

<b>Document Code &amp; Name</b>	Cutter Tool – Kali Linux Purple
<b>Issue Date</b>	2023/17/06
<b>Division Applied</b>	Cyber Security Services (CSS)
<b>Author(s)</b>	Porojnicu Constantin
<b>Owner(s)</b>	Porojnicu Constantin
<b>Last Modified</b>	2023/19/06
<b>Next Revision</b>	2023/20/06

Description .....	2
How to use cutter tool in Kali Linux .....	2
Dashboard .....	4
STRING.....	9
Imports .....	10
Dissassembly .....	10
Hexdump .....	13

## Description

In this article, you will find a detailed procedure for analyzing files using the Cutter tool. From opening the tool to interpreting the data, you'll be guided through the essential steps and techniques used to examine and extract information from files. Discover how to apply data mining and analysis, identifying patterns and gaining valuable insights from the information captured with Cutter.

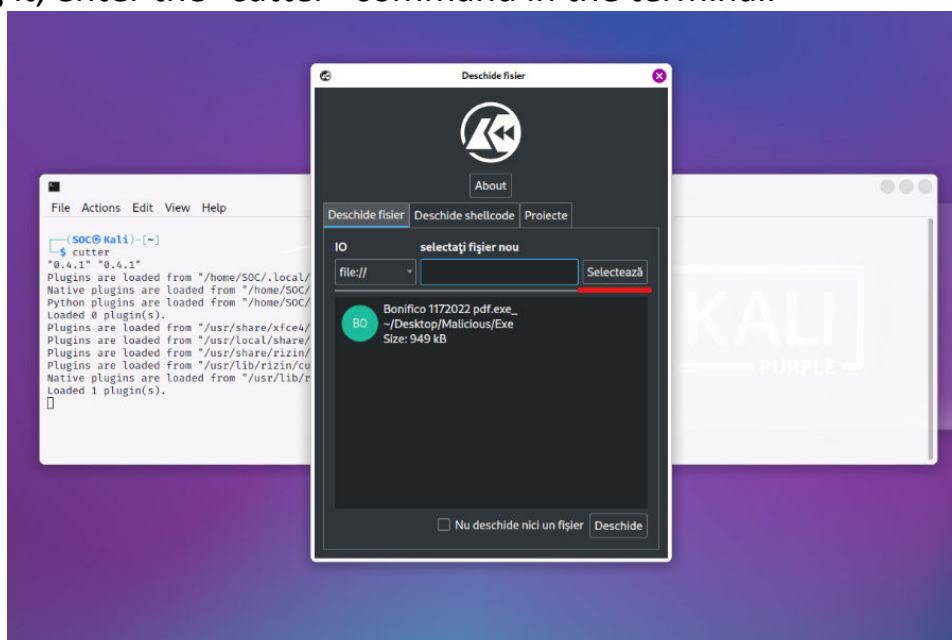
## How to use cutter tool in Kali Linux

Open Kali Linux Purple Terminal

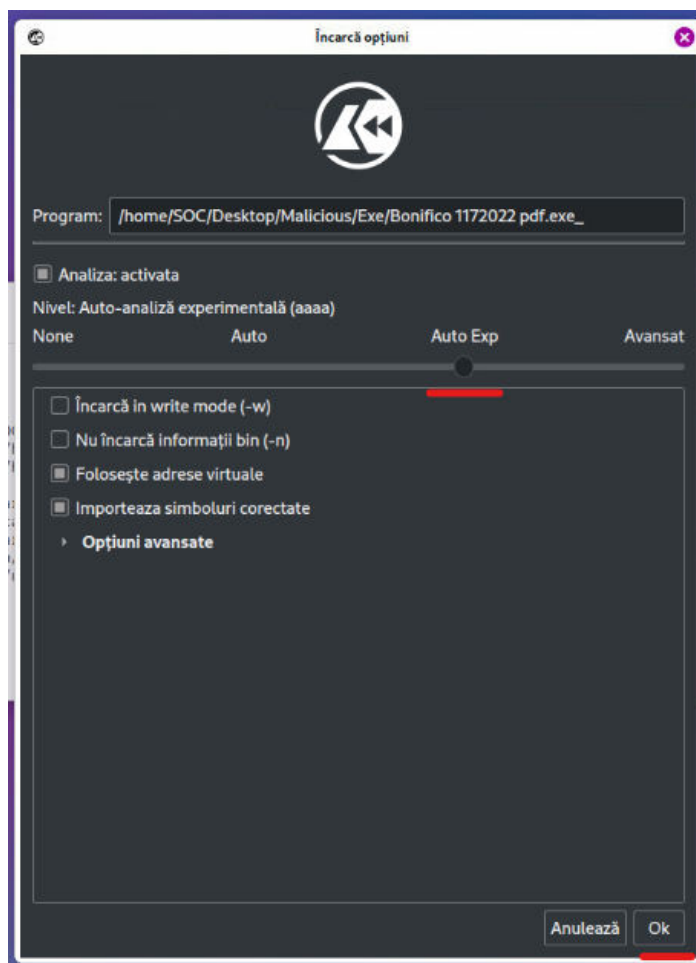
In the terminal execute the following command

```
sudo apt install cutter
```

After installing it, enter the "*cutter*" command in the terminal.



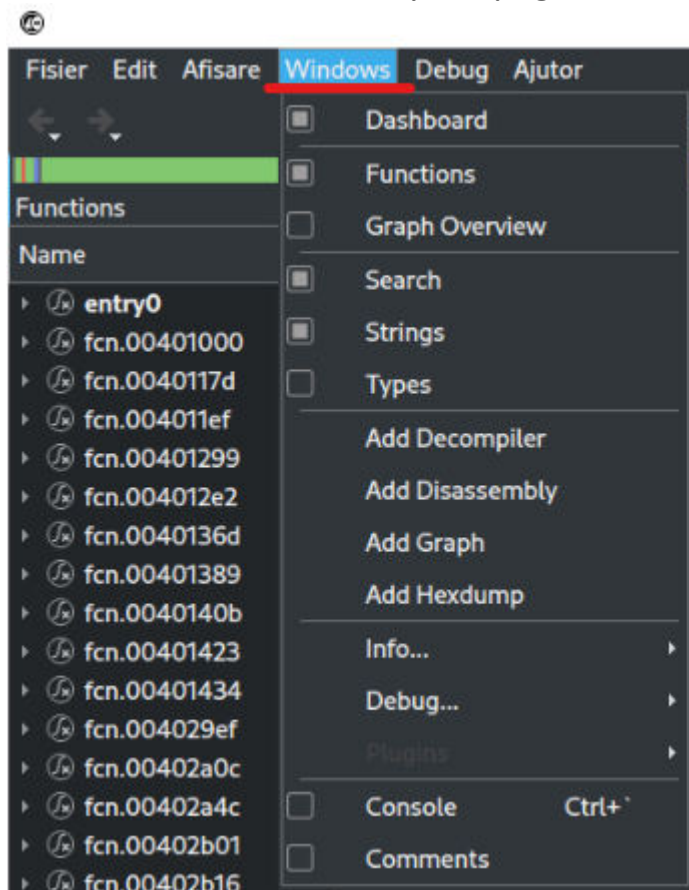
After starting the program, select "Select" and enter the executable file we want to analyze.



After selecting the file, we have a page for selecting the depth of the analysis. Select the analysis level to : Auto Exp

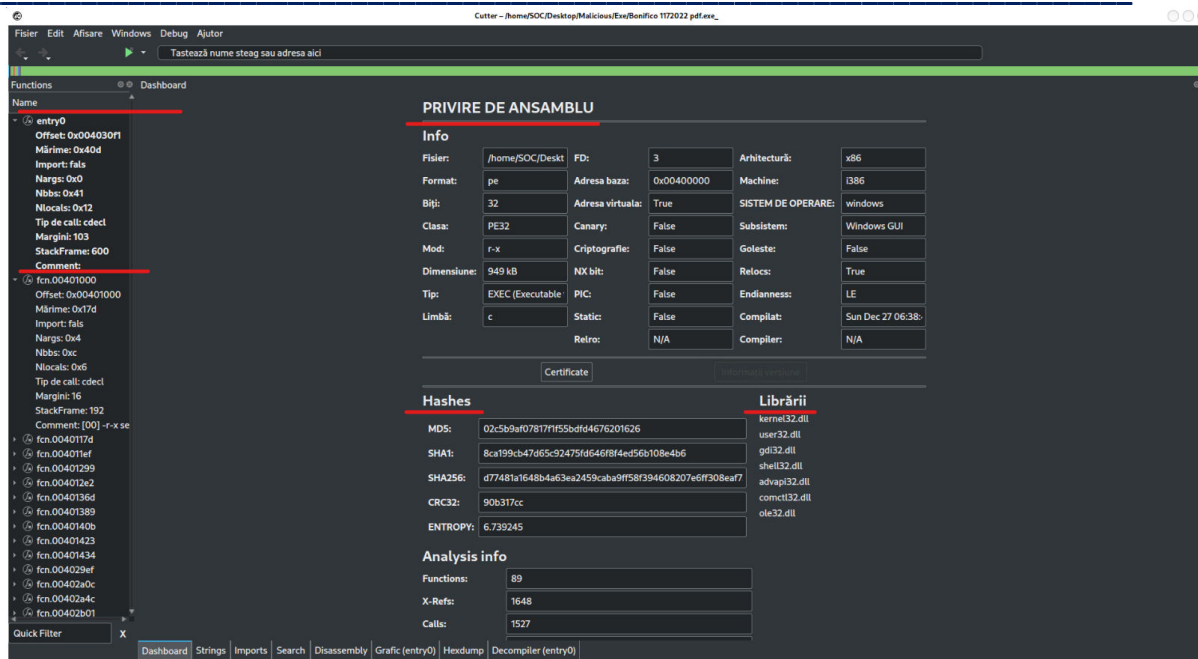
After selecting all the filters, we will go through the following subpages to obtain as much data as possible and find a concert verdict on the file:  
Dashboard, String, Imports, Disassembly, Hexdump

If you do not find the Dashboard page or other pages in the bottom of the tool, you can open the Windows window to add the necessary subpages



## Dashboard

In the Dashboard subpage we find the following information that will help us analyze the file and mark it as malicious or not.



Hashes		Librării
MD5:	02c5b9af07817f1f55bdfd4676201626	kernel32.dll user32.dll
SHA1:	8ca199cb47d65c92475fd646f8f4ed56b108e4b6	gdi32.dll shell32.dll
SHA256:	d77481a1648b4a63ea2459caba9ff58f394608207e6ff308eaf7	advapi32.dll comctl32.dll
CRC32:	90b317cc	ole32.dll
ENTROPY:	6.739245	
Analysis info		
Functions:	89	
X-Refs:	1648	
Calls:	1527	
Strings:	212	
Symbols:	153	
Imports:	153	
Analysis coverage:	22930 bytes	
Code size:	24576 bytes	
Coverage percent:	93%	

In the dashboard of the "cutter" tool in Kali Linux, the "Name" column displays the names of the functions identified in the analyzed binary code. These names are the

result of the parsing and decompiler used by "cutter" to translate the binary code into a more understandable form.

Examples of function names that may appear in the "Name" column include:

entry0: This is a special function that represents the entry point of the program. In most cases, this is the starting address of the parsed executable.

fcn.00401000: This is an example of a function name generated by "cutter" based on the function address in the binary code. In this example, "fcn" indicates that it is a function, and "00401000" is the address of that function in hexadecimal.

#### EXAMPLE :

Information	Malicious Value	Clean Value
Offset	0x00401023	0x004010A7
Size	256 bytes	128 bytes
Imports	kernel32.dll, user32.dll	user32.dll, gdi32.dll
Nargs	5	2
Nbbs	10	5
Nlocals	20	10
Call Type	Indirect call to dynamic address	Direct call
Boundaries	0x00401023 - 0x00401234	0x004010A7 - 0x00401127
Stackframe	Size: 512 bytes, Address: 0x7FFD	Size: 256 bytes, Address: 0x7FFD
Comment	Possible backdoor functionality	Message display function

- "nargs" probably refers to the number of arguments or parameters of a function. In your case, it looks like there are two different functions, each with a specific number of arguments: 5 arguments for the first function and 2 arguments for the second function.
- "nbbs" could mean the number of basic blocks. A basic block is a sequence of consecutive instructions that does not contain any branch or unconditional jump.
- "nlocals" probably refers to the number of local variables used within a function. Local variables are variables defined and used exclusively inside that function.
- "call type" refers to a - direct call to a function. This means that the function call address is known during compilation or disassembly of the code.

In the "Overview" tab we can extract the following data to continue with the analysis  
Bits, Size, Cryptography, Base Address

We can compare the size offered by the cutter with the size of an official program.

Example:

The Office Suite downloaded from a place other than the official source may have a different size compared to the Suite offered by Microsoft, and the size difference between them may raise suspicions.

## Hashes - we can see each Hash of the file CRC32 and Entropy

- **CRC32 :**

To use the CRC32 value in evaluating the maliciousness of a program, you can follow these steps:

Calculate the CRC32 value for the original program: Use the CRC32 functionality of "cutter" to calculate the CRC32 checksum of the original program.

Save CRC32 value: Write down the resulting CRC32 value for the original program.

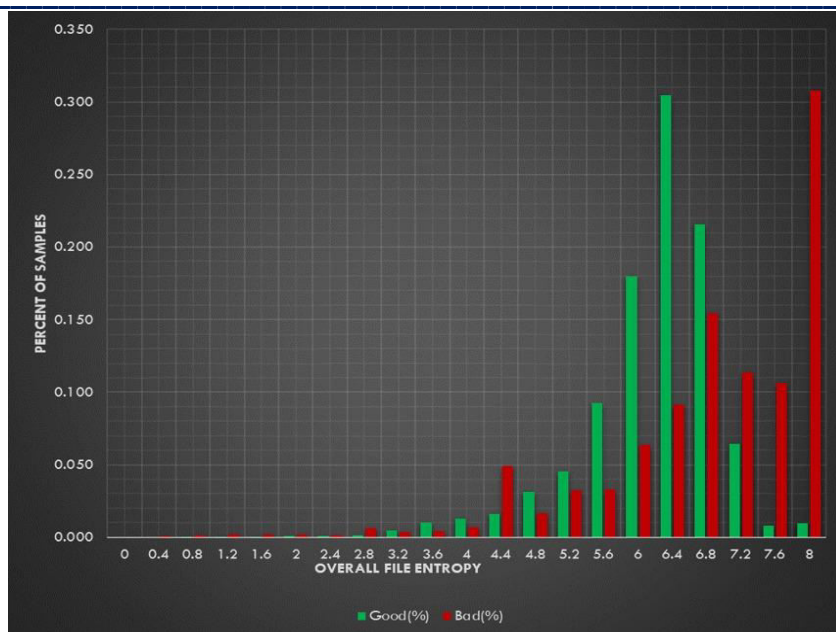
Check the CRC32 value of the current program: Recalculates the CRC32 value for the current or suspected malicious program.

Compare CRC32 value: Compares the CRC32 value of the current program with the CRC32 value of the original program. If these values are identical, there is a higher probability that the program has not been modified in the meantime.

- **Entropy :**

Entropy refers to the measure of uncertainty or complexity of the information contained in a file or area of binary code. Entropy is used to assess the degree of randomness or structure in data.

Entropy is measured on a scale from 0 to 8 :



A few things stand out in this graph:

- Legitimate files tend to have an entropy between 4.8 and 7.2.
- Files with an entropy above 7.2 tend to be malicious.
- Nearly 30% of all of the malicious samples have an entropy near 8.0 while only 1% of legitimate samples have an entropy near 8.0.
- Approximately 55% of all malicious samples have a entropy of 7.2 or more versus 8% of legitimate samples.

## Library

- The next subsection in Cutter - Dashboard is Library, where we can identify which libraries were used to create the program, from here we can identify malicious libraries or combinations of libraries that can have malicious outputs.

## Analysis Info

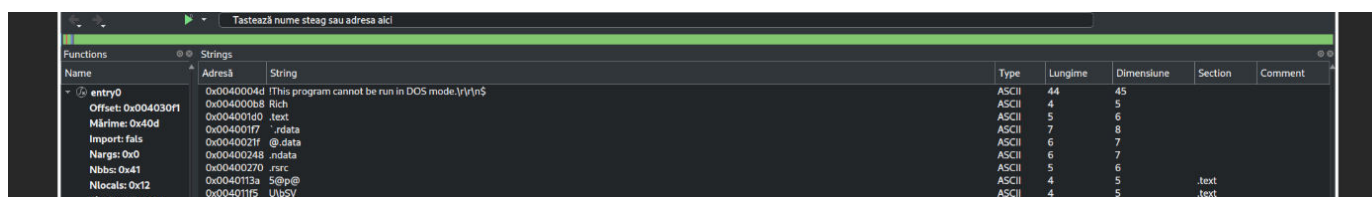
- **Functions:** Sections of code that perform specific tasks within a program. "cutter" identifies and provides details about functions, including names and addresses, aiding in understanding program logic and behavior.
- **Imports:** External libraries or modules used by the program. "cutter" displays imported libraries and their functions, helping understand external dependencies and interactions.
- **Symbols:** Named entities like variables or labels within the code. "cutter" identifies symbols, providing names, addresses, and data types, aiding in understanding data structures and program organization.
- **Calls:** Invocations or references to functions/subroutines in the code. "cutter" shows function calls, including source/destination addresses, helping analyze control flow and inter-function interactions.



- Coverage percent is the coverage percentage of the code analysis from the cutter side (In the beginning I selected Auto Exp for a higher percentage)

## STRING

To continue the analysis, we will go from the Dashboard page to the String page where we can analyze the data provided by the string with templates provided by the security vendors to see if there is any similarity.



Name	Address	String	Type	Length	Dimensions	Section	Comment
entry0	0x0040004d	This program cannot be run in DOS mode.\r\n	ASCII	44	45		
Offset: 0x004000b8	0x004000b8	Rich	ASCII	4	5		
Offset: 0x004001d0	0x004001d0	.text	ASCII	5	6		
Offset: 0x004001f7	0x004001f7	.rdata	ASCII	7	8		
Offset: 0x0040021f	0x0040021f	@.data	ASCII	6	7		
Offset: 0x00400248	0x00400248	.idata	ASCII	6	7		
Offset: 0x00400270	0x00400270	.rsrc	ASCII	5	6		
Offset: 0x0040113a	0x0040113a	5@p@	ASCII	4	5	.text	
Offset: 0x004011f5	0x004011f5	UlpSV	ASCII	4	5	.text	

**Address:** The address of the string indicates its location within the binary code. It can help us identify areas of interest or find connections between different parts of the code.

**String:** The actual contents of the string may reveal messages, memory addresses, URLs, or other important information. It helps us understand the functionality and behavior of the program.

**Type:** The type of the string (ASCII, Unicode, etc.) can provide clues about how it is used within the program. It may help us identify obfuscation or encryption techniques used.

**Length:** The length of the string can give us clues about its complexity and importance. Long or unusual strings may indicate the presence of malicious code or encrypted information.

**Size:** Size is the number of bytes occupied by the string in memory. It can help us understand the program's resource consumption or identify anomalies.

**Section:** Indicates the section of the binary file where the string is located (eg ".text", ".data", ".rodata", etc.). It helps us understand the organization and structure of the program.

**Comment:** Comments may contain additional information added by the user, such as observations or specific interpretations of the character string. They can help us better contextualize and interpret information.

## Imports

**Address** - indicates the memory address where the import is located - hexadecimal value - useful in identifying specific addresses

**Type** - Indicates if it is Function OR Data - Functions represent functions from other libraries while data are variables or constants imported from other libraries

**Library Name:** Indicates the external library - it can provide clues about the external functionalities used in the program

**Safety** - Marks when a tab is unsafe. We can analyze it by sending the respective function to Disassembly and analyzed separately.

**Inventory:** Description or specific details related to the import

## Dissassembly

In the disassembly section of Cutter, you will see the machine code of the opened binary file.

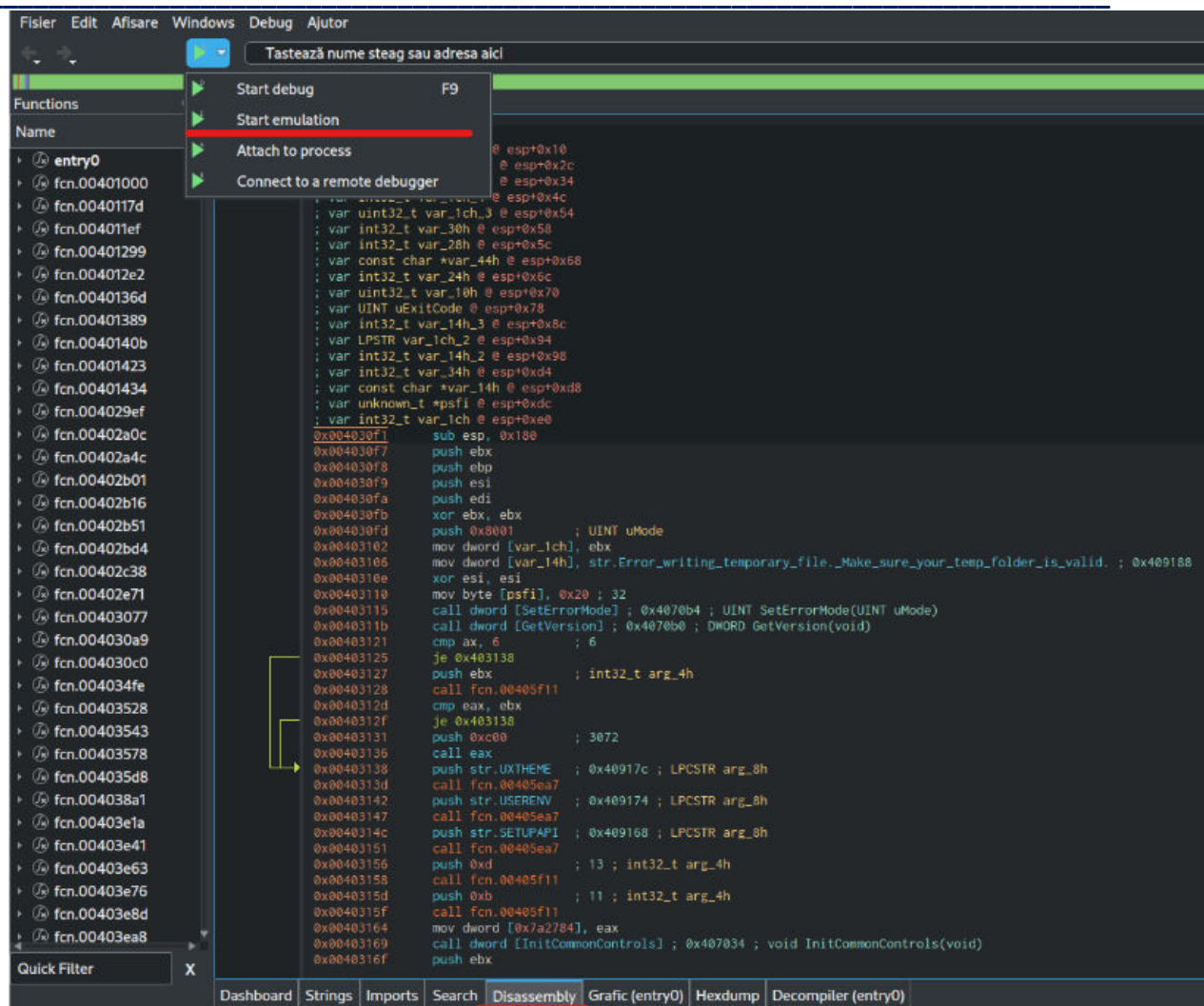
Here we have some important aspects that you can observe and inspect in the disassembly section of Cutter:

- Instructions: You can observe individual program instructions, such as loading and storing data, arithmetic, flow control, and function calls. These statements can be analyzed to understand how the program behaves and how data is manipulated.

- 
- Addresses: Each instruction has an associated address in memory. These addresses can be useful in tracing the flow of execution and identifying dependencies between different parts of the code.
  - Branches and Jumps: Disassembly allows the identification of decision points in the code, such as branching and jumping instructions. These can be analyzed to understand the logical structure of the program and to identify possible execution paths.

To help us with the tool as much as possible, at the top we have the Start emulation section

The "Start Emulation" section in Cutter is a facility that allows you to emulate and run the binary file in a controlled and simulated environment. This means that the program will not be run directly on the host operating system, but in an isolated environment within Cutter.



After running the emulation, we can see how Cutter underlines portions of code in red. There are several non-malicious reasons why code may be highlighted in red during emulation, such as:

- Programming errors
- Errors in the analyzed binary code
- Incompatibilities or configuration problems

However, in certain cases, **red underlining** of code during emulation can also indicate malicious or potentially dangerous behavior. In such situations, it is important to deeply investigate the causes of errors and analyze the code to determine if there is any malicious intent in the implementation.

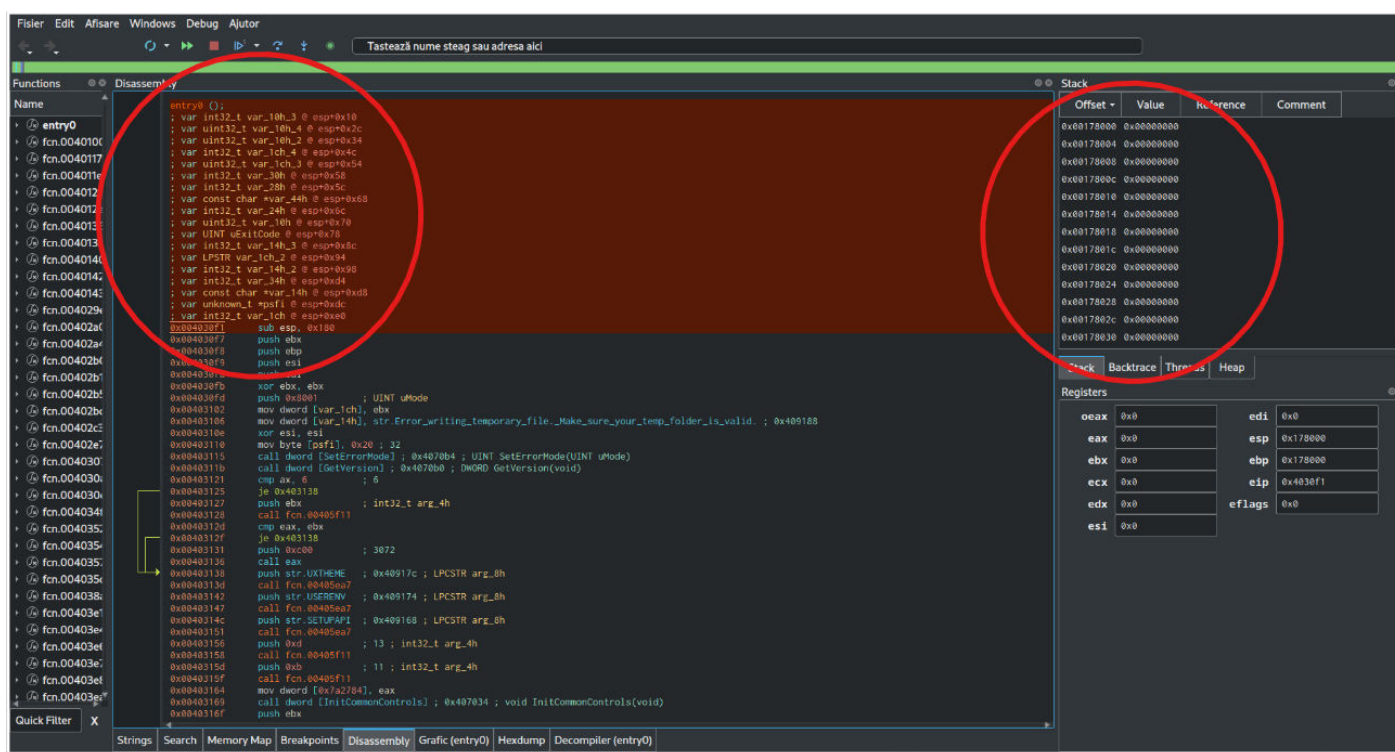
On the right side we have the two columns that indicate the location and value of this piece of code. Executable code may not be present in this area, but other components of the binary file, such as headers, symbol tables, data sections, or other structural information.

The base address 0x00000000 may contain some binary file metadata or information necessary for the program to function properly but is not always used for executable code.

However, it is possible for a binary to use the base address 0x00000000 maliciously in certain scenarios.

**Example:** An attacker may attempt to exploit a known vulnerability to override or redirect the flow of execution to this base address in an attempt to gain control of the program or execute malicious code.

Examples of vulnerabilities: Buffer Overflow, SQL Injection, Cross-Site Scripting (XSS)



## Hexdump

Hexdump is useful for inspecting and analyzing the binary contents of a file, including identifying specific data patterns or structures, searching for specific texts or values, detecting unauthorized changes, and more. By allowing you to see both the hexadecimal and the textual representation of the contents of the binary file, hexdump makes it easier to analyze and understand the contents in a more accessible and interpretable way.

These data can be analyzed using the templates inserted in the cutter and observing if there is any match between the template and the analyzed code.

