Hands On Lab Unit 5

MICS-252, Fall 2024

Prepared by: Karl-Johan Westhoff

email: kjwesthoff@berkeley.edu

UC Berkleley School of Information

MICS Course 252 Fall 2024 (Kristy Westphal)

1 Reverse engineering of executables

Two files are provided to be subjected to analysis.

2 'file' analysis

Linux has a simple utility for checking a file' type by its content, see results in Figure 1

```
$ file 'random file'*
random file:  PE32 executable (GUI) Intel 80386, for MS Windows
random file2: PE32+ executable (console) x86-64, for MS Windows
```

Figure 1: *Output from using 'file' in Linux on 'random_file' and 'random_file2'*

2.1 'file' Results

The 'PE32' and 'PE32+' indicate the file format is 'Portable Executable' (the '+' indicating version for 64bit memory structure)[1]. The PE32 standard includes some headers in the file so the Windows operating can execute the files, either as a stand alone .exe files or as part of other programs or things running on the OS as .dll etc.[1]

I.e. the files could contain many functions for something running on a Windows OS. The PE file (ELF files for Linux) contain headers with information of how the program should be laid out in memory on the OS.

3 'Virus Total'

I also threw the files into virus "Virus Total"[2] which both subjects the files to signature scanning and provides details and compares it to the contents to other uploads of the file.

[1] Windows file extensions for PE's include: .acm, .ax, .cpl, .dll, .drv, .efi, .exe, .mui, .ocx, .scr, .sys, .tsp, .mun[1]

3.1 Virus Total Results

Virus totals findings are listed in Appendix A, Figures 5 and 6. Results include various file SHA hashings, file formats etc. Furthermore, the is a 'Names' section where previous uploads of the same file listed, apart from other submissions of 'random file' Virus Total[2] finds the following file names:

- random_file = CNMSE.exe, a Canon printer network management software
- random_file2 = alfi_analyse.exe, not much on google regarding this file, ChatGPT mentions the AFL (American Fuzzing Lop)[3], a tool for software testing by sending unusual inputs to a program.

To confirm if the above findings are correct, next step is trying to reverse engineer the files.

4 Reverse engineering using Ghidra

Static reverse engineering includes de-compiling the executables, looking at the code, but not running it. I used 'Ghidra'[4] for the analysis. Ghidra is a software reverse engineering (SRE) tool created and maintained by the NSA[5] made publicly in 2019[3] Ghidra works as follows:

1. The Binary is 'disassembled', the machine code is read from the binary file
2. Assembly language code is constructed from the machine code and how data is moved around memory. The Assembly language is 'somewhat humanly readable' including comments, readable strings etc. and it is possible to read how data is manipulated by the processor and stored in memory it could look like:

   ```
   MOV AL, 61h       ; Load AL with 97 decimal (61 hex)
   ```

3. Based on the Assembly code, Ghidra constructs C-like code which is more humanly 'interpret-able':
   - It splits parts of the code execution into functions
   - Puts data into variables.
   - Provides information on which part of the code is imported from other code (Windows .dll's, C <headers.h> etc.)
   - It can even draw a graph on how the various functions are called when the program executes

4.1 Ghidra Treasure Hunt

Ghidra chewed away on the files and came up with a de-compilation solution, see example in Figure 2

[2] Looks like others have had the same idea and checked 'random_files'
[3] The existence of Ghidra was apparently published already in 2017 as part of a WikiLeaks leak[6]

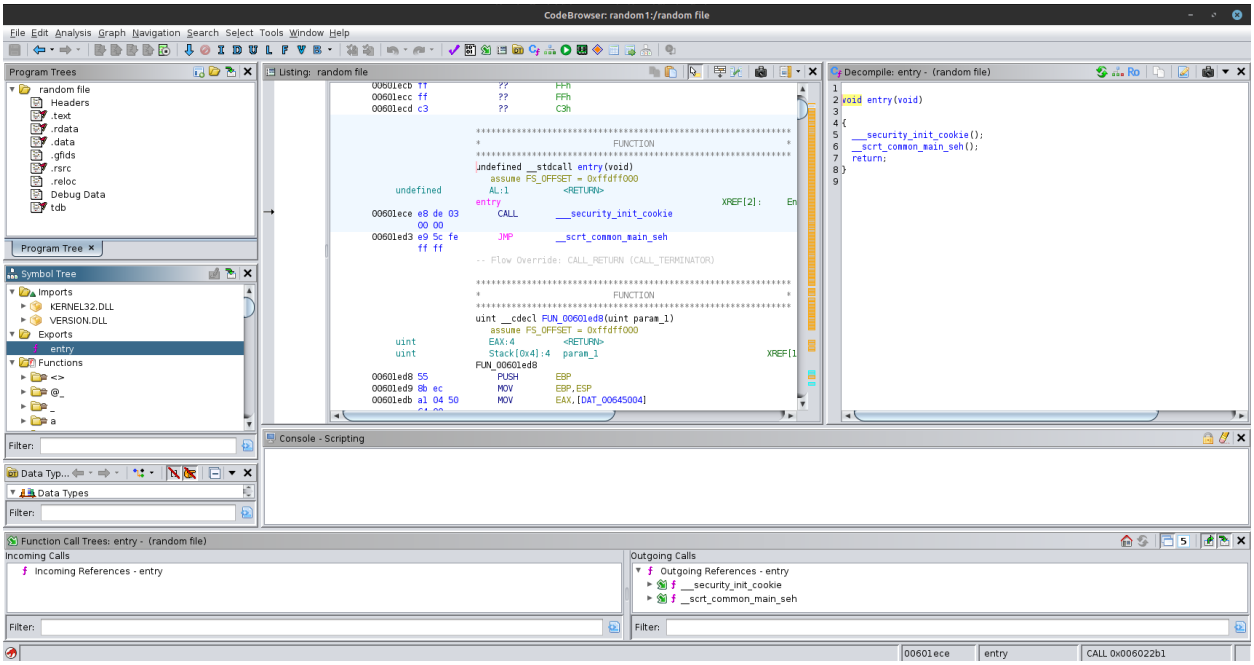

Figure 2: *Ghidra main windows, 'navigation' to the left (see Functions list) Assembly Code in the middle and C code representation to the right, funciton inputs and outputs in the bottom*

To analyze the findings I fist tried to find the entry point function in standard C this is defined as:

```
int main(void);
int main();
int main(int argc, char **argv);
int main(int argc, char *argv[]);
int main(int argc, char **argv, char **env);
```

Looking for functions structured like this was largely un-sucessful (I found multiple functions which fit the schema). Ghidra shows what functions are exposed by the Executable for other code to use as an API, here there is an 'entry' function which first checks some security cookie and then runs a function that "does a bunch of stuff" See a graphical tree representation in Appendix B, Figure 7. Figuring out what really goes on is subject to a deeper analysis including subject matter experts within the specific type of software (hypothesis from Section 3 above being that it is a Canon printer driver).

Another way to analyze the bunch of de-compiled code is to look for strings, in this case for 2 purposes:

1. Search for the 'Canon' name or 'print*' to confirm that the random file is a Canon printer software
2. Search for traces of malware. Trojans and malware that connects to a C2 server needs to know where to 'phone home' i.e. they need some hard coded URL's or a way to generate these (if they are trying o obfuscate). An example is the 'SolarWinds' attack, in which C2 url's were obfuscated using a custom

hash function to hide from code decompilation[7]. So I will look for suspicious URL's and Hash codes.

4.2 Ghidra Results, random_file

Searching for strings confirmed that the random_file most likely is a Canon printer software, see Figure 3. I did not find any suspicious hash codes the files authenticity can be verified against the vendors original file by comparing hash codes.

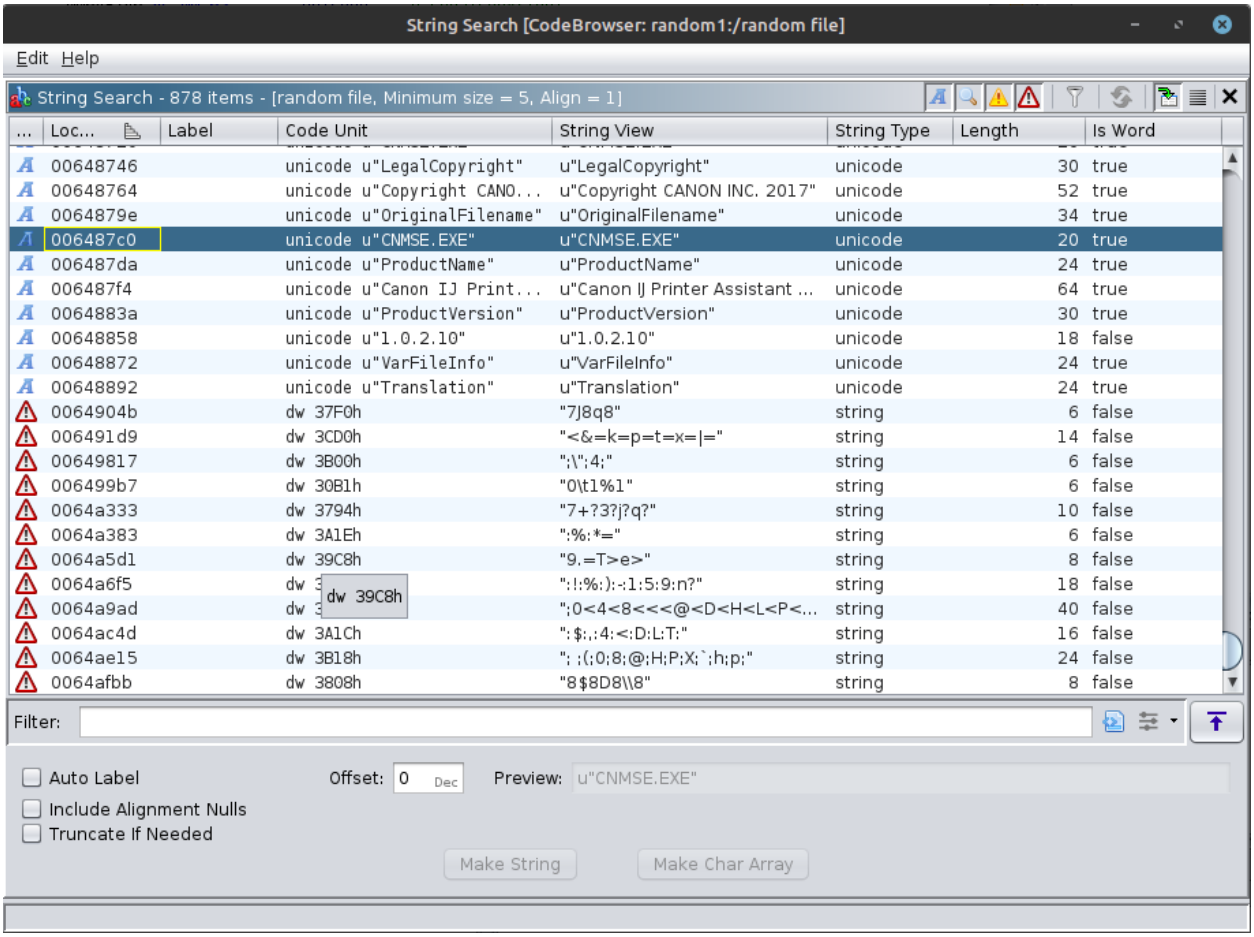

Figure 3: *String search for strings in the whole de-compiled code, note the vendor name 'Canon' and the CNMSE.EXE string*

4.3 Ghidra Results, random_file2

Using the same strategy as above, looking at strings reveal that the file is likely containing a Windows version of the AFL Fuzzing test software. The program flow tree is listed in Appendix C, Figure 8. Following the program flow, there are some conditionals that exit the program if conditions are not correct, this is probably reflected in the 'PROGRAM ABORT' strings in Figure 3.

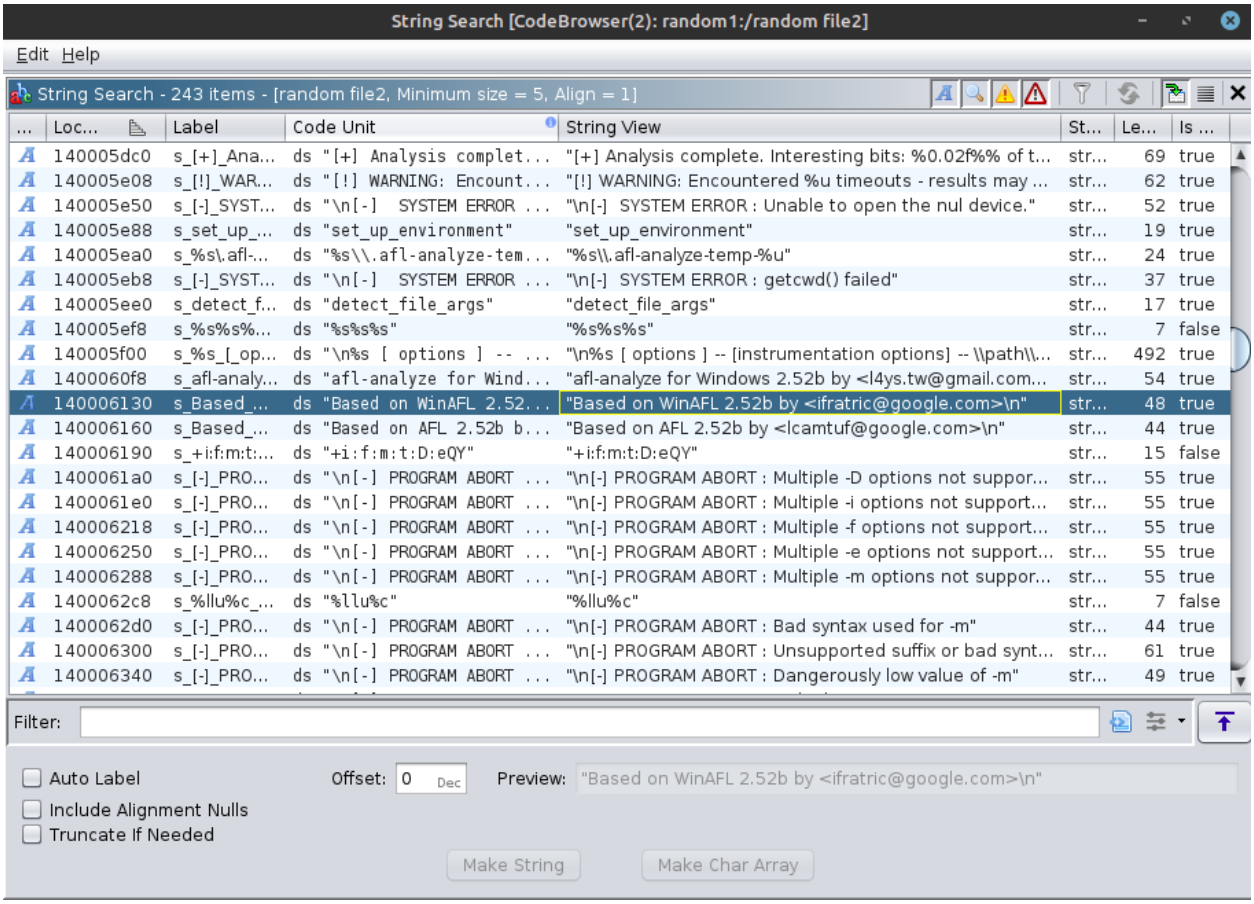

Figure 4: *String search for strings in the whole de-compiled code, note the afl_analyse and WinAFL strings indicating the file is a Windows version of the AFL Fuzzing test software*

5 Conclusion

The files were investigated using various tools: The preinstalled 'file' tool, the online 'Virus Total' and NSA's open source 'Ghidra' tool. The files origins were investigated and identified, further steps would include contacting the original providers of the software for comparison to ensure that no malicious code is hidden in the files (this is of course assuming that there is no malware in the original code either..).

References

[1] *Portable Executable, Wilipedia*. https://en.wikipedia.org/wiki/Portable_Executable. Accessed: 2024-23-13.

[2] *Portable Executable, Wilipedia*. https://www.virustotal.com/gui/home/upload. Accessed: 2024-25-13.

[3] Wikipedia contributors. *American Fuzzy Lop (software) — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=American_Fuzzy_Lop_(software)&oldid=1240165391. [Online; accessed 26-September-2024]. 2024.

[4] *NSA provided opensource reverse engineering tool*. https://ghidra-sre.org/. Accessed: 2024-23-13.

[5] *Ghidra Software Reverse Engineering Framework*. https://github.com/NationalSecurityAgency/ghidra. Accessed: 2024-25-13.

[6] Wikipedia contributors. *Vault 7 — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Vault_7&oldid=1244090880. [Online; accessed 26-September-2024]. 2024.

[7] *Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor*. https://cloud.google.com/blog/topics/threat-intelligence/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor/. Accessed: 2024-06-13.

Appendices

A Virus Total Results

Basic properties

MD5	a1f2d0552d2718ba114a49cd216b563b
SHA-1	34168f6c3b4f0dcdfa7384e4e9aeb0e03d6e2756
SHA-256	313207b2b10ea708facbf8cbee911fd149e69e3c855bfe7fb5a3962d3d1be641
Vhash	025066655d1515156az58pz3dz
Authentihash	e17849e5bda91c9a2ea4e9bcce7338529a03ec469c93c3577c2fcb37e51077d5
Imphash	8cd541ec6ef47f162c1ca1942cfd2e23
Rich PE header hash	63252e93d9dca16e039f4b15060b0475
SSDEEP	6144:DGLbF3xb9fmM4KYv7re4dqFfT0XH3tvPQ3:DGN3xb9OMVy7re0tQ3
TLSH	T1F3546B10B9C08432C6B338314679F2B24DBDA8315E355A9E53DC0DBA6F74491BB39B6B
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TrID	Windows Control Panel Item (generic) (63.7%) | Win64 Executable (generic) (11.6%) | Win32 Dynamic Link Library (generic) (7.2%) | Win16 NE executable (generic) (5.5…
DetectItEasy	PE32 | Compiler: EP:Microsoft Visual C/C++ (2013-2017) [EXE32] | Compiler: Microsoft Visual C/C++ (19.00.24210) [C] | Linker: Microsoft Linker (14.00.24210) | Tool: Vis…
Magika	PEBIN
File size	283.00 KB (289792 bytes)

History

Creation Time	2017-03-14 08:59:31 UTC
First Seen In The Wild	2017-03-14 09:59:31 UTC
First Submission	2017-10-09 02:38:39 UTC
Last Submission	2024-09-23 23:24:30 UTC
Last Analysis	2024-09-23 23:29:22 UTC

Names

CNMSE.EXE
random file
random_file.exe
cnmpase.exe
Prr4Uxhy.exe

Figure 5: *Virus Total details on 'random_file'*

Basic properties

MD5	354cd672f5ec15324fc2008ac6a539df
SHA-1	176781def7de7a4660bbf767133c2b47edcce260
SHA-256	d81c249b279f21f08215690e6e7ce4385cc8147b4c90773b6bf76b3fcec6def1
Vhash	034066555d1515551az26lz
Authentihash	11046ff2b9bf191cc9261c9c5db5280a560bf68a4c7be0fa93d19b2fe1e5ad17
Imphash	ce74ecdef1ff75ea4c1d2031f06eeb68
Rich PE header hash	5410fd4b90c3970696dfd99314d03a28
SSDEEP	768:Eou5UJILaE+VvHOGZI5WLiNdyBwUNF4ec/cj:588ILn+FuGlYiN
TLSH	T118D23CD1678610F4C927407ECF2A282BF43178F21B75D69FA3C64A9D0FE5390A926F85
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32+ executable (console) x86-64, for MS Windows
TrID	Win64 Executable (generic) (44.4%) | Win16 NE executable (generic) (21.3%) | Windows Icons Library (generic) (8.7%) | OS/2 Executable (generic) (8.5%) | Generic Win…
DetectItEasy	PE64 | Compiler: Microsoft Visual C/C++ (19.15.26729) [C] | Linker: Microsoft Linker (14.15.26729) | Tool: Visual Studio (2017 version 15.8)
Magika	PEBIN
File size	29.50 KB (30208 bytes)

History

Creation Time	2019-03-12 12:36:28 UTC
First Seen In The Wild	2020-06-11 13:11:24 UTC
First Submission	2019-04-11 21:41:05 UTC
Last Submission	2024-06-10 05:33:33 UTC
Last Analysis	2024-06-01 09:39:15 UTC

Names

afl-analyze.exe
random file2
random file2.exe
random file2 (3)
random file2 (2)
ChZXyu.exe

Figure 6: *Virus Total details on 'random_file2'*

B Ghidra Analyses of 'random_file'

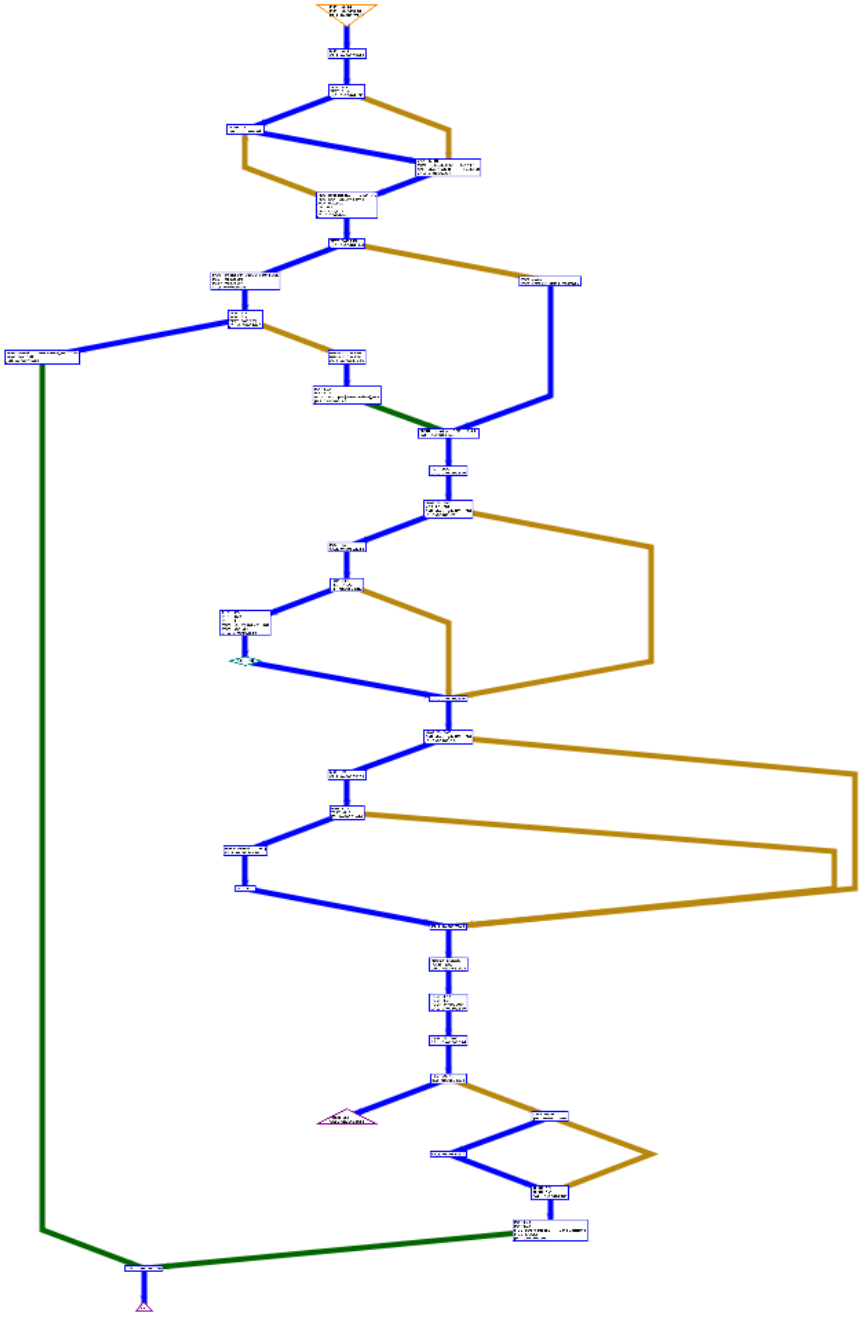

Figure 7: *'random_file' scrt_common_main_seh(void) Function as entry point*

C Ghidra Analyses of 'random_file2'

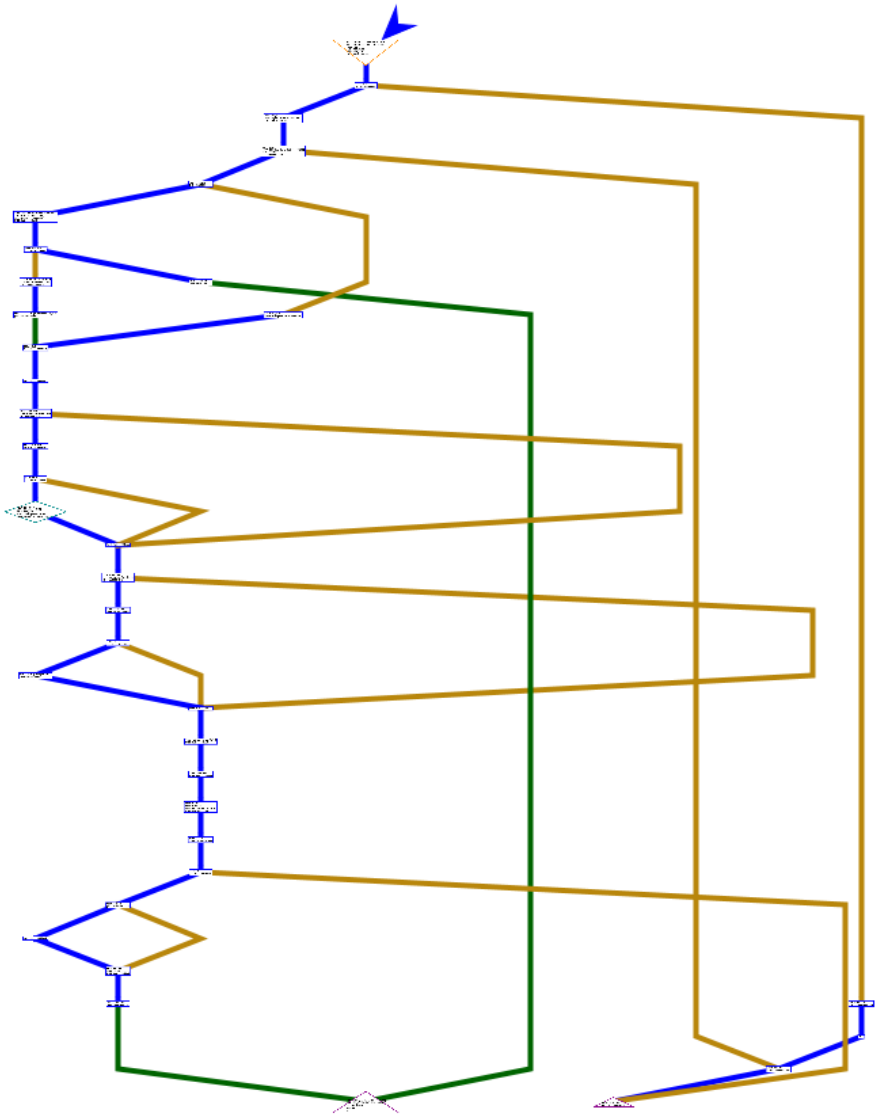

Figure 8: *'random_file2' 'FUN_140004618(void)' Function as entry point (obviously Ghidra picked the name..)*

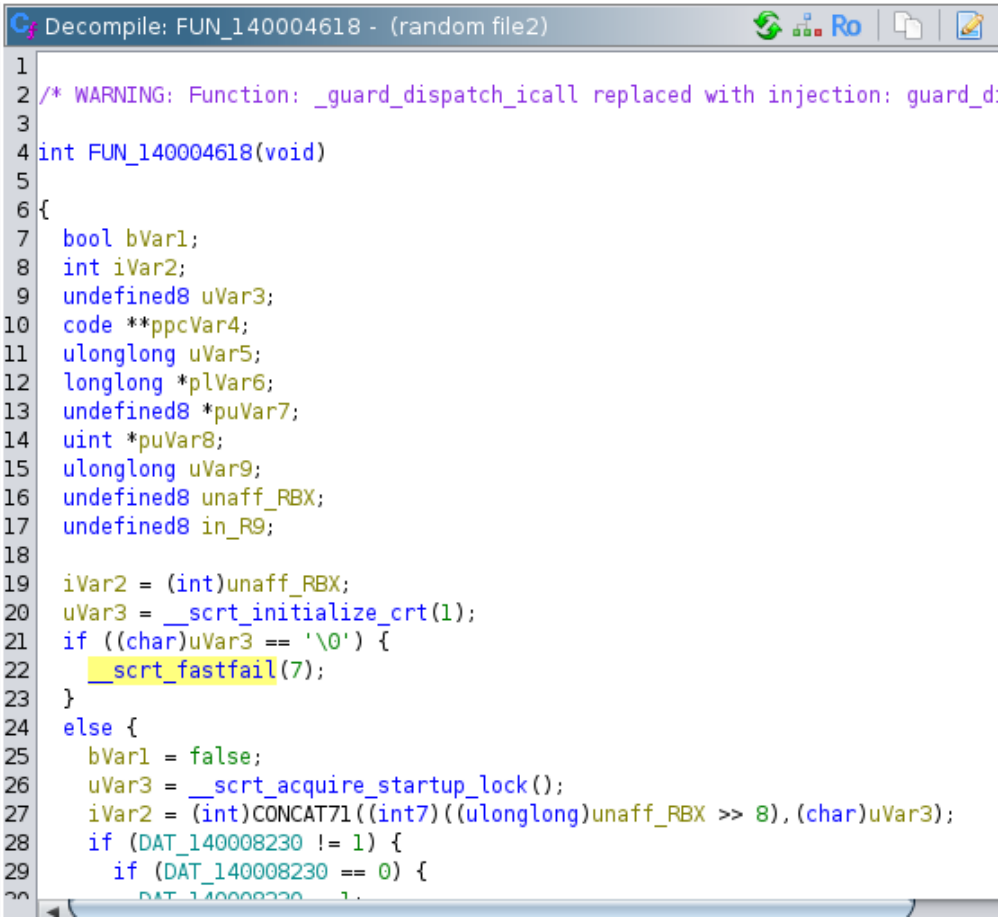

Figure 9: *'random_file2' 'FUN_140004618(void)' Function a portion of the conditionals execute the 'fastFail' function when they resolve to false*