Neutralizing Tofsee Spambot – Part 1 | Binary file vaccine

spamhaus.com/resource-center/neutralizing-tofsee-spambot-part-1-binary-file-vaccine/

April 6, 2023

The Spamhaus Malware Researchers have been busy in their lairs, reverse engineering Tofsee malware to provide you with the code required for two malware vaccines and a network-based kill switch. A hat trick of protection against this spambot! This is the first in this three-part series, and looks at how to inject a malware vaccine into the binary file.

An introduction to malware vaccines

This security concept works by proactively introducing a small piece of harmless code into a computer system to disrupt and prevent malware from executing and spreading. This is not dissimilar to how medical vaccines work (hence the use of the same terminology). Essentially, the premise is to “immunize” the system against specific types of malware by providing the system, in advance, with a form of defense.

There are various types of malware vaccines, including file-based, memory-based, and network-based. They can be delivered as standalone software tools or integrated into other security products such as antivirus software.

While malware vaccines can be an effective defense against certain types of malware, they should never be used as a substitute for other security measures such as keeping software and operating systems up to date, using strong passwords, and avoiding suspicious email attachments or downloads, to name but a few. It’s also important to note that as new malware strains are developed, the vaccines must be updated accordingly to remain effective.

Let’s move on to the malware taking center stage in this series…

An introduction to Tofsee

Tofsee, also known as **Gheg**, is a sophisticated modular malware primarily designed to send spam email along with other full-fledged botnet activities such as mining and stealing login and email credentials, as well as downloading further malware. Generally, the additional malware downloaded is either ransomware or banking Trojans.

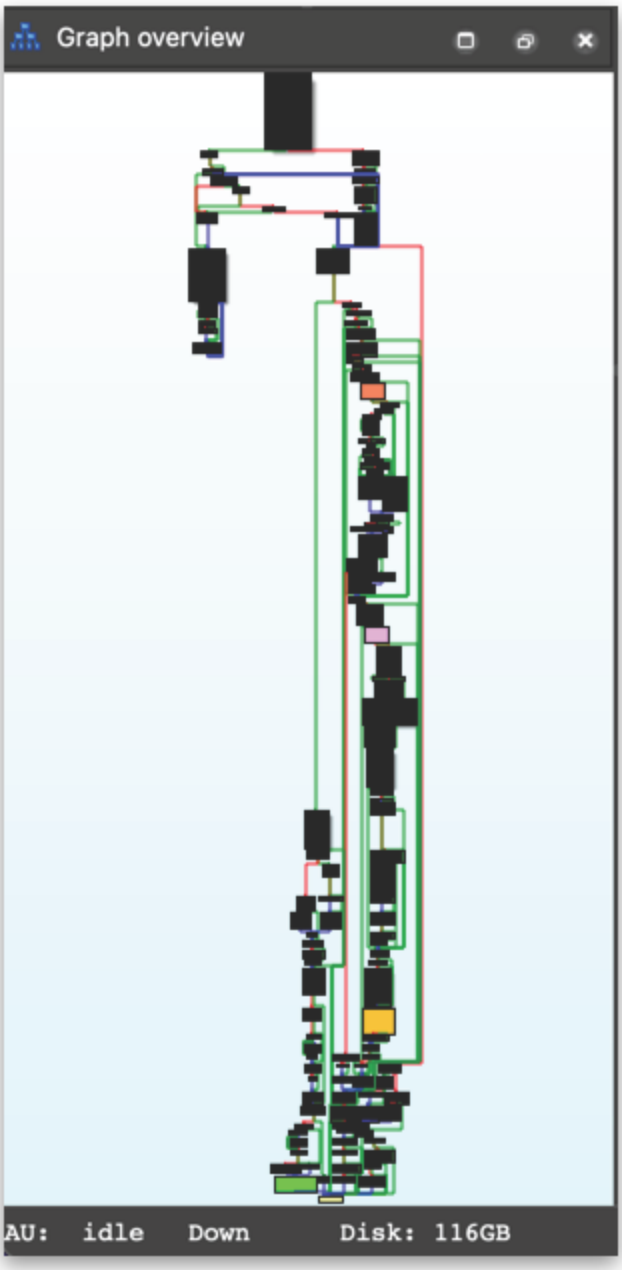
The malware is written in C/C++ and uses various techniques to avoid detection and remain persistent on infected systems.

Identifying where a vaccine can be “injected” in Tofsee

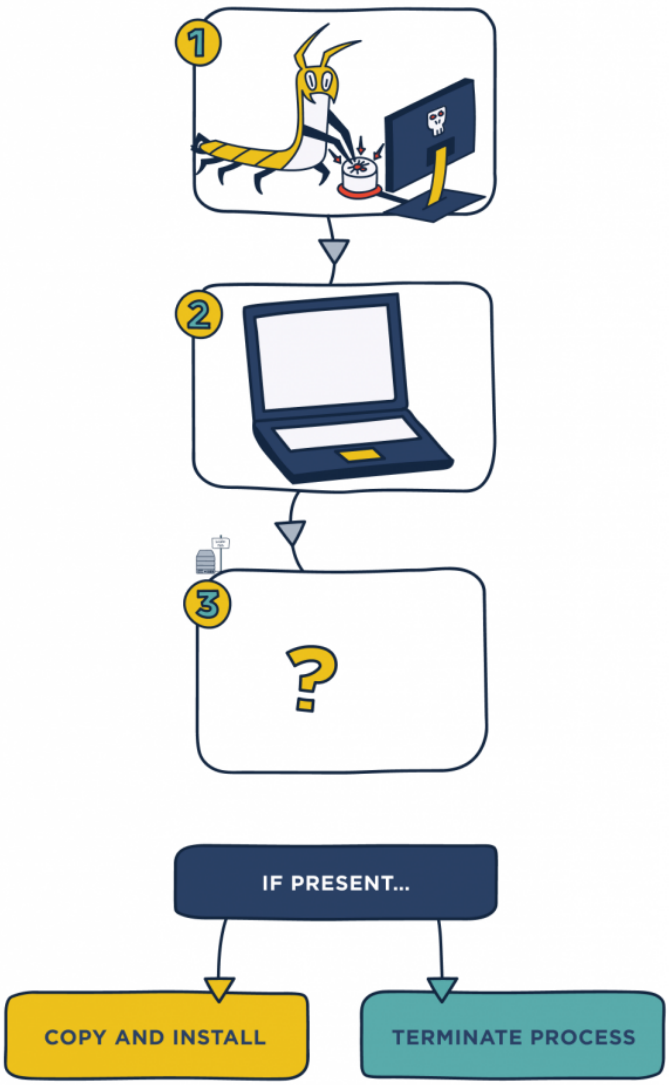
To create a vaccine for a malware family, you need to have the ability to mimic the existence of part of the malware, for example, its binary file. This tricks the malware into believing that an instance of the malware code is already running on the system and, therefore, won't try to re-infect it.

The first stage in identifying points to distract from the normal execution of the binary file is to reverse engineer the malware to understand the flow process of the code.

To explore the possibility of imitating the binary file, you need to check if it's in the installer/installed path.

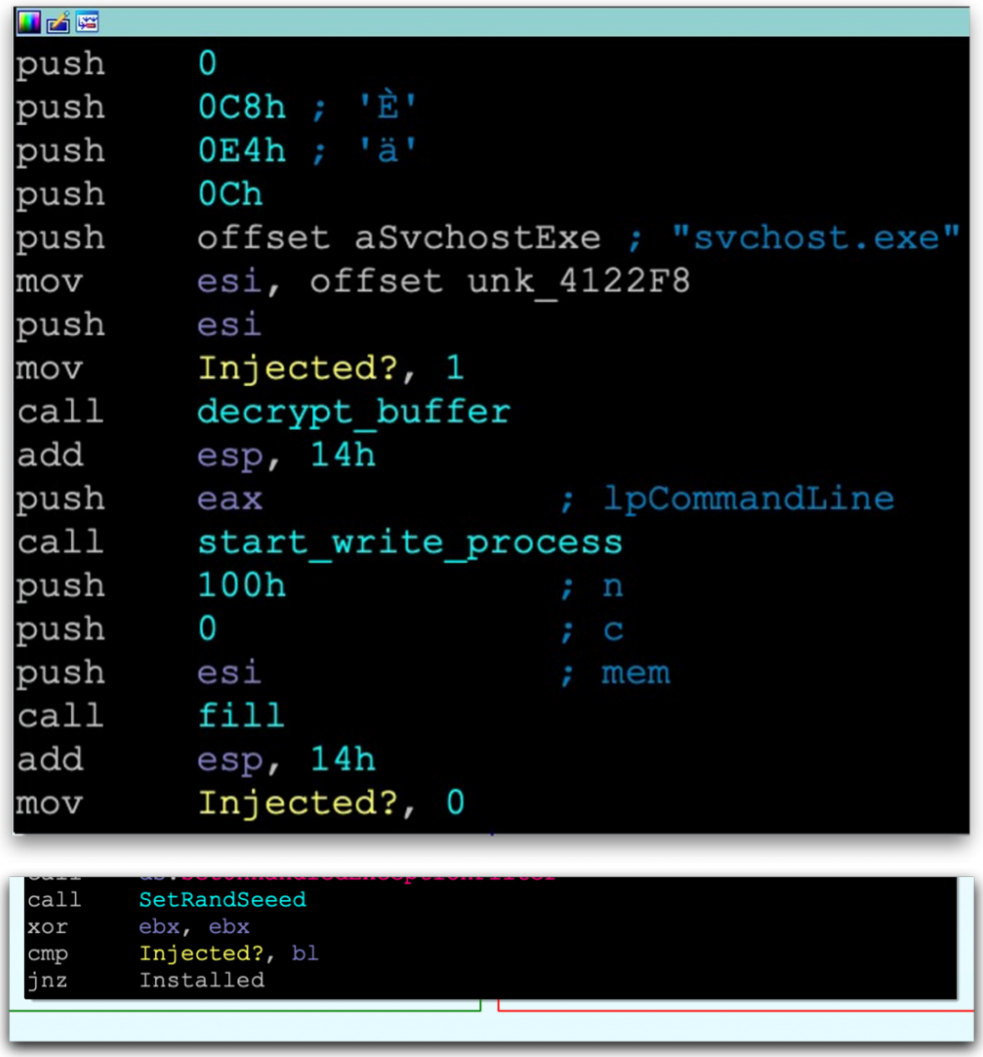


Installer/Installed path checks in Tofsee



Tofsee installer/installed paths deviate from the norm

When we ran these checks with Tofsee, we noticed a slight deviation from the typical routine. Instead of checking file or registry-based artifacts, Tofsee cross-checks against an in-memory variable injected during installation.



Installer checks Tofsee

This makes it impossible to imitate the binary file; however, it did make us ask the following question:

“How does Tofsee manage the duplicate runs of the same binary?”

The answer is that Tofsee handles this process using Inter-Process Communication (IPC) pipes [https://www.geeksforgeeks.org/ipc-technique-pipes/].

IPC communications initiate an exist

In the binary, we noticed a subroutine where Tofsee opens an IPC pipe and processes various data. The malware uses this IPC channel to communicate with another running instance to trigger an exist.

An algorithm is used to generate the pipe name, creating a name based on a predetermined value. This value is specific to the infected machine and is based on the hard drive’s volume serial number. The malware purposefully does this to make hardcoded indicator of

compromise (IOC) detection impossible on machines.

```
void Gen_Pipe_name(int len, char* dst, char salt) {
    char gen_str[len];
    int num2 = salt;
    if (num2 == 0) {
        num2 = rand();
    }
    char temp = num2;
    int i;
    for (i = 0; i < len - 1; i++) {
        int r = rand() % 26;
        gen_str[i] = 'a' + r;
    }
    gen_str[i] = num2;
    gen_str[len] = '\0';
    int r2 = (int)temp - 'a';
    for (i = 0; i < len - 1; i++) {
        r2 += gen_str[i];
        r2 %= 26;
        dst[i] = 'a' + r2;
    }
    dst[i] = num2;
    dst[len] = '\0';
}
```

Pipe name generation code

After generating the pipe name, the data received from the pipe is cross-checked as follows:

1. A 4-byte random integer is generated and sent across the pipe.
2. A 4-byte integer is read from the pipe.
3. The integrity of communication is checked using the following check (WRITE_DWORD >> 2) + WRITE_DWORD == READ_DWORD.
4. If the check is passed, another DWORD is written, which is generated from (READ_DWORD >> 2)
5. The calling process terminates.

A chink in Tofsee's armor

Here, where the data check creates the binary, there is the potential to leverage this process for the vaccine on the proviso that the binary isn't already running. If it is running, unfortunately, the opportunity to stop it is missed.

But let's focus on the scenario where the pipe doesn't exist; from here, an IPC pipe of the same name is created, and another set of data is received and cross-checked with specific parameters. These checks are a little more complex than the previous ones:

1. A 4-byte integer is read from the pipe.
2. A 4-byte integer is generated from right-shifting the integer by 2 and adding it back.
3. Two internally defined structures are read successively from the pipe. These structures are defined as follows:

At this point, the vaccine packet can be used.

```
struct __pipedata
{
    unsigned int IncPipeComTick; // Bool 1 : Recv another DWORD ,increate PIPEcomm
Tick varibale : 0 : Skip process exit
    unsigned int SuccessiveStructSize; // Size of packet to be recieved ( 0x0c for
vaccine packet )
};

struct __VaccinePacket
{
    const int TofseeMajVer = TOFSEE_VERSION; // malware version
    const int TofseeMinVer = TOFSEE_MIN;     // malware minor version
    bool Exit?; ( 0 : True , else False)
}
```

Tofsee Vaccine structures

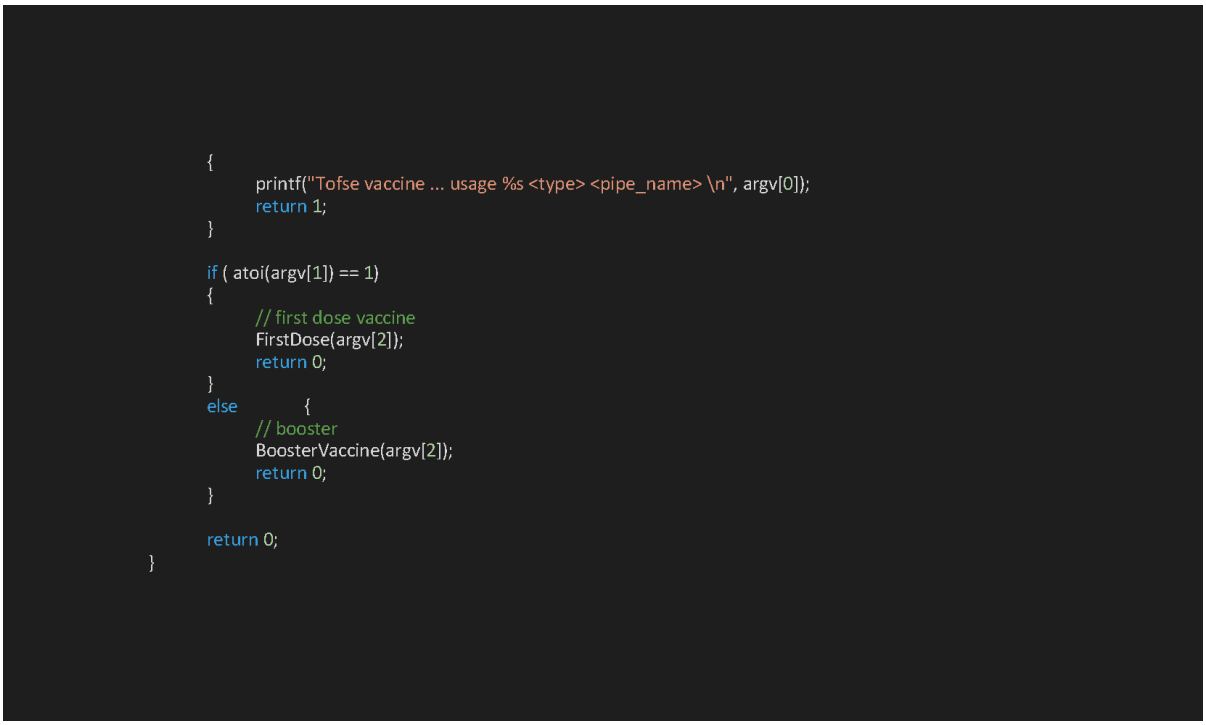
The entire code for the Tofsee vaccine

Below is the complete C code that you can use as a vaccine for new infections of Tofsee and existing ones, named as first dose vaccine and booster vaccine (ring any bells from the COVID days?!?).

```c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <stdbool.h>



int FirstDose(char *pipe_name)
{
	HANDLE pipe;
	int data, read_data;
	int confim_data;


	// Create the named pipe
	pipe = CreateNamedPipe(pipe_name, PIPE_ACCESS_DUPLEX, PIPE_TYPE_MESSAGE |
PIPE_READMODE_MESSAGE |  PIPE_WAIT,
                                                PIPE_UNLIMITED_INSTANCES, sizeof(int), sizeof(int), 0, NULL);

	if (pipe == INVALID_HANDLE_VALUE) {
		printf("Failed to create named pipe: %d\n", GetLastError());
		return 1;
	}

	// Wait for a client to connect
	if (ConnectNamedPipe(pipe, NULL) == 0) {
		printf("Failed to connect to named pipe: %d\n", GetLastError());
		CloseHandle(pipe);
		return 1;
	}

	// Read an integer from the pipe
	if (ReadFile(pipe, &data, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
		printf("Failed to read from named pipe: %d\n", GetLastError());
		CloseHandle(pipe);
		return 1;
	}

	// Modify the data to fulfil the tofsee checks
	data >>= 2;
	confim_data =   data;
	// Write the modified data back to the pipe
	if (WriteFile(pipe, &data, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
		printf("Failed to write to named pipe: %d\n", GetLastError());
		CloseHandle(pipe);
		return 1;
	}

	// Read the final confimation from tofsee
	if (ReadFile(pipe, &data, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
		printf("Failed to read from named pipe: %d\n", GetLastError());
		CloseHandle(pipe);
		return 1;
	}

	// cross check confirmation DWORD
	if ( data == (confim_data >> 2))                                {
		printf("Vacine successfuly applied .. malware terminated \n");
	}
	else{
		printf("Invalid sanity check ... exitting  \n");
		return 1;









	}
	// Close the pipe handle
```

```
        CloseHandle(pipe);

        return 0;
}

#define
TOFSEE_
VERSI
55;
#define
TOFSEE_
MIN 2


int BoosterVaccine(char *pipe_name)
{
        HANDLE pipe;
        int data, read_data;
        int RandWORD = 0xdeadbeef;


        struct _pipedata
        {
                unsigned int IncPipeComTick; // Bool 1 : Recv another DWORD ,increate PIPEcomm Tick varibale
: 0 :  Skip process exit
                unsigned int SuccessiveStructSize; // Size of packet to be recieved ( 0x0c for vaccine packet )
        }PipeData;

        struct _VaccinePacket
        {
                int TofseeMajVer; // malware version
                int TofseeMinVer;                   // malware minor version
                int KillMalware;
        }VaccinePacket;



        // Connect to the named pipe
        pipe = CreateFile(pipe_name, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);

        if (pipe == INVALID_HANDLE_VALUE) {
                printf("Failed to connect to named pipe: %d\n", GetLastError());
                return 1;
        }

        if (WriteFile(pipe, &RandWORD, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
                printf("Failed to write to named pipe: %d\n", GetLastError());
                CloseHandle(pipe);
                return 1;
        }

        if (ReadFile(pipe, &data, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
                printf("Failed to read from named pipe: %d\n", GetLastError());
                CloseHandle(pipe);
                return 1;
        }
        RandWORD = ((RandWORD >> 2) + RandWORD);
        // first data check
        if (RandWORD == data)









        {
                if (WriteFile(pipe, &RandWORD, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
                printf("Failed to write to named pipe: %d\n", GetLastError());
                CloseHandle(pipe);
                return 1;
                }
                RandWORD = ( (RandWORD >> 2) + RandWORD ) ;

                if (ReadFile(pipe, &data, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
                        printf("Failed to read from named pipe: %d\n", GetLastError());
```

```
                        CloseHandle(pipe);
                        return 1;
                }

                if ( RandWORD == data)
                {
                        // set up vaccine structures
                        VaccinePacket.KillMalware = 0;

                        VaccinePacket.TofseeMajVer = TOFSEE_VERSION;
                        VaccinePacket.TofseeMinVer = TOFSEE_MIN;

                        PipeData.IncPipeComTick = 1;
                        PipeData.SuccessiveStructSize = sizeof(VaccinePacket);


                          if (WriteFile(pipe, &PipeData, sizeof(PipeData), (LPDWORD) &read_data, NULL) == 0) {
                                printf("Failed to write to named pipe: %d\n", GetLastError());
                                CloseHandle(pipe);
                                return 1;
                                }

                        if (WriteFile(pipe, &VaccinePacket, sizeof(VaccinePacket), (LPDWORD) &read_data, NULL)
== 0) {
                                printf("Failed to write to named pipe: %d\n", GetLastError());
                                CloseHandle(pipe);
                                return 1;
                                }

                          if (ReadFile(pipe, &data, sizeof(int), (LPDWORD) &read_data, NULL) == 0) {
                                printf("Failed to read from named pipe: %d\n", GetLastError());
                                CloseHandle(pipe);
                                return 1;
                                }
                        if ( data == RandWORD)
                        {
                                printf("booster successfuly applied .. malware terminated \n");

                                CloseHandle(pipe);

                                  return 0;
                        }

                }
        }
        // Close the pipe handle
        CloseHandle(pipe);
        return 1;
}
int main(int argc, char **argv)
{
        if ( argc != 3)
```



```
{
        printf("Tofse vaccine ... usage %s <type> <pipe_name> \n", argv[0]);
        return 1;
}

if ( atoi(argv[1]) == 1)
{
        // first dose vaccine
        FirstDose(argv[2]);
        return 0;
}
else            {
        // booster
        BoosterVaccine(argv[2]);
        return 0;
}

return 0;
}
```

Happy vaccination coding!

In our next blog post, we'll look at a second vaccine you can use to protect against Tofsee. This one concentrates on injecting code into the memory configuration store.