Malware Analysis Report
“Sample2.exe”

Contro Filippo VR437055
Martini Michele VR437056

I. Executive summary

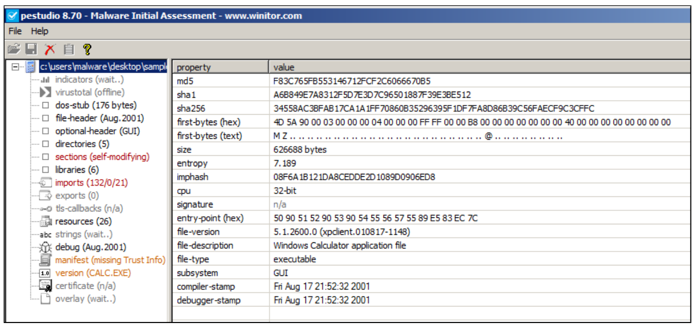
The malware “Sample2.exe” shows itself as a calculator app, with the appropriate icon and the expected behavior. Once it is opened by the user there is a simple GUI to perform basic arithmetic operations. Under the hood, however, the malware deactivates the Windows security center, makes many POST request to various internet sites and infects many other files in different locations, from the local folder to the system application’s executable.

The whole analysis process has been done on the provided virtual machine, which runs Windows 7 as operating system and contains a bunch of tools to make both static and dynamic analysis. The reverse engineering instead has been done on local machines.

II. Static analysis

We began the static analysis using *PEStudio*[1], a tool which performs malware initial assessment, studying headers, sections and many other features of the executable, which is never executed. This first analysis gave us many indicators of probable malicious behavior. First of all, in the main view we noticed an *entropy* value greater than 7, therefore it’s likely that either the file has been packed or some sections has been obfuscated. Entropy is a measure of the order of the application, and it is calculated with statistical operations on the bytes; As explained in this article[2]

“When a section of code of a file is compressed, the total entropy will increase while counting, i.e. , it will get closer to the standard sum of all bits - this will mean higher efficiency of information storage, i.e., possible compression of section of code.



property	value
md5	F83C765FB553146712FCF2C6066670B5
sha1	A6B849E7A8312F5D7E3D7C96501887F39E3BE512
sha256	34558AC3BFAB17CA1A1FF70860B35296395F1DF7FA8D86B39C56FAECF9C3CFFC
first-bytes (hex)	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes (text)	M Z .. .. @ .. ..
size	626688 bytes
entropy	7.189
imphash	08F6A1B121DA8CEDDE2D1089D0906ED8
cpu	32-bit
signature	n/a
entry-point (hex)	50 90 51 52 90 53 90 54 55 56 57 55 89 E5 83 EC 7C
file-version	5.1.2600.0 (xpclient.010817-1148)
file-description	Windows Calculator application file
file-type	executable
subsystem	GUI
compiler-stamp	Fri Aug 17 21:52:32 2001
debugger-stamp	Fri Aug 17 21:52:32 2001

The maximum value for entropy in PEStudio is 8, so this high value is a clear evidence of packing and/or obfuscation.

[1]https://www.winitor.com/

[2]http://n10info.blogspot.com/2014/06/entropy-and-distinctive-signs-of-packed.html

The executable lacks of any certificate, while the information dealing with the version disguise it as a “Windows calculator application file”, with the legal copyright by Microsoft, but there is not a date. The header instead has a compilation time-stamp dating August 2001.

A. Sections

The malware’s code is made up of four sections:

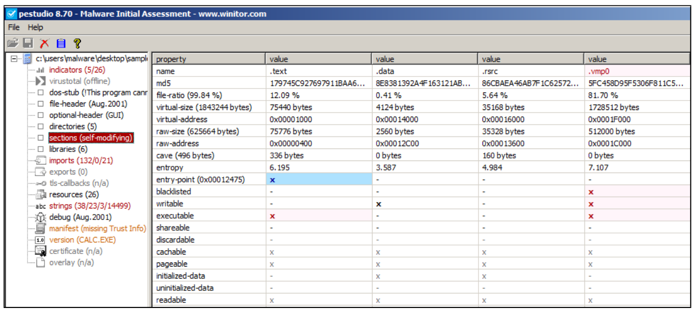
- **.text**: the code in clear;
- **.data**: contains global variables;
- **.rsrc**: contains various resources, like images or icons;
- **.vmp0**: main section of the code, obfuscated to make reverse-engineering more difficult.

The last section is the largest with a file-ratio of 81.70%; it has an entropy greater than 7 and PEStudio marks it as blacklisted. Moreover it is both writable and executable, which is often a malicious evidence, as it indicates self modifying code.

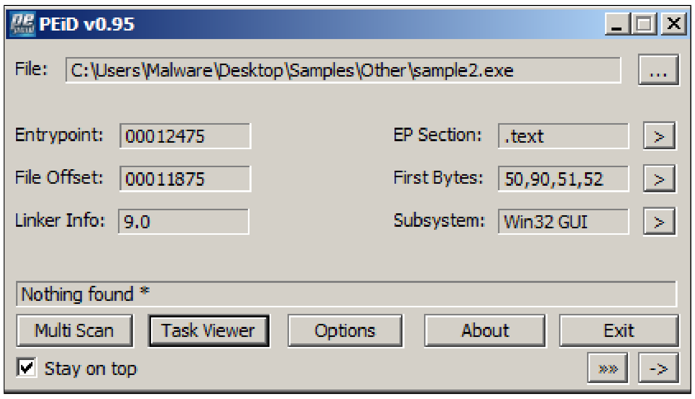
After a little search on the internet we discovered that the name “vmp” refers to an obfuscation program called *VMProtect*[3]. This program cypher the code exectuting it in a virtual CPU that is different from $x86$. This code obfuscator makes the reverse engineering extremely difficult.

Nevertheless, the presence of these sections allows us to suppose that the code has not been packed and its high entropy is due only to the “vmp0” section, which is obfuscated. In order to validate this theory, we used *PEiD*, a software whose goal is to detect any possible packing. This tool provides three levels of analysis varying in depth. Every level, however, gave us the same result: “nothing found”, meaning that PEiD could not detect any packing mechanism.

[3]https://vmpsoft.com/



property	value	value	value	value
name	.text	.data	.rsrc	.vmp0
md5	179745C927697911BAA6...	8E8381392A4F163121AB...	86CBAEA46AB7F1C62572...	5FC458D95F5306F811C5...
file-ratio (99.84 %)	12.09 %	0.41 %	5.64 %	81.70 %
virtual-size (1843244 bytes)	75440 bytes	4124 bytes	35168 bytes	1728512 bytes
virtual-address	0x00001000	0x00014000	0x00016000	0x0001F000
raw-size (625664 bytes)	75776 bytes	2560 bytes	35328 bytes	512000 bytes
raw-address	0x00000400	0x00012C00	0x00013600	0x0001C000
cave (496 bytes)	336 bytes	0 bytes	160 bytes	0 bytes
entropy	6.195	3.587	4.984	7.107
entry-point (0x00012475)	x	-	-	-
blacklisted	-	-	-	x
writable	-	x	-	x
executable	x	-	-	x
shareable	-	-	-	-
discardable	-	-	-	-
cachable	x	x	x	x
pageable	x	x	x	x
initialized-data	-	x	x	-
uninitialized-data	-	-	-	-
readable	x	x	x	x


PEiD v0.95

File: C:\Users\Malware\Desktop\Samples\Other\sample2.exe

Entrypoint:	00012475	EP Section:	.text
File Offset:	00011875	First Bytes:	50,90,51,52
Linker Info:	9.0	Subsystem:	Win32 GUI

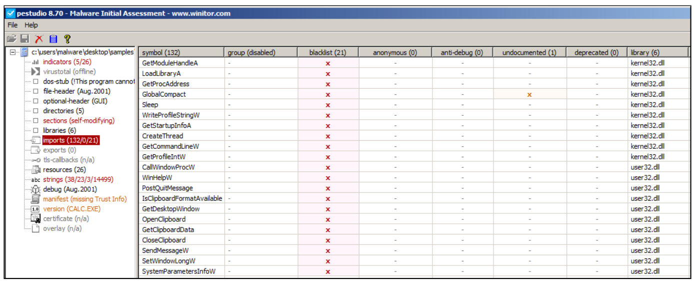
Nothing found *

B. Imports

The executable imports 6 libraries in total, and in particular PEStudio marks as blacklisted 21 functions belonging to *Kernel32.dll* and *user32.dll*. The most interesting are:

- **getModuleHandleA**, to call external files, such as *dll* or *exe*;
- **loadLibraryA**, similar to that one above;
- **getProcAddress**, to get the address of a procedure inside a library;
- **getStartupInfoA**, to get information about startup;
- **getCommandLineW**, to let the program execute code on the command line;

Moreover there are also imports of functions dealing with *Threads*, *Clipboard* and *Windows*. This suggests a suspicious behavior, due to the possibility of the program to access to all the libraries of the installed operating system. In the dynamic analysis we will see what actions are actually performed. The figure below contains all the blacklisted imports detected by PEStudio.


pestudio 8.70 - Malware Initial Assessment - www.winitor.com

symbol (132)	group (disabled)	blacklist (21)	anonymous (0)	anti-debug (0)	undocumented (1)	deprecated (0)	library (6)
GetModuleHandleA	-	x	-	-	-	-	kernel32.dll
LoadLibraryA	-	x	-	-	-	-	kernel32.dll
GetProcAddress	-	x	-	-	-	-	kernel32.dll
GlobalCompact	-	x	-	-	x	-	kernel32.dll
Sleep	-	x	-	-	-	-	kernel32.dll
WriteProfileStringW	-	x	-	-	-	-	kernel32.dll
GetStartupInfoA	-	x	-	-	-	-	kernel32.dll
CreateThread	-	x	-	-	-	-	kernel32.dll
GetCommandLineW	-	x	-	-	-	-	kernel32.dll
GetProfileIntW	-	x	-	-	-	-	kernel32.dll
CallWindowProcW	-	x	-	-	-	-	user32.dll
WinHelpW	-	x	-	-	-	-	user32.dll
PostQuitMessage	-	x	-	-	-	-	user32.dll
IsClipboardFormatAvailable	-	x	-	-	-	-	user32.dll
GetDesktopWindow	-	x	-	-	-	-	user32.dll
OpenClipboard	-	x	-	-	-	-	user32.dll
GetClipboardData	-	x	-	-	-	-	user32.dll
CloseClipboard	-	x	-	-	-	-	user32.dll
SendMessageW	-	x	-	-	-	-	user32.dll
SetWindowLongW	-	x	-	-	-	-	user32.dll
SystemParametersInfoW	-	x	-	-	-	-	user32.dll

C. Strings

Regarding strings we notice that those in .text, .data and .rsrc sections are in clear; furthermore many of them are function calls or values used to modify system registers. Followin them there are thousands of obfuscated strings, seemingly meaningless, which belong to the ".vpm0" section. This is another clue of code obfuscation.

III. Dynamic Analysis

The dynamic analysis consists of the execution of the malware in a controlled and isolated system to observe the behavior and the malicious actions performed.

To accomplish this task we used *VirtualBox*, a virtualization program that let us run a Windows 7 machine. VirtualBox handles the snapshots too, so that after every test the machine was restored to its initial state. The machine was also isolated from internet in order to avoid leaks.

The machine was equipped with various tools to monitor the execution of the malware. The main ones were:

- **regshot**, to detect files and registers alterations between a time lapse;
- **procmon**, to log system functions called by the malware;
- **fakenet**, to track internet traffic in a simulated network.

These programs were executed with administrator permission in order to detect access to restricted locations.

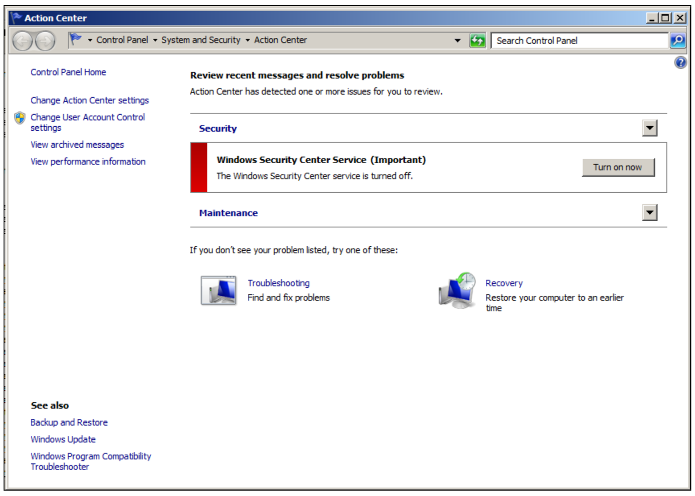
The tools can cooperate but they need to be executed is a certain schedule, in order to avoid conflicts among them. The analysis proceeded following these phases:

1) launch Fakenet;
2) launch and setup Procmon;
3) launch Regshot, setup path and run of its first shot;
4) start Procmon analysis and launch of the malware;
5) interaction with calculator by the GUI;
6) stop Procmon tracking
7) second Regshot shot;
8) stop Fakenet;

The first tests did not last too long: the GUI remained opened for just a few seconds. These tests did not give us evidence of malicious behavior, probably beacuse of the poor performance of the host machine.

Then we ran other test leaving the malware run for a couple of minutes, leaving all the time needed to permorm bad actions. Then we collect the logs and analyzed them one by one.

The first evidence of malicious behavior has been observed without any tools. After 20-30 seconds from the program execution the operating system prompted us a message about the *Windows security center*; it was disabled and, from the control panel, it could not be re-activated. This is a seriuos threat to the security of the machine, leaving it opened to many other infections.


Action Center

Control Panel ▸ System and Security ▸ Action Center
Search Control Panel
Control Panel Home
Change Action Center settings
Change User Account Control settings
View archived messages
View performance information

Review recent messages and resolve problems

Action Center has detected one or more issues for you to review.

Security

Windows Security Center Service (Important)

The Windows Security Center service is turned off.

Maintenance

If you don't see your problem listed, try one of these:

Troubleshooting
Find and fix problems
Recovery
Restore your computer to an earlier time
See also
Backup and Restore
Windows Update
Windows Program Compatibility Troubleshooter

A. Fakenet

FakeNet is a dynamic network analysis tool for malware analysts and penetration testers. It simulates a network logging the traffic generated by the machine. Every request is a file containing:

- Request type
- Destination URL
- Protocol
- User agent

The user agent field contains various informations about the local machine, such as architecture, operating system, product and so on. This string is

```
User-Agent: Mozilla/4.0
(compatible; MSIE 28;...
```

There are other nubers dealing with the versions that have not been written for simplicity.

The registered requests are over 900, whereas the destinations are 300, mainly russian sites. Unluckily the body is empty, and we cannot know the expected result due to the presence of Fakenet, which builds a fake http page as response to every request.

Fakenet generated also a *.pcap* file, containing the captured packets. Examinating it with *Wireshark* we noticed all the TCP streams related to the POST requests, and other packets irrelevant with the malware, such as DHCP, ARP and DNS. One frequent message was a request to *www.msftncs.com*, asking for a txt file called *ncsi*. This sort of message is generated by Windows as a heat beat, keeping the machine constantly connected to the internet.

This whole internet traffic should not be present, due to the absence of any socket library imported by the executable. Thus, in order to send packets over the network the malware needs to load external libraries dynamically, leaving a trace on the system that we capture using *Procmon*.

B. Procmon

Process Monitor is an advanced monitoring tool for Windows that shows real-time file system, registry and process/thread activity. Procmon offers also a filtering function, and we set it to keep only the actions made by the malware.

We can group these actions in 3 main sets: dll, registry and file.

Dll stands for dynamic link library. These are particular files, shared among all the programs of the Windows operating system, that expose useful functions to programs which can call them to perform frequent operations. Every time that the malware loads a dll the system keeps a track of the action, thus using Procmon we were able to collect them all, identifying 46 different dlls used with the "Load Image" function. This primitive allows the program to load also other executables but, in our case, the only one loaded by the malware was the malware itself.

Overall we counted 46 dll files loaded, among wich the most interesting are the cypher family, like *cryptbase* and *crypt32*, to handle cryptography and certificates.

In order to perform network operations the malware needs a library to handle internet connections. This is made possible by the load of *ws2_32.dll*, which stands for web socket. This library provides the APIs (Application programming interface) needed to do POST request to all of the different URLs listed before.

Among the actions concerning with the registers Procmon logged many Open-Read-Close operations on many different ones, but only a few were written directly by the malware. These ones are about the "Language list" and "Internet settings". This last one manage the permission over the internet among the 4 different zones (Intranet, Trusted, Internet, Restricted), setting the value to 0 which means "Accept all". This is dangerous as exposes the machine to possible attacks over the net.

Procmon registered also accesses to thousands of file, and we noticed that many one of them were infected. The infection procedure followed these steps:

- Read the *.exe* victim file.
- Write of the content plus the infected part in a *.vir* file with the same name.
- Copy of the content of the *.vir* file to the *.exe* one changing the EOF location.
- Set of fake information on the executable such as creation an last access time.
- Delete the *.vir* file.

After the infection the victim had another section called *.vmp0*, having a similar size to the original one, which is both executable and writable. The infected files are hundreds and they are stored in many different locations. Many of them are common applications such as Windows Media Player, Internet Explorer, Windows Defender, Windows Mail, Windows Photo Viewer and other files in the System32 directory.

C. RegShot

Regshot is an open-source (LGPL) registry compare utility that allows you to quickly take a snapshot of your registry and then compare it with a second one - done after doing system changes or installing a new software product.

In our use-case we recorded the first shot just before the execution of the malware. Then we ran it for some minutes and, after the program exit, we took the second shot, saving the result of the comparison in a *.txt* file.

This file contains variuos information about registries files and folders. The downside of Regshot is that it takes care of all the system changes, but it cannot associate each change to the process that caused it, so there is no way to separate the malware generated events from the system ones. The only clue that we can use is a comparison to the logfile generated by Procmon.

Analyzing the output there are 6 new key added to the registry, 4 related to error reporting, one to instrumentation and one to the Security Center. This last one is the most interesting because it is the trace left by the deactivation of the security center that we noticed during the malware execution. In fact among the values added to the registries there is "Enable Notifications" set to zero. Then we found all the changes reported by Procmon about the internet zones, where every value is set to 0 (Allow anything).

Another malicious evidence that we found is that the value "Start" of the *wscsvc* service set to 4. This service is the security center, and the value 4 stands for disabled.

The weired thing is that Procmon did not log those events, so they are not made by the malware itself, but by another process. In order to validate this hypotesis we ran Procmon and exectued the sample another time, keeping even the action perfomed by other processes. We discovered that those registry modifications had been made by "services.exe" and "dllhost.exe". They are not malicious process, but they are likely to be corrupted by the malware to deactivate the security center.

The file attributes that are modified are those related to the infected files and thus they have a different creation time.

IV. Reverse Engineering

In order to analyze the source code of the malware we used a tool called IDA, which is a feature rich, cross-platform, multi-processor disassembler and debugger developed by Hex-Rays. We also used OllyDBG, a 32-bit assembler-level debugger, which turned out fundamental to decode hidden instructions.

The whole process of reverse engineering is very difficult due to the obfuscation of the code and to the presence of *VMProtect* which encrypts the core of the program. Analyze .vmp0 section is not possible until the code in .text is executed, therefore we had to use OllyDBG on the malware's first section and obtain a decrypted version of the code. After that, we generated a dump of the memory so we could analyze the whole code using IDA.

A. Code rebuilding

The process of reverse engineering began with the disassembly phase, made possible by IDA. The tool shows the entry point and builds a cfg for each function. We analyzed the main one, obtaining a pseudo code of the function itself (See **??**). This main procedure manages the de-obfuscation of the real malware. The *.vmp0* section is decrypted through a cycle that perform an arithmetic xor of the code with a certain key. The cycle is repetead 7 times, but during the last one the key is incremented by one. The key is $0x58$. After this cycle there are two more that simply move portions of code. Next we observed a strange behavior: there are not return statements, but after the main procedure there are many functions. Following the code execution with a debugger we discovered that the program executes the code inside those functions like simple statements of the main procedure. Hence, the disassembler is unable to build a consistent cfg. At this point we decided to stop the code rebuilding and we began a debugging phase.

B. Debugging

In order to perform this analysis we began executing the malware with OllyDBG one instruction at the time, keeping procmon active alongside in order to detect malicious actions. Every time we found a function call we stepped over, tracking the results in order to isolate the interesting ones. Then we restarted the machine and repeat the whole process entering the target function, looking for nested calls to analyze.

This malware uses a lot of external functions, imported from various dynamic link libraries. This functions are loaded with two primitives in Windows: *loadLibrary* and *getProcAddress*. The malware then stores the pointers to those functions as global values in the *.data* section, thus the disassembler cannot statically detect the imported function. The only way to bypass it is to execute the malware with a debugger who can get the function name whenever it is called.

Our aims were the infection procedure, which lets the malware copy its code inside other files, and the method used to stop the Windows Security Center.

We discovered that the first thread was responsible for the GUI, whereas the second one did the real infection. In order to bypass loops we often put breakpoints after them. Eventually we achieved a procedure to debug the infection function:

1) breakpoint in $0101273A$; then after the *RET* the malware enters the obfuscated section.
2) breakpoint in $010AAB30$; then there is the creation of the second thread which is the analyzed one.
3) breakpoint in $010BD0A9$ which is the begin target function

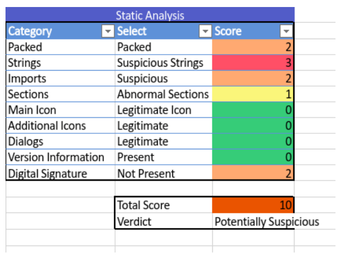
In particular we discovered that the thread calls FUN_010ACABF, which then calls FUN_0109F059, which then calls iteratively FUN_010B0CAD; this last one contains FUN_010BD0A9 which is the target function that performs the malicious actions.

The infection function is FUN_10BD636. It has only the target name as parameter. This procedure is huge, with too much code to reverse; it even contains a recursive call inside.

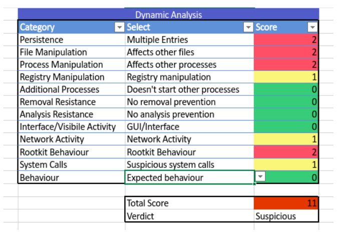
The second target of our analysis is the deactivation of the security center. During the dynamic analysis we discovered that this was done by another process: *services.exe*. Thus we repetead the debugging process keeping *procmon* active, filtering the actions made by that process. We executed the malware step by step, finding that the instruction which performs the action is located at $010B10C7$. At this point the program calls an exteran function which is *StartServiceA* from *ADVAPI32.dll*. This instruction is executed two times without any apparent effect, but the third time immediately after the return the Security Center is turned off. Reading the documentation of the dll we found that this function is able to close services, but only if the caller has the rights to do it. However the malware is executed without administrative priviledge, so it's likely to have a priviledge escalation from previous functions.

V. Conclusion

Summing up the result of our analysis we can describe the malware as a polymorphic one, which performs variuos malicious actions such as internet connections, replications on other system programs and deactivates the security center. It disguise itself as a calculator, fooling the average user. The following table sums up its characteristics.



Static Analysis		
Category	Select	Score
Packed	Packed	2
Strings	Suspicious Strings	3
Imports	Suspicious	2
Sections	Abnormal Sections	1
Main Icon	Legitimate Icon	0
Additional Icons	Legitimate	0
Dialogs	Legitimate	0
Version Information	Present	0
Digital Signature	Not Present	2
	Total Score	10
	Verdict	Potentially Suspicious



Dynamic Analysis		
Category	Select	Score
Persistence	Multiple Entries	2
File Manipulation	Affects other files	2
Process Manipulation	Affects other processes	2
Registry Manipulation	Registry manipulation	1
Additional Processes	Doesn't start other processes	0
Removal Resistance	No removal prevention	0
Analysis Resistance	No analysis prevention	0
Interface/Visibile Activity	GUI/Interface	0
Network Activity	Network Activity	1
Rootkit Behaviour	Rootkit Behaviour	2
System Calls	Suspicious system calls	1
Behaviour	Expected behaviour	0
	Total Score	11
	Verdict	Suspicious

```c
void entry(void){
  unsigned short uVar1;
  int iVar2;
  unsigned int uVar3;
  int iVar4;
  unsigned int local_50;
  unsigned int local_48;
  int *local_3c;

  //Constant pointer to the code: 0x101E00B
  char *code_start = 0x183A+0x1FB4+0x101BADF-0x12C2;

  unsigned int i, j;

  for (i=0; i<7; i++){
    for (j=0x33cc1; j>0x12c5; j--){
    code_start[j] ^= (char)(i / 6) + 0x58U;
    }
  }

  for (i=0; i<0x24880; i++)
    DAT_0109c000[i] = DAT_0101f6cd[i];

  local_3c = &DAT_0104e6cd;

  for (i=0; i<0xa1c8; i++)
    DAT_011b9000[i] = DAT_010440cd[i];
}
```