTHONEXT=.py install
```

ساده‌ترین راه برای اجرای FlawFinder در ویندوز، استفاده‌ی مستقیم از پایتون است. پایتون 2 (نسخه 2.5 یا بالاتر) را نصب کرده و اسکریپت FlawFinder را در خط فرمان، اجرا کنید. به‌عنوان مثال:

```
C:\Python27\Python.exe flawfinder -H -- savehitlist=ReportFolder\hitReport.hit
C:\MySourcesFolder
```

2.2 آسیب‌پذیری‌های کشف‌شده توسط FlawFinder

نرم‌افزار Splint، قادر به کشف کلاس‌های مختلفی از اشکالات برنامه‌نویسی است که می‌توانند منجر به آسیب‌پذیری شوند. این کلاس‌ها عبارتند از:

- سرریز بافر
- اجراهای Shell
- فایل‌های موقتی ناامن
- شرایط رقابتی
- نقض دسترسی
- رمزنگاری ضعیف
- ورودی ناامن کاربر

3.2 نحوه‌ی استفاده از FlawFinder

پس از راه‌اندازی نرم‌افزار Splint، می‌توان با استفاده از یکی از دستورات زیر، کد برنامه‌ی C/C++ را جهت تجزیه و تحلیل امنیتی، به آن وارد کرد:

```
flawfinder *.c
flawfinder source/
```

یکی از دو دستور بالا را بنویسید و نام یا آدرس برنامه‌ی موردنظر را جلوی آن دستور، قرار دهید. با این‌کار، برنامه‌ی موردنظر، در اختیار Splint قرار می‌گیرد.

4.2 عملکرد FlawFinder

FlawFinder، با استفاده از یک پایگاه داده‌ی داخلی C/C++، که دربردارنده‌ی مشکلات شناخته‌شده است، کار می‌کند. این مشکلات شامل مواردی مانند خطرات سرریز بافر (مانند دستورات *strcpy()*، *strcat()*، *get()*، *sprintf()* و خانواده‌ی *scanf()*)، مشکلات قالب رشته‌ها (مانند دستورات *printf[f][v]*، *[v]*، *snprintf()* و*syslog()*)، شرایط مسابقه (مانند دستورات *access()*، *chown()*، *chgrp()*، *chmod()*، *tmpnam()*، *tempnam()* و *mktemp()*)، خطرات بالقوه‌ی پوسته (اکثر دستورات مربوط به خانواده‌های *exec()*، *system()* و *popen()*) و دستورات ضعیف مربوط به تولید اعداد تصادفی (مانند *random()*) هستند. نکته‌ی مثبت ماجرا این است که شما مجبور نیستید این کتابخانه را ایجاد کنید، زیرا این کتابخانه همراه با نرم‌افزار، نصب می‌گردد.

این نرم‌افزار، کد برنامه‌ی موردنظر را بررسی کرده و لیستی از نقص‌های امنیتی احتمالی را که بر اساس شدت خطر، مرتب شده‌اند، تهیه می‌کند. شدیدترین خطرها، در بالای لیست قرار دارند. درجه‌ی شدت خطر از 0 (کمترین میزان خطر) تا 5 (بیشترین شدت خطر) تغییر می‌یابد. این سطح خطر به عملکرد یک تابع و پارامترهای موجود در آن، بستگی دارد. به‌عنوان مثال، رشته‌های ثابت، خطر بیشتری نسبت به رشته‌های با سایز متغیر دارند. FlawFinder برای رفع این مشکل، رشته‌ها را از طریق Gettext (یک کتابخانه‌ی مشترک برای برنامه‌های بین‌المللی) دریافت می‌کند و آنها را پردازش می‌نماید. با این‌کار، تعداد اشکلات پیداشده‌ی کاذب، کاهش می‌یابد.

FlawFinder حتی نسبت به ابزار «grep» در لینوکس نیز، اطلاعات و اولویت‌بندی بهتری را فراهم می‌سازد. همچنین، می‌داند که باید نظرات[3] را نادیده بگیرد و پارامترهایی را برای ارزیابی خطر، بررسی نماید. با این‌وجود، FlawFinder اساسا یک برنامه‌ی ساده است. این نرم‌افزار حتی در مورد انواع داده‌ها نیز اطلاعاتی ندارد و نمی‌تواند کنترل جریان یا تجزیه و تحلیل جریان داده‌ها را انجام دهد.

همان‌طور که در بالا ذکر شد، FlawFinder در واقع معنای کد را درک نمی‌کند و تنها با تطبیق الگوی متن ساده و نادیده گرفتن نظرات و رشته‌ها، برخی مشکلات کد را می‌یابد. با این‌وجود، FlawFinder می‌تواند کمک شایانی در یافتن و رفع آسیب‌پذیری‌های امنیتی به‌حساب آید.

[3] Comments

5.2 ورودی برنامه FlawFinder

نرم‌افزار Splint، متن ساده‌ی کد برنامه‌ی C/C++ را به‌عنوان ورودی دریافت می‌کند.

6.2 خروجی برنامه‌ی FlawFinder

این برنامه، آسیب‌پذیری‌های امنیتی بالقوه را بر اساس شدت خطر، در خروجی نشان می‌دهد. این برنامه، نام فایل، شماره‌ی خط برنامه، میزان شدت خطر، کلاس (طبقه‌بندی) آسیب و نام تابع آسیب‌پذیر را مشخص می‌نماید. سپس، توضیح می‌دهد که چرا این تابع دارای مشکل است. همچنین، اطلاعاتی شامل تعداد خطوط برنامه، تعدا کل آسیب‌پذیری‌های کشف‌شده، آسیب‌پذیری‌های مبتنی بر شدت خطر، و آسیب‌پذیری‌های موجود در هر خط را نشان می‌دهد.

7.2 اجرای چند مثال توسط FlawFinder

در این بخش سعی داریم با اجرای چندین مثال در محیط FlawFinder، با محیط کار این برنامه آشنا شویم. برنامه‌ها، در نرم‌افزار Visual Studio.net نسخه‌ی 2013، اجرا و تست در FlawFinder نسخه‌ی 2.0.2 ، تحت سیستم‌عامل اوبونتو، تست شده‌اند.

2.7.1 اجرای کد عاری از اشکال

در ابتدا، برنامه‌ی ساده‌ای را که فاقد هرگونه اشکال امنیتی است، مورد بررسی قرار می‌دهیم. تصویر (1)،

```
#include "stdafx.h"
#include "iostream"
#include "conio.h"
using namespace std;

int main()
{
    cout << "Hello!";
    _getch();
    return 0;
}
```

شکل 1: کد برنامه‌ی صحیح

کد برنامه‌ی صحیح و عاری از مشکل را نشان می‌دهد.

در مرحله‌ی بعد، برنامه‌ی فوق را در FlawFinder اجرا می‌نماییم. تصویر (2)، نتیجه‌ی حاصل از این اجرا را نشان می‌دهد. همان‌گونه که در تصویر با کادر قرمز نشان داده شده‌است، برنامه با اعلان پیام "No hits found" اعلام می‌دارد که کد، فاقد خطا است.

```
Setting up flawfinder (1.31-1) ...
ubuntu@ubuntu:/mnt/hgfs/SharedUbuntu/flawfinder-2.0.2$ flawfinder ConsoleApplica
tion1/
Flawfinder version 1.31, (C) 2001-2014 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 169
Examining ConsoleApplication1/ConsoleApplication1.cpp
Examining ConsoleApplication1/ConsoleApplication1/stdafx.cpp
Examining ConsoleApplication1/ConsoleApplication1/stdafx.h
Examining ConsoleApplication1/ConsoleApplication1/targetver.h

FINAL RESULTS:


ANALYSIS SUMMARY:

No hits found.
Lines analyzed = 47 in approximately 0.12 seconds (387 lines/second)
Physical Source Lines of Code (SLOC) = 32
Hits@level = [0]   0 [1]   0 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   0 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+]   0 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Minimum risk level = 1
There may be other security vulnerabilities; review your code!
See 'Secure Programming for Linux and Unix HOWTO'
(http://www.dwheeler.com/secure-programs) for more information.
ubuntu@ubuntu:/mnt/hgfs/SharedUbuntu/flawfinder-2.0.2$
```

شکل 2: نتیجه‌ی اجرای کد سالم تصویر (1) در FlawFinder

2.7.2 اجرای کدهای دارای اشکال

در این قسمت، قصد داریم چند نمونه از کدهای دارای اشکال را در FlawFinder اجرا نماییم. سرریز بافر، کدهای دارای shell، ورودی‌های ناامن کاربر، و فرمت داده مثال‌هایی هستند که در این بخش از آنها استفاده خواهیم نمود.

2.7.2.1 کد دارای اشکال shell

تصویر (3)، کد برنامه‌ای را در نشان می‌دهد که از اشکال shell رنج می‌برد. همان‌گونه که در توضیحات مشاهده می‌شود، این کد باعث می‌شود تا برنامه‌ی جدید اجرا گردد، که استفاده‌ی ایمن از آن، دشوار است. بنابراین، برنامه پیشنهاد می‌دهد که کتابخانه‌ای فراخوانی شود که همان کاربرد را پیاده‌سازی می‌نماید. پس از اجرای آن در FlawFinder، این برنامه به کاربر اعلام می‌دارد که برنامه دارای اشکال shell است و پیام

"Hits = 1" بیان‌گر وجود تنها یک مشکل در کد است. نتایج حاصل از اجرای برنامه‌ی فوق در FlawFinder، در شکل (4) مشاهده می‌شود.

```
#include "stdafx.h"
#include "iostream"
#include "conio.h"
using namespace std;
int num = 11;
unsigned long long int number = 22;
int  Divisor()
{
    int result;
    result = number%num;

    if (result == 0 && num < 21)
    {
        num + 1;
        Divisor();

        if (num == 20 && result == 0)
        {
            return number;
        }
    }
    else if (result != 0)
    {
        number++;
        Divisor();
    }}
int main()
{
    Divisor();
    cout << endl << endl;
    system("PAUSE");
    _getch();
    return 0;}
```

شکل 3: کد برنامه‌ی دارای اشکال shell

```
ubuntu@ubuntu:/mnt/hgfs/SharedUbuntu/flawfinder-2.0.2$ flawfinder ConsoleApplica
tion1/
Flawfinder version 1.31, (C) 2001-2014 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 169
Examining ConsoleApplication1/ConsoleApplication1/ConsoleApplication1.cpp
Examining ConsoleApplication1/ConsoleApplication1/stdafx.cpp
Examining ConsoleApplication1/ConsoleApplication1/stdafx.h
Examining ConsoleApplication1/ConsoleApplication1/targetver.h

FINAL RESULTS:

ConsoleApplication1/ConsoleApplication1/ConsoleApplication1.cpp:40:  [4] (shell)
 system:
  This causes a new program to execute and is difficult to use safely
  (CWE-78). try using a library call that implements the same functionality
  if available.

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 75 in approximately 0.09 seconds (841 lines/second)
Physical Source Lines of Code (SLOC) = 60
Hits@level = [0]   0 [1]   0 [2]   0 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   1 [1+]   1 [2+]   1 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 16.6667 [1+] 16.6667 [2+] 16.6667 [3+] 16.6667 [4+] 16.
6667 [5+]   0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming for Linux and Unix HOWTO'
(http://www.dwheeler.com/secure-programs) for more information.
```

شکل 4: نتایج حاصل از اجرای کد شکل (3) در FlawFinder

2.7.2.2 کد حاوی اشکال فرمت داده

در این بخش، کد برنامه‌ای ارائه داده می‌شود که دارای مشکل فرمت داده است. کد برنامه، در تصویر (5) مشاهده می‌شود.

```
#include"iostream"
#include"stdafx.h"
#include"conio.h"

using namespace std;
int main(int argc, char* argv[])
{
    if (argc > 1)
        printf(argv[1]);
    _getch();
    return 0;
}
```

شکل 5: کد حاوی اشکال فرمت داده

پس از اجرای کد در FlawFinder، نتیجه به‌صورت شکل (6) خواهد بود. همان‌گونه که در تصویر مشاهده می‌شود، FlawFinder اعلام می‌دارد که تنها یک مشکل (“Hits = 1”) و آن هم در فرمت *printf* وجود دارد. در واقع، برنامه اعلام می‌دارد، اگر فرمت‌های رشته‌ای، مورد استفاده‌ی مهاجم قرار گیرند، قابل

```
ubuntu@ubuntu:/mnt/hgfs/SharedUbuntu/flawfinder-2.0.2$ flawfinder ConsoleApplica
tion1/
Flawfinder version 1.31, (C) 2001-2014 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 169
Examining ConsoleApplication1/ConsoleApplication1/ConsoleApplication1.cpp
Examining ConsoleApplication1/ConsoleApplication1/stdafx.cpp
Examining ConsoleApplication1/ConsoleApplication1/stdafx.h
Examining ConsoleApplication1/ConsoleApplication1/targetver.h

FINAL RESULTS:

ConsoleApplication1/ConsoleApplication1/ConsoleApplication1.cpp:11:  [4] (format
) printf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification.

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 47 in approximately 0.07 seconds (657 lines/second)
Physical Source Lines of Code (SLOC) = 34
Hits@level = [0]   0 [1]   0 [2]   0 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   1 [1+]   1 [2+]   1 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 29.4118 [1+] 29.4118 [2+] 29.4118 [3+] 29.4118 [4+] 29.
4118 [5+]   0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming for Linux and Unix HOWTO'
(http://www.dwheeler.com/secure-programs) for more information.
```

شکل 6: نتیجه‌ی حاصل از اجرای کد تصویر (5) در FlawFinder

سوءاستفاده خواهند بود. از این‌روی، پیشنهاد می‌دهد که از یک ثابت، به‌عنوان فرمت *printf*، استفاده گردد.

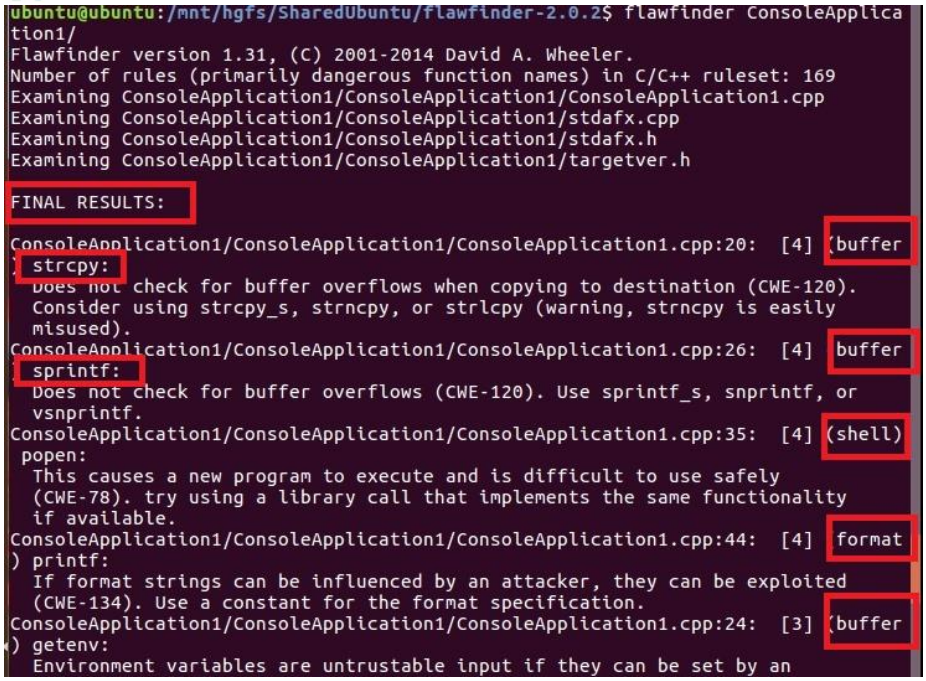
2.7.2.3 کد دارای اشکالات سرریز بافر، Shell، فرمت داده، و ورودی ناامن کاربر

در شکل (7)، کد برنامه‌ای را مشاهده می‌کنید که دارای اشکالات سرریز بافر، shell، فرمت داده، و ورودی ناامن کاربر است.

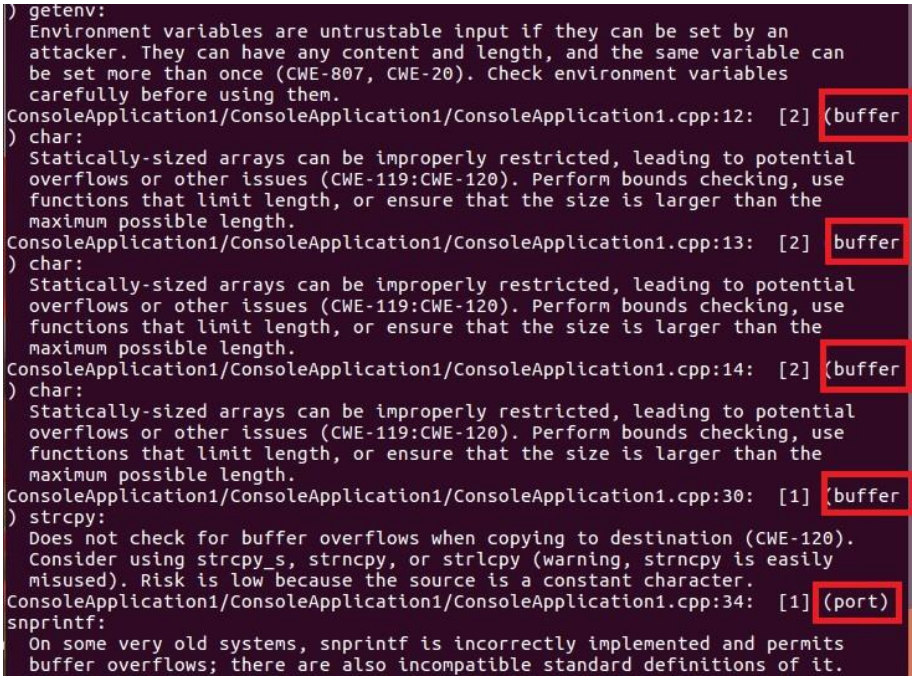
```
#include "stdafx.h"
#include "iostream"
#include "conio.h"
#include <stdio.h>
#include <stdlib.h>
using namespace std;
int main(int argc, char *argv[])
{
    char dir[1024];
    char cmd[1200];
    char buff[1024];
    FILE *fp = NULL;
    int i = 0;

    if (argc == 2)
    {
        strcpy(dir, argv[1]);
    }
    else
    {
        if (getenv("HOME") != NULL)
        {
            sprintf(dir, "%s", getenv("HOME"));
        }
        else
        {
            strcpy(dir, "/");
        }
    }

    snprintf(cmd, sizeof(cmd)-1, "%s %s", "ls", dir);
    fp = popen(cmd, "r");
    if (fp == NULL)
    {
        if (getenv("HOME") != NULL)
        {
            sprintf(dir, "%s", getenv("HOME"));
        }
        else
        {
            strcpy(dir, "/");
        }
    }

    snprintf(cmd, sizeof(cmd)-1, "%s %s", "ls", dir);
    fp = popen(cmd, "r");
    if (fp == NULL)
    {
        printf("Failed to invoke: %s\n", cmd);
        return -1;
    }

    while (i = fread(buff, 1, sizeof(buff), fp))
    {
        printf(buff);
    }

    pclose(fp);

    return 0;
}
```

شکل 7: کد برنامه‌ای که دارای اشکالات سرریز بافر، shell، فرمت داده، و ورودی ناامن کاربر است

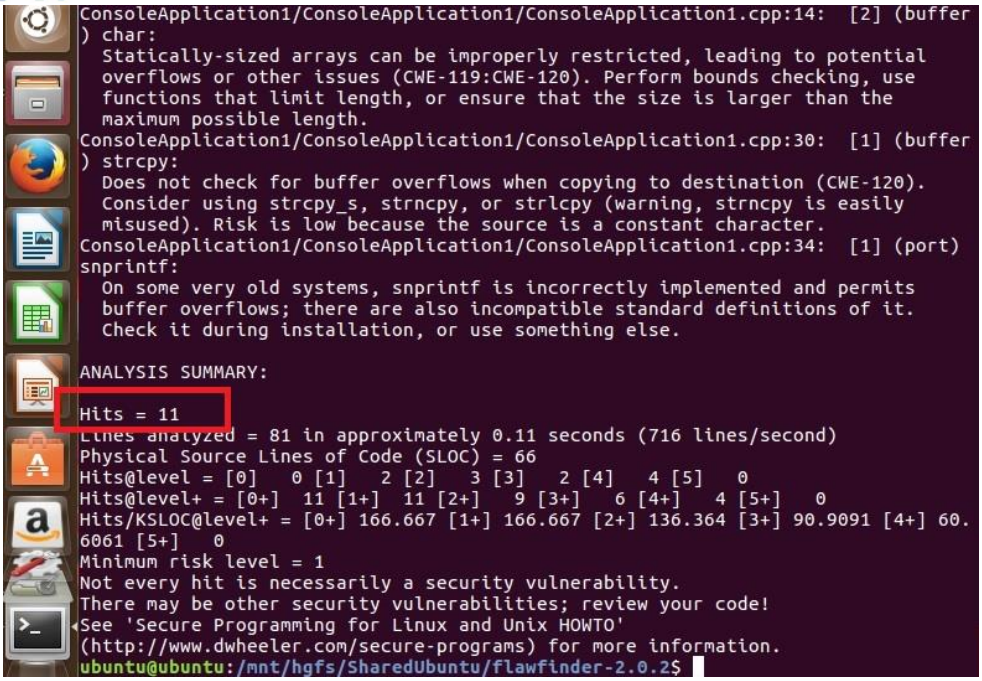
پس از اجرای کد در FlawFinder، برنامه با اعلان پیام‌های "buffer"، "shell"، "format"، و "port"، به‌ترتیب اعلام می‌دارد که کد، از مشکلات سرریز بافر، shell، فرمت داده، و ورودی ناامن کاربر، رنج می‌برد. این نتایج، در تصاویر (8) و (9) قابل مشاهده هستند. همچنین، FlawFinder این قابلیت را دارد که تعداد اشکالات کد را اعلام نماید. ویژگی مذکور، در شکل (10) مشاهده می‌شود.



شکل 8: نتایج حاصل از اجرای کد تصویر (7) در FlawFinder



شکل 9: نتایج حاصل از اجرای کد تصویر (7) در FlawFinder



شکل 10: نمایش تعداد خطاهای حاصل از اجرای کد تصویر (7) در FlawFinder

3 ابزار RATS

نرم‌افزار RATS[4]، یک ابزار منبع‌باز است که توسط مهندسان نرم‌افزار امنیتی طراحی و تولید شده‌است. این نرم‌افزار می‌تواند زبان‌های مختلفی، از جمله C، C ++، Perl، PHP و پایتون، را مورد بررسی قرار دهد. نرم‌افزار RATS، بسیار سریع است و به‌راحتی می‌تواند بدون ایجاد سربار قابل توجهی، با فرآیند تولید برنامه ادغام گردد.

همان‌طور که از نام این نرم‌افزار پیداست، تنها قادر به یافتن اشکالات بزرگ و ناهنجار زبان‌های برنامه‌نویسی است. به‌عنوان مثال، قادر به یافتن معایب طراحی الگوریتم در زبان C/C++ نیست. در نتیجه، همچنان به بررسی دستی کد، جهت یافتن اشکالات احتمالی، نیازمندیم.

RATS، یک نرم‌افزار رایگان و متن باز است و می‌توانید تغییرات دلخواه خود را بر اساس GNU Public License در آن ایجاد نمایید.

[4] Rough Auditing Tool for Security (RATS)

1.3 نصب و راه‌اندازی RATS

RATS برای نصب و اجرا به یک سیستم expat نیاز دارد. سیستم expat، یک کتابخانه‌ی تحلیل‌گر[5] جریان‌گرای[6] مبتنی بر XML است که با زبان C نوشته شده‌است. معمولا Expat در مسیر "usr/local/lib/" و "usr/local/include/" نصب می‌شود. در برخی از سیستم‌ها، باید تنظیمات نصب را با استفاده از گزینه‌های expat-lib و expat-include انجام دهید تا نصب و راه‌اندازی کتابخانه و سرآیندها (هِدرها)، قابل شناسایی باشد.

نصب و راه‌اندازی نرم‌افزار RATS، بسیار ساده است. برای راه‌اندازی این نرم‌افزار، کافی است که اسکریپت تنظیمات مربوط به پوسته‌ی برنامه را در مسیر اصلی، اجرا کنید:

```
./configure
```

اسکریپت تنظیمات، یک اسکریپت عمومی خودکار است که گزینه‌های زیادی دارد. می‌توانید با استفاده از دستور *help–*، تمامی گزینه‌های موجود در این اسکریپت را ببینید.

به‌محض این‌که کار اسکریپت تنظیمات تمام شد، می‌توانید با نوشتن دستور *make* در مسیر اصلی، این برنامه را راه‌اندازی نمایید:

```
make
```

به‌طور پیش‌فرض، نرم‌افزار RATS در مسیر "usr/local/bin/" و پایگاه داده‌ی آسیب‌پذیری آن، در مسیر "usr/local/lib/" نصب می‌شود. می‌توانید با استفاده از دستور *prefix–*، مسیر نصب برنامه را تغییر دهید. همچنین، می‌توانید با استفاده از دستورات *bindir–* و *datadir–*، برای فایل‌های نصب‌شده روی سیستم، مسیرهای دقیق‌تری را انتخاب نمایید. برای نصب این نرم‌افزار، می‌توانید از دستور زیر استفاده کنید:

```
make install
```

این‌کار باعث می‌شود تا نسخه‌ی باینری فایل rats در مسیر نصب binary آن و همچنین فایل rats.xml، که فایل پایگاه داده‌ی آسیب‌پذیری است، در مسیر نصب data، کپی شوند.

[5] Parser
[6] Stream-Oriented

2.3 آسیب‌پذیری‌های کشف‌شده توسط RATS

برخی از اشکالاتی که توسط این نرم‌افزار پیدا می‌شوند عبارتند از:

زبان برنامه‌نویسی	اشکال
C/C++	سرریز بافر (Time of Check, Time of Use) TOCTOU اجراهای Shell فایل‌های موقتی ناامن شرایط رقابتی نقض دسترسی رمزنگاری ضعیف ورودی ناامن کاربر
PHP	متاسفانه، قابلیت زیادی در یافتن اشکالات زبان PHP، مانند اشکالات مربوط به Cross-Site یا دستورات مربوط به تزریق SQL، را ندارد.
پایتون	تنها قادر به یافتن اشکالات مربوط به توابع داخلی/کتابخانه‌ای پایتون است.
Perl	اشکالات مربوط به توابع داخلی Perl را می‌یابد.

3.3 نحوه‌ی استفاده از RATS

برای اجرای این برنامه و تحلیل کد موردنظر، می‌توان از یکی از دستورات زیر استفاده نمود:

```
rats *.c
rats source/
```

این نرم‌افزار، دارای دستورات خط فرمان زیادی است که برای بررسی کد برنامه‌ی موردنظر و یافتن اشکالات احتمالی، از آنها استفاده می‌شود. اگر هیچ کد برنامه‌ای برای بررسی تعیین نشود، این برنامه به‌صورت پیش‌فرض، فایلی به نام stdin را مورد بررسی قرار می‌دهد. شکل کلی یکی از دستورات خط فرمان نرم‌افزار RATS، به‌صورت زیر است:

```
usage: rats [options] [file]...
```

4.3 عملکرد RATS

این نرم‌افزار، دارای پایگاه داده‌ای از آسیب‌پذیری‌های امنیتی احتمالی است. می‌توان از چندین پایگاه داده نیز استفاده نمود. در این حالت، تمامی پایگاه‌های داده‌ی موردنظر، بارگذاری و استفاده می‌شوند.

با استفاده از دستور *-i, --input*، می‌توان توابعی را ایجاد نمود که ورودی‌ها را دریافت و آنها را مورد پردازش قرار می‌دهند و سپس، نتایج آن را در قالب گزارش آسیب‌پذیری، به نمایش می‌گذارند.

نرم‌افزار RATS، دارای سه سطح هشدار است که شدت خطر آسیب‌پذیری را مشخص می‌کنند: سطح 1، شامل آسیب‌پذیری‌های شدید؛ سطح 2، شامل آسیب‌پذیری‌های متوسط؛ و سطح 3، شامل آسیب‌پذیری‌های سطح پایین. در هنگام کار با این ابزار، می‌توان سطح هشدار را مشخص نمود.

RATS، تمام فایل‌های مشخص‌شده در خط فرمان را اسکن می‌کند و پس از پایان اسکن، یک گزارش ارائه می‌دهد. آسیب‌پذیری‌های موجود در گزارش نهایی، به اطلاعات موجود در پایگاه داده‌ی آسیب‌پذیری یا پایگاه‌های داده‌ی مورد استفاده و سطح هشدار استفاده‌شده، بستگی دارند.

5.3 ورودی برنامه RATS

ورودی این برنامه می‌تواند به‌صورت کد ساده‌ی C/C ++/Perl/PHP/Python باشد. فایل‌ها باید با پسوند مناسبی تعریف شوند تا بتوانند مورد تحلیل قرار گیرند. این پسوندها می‌توانند c.، cpp.، php.، pl. و py. باشند. در این‌حالت، باید گزینه‌ی زبان ابزار RATS، با استفاده از دستور *-l <lang>, --language <lang>* تنظیم گردد تا بتواند کد آن زبان را تحلیل و عیب‌یابی نماید.

6.3 خروجی برنامه RATS

پس از تحلیل کد موردنظر، برنامه‌ی RATS، گزارشی از آسیب‌پذیری‌های احتمالی ارائه می‌دهد که به‌ترتیب، بر اساس شدت خطر، نام تابع، نام فایل، و شماره‌ی خط کد، مرتب شده‌اند. سپس، توضیحاتی درباره‌ی هر یک از گزینه‌های آسیب‌پذیری مطرح‌شده، ارائه می‌شود. در نهایت، نرم‌افزار RATS، تعدا کل خطوط بررسی و زمان مصرفی را نشان می‌دهد.

4 ابزار ITS4

نرم‌افزار ITS4، برنامه‌ای برای بررسی کدهای C/C++ جهت یافتن آسیب‌پذیری‌های امنیتی است. این نرم‌افزار دارای دقت و صحت عملکرد بیشتری نسبت به سایر برنامه‌های مشابه است. همچنین، از سرعت قابل توجهی برخوردار است. ITS4، 9000 خط کد را در هر ثانیه، پردازش می‌کند. تکنیک این نرم‌افزار بسیار ساده است و به‌رغم پیچیدگی‌های ذاتی زبان، به‌راحتی روی کدهای C++ اعمال می‌شود. از این نرم‌افزار برای یافتن آسیب‌پذیری‌های جدید در طیف گسترده‌ای از بسته‌های نرم‌فزاری، مانند نرم‌افزارهای تجارت الکترونیک، استفاده شده‌است.

ITS4، یک ابزار کاربردی است که به‌منظور شناسایی ساختارهایی که به‌طور بالقوه در کدهای C و C++ ناامن هستند، به‌طور گسترده‌ای توسعه یافته‌است.

پیش از نرم‌افزار ITS4، از نرم‌افزار grep، که یک برنامه‌ی متنی و تحت خط فرمان بود، استفاده می‌شد. هدف، یافتن مکان‌هایی از برنامه بود که ممکن بود دچار خطاهای شناسایی‌شده‌ی احتمالی باشند. برای این‌کار، از کتابخانه‌ها یا پایگاه‌های داده‌ای استفاده می‌شد که لیستی از آسب‌پذیری‌های احتمالی را در خود نگه می‌دارند.

نرم‌افزار ITS4، کد برنامه‌ی موردنظر را به‌صورت کاملا ساده، تجزیه و تحلیل می‌کند. دلیل این سادگی، استراتژی مورد استفاده در این نرم‌افزار است.

1.4 نصب و راه‌اندازی ITS4

برای نصب و راه‌اندازی نرم‌افزار ITS4، به یک کامپایلر C/C++ نیاز دارید. ابتدا، فایل نصب ITS4 را دانلود کرده و آن را باز کنید و به مسیر اصلی سیستم خود منتقل نمایید. سپس، آن را اجرا کنید. اکنون، فایل configure/. را اجرا نمایید. پس از آن، در مسیر root قرار گیرید و دستورات "*make*" و "make install" را اجرا کنید. ITS4، به‌طور پیش‌فرض، در مسیر "usr/local/" نصب می‌شود. فراموش نکنید که باید اسکریپت emacs script/its4.el را از محل فایل دانلودشده، به فایل emacs./~ خود، منتقل نمایید.

2.4 آسیب‌پذیری‌های کشف‌شده توسط ITS4

- سرریز بافر

- مشکلات مربوط به فرمت رشته‌ها
- اجرای Shell
- TOCTOU
- استفاده از مولد ضعیف اعداد تصادفی
- ورودی‌های ناامن کاربر

3.4 نحوه‌ی استفاده از ITS4

ITS4 را می‌توان در خط فرمان، مورد استفاده قرار داد و همراه با برنامه‌هایی، همچون Emacs و Microsoft Visual Studio، استفاده نمود. با تایپ دستور "M-x its4-scan" در Emacs، فایل (ها) را به ITS4 وارد کنید. سپس، ITS4، فایل را تجزیه و تحلیل می‌کند و نتایج را نمایش می‌دهد. اکنون، می‌توانید با حرکت دادن مکان‌نما روی خطا و فشردن «Enter»، یا با کلیک ماوس روی خطا، به خط منبع مربوط به خطا، پرش کنید.

اگر بخواهید برای اجرای ITS4 از خط فرمان استفاده نمایید، باید دستور زیر را تایپ کنید:

```
its4 *.c
```

ITS4، فایل‌ها را اسکن می‌کند و مشکلات موجود را نمایش می‌دهد.

4.4 عملکرد ITS4

در هر نوبت، یک یا چند فایل پردازش‌نشده‌ی C/C++ را از ورودی دریافت کرده و آنها را به یک دسته از نشانه‌های واژگانی، تقسیم می‌کند. سپس، این نشانه‌ها را با یک پایگاه داده، شامل الگوهای آسیب‌پذیری احتمالی، مطابقت می‌دهد. اطلاعات این پایگاه داده، با دست اضافه می‌شود. در نتیجه، الگوهای غیرمعمول نیز، قابل شناسایی خواهند بود. برای انجام تجزیه و تحلیل، یک درخت Parser به کار می‌رود. البته، این تطابق چندان هم دقیق نیست، زیرا ممکن است مواردی پیدا شوند که از نظر امنیتی مشکلی نداشته‌باشند. واضح‌ترین مثال، نام متغیرها است. به کد زیر توجه کنید:

```
#include "test.h"
int main()
{
	int strcpy;
	return 0;
}
```

با اجرای ITS4 روی این کد، نتیجه‌ی زیر حاصل می‌گردد:

```
[viega@lima c]$- its4 test.c
test.c:3:(Very Risky) strcpy
This function is high risk for buffer overflows.
Use strncpy instead.
```

واضح است که ما دوست داریم که از ایجاد این نتایج مثبت کاذب، جلوگیری به‌عمل آوریم، اما از آنجاکه نرم‌افزار قادر است ارجاعات خطرناک به متغیرها را نیز تشخیص دهد، از این عوارض ناخواسته، چشم‌پوشی می‌کنیم.

در حال حاضر، پایگاه داده‌ای که ITS4 از آن استفاده می‌کند، دارای 131 مورد شناسایی آسیب‌پذیری است که از منابعی مانند آرشیو Bugtraq، گردآوری شده‌است. Bugtraq، بزرگترین و جامع‌ترین پایگاه اینترنتی دانش و منابع امنیتی کامپیوتر است که برای عموم، آزاد است. بزرگترین کلاس آسیب‌پذیری‌ها در این پایگاه، مشکلات مربوط به شرایط رقابتی فایل‌ها است. درجه‌ی دوم این مشکلات، متعلق به کلاس سرریز بافر است.

5.4 ورودی برنامه ITS4

ورودی برنامه‌ی ITS4 باید به‌صورت کد ساده‌ی C/C++ باشد.

6.4 خروجی برنامه ITS4

برنامه‌ی ITS4، فایل ورودی را تحلیل کرده و سپس، گزارشی ارائه می‌کند که شامل موارد زیر است:

- شرح مختصری از آسیب‌پذیری
- توضیحات جامع‌تری درباره‌ی کدنویسی مربوط به آسیب‌پذیری
- ارزیابی نسبی شدت آسیب‌پذیری، که یکی از موارد زیر خواهد بود: بدون خطر، کم‌خطر، خطر متوسط، خطرناک، خطر زیاد، خطر بسیار زیاد.
- علامتی که مشخص می‌کند آسیب‌پذیری از چه کلاسی است.
- تشخیص این‌که آیا توابع موجود در برنامه می‌توانند ورودی‌های یک منبع خارجی مانند فایل یا سوکت را بازیابی نمایند یا خیر.

5 ابزار Splint

نرم‌افزار Splint، یک ابزار جالب برای بررسی ایستای وجود آسیب‌پذیری‌های امنیتی و اشتباهات برنامه‌نویسی در برنامه‌های C/C++ است. این نرم‌افزار، توسط گروه برنامه‌ریزی امن دانشگاه ویرجینیا توسعه داده‌شده‌است. دیوید اِوانس[7]، مدیر پروژه و توسعه‌دهنده‌ی اصلی Splint است. Splint، جانشینی برای پروژه‌ی LCLint است، که در ابتدا به‌عنوان یک پروژه‌ی تحقیقاتی مشترک بین موسسه فناوری ماساچوست و مرکز تحقیقات سیستم‌های دیجیتال طراحی شد. ابزار Splint، شامل کنترل‌کننده‌ی اصلی LCL است. نام Splint، از عبارت Specification Lint گرفته شده‌است. Lint یک ابزار برنامه‌نویسی رایج، برای شناسایی ناهنجاری‌ها در برنامه‌های C است.

1.5 نصب و راه‌اندازی Splint

برای دانلود ابزار Splint، می‌توانید به یکی از دو آدرس http://www.splint.org/linux.html یا http://www.splint.org/source.html مراجعه کنید. پس از دانلود، آن را از حالت فشرده خارج نمایید و در مسیر اصلی سیستم قرار دهید. اکنون، عبارت configure/. را تایپ کنید و دکمه‌ی Enter را بفشارید. سپس، دستور *make* را اجرا نمایید. برنامه اجرا می‌شود.

اگر از نسخه‌ی باینری ابزار Splint استفاده می‌کنید، باید تنظیمات زیر را انجام دهید:

- مقدار LARCH_PATH را به $HOME/src/splint-3.1.1/lib/ تنظیم کنید.
- مقدار LCLIMPORTDIR را به $HOME/src/splint-3.1.1/imports/ تنظیم کنید.
- $HOME/src/splint-3.1.1/lib/ را به PATH اضافه کنید.
- اکنون می‌توانید نسخه‌ی باینری Splint خود را اجرا کنید.

2.5 آسیب‌پذیری‌های کشف‌شده توسط Splint

Splint، حاوی بسیاری از کنترل‌های سنتی Lint، شامل اعلان‌های بلااستفاده، عدم تطابق نوع، استفاده قبل از تعریف، کد غیرقابل دسترسی، مقادیر بازگشتی نادیده گرفته‌شده، مسیرهای اجرای بدون بازگشت، حلقه‌های بی‌نهایت، و عدم دست‌یابی به نتیجه در case بعدی، است. با استفاده از اطلاعات اضافی به‌دست‌آمده از کد، می‌توان کنترل‌های دقیق‌تری روی آن انجام داد.

[7] David Evans

مشکلاتی که توسط نرم‌افزار Splint شناسایی می‌شوند، عبارتند از:

- ارجاع اشاره‌گر تهی
- استفاده از حافظه‌ی تعریف‌نشده
- عدم تطابق نوع داده
- نقض پنهان بودن اطلاعات
- اشتباهات مدیریت حافظه، شامل ارجاعات سرگردان و نشت حافظه
- نام‌گذاری‌های خطرناک
- استفاده یا تغییر متغیرهای عمومی، که منجر به ایجاد تناقض در مقدار می‌شود
- حلقه‌های بی‌نهایت و عدم دست‌یابی به نتیجه در case بعدی
- سرریز بافر
- پیاده‌سازی یا اعلان‌های خطرناک
- نقض قرارداد نام‌گذاری سفارشی

3.5 نحوه‌ی استفاده از Splint

نرم‌افزار Splint را همراه با کد مورد بررسی، به‌صورت زیر، فراخوانی نمایید.

```
splint *.c
```

در بسیاری از موارد، هنگام اجرای برنامه و بررسی کد، هشدارهایی ظاهر می‌شوند. برای خاموش کردن بخش هشداردهنده‌ی نرم‌افزار Splint، از دستور زیر استفاده نمایید:

```
splint -weak *.c
```

4.5 عملکرد Splint

Splint برای انجام وظایف خود، از یک لیست استفاده می‌کند که شامل انواع کلاس‌های آسیب‌پذیری است و با تطبیق این لیست با کد مورد بررسی، اشکالات احتمالی را می‌یابد.

می‌توان Splint را به نحوی تنظیم نمود که تنها، برخی از کلاس‌های آسیب‌پذیری را مورد بررسی قرار دهد و به دنبال سایر مشکلات در کد موبوطه نگردد. همچنین، برنامه‌نویسان می‌توانند کلاس‌های جدیدی از آسیب‌پذیری را برای Splint تعریف نمایند و بدین‌طریق، قدرت Splint را افزایش دهند.

5.5 ورودی برنامه Splint

Splint، به کد برنامه‌ی مورد بررسی و تمامی هدرهای استفاده‌شده در آن، به‌عنوان ورودی، نیاز دارد.

6.5 ورودی برنامه‌ی Splint

این برنامه، پس از بررسی و تحلیل کد مورد نظر، گزارشی از آسیب‌پذیری‌های احتمالی، ارائه می‌دهد. این گزارش، شامل موارد زیر است:

- خلاصه‌ای از تمام اشکالات گزارش‌شده همراه با شماره‌ی خط
- نام تابعی که دارای اشکال است
- تعداد کل اشکالات پیداشده
- نام فایل بررسی‌شده
- تعدا کل خطوط برنامه‌ی بررسی‌شده
- زمان مصرفی

6 مقایسه

برای این‌که بتوان چهار ابزار معرفی‌شده را با یکدیگر مقایسه نمود، ابتدا باید مفاهیم و معیارهای ارزیابی را توضیح داد. این معیارها، در قالب جداولی نشان داده‌می‌شوند.

1.6 کلاس آسیب‌پذیری

کد	کلاس آسیب‌پذیری
V1	سرریز بافر ناشی از عملیات روی پشته
V2	سرریز بافر ناشی از عملیات روی هیپ[8]
V3	ارجاع به اشاره‌گر سرگردان

[8] Heap

V4	آسب‌پذیری‌های مربوط به فرمت رشته‌ها
V5	اشکالات مربوط به اعداد صحیح

2.6 محدودیت‌ها

کد	کلاس محدودیت
A1	به کد منبع نیاز دارد.
A2	تنها از توابع رشته‌ای کتابخانه‌ی libc پشتیبانی می‌کند.
A3	به کدنویسی اضافه جهت اجرای ابزار، نیاز دارد.

3.6 محدودیت‌های امنیتی

کد	محدودیت‌های امنیتی
P1	هیپ به‌صورت تصادفی انتخاب نمی‌شود.
P2	امکان جواب مثبت کاذب وجود دارد. یعنی، امکان دارد اشکالی پیدا شود که در واقع، یک اشکال امنیتی نیست.

4.6 نوع

کد	نوع
T1	تشخیص: تلاش‌هایی که برای بهره‌برداری از پارامترهای کد صورت می‌پذیرد، شناسایی شوند.
T2	پیشگیری: از آسیب‌پذیری، پیشگیری به‌عمل آید.
T3	کاهش تداخل: اعمال نفوذ مهاجم، محدود گردد.

5.6 واکنش

کد	واکنش
R1	برنامه خاتمه می‌یابد.
R2	برنامه خراب می‌شود.
R3	یک مشکل، گزارش می‌گردد.

6.6 مقایسه‌ی ابزارها

ابزار	آسیب‌پذیری	هزینه‌ی محاسباتی	هزینه‌ی سربار حافظه	محدودیت	محدودیت امنیتی	نوع	واکنش
FalwFinder	V1+V2+V4	خیلی پایین	خیلی پایین	A1+A2	P2	T2	R3
RATS	V1+V2+V4	خیلی پایین	خیلی پایین	A1+A2	P2	T2	R3
ITS4	V1+V2+V4	خیلی پایین	خیلی پایین	A1+A2	P2	T2	R3
Splint	V1+V2+V4	پایین	پایین	A1+A3	P2	T2	R3

7 نتیجه‌گیری

برای دست‌یابی به کدهای امن، باید استانداردها و معیارهای مشخصی جهت ارزیابی کد وجود داشته‌باشد. این استانداردها، توسط سازمان‌های معتبر بین‌المللی تدوین شده‌اند. در این میان، ابزارهایی مانند ITS4، Splint، RATS، و Falwfinder وجود دارند که با بهره‌گیری از استانداردهای مذکور، امکانی را فراهم می‌سازند تا بررسی آسیب‌پذیری‌های ناشی از عدم رعایت استانداردها، به‌صورت خودکار انجام پذیرد. اگر راه درازی تا کشف تمامی آسیب‌پذیری‌های امنیتی درپیش است، اما پیشرفت و گسترش این قبیل ابزارها، درخور توجه است.

مراجع

[1] http://www.dwheeler.com/flawfinder
[2] http://www.securesw.com/rats
[3] http://www.rstcorp.com/its4
[4] http://www.splint.org
[5] https://www.debian.org/security/audit/tools
[6] http://www.securiteam.com/tools/6E00L0K01K.html
[7] http://www.securityfocus.com/tools/2047
[8] https://www.security-database.com/toolswatch/RATS-v2-3-Rough-Auditing-Tool-for.html