

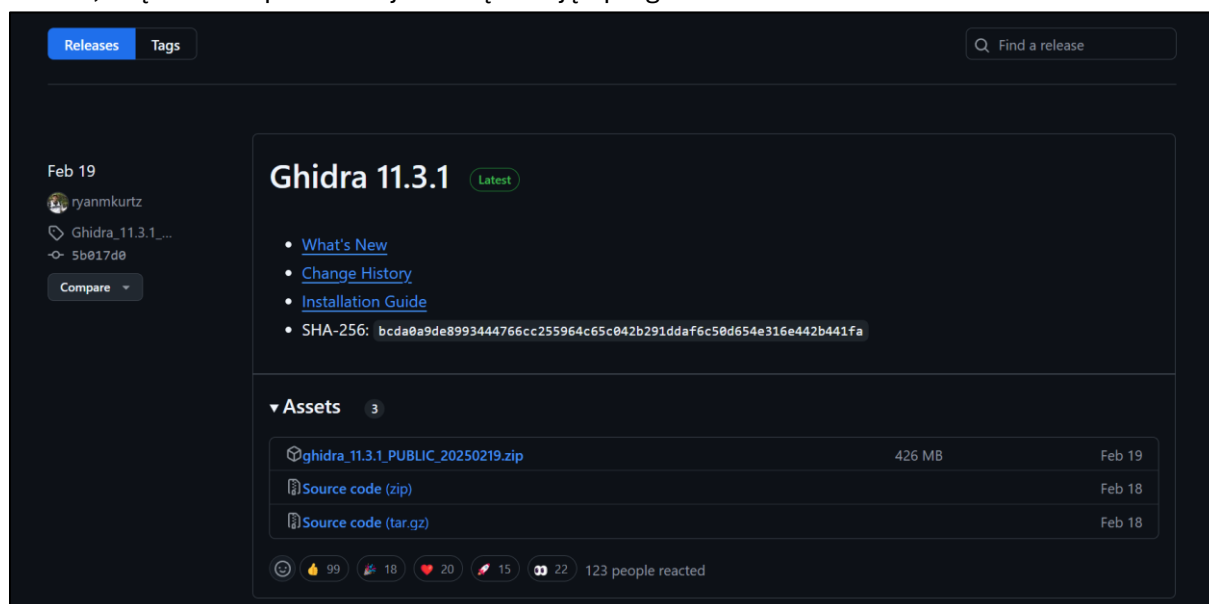
## Programowanie niskopoziomowe – Lab 3/4

### Z wykonanego laboratorium należy wykonać sprawozdanie

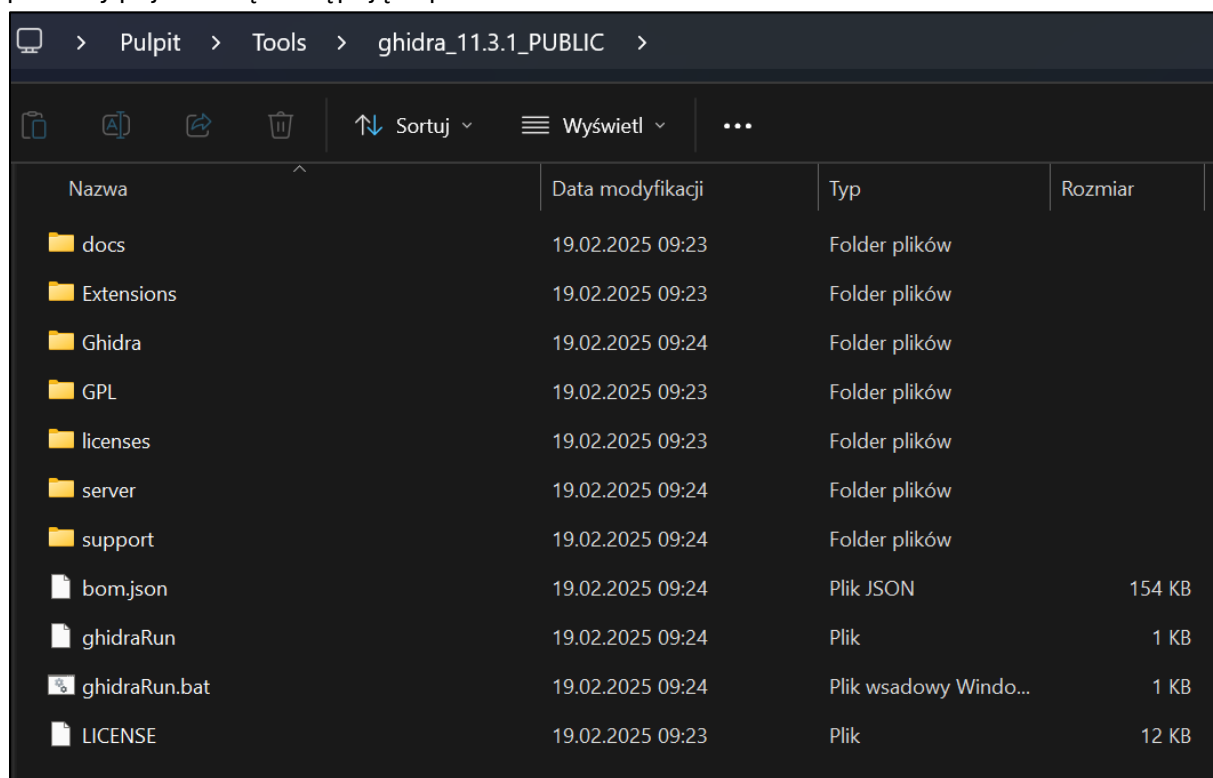
Aby zainstalować Ghidrę w systemie Windows, należy przejść na oficjalną stronę internetową [Ghidra](#). Powinna wyświetlić się strona podobna do poniższej.



Należy wybrać „Download from Github”, co spowoduje przejście na stronę Ghidra w serwisie Github, skąd można pobrać najnowszą wersję oprogramowania.

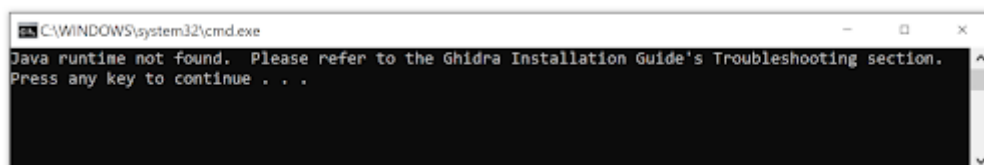
The image shows the GitHub release page for Ghidra 11.3.1. The page has a dark theme. At the top, there are tabs for "Releases" and "Tags", and a search bar. The main content area shows the release "Ghidra 11.3.1" with a "Latest" badge. Below the title, there are links for "What's New", "Change History", and "Installation Guide". A SHA-256 hash is displayed: bcd0a9de8993444766cc255964c65c042b291dda6c50d654e316e442b441fa. Under the "Assets" section, there are three assets: "ghidra\_11.3.1\_PUBLIC\_20250219.zip" (426 MB, Feb 19), "Source code (zip)" (Feb 18), and "Source code (tar.gz)" (Feb 18). At the bottom, there are reaction icons (thumbs up, thumbs down, heart, etc.) and a count of "123 people reacted".

Należy pobrać plik zip i rozpakować jego zawartość w wybranej lokalizacji. Po rozpakowaniu powinny pojawić się następujące pliki:



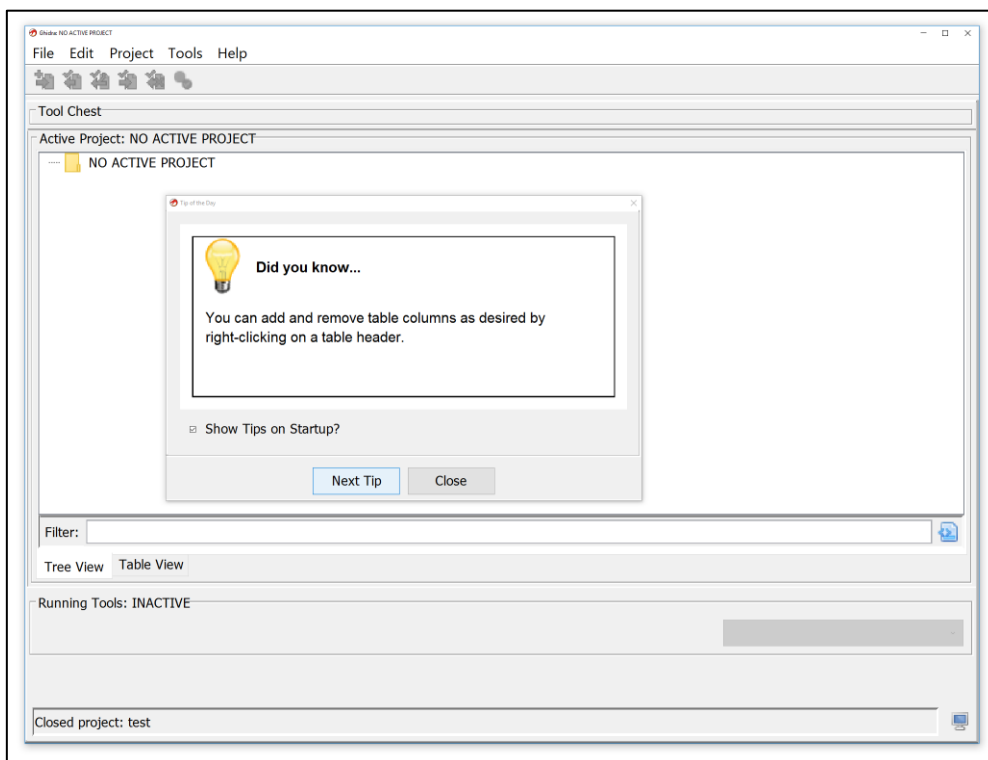
Nazwa	Data modyfikacji	Typ	Rozmiar
docs	19.02.2025 09:23	Folder plików	
Extensions	19.02.2025 09:23	Folder plików	
Ghidra	19.02.2025 09:24	Folder plików	
GPL	19.02.2025 09:23	Folder plików	
licenses	19.02.2025 09:23	Folder plików	
server	19.02.2025 09:24	Folder plików	
support	19.02.2025 09:24	Folder plików	
bom.json	19.02.2025 09:24	Plik JSON	154 KB
ghidraRun	19.02.2025 09:24	Plik	1 KB
ghidraRun.bat	19.02.2025 09:24	Plik wsadowy Windo...	1 KB
LICENSE	19.02.2025 09:23	Plik	12 KB

Aby uruchomić Ghidrę, należy dwukrotnie uruchomić plik wsadowy Windows o nazwie ghidraRun. Może wtedy wystąpić błąd podobny do przedstawionego na ilustracji.

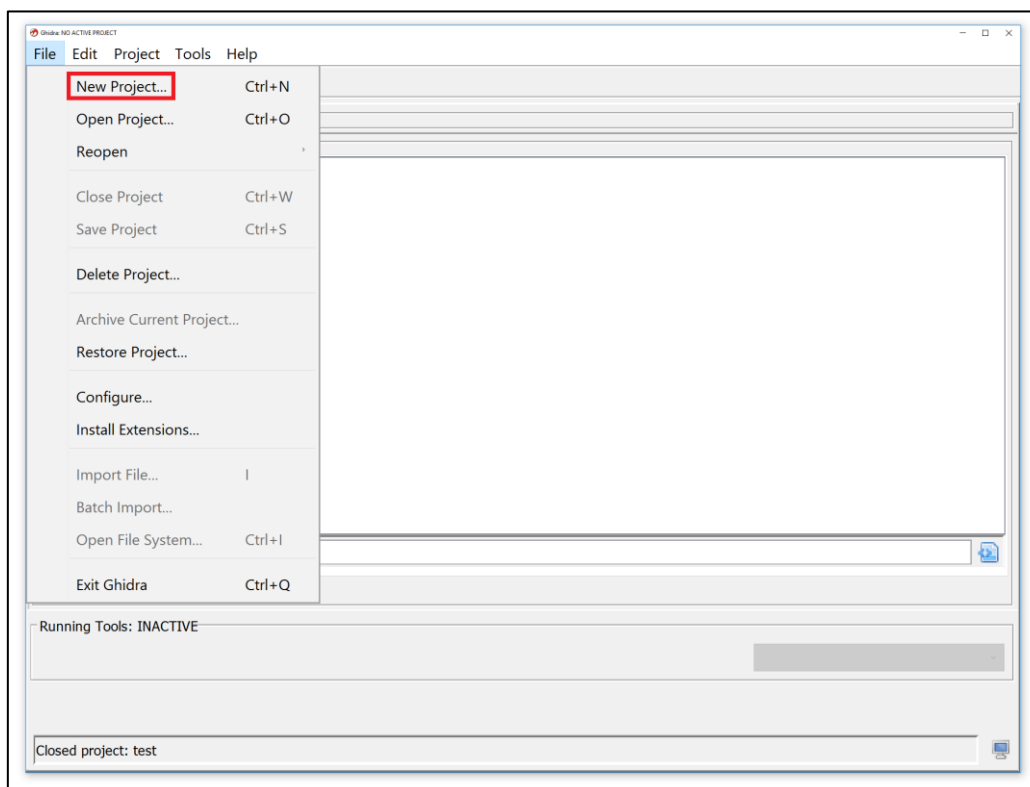


W przypadku pojawienia się takiego komunikatu można skorzystać z dostępnego materiału [video](#), który przedstawia sposób szybkiego rozwiązania problemu.

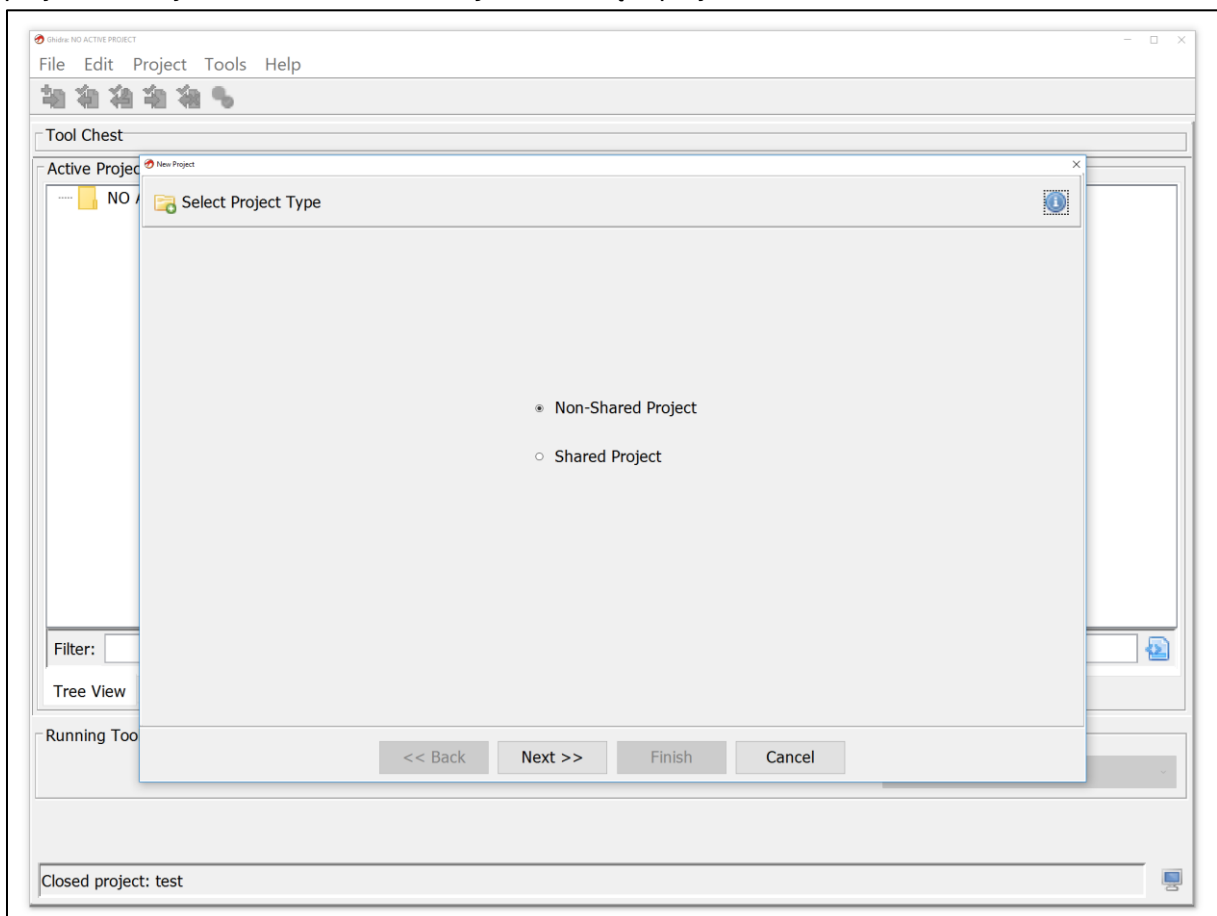
Po uruchomieniu Ghidry wyświetlane są dwa okna: **Active Project** oraz **Tip of the Day**. Okno **Tip of the Day** zawiera wskazówki i może zostać zamknięte.



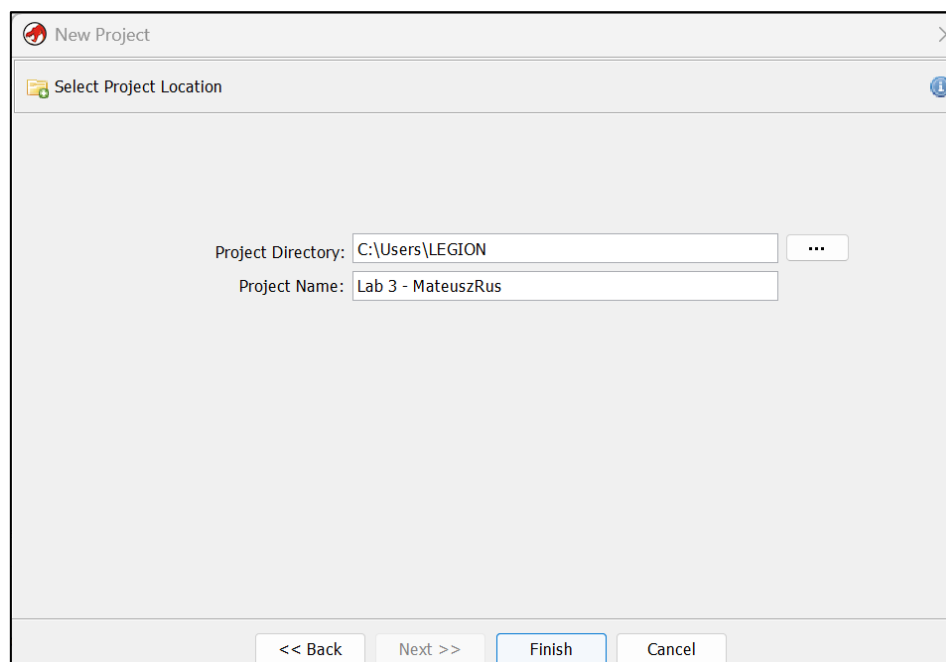
Aby rozpocząć analizę próbki w Ghidrze, należy najpierw utworzyć projekt, w którym przechowywana będzie zarówno sama próbka, jak i wszelkie pliki wygenerowane przez Ghidrę. W tym celu należy wybrać **File**, a następnie **New Project**.



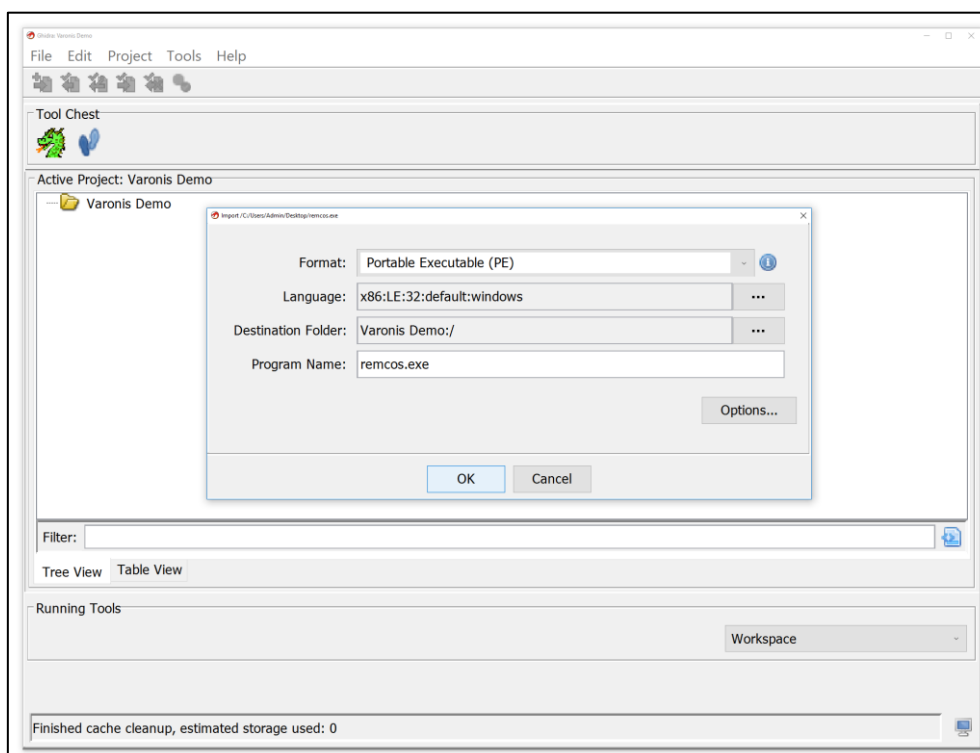
Należy wskazać, czy projekt ma być współdzielony (ang. *shared*), czy nie. W przedstawionym przykładzie wybrano **Non-Shared Project** i kliknięto przycisk **Next**.



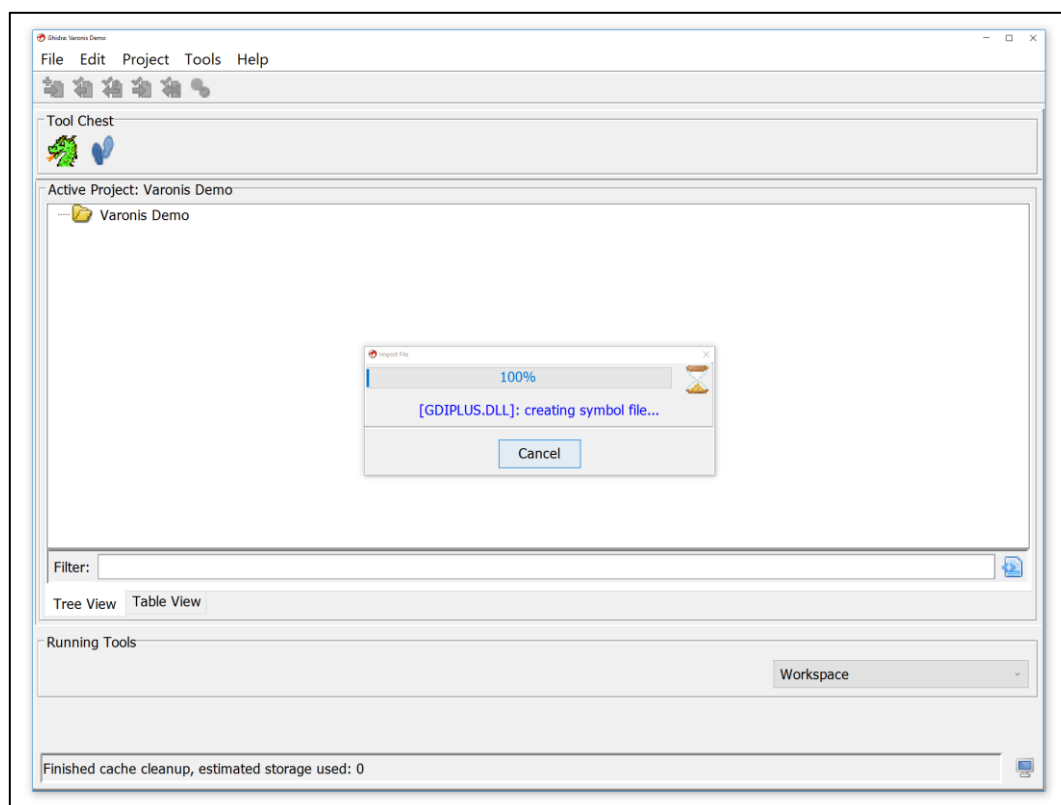
Następnie należy nadać projektowi nazwę, np. odpowiadającą analizowanemu złośliwemu oprogramowaniu, i wybrać **Finish**. W przytoczonym przykładzie użyto nazwy *Lab 3 – ImięNazwisko*



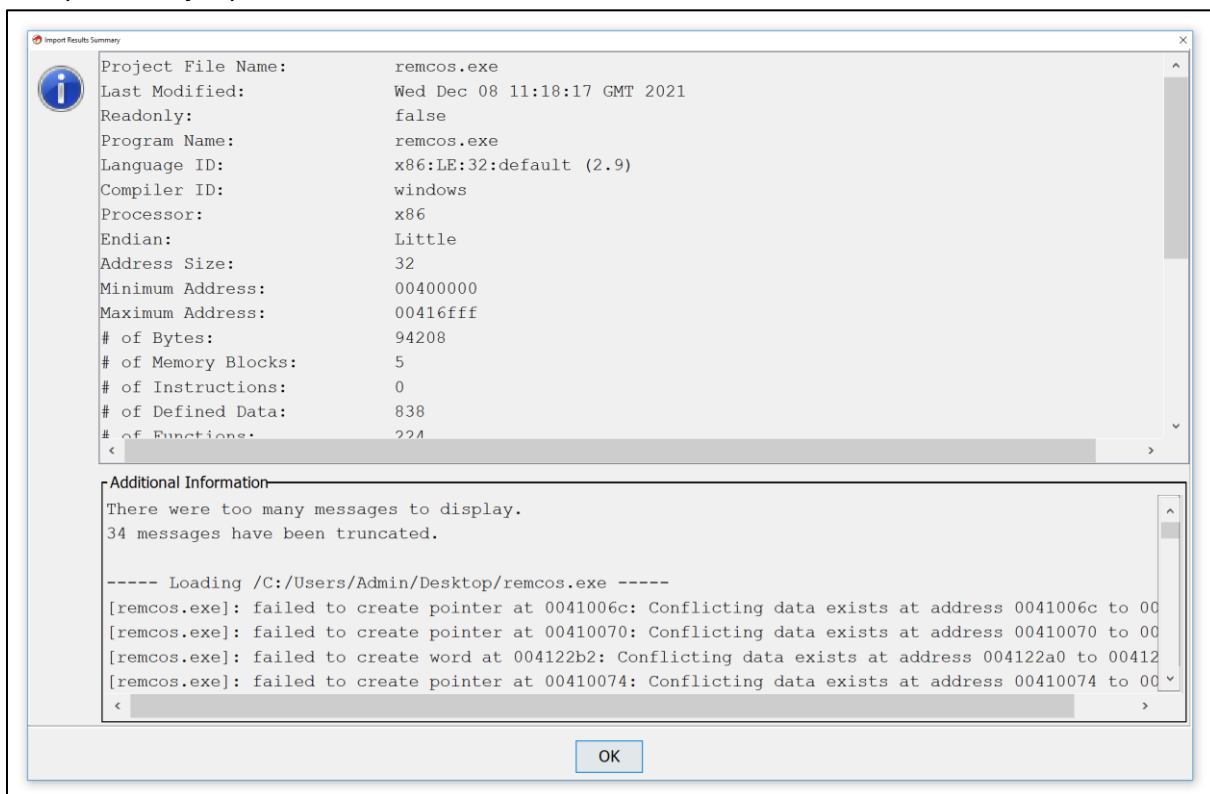
Po utworzeniu nowego projektu można do niego przeciągać próbki przeznaczone do analizy. Na ilustracji przedstawiono plik *remcos.exe* dodany do projektu.



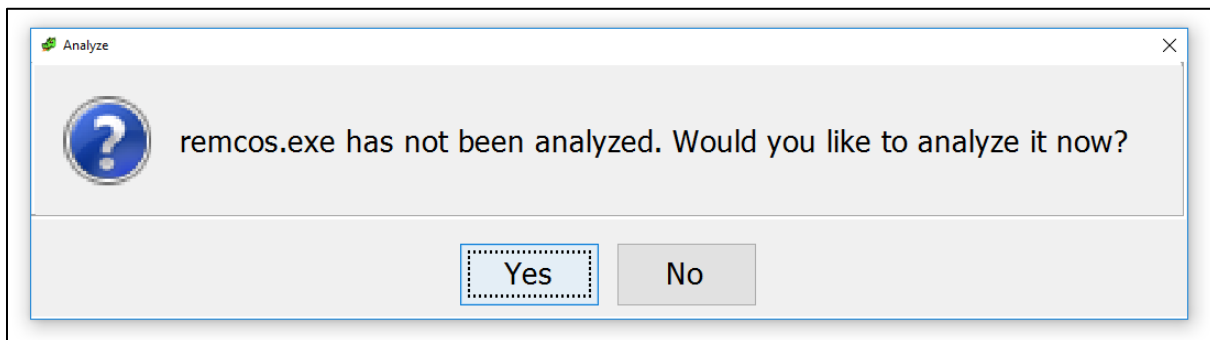
Ghidra rozpoznała go jako 32-bitowy plik PE systemu Windows. Po wybraniu opcji **OK** Ghidra rozpoczyna importowanie pliku, a następnie wyświetlany jest pasek postępu.



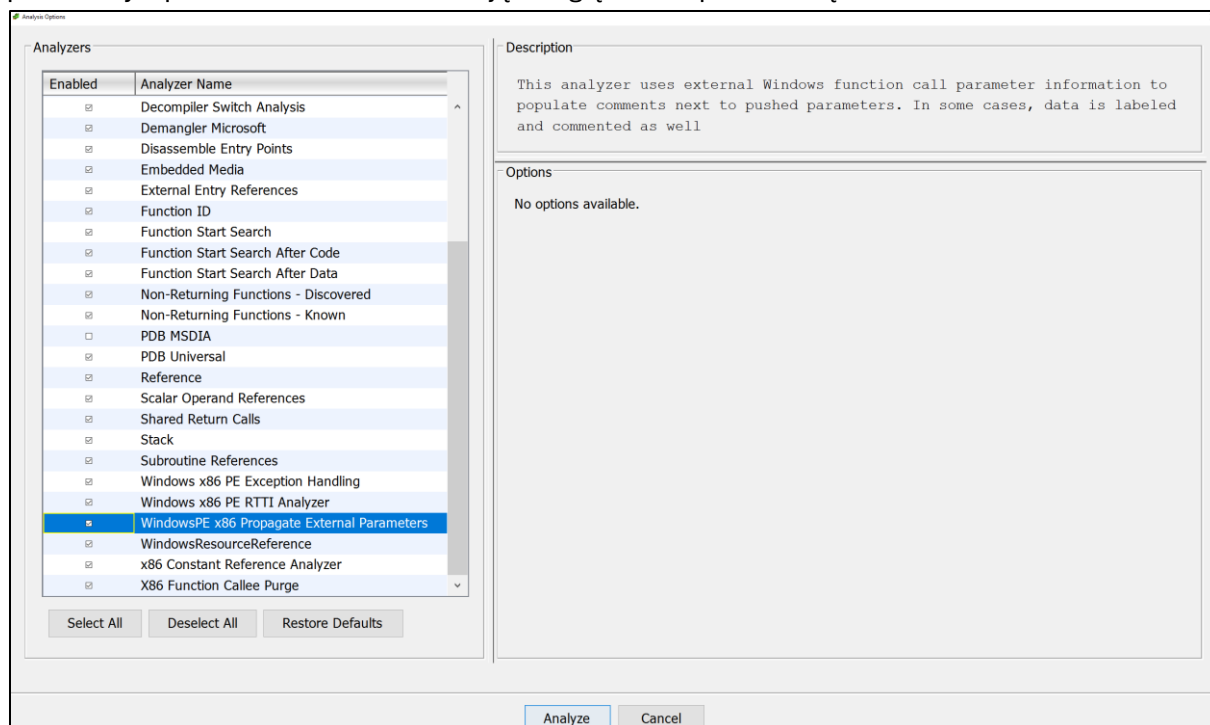
Po pomyślnym zaimportowaniu pliku do Ghidry pojawia się okno z informacjami o zaimportowanym pliku.



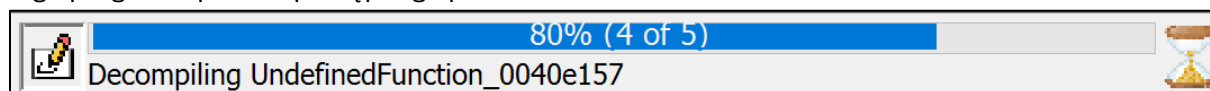
Należy wybrać **OK**, a następnie dwukrotnie kliknąć nazwę zaimportowanego złośliwego oprogramowania lub użyć ikony z logo smoka, aby otworzyć przeglądarkę kodu (*Code Browser*). Zostanie wyświetlone powiadomienie, że Ghidra nie przeanalizowała jeszcze pliku, a użytkownik otrzymuje zapytanie, czy przeprowadzić analizę w tym momencie. Należy wybrać **Yes**.



W kolejnym oknie należy skonfigurować opcje analizy. Zaleca się zaznaczenie opcji **WindowsPE x86 Propagate External Parameters**, co ułatwia analizę importowanych funkcji, ponieważ parametry wprowadzane na stos zostają uwzględnione przez narzędzie.



Należy wybrać **Analyze**, aby Ghidra rozpoczęła analizę pliku. Pasek stanu w prawym dolnym rogu programu pokaże postęp tego procesu.



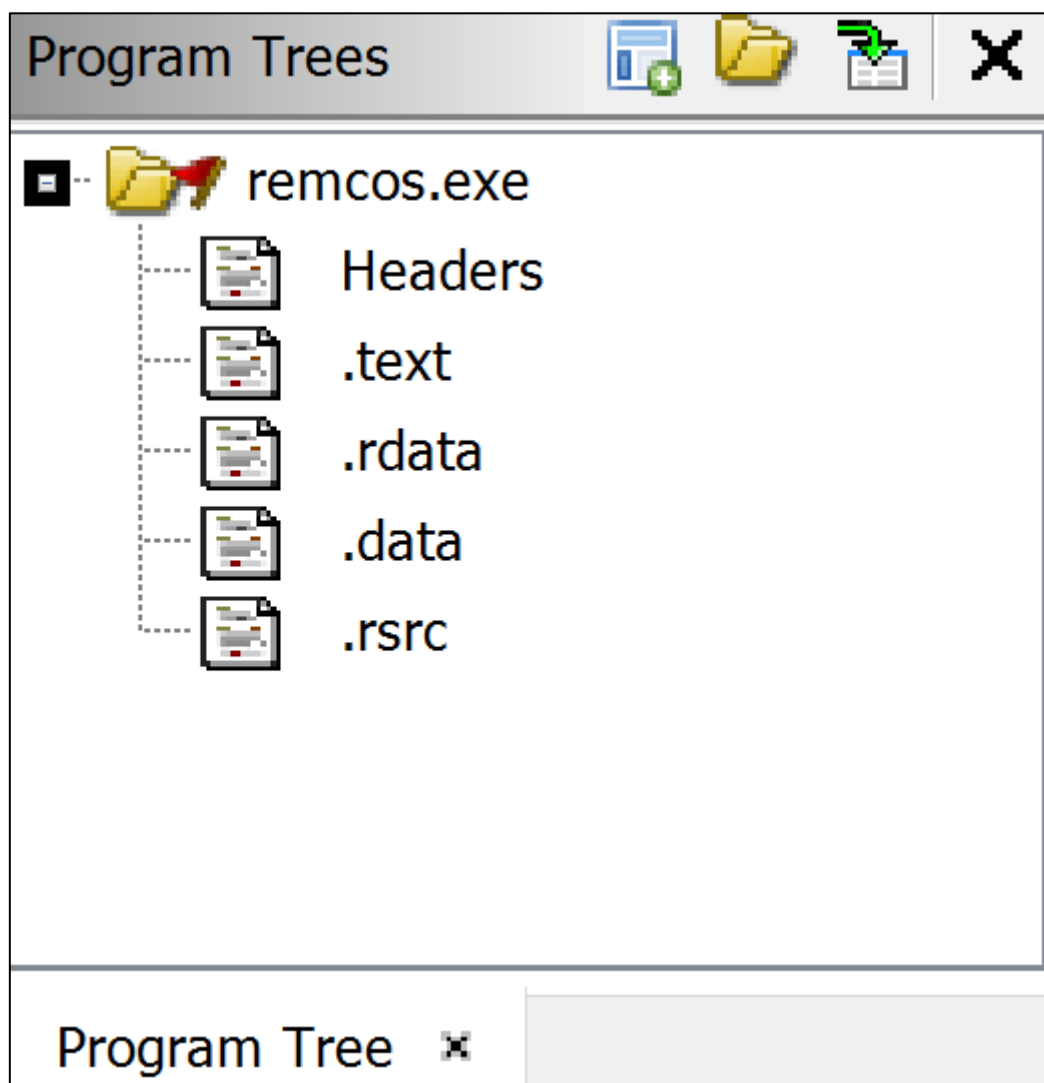
Po ukończeniu analizy Ghidra jest gotowa do prowadzenia inżynierii wstecznej oprogramowania.

## Reverse Engineering Using Ghidra

Po otwarciu w Ghidrze oprogramowania, które zostało wcześniej zaimportowane i przeanalizowane, wyświetlanych jest kilka okien.

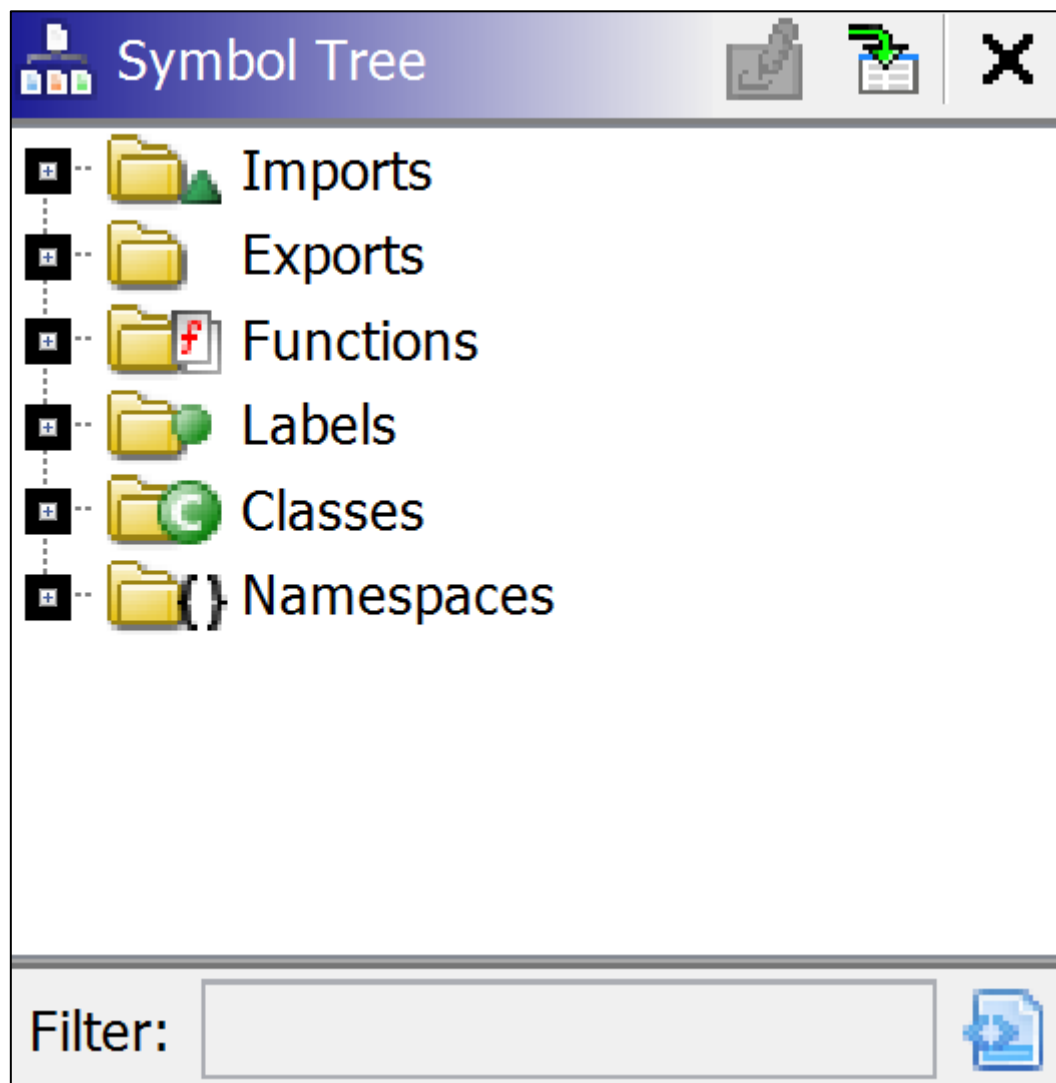
### Główne okna Ghidry

W lewym górnym rogu wyświetlane jest okno zawierające sekcje danego pliku wykonywalnego, oznaczone jako **Program Trees**.

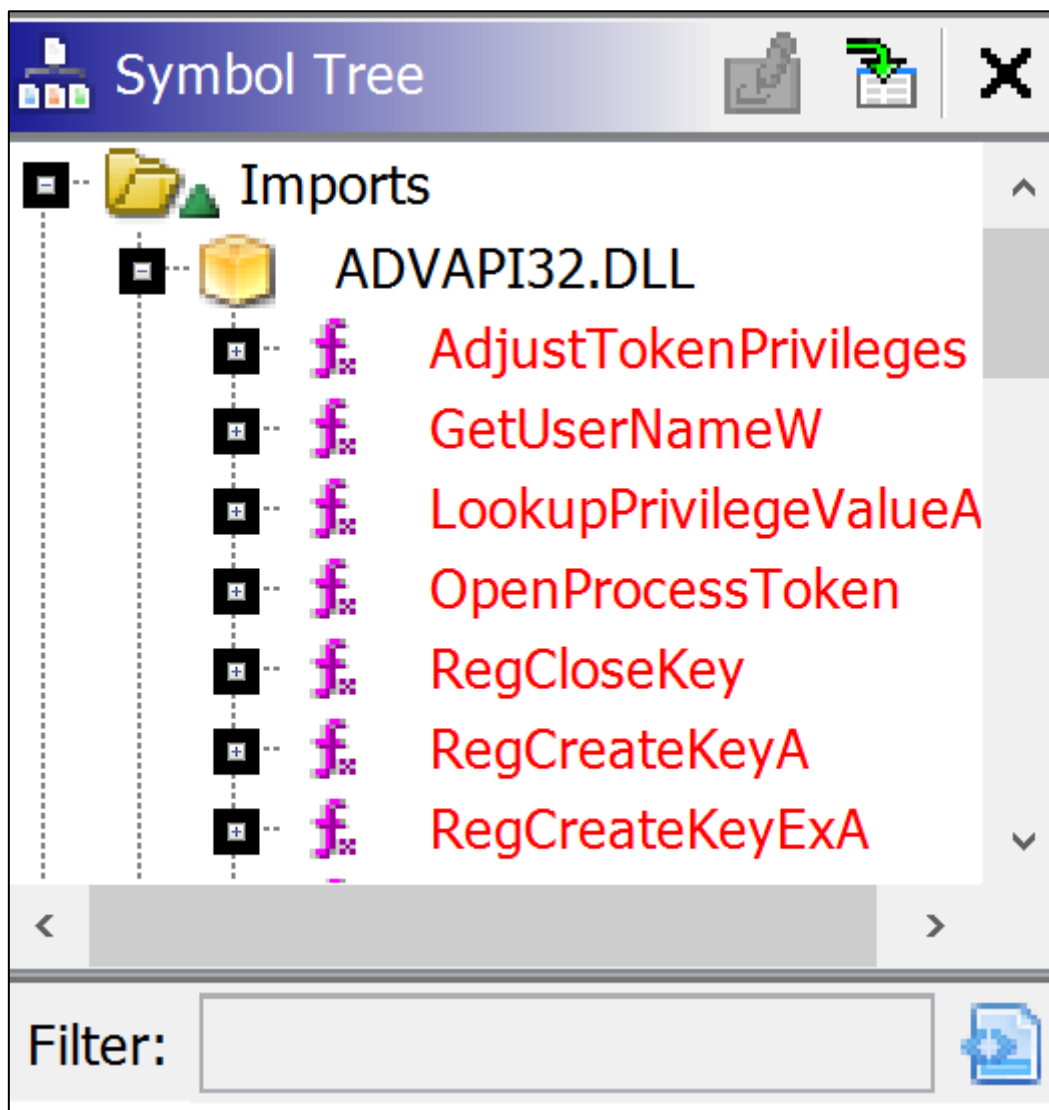




Okno **Symbol Tree** okazuje się bardzo przydatne, gdyż zawiera informacje o importach, eksportach oraz funkcjach wykorzystywanych przez złośliwe oprogramowanie w działaniach o charakterze złośliwym.

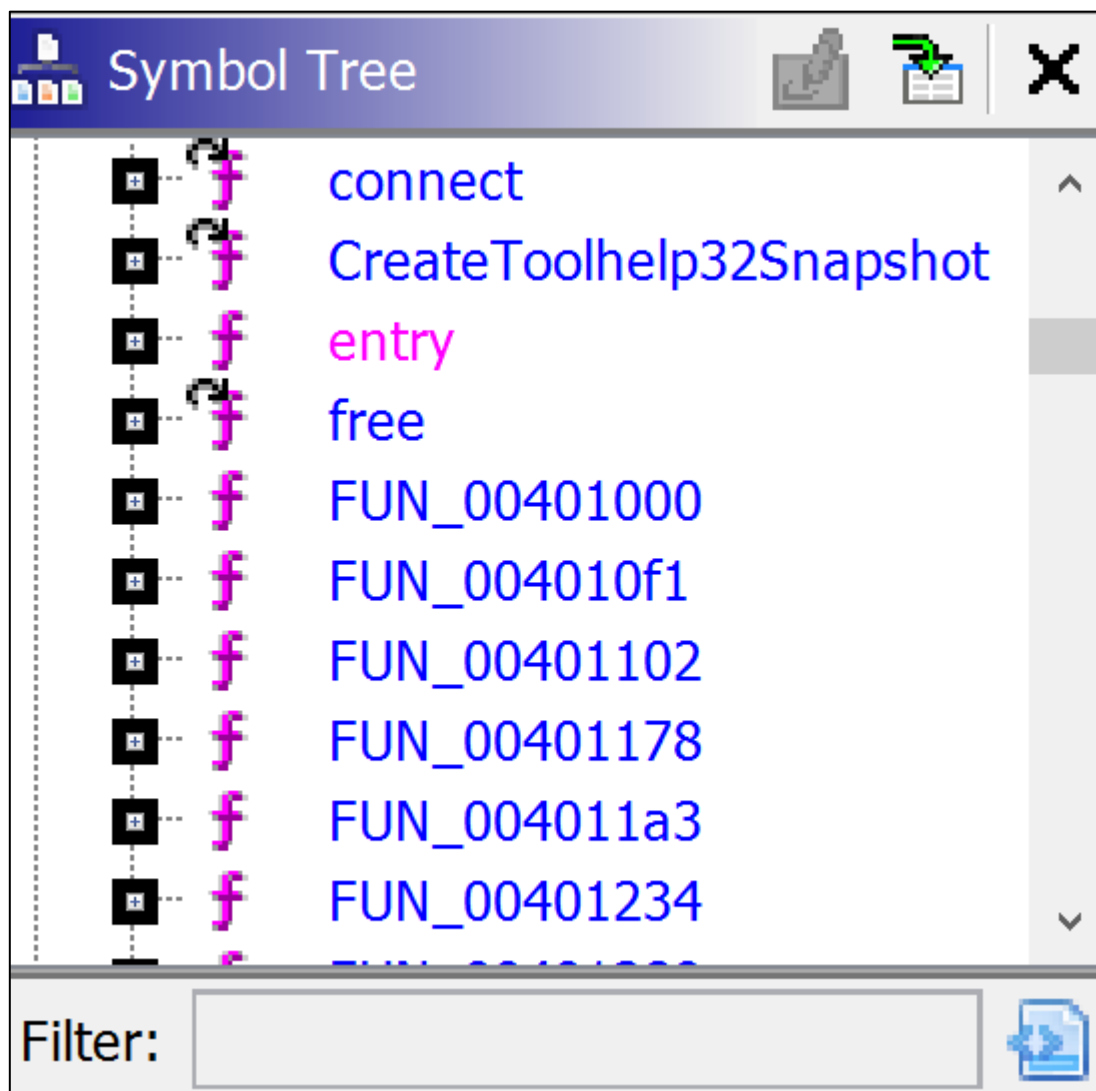


Po wybraniu zakładki **Imports** można sprawdzić, które biblioteki zostały zaimportowane przez złośliwe oprogramowanie. Kliknięcie wybranej biblioteki DLL pozwala zobaczyć zaimportowane funkcje powiązane z daną biblioteką.



Dzięki analizie importów można zidentyfikować interesujące funkcje wykorzystywane przez złośliwe oprogramowanie. Umożliwia to określenie, czy i w jaki sposób dana funkcja jest faktycznie używana, co pozwala zrozumieć potencjalne działania podejmowane na zainfekowanym komputerze. Podobne postępowanie dotyczy zakładki **Exports**, choć w przedstawionym przykładzie żadne funkcje nie są eksportowane.

W oknie **Symbol Tree** znajduje się również lista wszystkich funkcji napisanych przez autora złośliwego oprogramowania. Podczas importowania i analizy pliku Ghidra podejmuje próbę przypisywania nazw niektórym funkcjom, zazwyczaj na podstawie zauważonych importów lub rozpoznanych wzorców. Przykładem może być funkcja *CreateToolhelp32Snapshot*, która jest oficjalną nazwą importowanej funkcji służącej do wyliczania procesów działających w systemie.



W widocznym zestawieniu można też zaobserwować funkcje o nazwach rozpoczynających się od *FUN\_* oraz ciągu cyfr. Takie funkcje nie zostały zidentyfikowane przez Ghidrę, dlatego narzędzie nadaje im ogólną nazwę *FUN\_*, a następnie wartość szesnastkową odpowiadającą lokalizacji w pliku binarnym.

W drzewie symboli znajduje się również pozycja *entry*, czyli punkt wejścia programu. Po dwukrotnym kliknięciu tego wpisu okno **Listing** zostaje zaktualizowane, prezentując kod asemblera w punkcie startowym złośliwego oprogramowania.

<i>entry</i>		XREF[2]:		Entry Point(*)
				00400130 (*)
0040fd88	55	PUSH	EBP	
0040fd89	8b ec	MOV	EBP, ESP	
0040fd8b	6a ff	PUSH	-0x1	
0040fd8d	68 18 19...	PUSH	DAT_00411918	; = FFh
0040fd92	68 60 fc...	PUSH	DAT_0040fc60	; = FFh
0040fd97	64 a1 00...	MOV	EAX, FS:[0x0]	
0040fd9d	50	PUSH	EAX	
0040fd9e	64 89 25...	MOV	dword ptr FS:[0x0], ESP	
0040fda5	83 ec 68	SUB	ESP, 0x68	
0040fda8	53	PUSH	EBX	
0040fda9	56	PUSH	ESI	
0040fdaa	57	PUSH	EDI	
0040fdab	89 65 e8	MOV	dword ptr [EBP + local_1c], ESP	
0040fdae	33 db	XOR	EBX, EBX	
0040fdb0	89 5d fc	MOV	dword ptr [EBP + local_8], EBX	
0040fdb3	6a 02	PUSH	0x2	; int param
0040fdb5	ff 15 10...	CALL	dword ptr [->MSVCRT.DLL: __set_app_type]	
0040fdbb	59	POP	ECX	
0040fdbc	83 0d 1c...	OR	dword ptr [DAT_00415c1c], 0xffffffff	
0040fdc3	83 0d 20...	OR	dword ptr [DAT_00415c20], 0xffffffff	
0040fdca	ff 15 14...	CALL	dword ptr [->MSVCRT.DLL: __p_fmode]	
0040fdd0	8b 0d 14...	MOV	ECX, dword ptr [DAT_00415c14]	

Tego rodzaju analiza pozwala ustalić, jakie działania wykonuje złośliwe oprogramowanie, koncentrując się na potencjalnie podejrzanych wywołaniach funkcji API w oknie **Symbol Tree**.

Po wybraniu *entry* zawartość okna **Decompile** zostaje zaktualizowana, prezentując kod, który Ghidra próbuje przekonwertować z assemblera (widocznego w oknie **Listing**) na kod w języku C. Taka reprezentacja może ułatwić zrozumienie, jak mógł wyglądać oryginalny kod stworzony przez autora złośliwego oprogramowania.

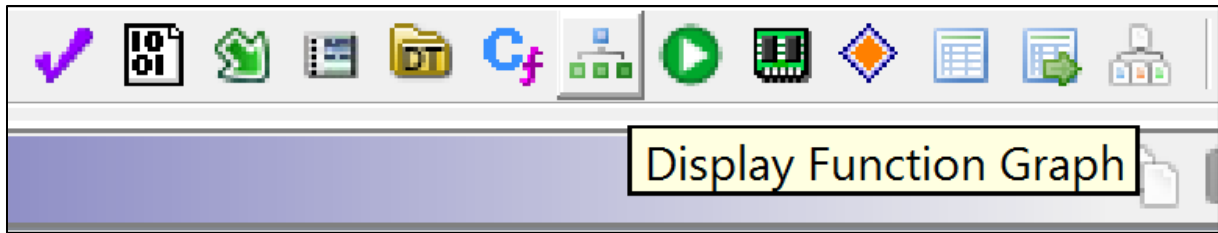


The screenshot shows the Ghidra Decompiler window titled "Decompile: entry - (remcos.exe)". The window contains a list of C code statements, each preceded by a line number from 22 to 49. The code includes variable declarations, pointer assignments, function calls, and conditional logic. A yellow highlight is present on line 36, marking the function call `FUN_0040ff13()`. The code is as follows:

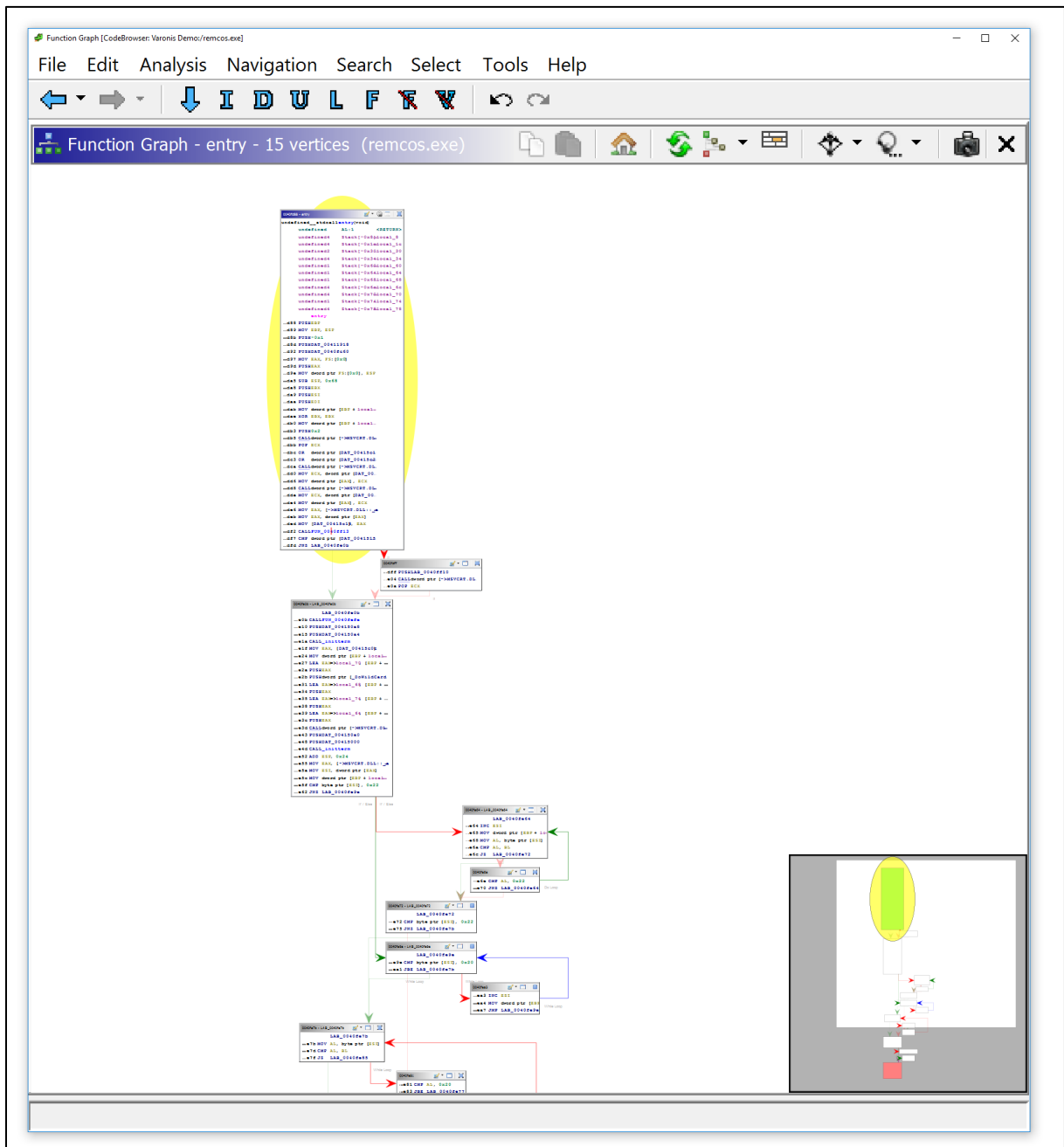
```
22  puStack12 = &DAT_00411918;
23  puStack16 = &DAT_0040fc60;
24  uStack20 = *in_FS_OFFSET;
25  *in_FS_OFFSET = &uStack20;
26  local_1c = &stack0xffffffff78;
27  local_8 = 0;
28  __set_app_type(2);
29  _DAT_00415c1c = 0xffffffff;
30  _DAT_00415c20 = 0xffffffff;
31  puVar1 = (undefined4 *) __p__fmode();
32  *puVar1 = DAT_00415c14;
33  puVar1 = (undefined4 *) __p__commode();
34  *puVar1 = DAT_00415c10;
35  _DAT_00415c18 = *(undefined4 *) _adjust_fdiv_exref
36  FUN_0040ff13();
37  if ( _DAT_00415150 == 0 ) {
38      __setusermatherr(&LAB_0040ff10);
39  }
40  FUN_0040fefe();
41  __initterm(&DAT_004150a4, &DAT_004150a8);
42  local_70 = (int) DAT_00415c0c;
43  __getmainargs(&local_64, &local_74, &local_68, _DoWi
44  __initterm(&DAT_00415000, &DAT_004150a0);
45  local_78 = *(byte **) _acmdln_exref;
46  if (*local_78 != 0x22) {
47      do {
48          if (*local_78 < 0x21) goto LAB_0040fe7b;
49          local_78 = local_78 + 1;
```

### Function Graph

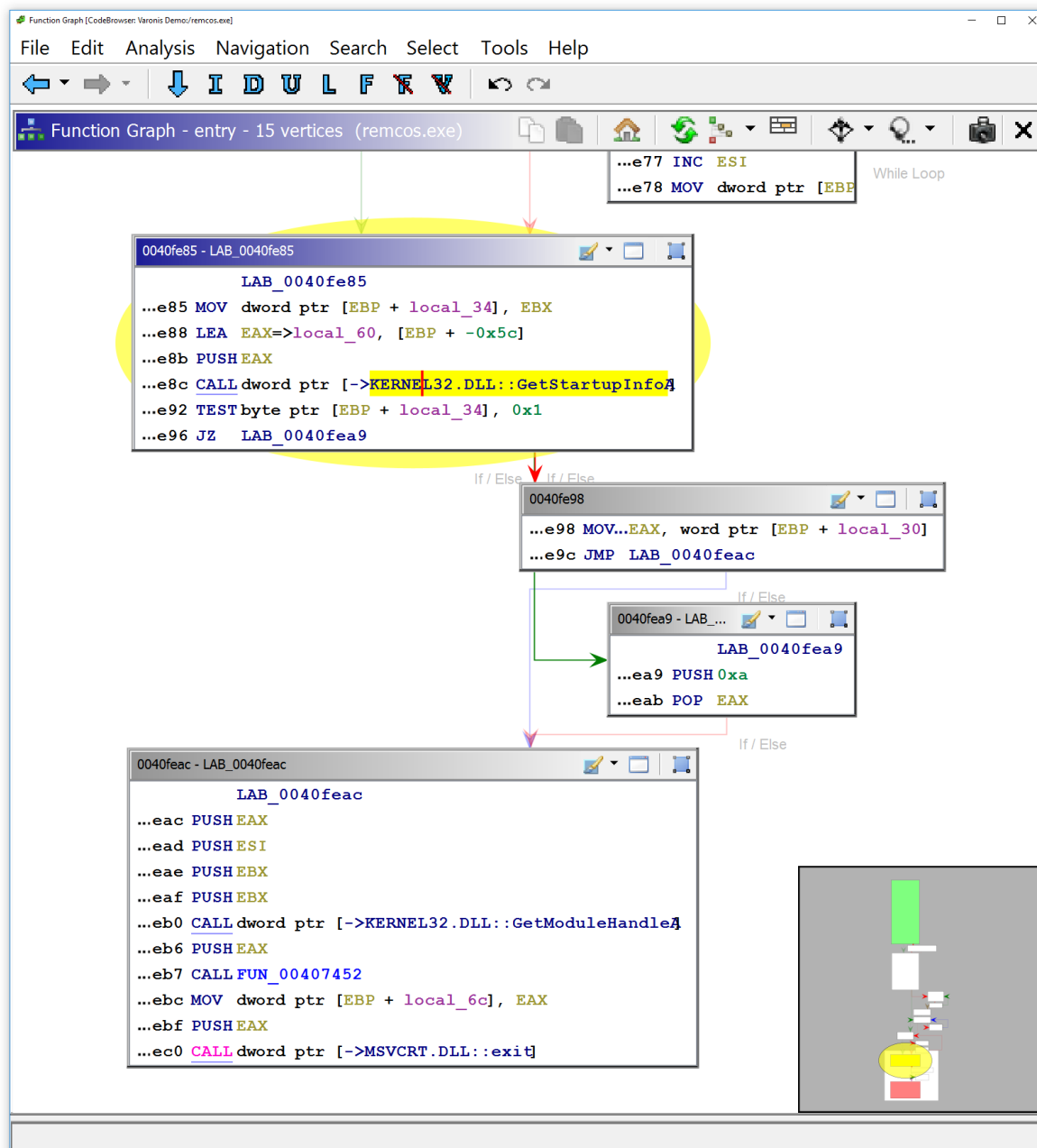
Kolejną przydatną funkcją w procesie analizy jest graficzne przedstawienie kodu funkcji (*Function Graph*). Można je wywołać, klikając **Display Function Graph** na pasku narzędzi Ghidry.



W rezultacie pojawia się graficzna reprezentacja aktualnie wyświetlanej funkcji.



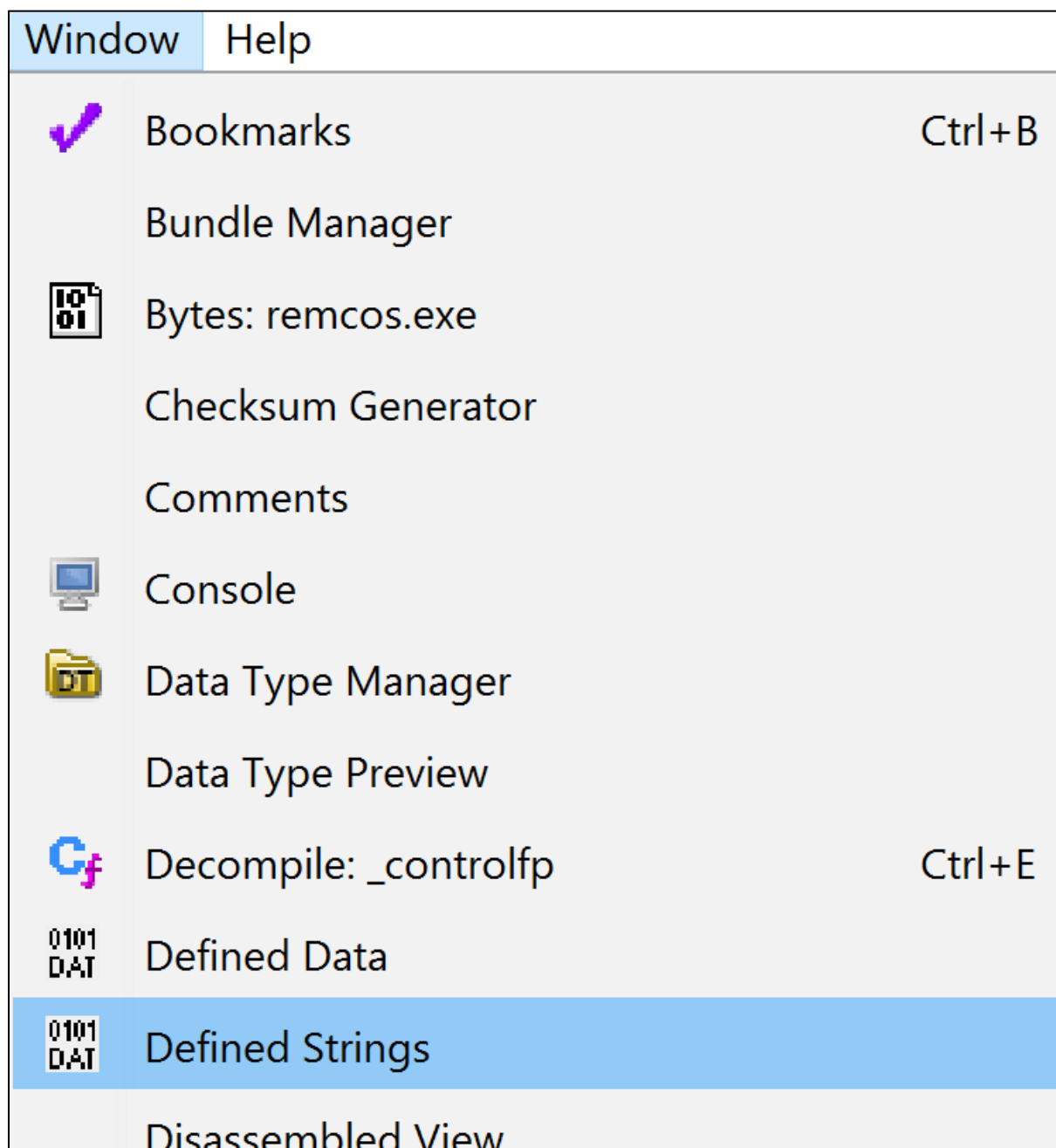
Po przybliżeniu widoku grafu możliwe jest bardziej szczegółowe prześledzenie decyzji podejmowanych przez złośliwe oprogramowanie oraz tego, w jaki sposób przebiega jego wykonanie w zależności od określonych warunków.



Na powyższym zrzucie ekranu wskazano wywołanie funkcji *GetStartupInfoA*, co sugeruje, że badany fragment kodu pobiera informacje o uruchomieniu systemu na zaatakowanej maszynie. Dwukrotne kliknięcie nazwy dowolnej funkcji w grafie powoduje przejście do definicji tej funkcji i aktualizuje widok prezentowany w Ghidrze.

## Wyszukiwanie łańcuchów znaków

W menu **Windows** na pasku narzędzi znajduje się opcja **Defined Strings**, która pozwala wyświetlić listę łańcuchów znaków zawartych w pliku wykonywalnym.



Jest to szczególnie przydatne w przypadku niezapakowanego złośliwego oprogramowania, gdyż często zawiera ono łatwe do odczytania ciągi znaków sugerujące potencjalne zachowanie lub cele programu.



Po wybraniu tej opcji wyświetlane jest dedykowane okno z listą odnalezionych ciągów znaków.

Defined Strings - 590 items			
Location	String Value	String Representation	Data T...
00410c94	\cookies.sqlite	"\cookies.sqlite"	ds
00410ca4	[Firefox Cookies not found]	"\n[Firefox Cookies not found]"	ds
00410cc8	[IE cookies cleared!]	"\n[IE cookies cleared!]"	ds
00410ce0	[IE cookies not found]	"\n[IE cookies not found]"	ds
00410cf8	Software\Microsoft\Windows\CurrentVer...	"Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\User Shell Folde..."	ds
00410d40	Cookies	"Cookies"	ds
00410d4c	[Cleared all cookies & stored logins!]	"\n[Cleared all cookies & stored logins!]\n"	ds
00410d78	getfunlib	"getfunlib"	ds
00410d84	funready	"funready"	ds
00410d90	funfunc	"funfunc"	ds
00410d98	FunFunc	"FunFunc"	ds
00410da0	fundlldata	"fundlldata"	ds
00410dac	Software\Microsoft\Windows\CurrentVer...	"Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer\\Run\\"	ds
00410df0	Userinit	"Userinit"	ds
00410dfc	C:\WINDOWS\system32\userinit.exe,	"C:\\WINDOWS\\system32\\userinit.exe, "	ds
00410e20	Software\Microsoft\Windows NT\Current...	"Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon\\"	ds
00410e58	Shell	"Shell"	ds
00410e60	explorer.exe,	"explorer.exe, "	ds
00410e70	Software\Microsoft\Windows\CurrentVer...	"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\"	ds
00410ea0	exit	"exit \n"	ds
00410ea8	del %0	"del %0 \n"	ds
00410eb4	start ""	"start \"\" \" "	ds
00410ecc	PING 127.0.0.1 -n 2	"PING 127.0.0.1 -n 2 \n"	ds
00410eec	\\install.bat	"\\install.bat"	ds
00410efc	@RD /Q "	"@RD /Q \\"	ds
00410f08	if exist "	"if exist \\"	ds
00410f14	" goto Repeat	"\" goto Repeat\n"	ds
00410f24	del "	"del \\"	ds

Po wybraniu tej opcji wyświetlane jest dedykowane okno z listą odnalezionych ciągów znaków. Na przedstawionym przykładzie można dostrzec, że złośliwe oprogramowanie prawdopodobnie tworzy mechanizm auto-startu, ponieważ odwołuje się do klucza rejestru Software\Microsoft\Windows\CurrentVersion\Run. Taki sposób jest powszechnie wykorzystywany przez malware do zapewnienia sobie automatycznego uruchamiania w systemie. Ponadto wśród łańcuchów znajduje się plik install.bat, co może wskazywać na dodatkowe działania wykonywane na dysku.

Na poniższym zrzucie ekranu dwukrotnie kliknięto łańcuch *install.bat*, co spowodowało aktualizację okna **Listing** i wskazanie miejsca w pliku binarnym, w którym znajduje się ten tekst.

```

*****
* param_2 parameter of basic_string<char,struct_st...
*
*****
s_\\install.bat_00410eec          XREF[1]:  FUN_0040650d:00406750(*)
00410eec 5c 69 6e...  ds      "\\install.bat"
00410ef9 00          ??      00h
00410efa 00          ??      00h
00410efb 00          ??      00h

```

Na poniższym zrzucie ekranu dwukrotnie kliknięto łańcuch *install.bat*, co spowodowało aktualizację okna **Listing** i wskazanie miejsca w pliku binarnym, w którym znajduje się ten tekst. Zauważalna jest również adnotacja *XREF[1]* wraz z nazwą funkcji. Informuje to, że odnaleziono jedno odwołanie (*cross-reference*) do łańcucha *install.bat*, a w nawiasie widnieje nazwa funkcji, w której go użyto. Po dwukrotnym kliknięciu tej nazwy Ghidra przenosi do odpowiedniego miejsca w oknie **Listing** i pokazuje, gdzie dokładnie łańcuch został użyty w kodzie. Załóżmy, że w trakcie analizy ustalono, iż plik *install.bat* jest zapisywany na dysk i uruchamiany w celu dodania klucza *run* do rejestru. W efekcie złośliwe oprogramowanie startuje wraz z systemem. Nazwa funkcji *FUN\_00040560d* nie jest jednak pomocna w zapamiętaniu przeznaczenia tej funkcji, dlatego warto zastąpić ją bardziej opisową etykietą. Aby zmienić nazwę funkcji, należy kliknąć ją prawym przyciskiem myszy i wybrać **Edit Function**.

Instruction Info...	
Patch Instruction	Ctrl+Shift+G
Processor Manual...	
Processor Options...	
Edit Function...	F
Function	
Set Data Type	
Program Selection	

W efekcie pojawia się okno dialogowe umożliwiające edycję nazwy funkcji:

undefined **FUN\_0040650d** (char param\_1, char \* param\_2, char \* param\_3, char param\_4, char param\_5)

Function Name:

Calling Convention:

Function Attributes:

- ☐ Varargs
- ☐ In Line
- ☐ No Return
- ☐ Use Custom Storage

Function Variables

Index	Datatype	Name	Storage
	undefined	<RETURN>	AL:1
1	char	param_1	Stack[0x4]:1
2	char *	param_2	Stack[0x8]:4
3	char *	param_3	Stack[0xc]:4
4	char	param_4	Stack[0x10]:1
5	char	param_5	Stack[0x14]:1

Call Fixup:

OK Cancel

W polu **Function Name** można wprowadzić nową, bardziej zrozumiałą nazwę:

undefined **persistence\_install.bat** (char param\_1, char \* param\_2, char \* param\_3, char param\_4, char param\_5)

Function Name:

Calling Convention:

Function Attributes:

- ☐ Varargs
- ☐ In Line
- ☐ No Return
- ☐ Use Custom Storage

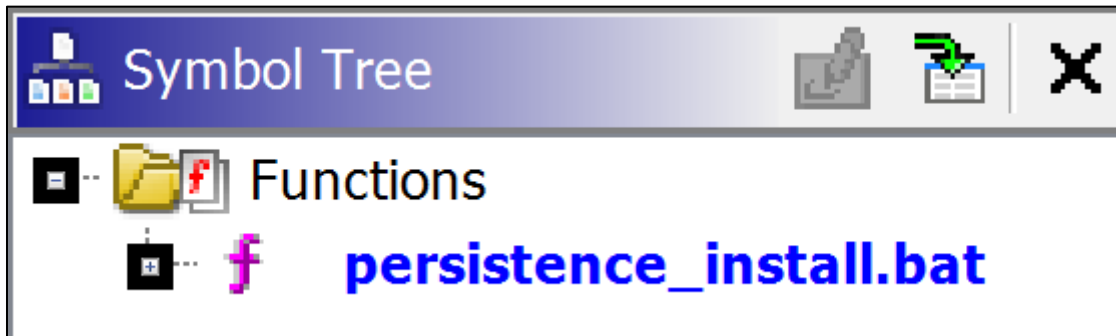
Function Variables

Index	Datatype	Name	Storage
	undefined	<RETURN>	AL:1
1	char	param_1	Stack[0x4]:1
2	char *	param_2	Stack[0x8]:4
3	char *	param_3	Stack[0xc]:4
4	char	param_4	Stack[0x10]:1
5	char	param_5	Stack[0x14]:1

Call Fixup:

OK Cancel

Zmodyfikowana nazwa jest następnie wyświetlana w całym projekcie w Ghidrze, co ułatwia orientację w kolejnych etapach analizy. Przykład takiej zmiany widać w sekcji **Symbol Tree**, gdzie dotychczasową nazwę zastąpiono nową. W ten sposób praca z projektem staje się łatwiejsza i bardziej intuicyjna podczas dalszej analizy.



Ghidra omówiona na wideo: <https://ghidra-sre.org/GhidraGettingStartedVideo/GhidraGettingStartedVideo.mp4>

Ghidra cheatsheet:

<https://htmlpreview.github.io/?https://github.com/NationalSecurityAgency/ghidra/blob/stable/GhidraDocs/CheatSheet.html>

Dodatkowe źródła odnośnie Ghidry i Reverse Engineering:

[https://static.grumpycoder.net/pixel/docs/GhidraClass/Beginner/Introduction\\_to\\_Ghidra\\_Student\\_Guide\\_withNotes.html#Introduction\\_to\\_Ghidra\\_Student\\_Guide.html](https://static.grumpycoder.net/pixel/docs/GhidraClass/Beginner/Introduction_to_Ghidra_Student_Guide_withNotes.html#Introduction_to_Ghidra_Student_Guide.html)

<https://github.com/AllsafeCyberSecurity/awesome-ghidra>

<https://www.youtube.com/watch?v=UUFoxZpxKhg&list=PL7iSco3duZcrs-SgnOXaX9qLyB97tnYLO&index=2>

<https://hackaday.io/course/172292-introduction-to-reverse-engineering-with-ghidra>

<https://beginners.re/RE4B-PL.pdf>

## Zadanie 1 – Zapoznanie z programem Ghidra

### 1. Instalacja i przygotowanie środowiska:

- Pobierz i zainstaluj Ghidrę z oficjalnej strony (lub repozytorium GitHub).
- Uruchom Ghidrę, a następnie utwórz nowy projekt (Non-Shared lub Shared – w zależności od potrzeb).

### 2. Import pliku wykonywalnego (próbki):

- Przygotuj przykładowy plik wykonywalny (może to być niewielka, niegroźna próbka złośliwego oprogramowania bądź testowy plik .exe).
- Zaimportuj plik do nowo utworzonego projektu w Ghidrze.
- Sprawdź komunikaty w oknie informacyjnym Ghidry dotyczące importu (np. rozpoznany format PE, architektura 32-/64-bitowa itp.).

### 3. Wstępna konfiguracja analizy:

- Przy pierwszym otwieraniu pliku wykonywalnego wybierz odpowiednie ustawienia analizy, zwracając uwagę na moduły sugerowane przez Ghidrę (np. *WindowsPE x86 Propagate External Parameters*).
- Uruchom proces analizy i obserwuj pasek postępu. Zanotuj, ile czasu trwa analiza oraz czy pojawiają się dodatkowe komunikaty.

### 4. Przegląd struktury pliku w Ghidrze:

- Obejrzyj okno **Program Trees** i sprawdź, jak podzielona jest struktura pliku (sekcje, segmenty).
- Zwróć uwagę na **Symbol Tree** – przejrzyj listę **Imports**, **Exports** oraz **Functions**:
  - Zidentyfikuj funkcje importowane z bibliotek systemowych (np. kernel32.dll, user32.dll).
  - Sprawdź, czy plik ma jakieś funkcje eksportowane (jeżeli tak, spróbuj je zinterpretować).

### 5. Analiza punktu wejścia (entry point):

- Przejdź do punktu wejścia programu (pozycja *entry* w **Symbol Tree**).
- W oknie **Listing** prześledź instrukcje asemblera wywoływane na początku działania programu.
- W oknie **Decompile** zapoznaj się z próbą dekompilacji kodu do postaci języka C. Oceń, czy Ghidra zidentyfikowała jakieś konkretne funkcje/zmienne.

## 6. Wyszukiwanie łańcuchów znaków (strings):

- Użyj opcji **Windows** → **Defined Strings** w Ghidrze, aby wyświetlić listę łańcuchów znaków.
- Wybierz kilka ciekawych lub podejrzanych wpisów (np. ścieżki plików, klucze rejestru, nazwy funkcji API) i przejdź do nich w oknie **Listing**.
- Sprawdź, w jakich funkcjach pojawiają się te łańcuchy (tzw. *cross-references*, XREF). Zapisz wnioski lub hipotezy dotyczące sposobu ich wykorzystania przez analizowany program.

## 7. Modyfikacja nazw funkcji:

- Znajdź w **Symbol Tree** funkcje o nazwach zaczynających się od *FUN\_* lub inne niezidentyfikowane (np. *sub\_* w niektórych przypadkach).
- Za pomocą opcji **Edit Function** zmień nazwy tych funkcji na bardziej opisowe (np. *CheckPersistence*, *InstallBatch*, *InitNetwork*), jeśli na podstawie kodu dekompilowanego domyślasz się ich roli.
- Zwróć uwagę na to, jak zmiana nazwy wpływa na **Symbol Tree** i **Listing**.

## 8. Analiza grafu funkcji (Function Graph):

- Wybierz jedną z interesujących funkcji i użyj przycisku **Display Function Graph**, aby wyświetlić graficzny przeptyw jej działania.
- Przybliż i zorientuj się w blokach podstawowych (basic blocks) oraz ścieżkach przepływu programu (warunki skoków, wywołania funkcji).
- Zidentyfikuj wywołania funkcji systemowych w danej ścieżce i spróbuj ustalić, co dzieje się z danymi wejściowymi/wyjściowymi.

## 9. Zgodnie z tutorialiem na YouTube rozwiąż pierwsze przykładowe zadanie (tego kroku nie trzeba zamieszczać w sprawozdaniu)

Video Tutorial i zadanie na start:

<https://www.youtube.com/watch?v=fTGTrnjuGA>

<https://crackmes.one/crackme/5b8a37a433c5d45fc286ad83> - plik znajduje się w repozytorium github pod nazwą rev50\_linux64-bit

## **Zadanie 2**

### **Zadanie 2**

W repozytorium GitHub dostępnym pod adresem <https://github.com/Sptimus/Low-level-programming/tree/main/Lab%203> znajduje się dziesięć programów. Należy przeprowadzić ich inżynierię wsteczną, korzystając z dowolnych narzędzi, na przykład Ghidra czy radare2 lub inne.

**Po wykonaniu zarówno zadania 1, jak i zadania 2, przygotuj sprawozdanie, w którym krok po kroku zaprezentujesz przebieg prac oraz uzyskane wyniki.**