Imminent Monitor 4 RAT Analysis – A Glance

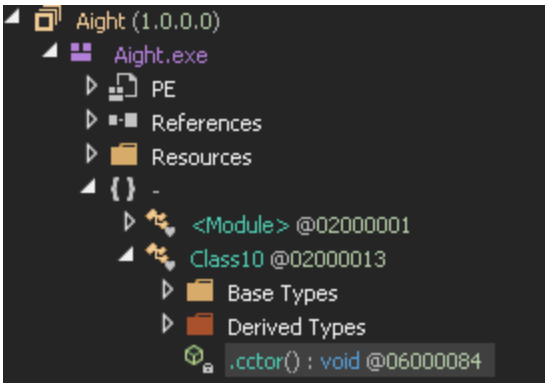
itsjack.cc/blog/2016/01/imminent-monitor-4-rat-analysis-a-glance/

January 23, 2016

January 23, 2016

As I stated in this video I'm going to be focusing much more on RAT analysis. My reasons are given in the second portion of the video for anyone who is interested. I decided to look at a fairly popular RAT called 'Imminent Monitor'. I downloaded the cracked version which many should note as this may differ from the original in some ways, I've tried to only include things which I'm sure are wrong with the original as well. There is something positive about analysing cracked malware, as the majority of criminals probably don't want to pay for RAT's.

Darkcomet RAT is still actively used today even though it is no longer developed and has been proven to have exploits in this. With this in mind, this version of cracked software will most likely distribute around the internet and be used more than the paid version due to it simply being free. I thought I was going to have a fairly long wait to find how to extract the config out of the RAT but it was really as easy as 1-2-3. Literally looking at the first classes constructor.



I was able to see port, ip/hostname, startup key, startup name, startup location, mutex and version (and other less important things).

```
Class10.string_9 = "80728c24-febf-4677-89df-9f8aff30b668"
arg_221_0 = -1220711842;
continue;
e 21:
goto IL_291;
e 22:
Class10.string_1 = "Default";
arg_221_0 = -1220711865;
continue;
e 23:
Class10.string_6 = "MyFile.exe";
Class10.bool_1 = true;
Class10.bool_2 = true;
arg_221_0 = -1220711861;
continue;
e 24:
Class10.string_0 = "127.0.0.1";
```

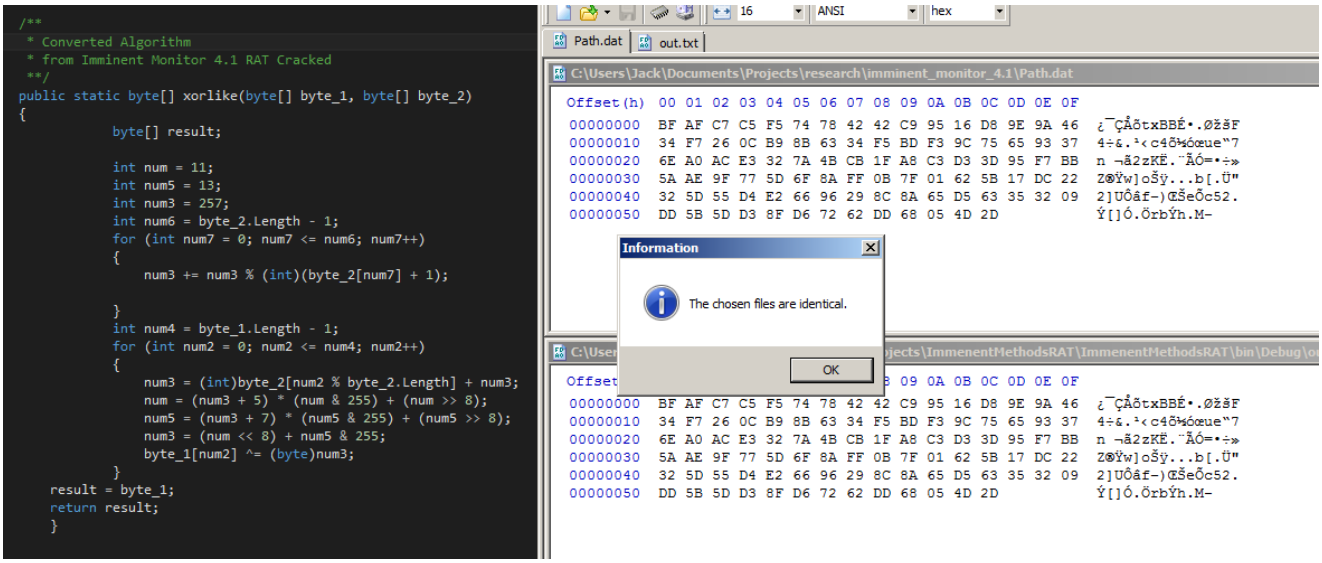
One of the first threat indicators I can provide to malware guys is that Imminent RAT creates a folder which is always the same, it is quite unique. It creates a directory called “Imminent” in the application data of the current user. This does not hold the executable but instead holds information to be sent over to the client. Like logs.

```
GClass20.smethod_5(Environment.SpecialFolder.ApplicationData), "\\Imminent\\Logs\\"))
```

The binary uses cases to make it harder for researchers to make out whats going. Imminent uses a XOR like function but to what I see isn’t *exactly* XOR. I’m thinking of creating a deobfuscator for this sort of code later on, but for now I solved it manually, which wasn’t particularly challenging. The more I look at it the more it looks like XOR, but I’m terrible at checking these things so I’m just going to call it XOR-like.

```
int num3;
int num5 = (num3 + 7) * (num5 & 255) + (num5 >> 8);
arg_196_0 = 2021547055;
continue;
}
case 10:
{
int num2;
int num3 = (int)byte_2[num2 % byte_2.Length] + num3;
num = (num3 + 5) * (num & 255) + (num >> 8);
arg_196_0 = 2021547051;
continue;
```

This XOR like functions is used to encrypt logs and other parts included in the “Imminent” application data. I successfully tested my algorithm against the original and got the same result, this was also verified by breakpointing the function to where it was encrypting to make sure the plain text was the same.



With the same algorithm we can decrypt logs that Imminent creates. It does it by a format of month/day/year, showing the author could be American origin. An interesting thing about the use of what looks to be XOR is the key provided for the operation, the developer decided to use “sampleKey”. Or did he? A person who doesn’t copy and paste an algo usually chooses a different key than sampleKey, infact someone who **does** copy and paste an algorithm usually changes the key.

> byte[] path = System.Text.Encoding.UTF8.GetBytes(“C:\\Documents and Settings\\L!NK\\Local Settings\\Application Data\\Startup Folder Name\\MyFile.exe”);
> byte[] pass = System.Text.Encoding.UTF8.GetBytes(“sampleKey”);
> byte[] ax = xorlike(path, pass);
> System.IO.File.WriteAllBytes(“out.txt”, ax);

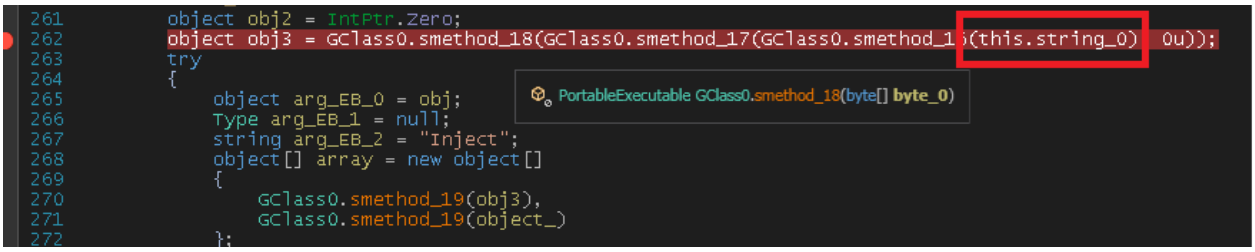
```
        public static byte[] xorlike(byte[] byte_1, byte[] byte_2)
{
byte[] result;

int num = 11;
int num5 = 13;
int num3 = 257;
int num6 = byte_2.Length – 1;
for (int num7 = 0; num7 <= num6; num7++)
{
num3 += num3 % (int)(byte_2[num7] + 1);

}
int num4 = byte_1.Length – 1;
for (int num2 = 0; num2 <= num4; num2++)
{
num3 = (int)byte_2[num2 % byte_2.Length] + num3;
num = (num3 + 5) * (num & 255) + (num >> 8);
num5 = (num3 + 7) * (num5 & 255) + (num5 >> 8);
num3 = (num << 8) + num5 & 255;
byte_1[num2] ^= (byte)num3;
}
result = byte_1;
return result;
}
```

I've decided to show an example of it's operation. You can use this for any defense against Imminent if you choose to make one. The code is obviously deciding to encrypt the startup path and saved in the Imminent folder as "Path.dat", a rough location of a user is also saved "Geo.dat", which uses a site called iptrackeronline.com to longitude, lattitude, city and finally country.

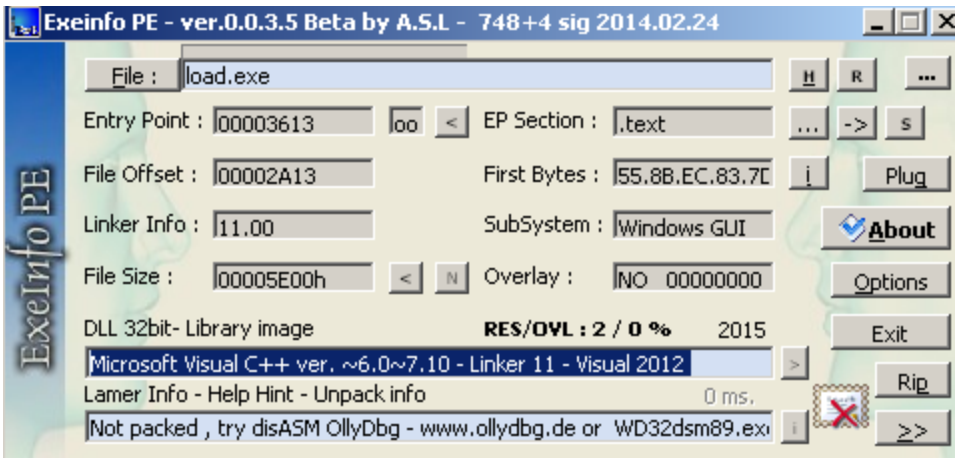
I discovered a file which deals with processes, I'm not sure at the moment, if its the file updater, download and execute or persistence. I'm more leaning towards it being a process watchdog but have many things to explore. I discovered it here:



Turns out there is a base64 string which is then decompressed using the LOADLZ library it has with it. There's a reason why this thing can be 400kb.

```
ase 1:
    this.string_0 = "UUNMWgMAAAD5PQAAAF4AAAEAAAAAAAAAAAAAAAAAAAAAAgoiATVqQAAMAAAAEEP//AAC4XwAAAAB
    /dAgFCkBBABBZUhkKoYp/CjmJSjI5ffwPhcClAjPAhwPpt0oCBMxBkOmzHIN9DAEPhaYoIkiL98oOAQP2RwMz9kY5ihR
    AgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    this.random_0 = GClass0.smethod_0();
    arg_2F_0 = -1636407252;
    continue;
```

The file was written in C++ which is interesting as most .NET malware doesn't drop anything like C++, turns out to be a dll which I assume is injected for persistence, although as I say I don't want to rule out anything yet.



I couldn't debug in Immunity properly because I use an XP machine with some of the C++ redistributables not installed. This file required GetModuleHandleEx which was provided in 2011 from my research, but I might be wrong.

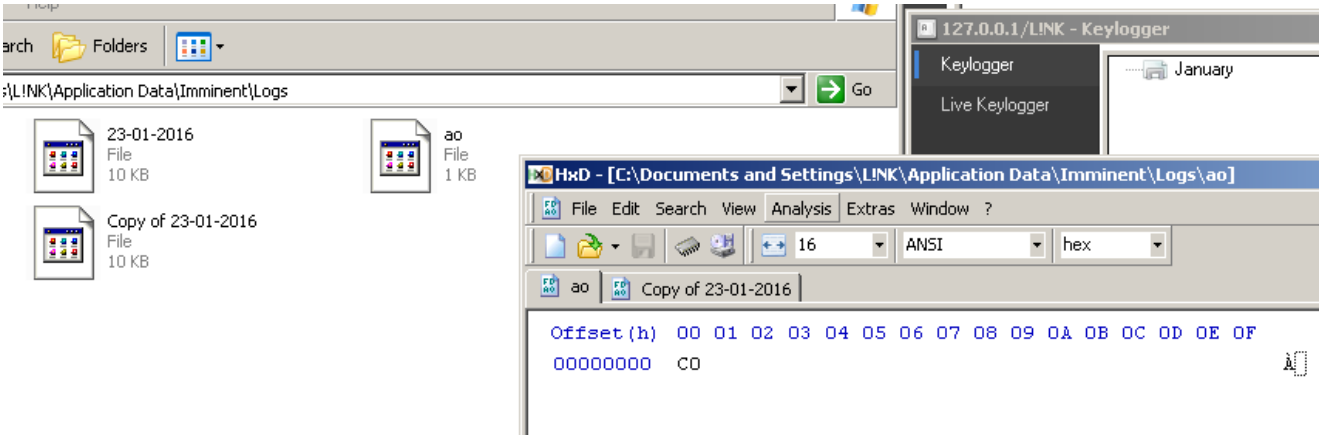
C	N/A	640 (0x0280)	GetModuleHandleExW	Not
C	N/A	756 (0x02F4)	GetSystemTimeAsFileTime	Not
C	N/A	785 (0x0311)	GetTickCount64	Not
C	N/A	899 (0x0383)	IsDebuggerPresent	Not
C	N/A	904 (0x0388)	IsProcessorFeaturePresent	Not
C	N/A	931 (0x03A3)	K32GetModuleFileNameExW	Not
C	N/A	1032 (0x0408)	OpenProcess	Not
C	N/A	1084 (0x043C)	QueryPerformanceCounter	Not
C	N/A	1375 (0x055F)	Sleep	Not
?	MSVCP110.DLL	Error opening file. The system cannot find the file specified (2).		
?	MSVCR110.DLL	Error opening file. The system cannot find the file specified (2).		
	KERNEL32.DLL	06/21/2011 9:52a	03/21/2009 5:59a	991,744 A
	LOAD.EXE	01/20/2016 6:20a	07/05/2015 2:15p	24,064 A
	NTDLL.DLL	12/09/2010 7:15a	12/09/2010 7:15a	718,336 A

I'm not going to show a lot of IDA, but it checks for a debugger, creates a thread, opens processes and creates them too. It is also uniquely identifiable because of some of the strings that are given. That is indeed a md5 hash which is "killswitch" in plain text.

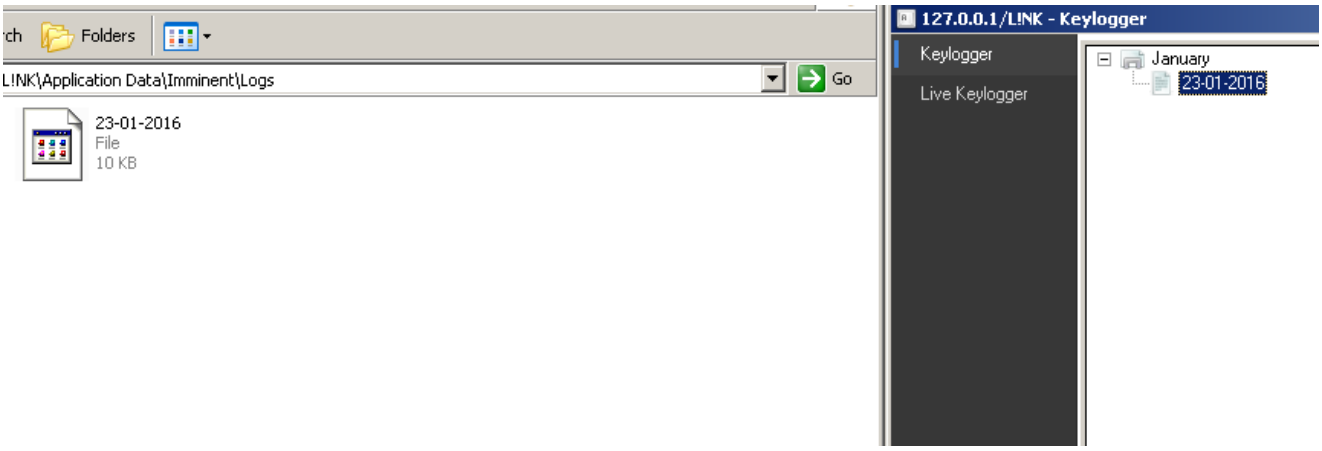
```
.rdata:10004232                 align 4
.rdata:10004234 aIostream       db 'iostream',0         ; DATA XREF: su
.rdata:1000423D                 align 10h
.rdata:10004240 aIostreamStream db 'iostream stream error',0 ; DATA XRE
.rdata:10004256                 align 4
.rdata:10004258 aSystem         db 'system',0           ; DATA XREF: su
.rdata:1000425F                 align 10h
.rdata:10004260 ; char VarName[]
.rdata:10004260 VarName         db 'APPDATA',0          ; DATA XREF: su
.rdata:10004260                                         ; sub_100015A0+
.rdata:10004268 aImminentPid_da db '\Imminent\PID.dat',0
.rdata:10004268                                         ; DATA XREF: su
.rdata:10004268                                         ; sub_10002F50+
.rdata:1000427A                 align 4
.rdata:1000427C aF824601a639c6c db 'f824601a639c6ca96752197f8868cf9c',0
.rdata:1000427C                                         ; DATA XREF: su
.rdata:1000429D                 align 10h
.rdata:100042A0 ; char Format[]
.rdata:100042A0 Format          db 'CreateProcess failed (%d).',0Ah,0
.rdata:100042A0                                         ; DATA XREF: sub_100015A0+232↑o
.rdata:100042BC ; char aProcessRestart[]
.rdata:100042BC aProcessRestart db 'Process Restarted (%d).',0Ah,0
.rdata:100042BC                                         ; DATA XREF: sub_100015A0+2CA↑o
```

You can see that correctly, the file is also FUD and still is as I write this article 3 days later.

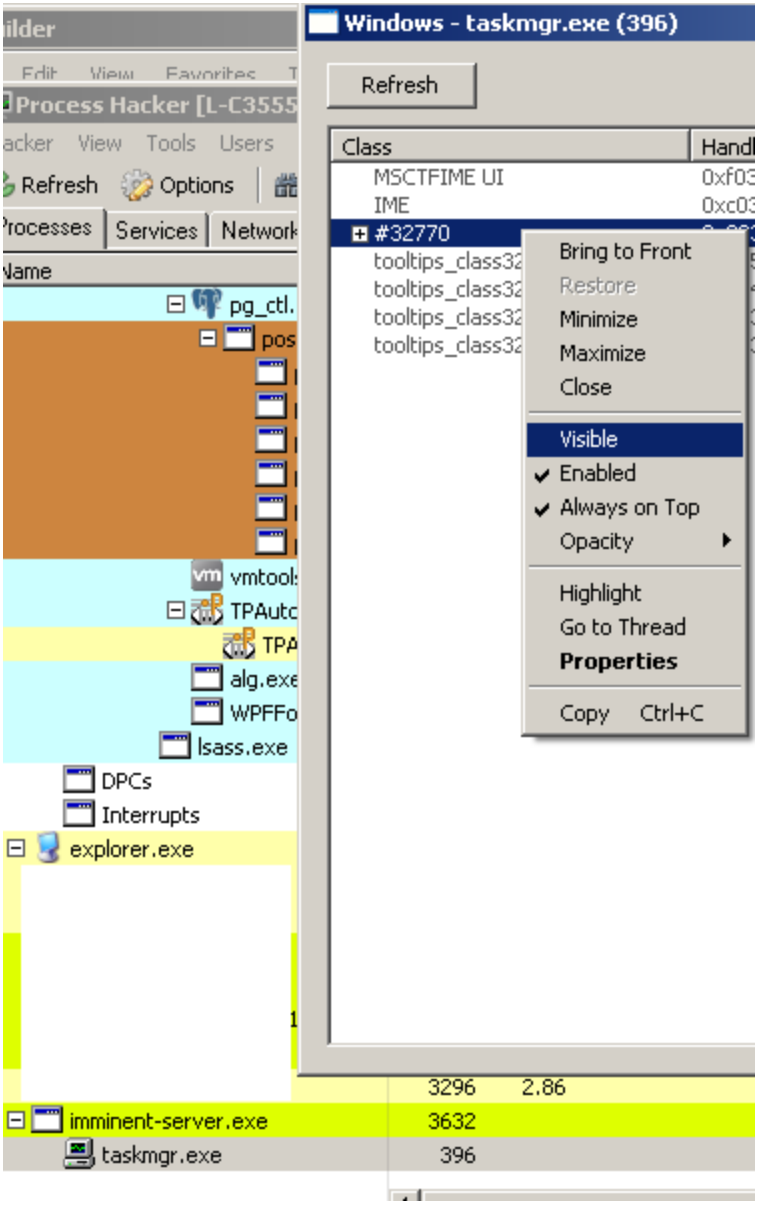
I was starting to test out how I could break Imminent Monitor as well and found that the RAT does not like showing files when a file is out of its format, I named a file in the keylogger file “ao” and entered a small amount of data. It proceeded to not show any of the keylogging files that were available to it. This is it in a broken state.



This is it in the normal state.



The task manager disabler functionality that Imminent provides is also quite bad. It simply executes the task manager and makes it invisible. Not exactly to rootkit standards, but I guess does the trick to the average joe?



In the cracked version the process protection is broken or non-existant, I am able to kill the process with ease. I am to do more research on Imminent Monitor, it's been fun to look at so far and I haven't looked through all of it because of time constraints in my life. What I have provided is some threat indicators for AV's so that this RAT can be removed from peoples computers with ease, even if crypted. The static path declaration provides an opportunity to remove this RAT from most machines.

I've been looking more at how Imminent can be broken remotely, I've had a few sucesses with random things, but must look at them in more depth before I decide to release them publicly. A reminder for anyone reading this, if you have an original build that isn't from a

cracked client, I would like to have it so please contact me.

Thanks for looking.