Escuela Colombiana de Ingeniería Julio Garavito

Demo Company
IT Security and Privacy

Reverse Engineering Report

Students:

Laura Sofia Gil Chaves

Camilo Castaño Quintanilla

Teacher:

Ing. Daniel Vela

Business Confidential

September 25th, 2024

Project 01-11

Version 1.0

Table of Contests

Assessment Overview

From Thursday, September 19 to Monday, September 23, a vulnerability analysis of the programs Bricked Up and Vasa's Very Easy Crackme was carried out through the windows system, downloading the Crackme web page file. Bricked Up and Vasa's Very Easy Crackme was performed through the windows system, by downloading the file from the Crackme webpage from discovery to accessing the shell.

1.Planning: Identification of the necessary steps to identify the vulnerabilities of the shell documents and exploit them.

2.Discovery: Command lines where we can route the path to the solution.

3.Attacking: Changing the file path to be able to enter the different shells.

4.Reporting:Documentation of vulnerability analysis and possible mitigation options.

Plan → Discovery → Attack →Report

Assessment Components

External Penetration Test

During external penetration testing on a Windows system, a shell program was downloaded and executed to gain initial access. Using advanced tools such as x64dbg and IDA, vulnerabilities in the system code were identified and key command lines were traced that allowed the path to be modified. This facilitated the execution of a reverse shell. Thanks to these modifications, the shell provided full access to the system, completely compromising the security of the Windows environment. This process revealed vulnerabilities that were successfully exploited to take unauthorized control of the system.

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	The vulnerability allowed full access to the system using a reverse shell and code modifications, compromising the security of the Windows environment. This was evidenced during vulnerability analysis of the Bricked Up and Vasa's Very Easy Crackme programs, where a discovery was made that led to shell access.
High	7.0-8.9	Vulnerabilities that allowed modifying paths and executing commands but required advanced knowledge and tools such as x64dbg and IDA. This analysis was carried out by downloading the file from the Crackme website.
High	7.0-8.9	Vulnerabilities were found in the system's security configuration that could be exploited to gain unauthorized access to critical information.
High	7.0-8.9	Identification of input validation flaws that allowed command injections and access to sensitive data.
Medium	4.0-6.9	Vulnerabilities that could be exploited for limited access, but without completely compromising the system.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Scope

Assessment	Details
External Penetration Test	crackmes.one

Scope Exclusions

No scope exclusions will be established in relation to cracking, as all testing will be conducted in a secure and controlled laboratory environment. The primary focus will be on practical cracking techniques.

Client Allowances

The permissions required to perform the tests will be provided by the crackme.one page, which will allow full access to all necessary files and resources.

Executive Summary

During external penetration testing on a Windows system, a process was carried out to identify and exploit security vulnerabilities. A shell program was downloaded and executed, allowing initial access to the system. Using advanced tools such as x64dbg and IDA, weaknesses in the system code were detected and key commands were identified that facilitated the modification of access paths. This allowed a reverse shell to be executed, granting full access to the system and compromising its security. As a result, vulnerabilities were discovered and exploited that allowed unauthorized control of the system.

Attack Summary Bricked-Up

The following table describes how we gained internal network access, step by step:

Step	Action	Recommendation
1	The Bricked-Up program was chosen from the crackmes.one site, written in C/C++.	Continue to use programs from the same source for controlled cracking practices.
2	The program prompts for a password and returns “Not ok!” if incorrect, terminating execution.	Identify and document the program's responses to different inputs for future reference.
3	Used x32dbg to analyze the program and search for string references.	Learn to use other debugging tools to expand the software analysis skill set.
4	The program flow is modified to allow login regardless of the password provided.	Consider implementing protection techniques in the software to avoid similar vulnerabilities.
5	Verify that, after modification, the program allows access to the system regardless of the password entered.	Perform additional tests to ensure that there are no other vulnerabilities that compromise security.

Attack Summary Vasa's Very Easy Crackme

The following table describes how we gained internal network access, step by step:

Step	Action	Recommendation
1	The Vasa's Very Easy Crackme program was chosen from the crackmes.one site, written in C/C++.	Continue to use programs from the same source for controlled cracking practices.
2	Run the program and enter an incorrect user and password to see the error message "No, that's not it. Try again" .	Log all error messages for analysis and understanding of the program flow.
3	Use IDA to disassemble the code and understand the program flow, looking for the relevant sections.	Become familiar with IDA to identify functions and critical points in the code.
4	Switch to x64dbg to analyze the program and locate the memory address that generates the error message.	Learn how to use x64dbg for effective debugging and observe code behavior.
5	Identify and analyze code breaks that can be exploited to bypass access verification.	Search for flow control patterns that can facilitate the bypassing of protections.
6	Modify the code in the identified address and save the changes to the file.	Document each modification for future reference and analysis.
7	Run the modified program to verify that unrestricted access has been obtained.	Confirm that the changes have been successful

Security Weaknesses

Inadequate Validation Protections

The lack of robustness in protections allows attackers to easily bypass these checks by simply trying different combinations or analyzing the program flow. Implementing more stringent and complex validation measures could help prevent unauthorized access.

Revealing Error Codes

Error messages that programs return in response to incorrect input are often explicit, providing valuable clues about the program's internal logic, making it easier to identify vulnerabilities. A best practice would be to provide generic error messages that do not divulge sensitive information about the state of the program.

Lack of Code Encryption

The absence of encryption techniques in the source code makes it easier to analyze. This allows reverse engineering to be performed more easily, identifying and modifying critical sections of the program without difficulty. Implementing more encryption techniques and transforming the code into a less readable format can help protect the program logic and make unauthorized access more difficult.

External Penetration Test Findings

External Penetration Test Findings

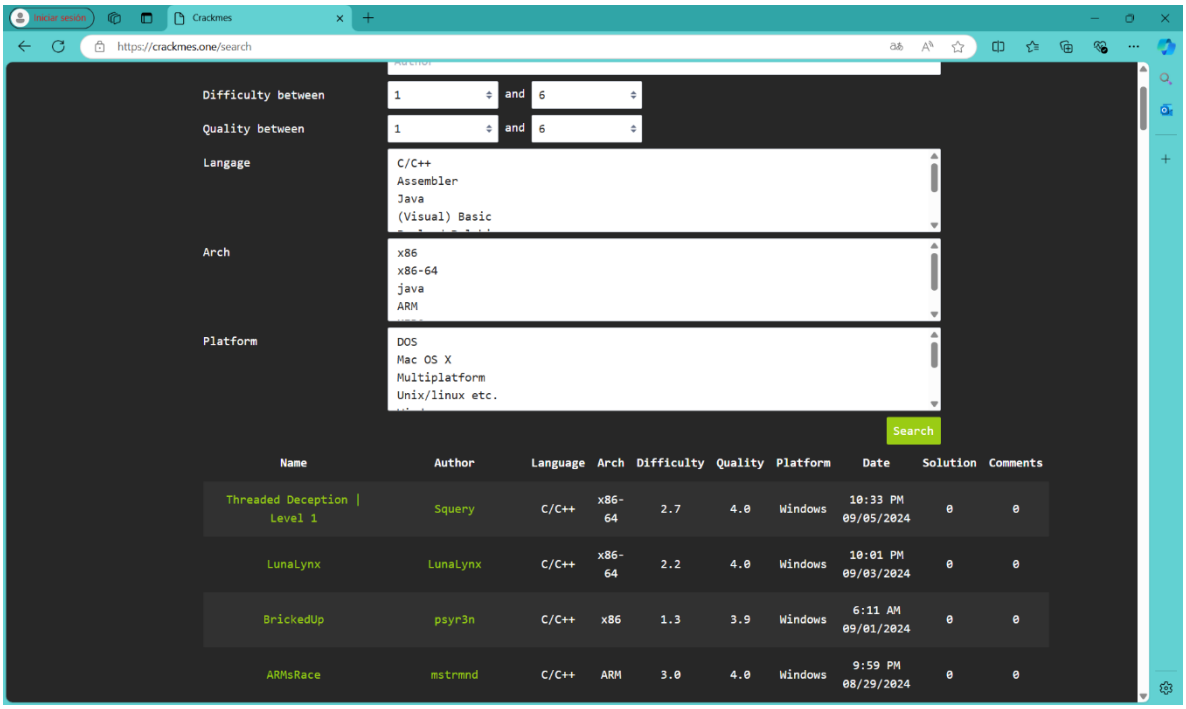
Description:	An analysis of Vasa's Very Easy Crackme and Bricked up program, which requests login credentials, was performed. After observing the error message, x64dbg and/or IDA were used to identify and modify the code at a specific address, avoiding the access verification. Finally, the modified program was executed, obtaining successful access.
Impact:	Critical
System:	Windows
References:	Bricked-Up Vasa's very easy Crackme

Exploit Proof of Concept Bricked-Up

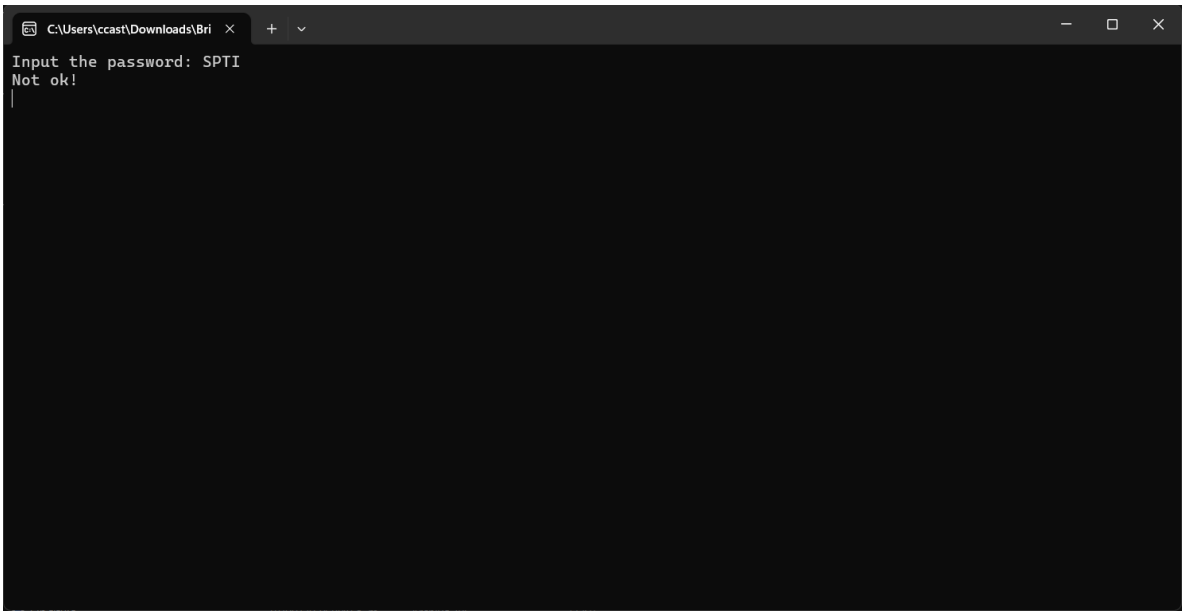
Crackmes

The Proof-of-Concept that follows highlights our ability to carry out remote code execution and take over the target system by exploiting the windows programs.

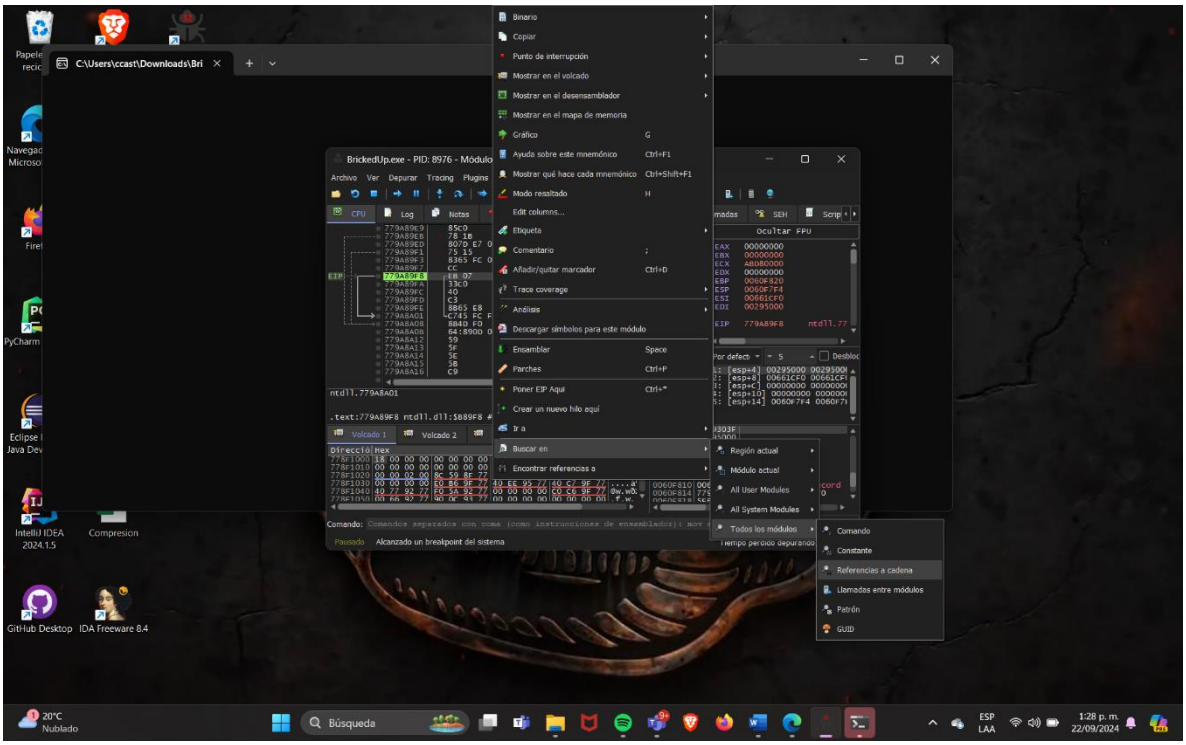
By means of an analysis of the Bricked-Up program, a Crackme designed for Windows and written in C/C++. The objective is to explore the vulnerabilities of this program by manipulating its authentication logic.



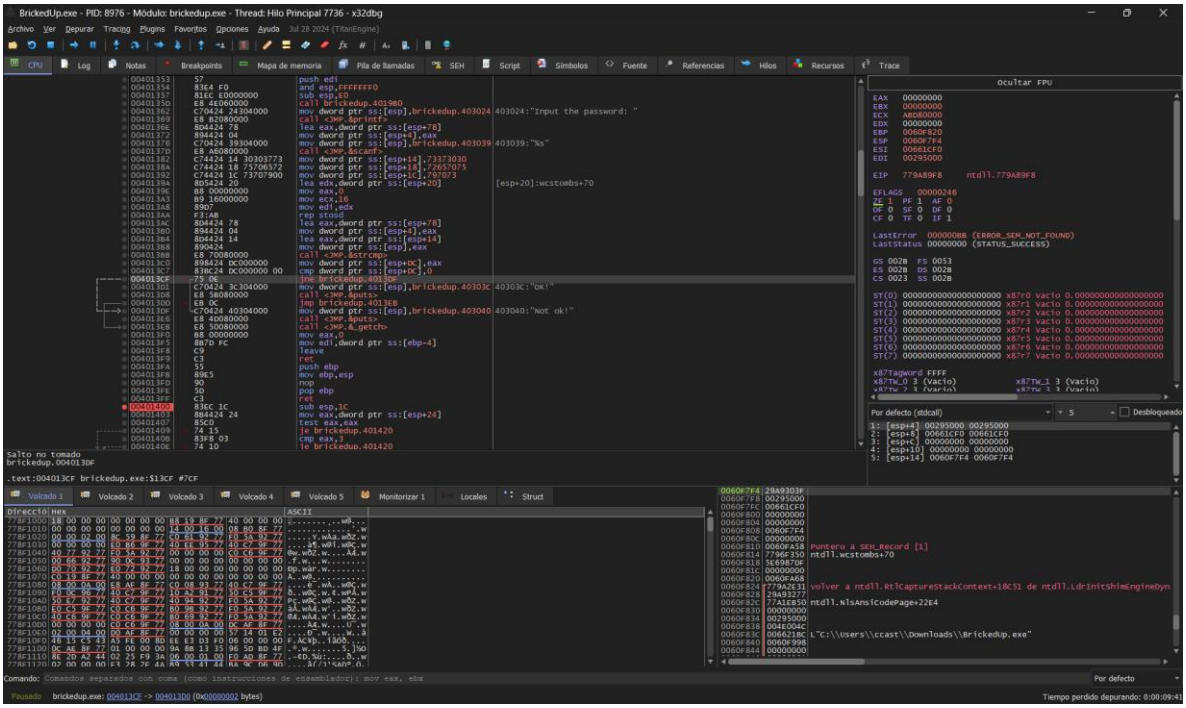
When starting Bricked Up, the user is prompted for a password, and if the password is incorrect, the message “Not ok!” is displayed and the program closes.



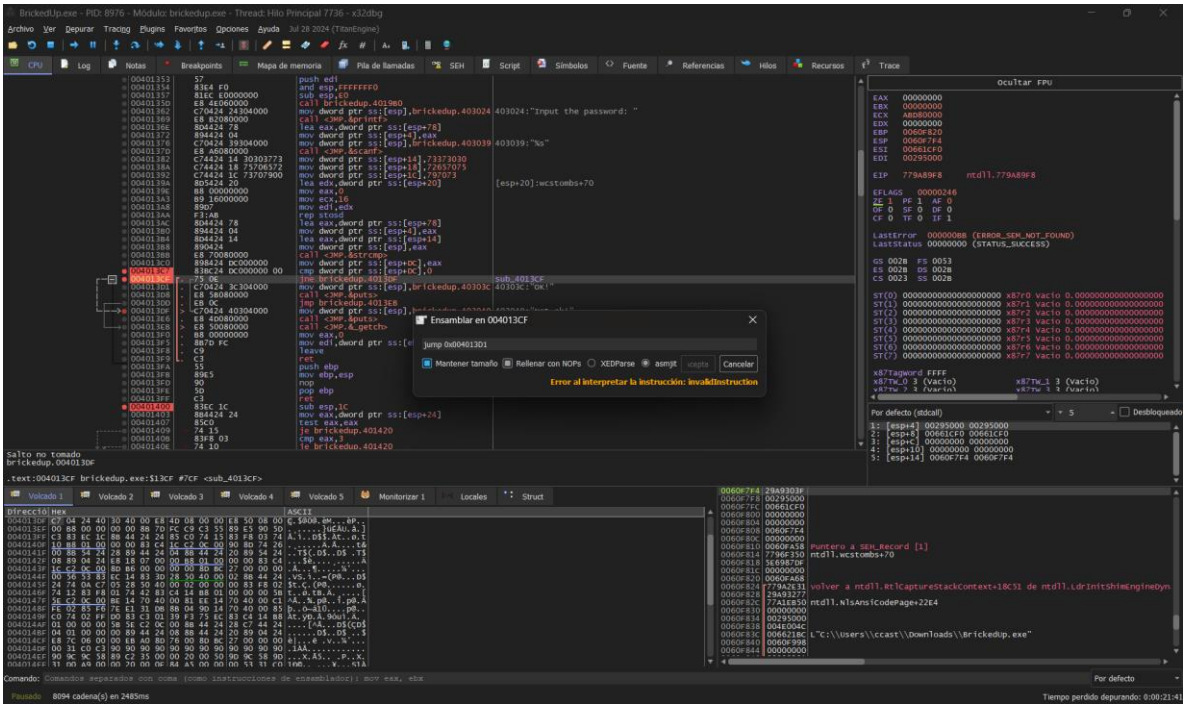
The x32dbg debugging environment was used to perform a thorough analysis of the program. To understand the behavior of the program, all references to strings in the code were examined.

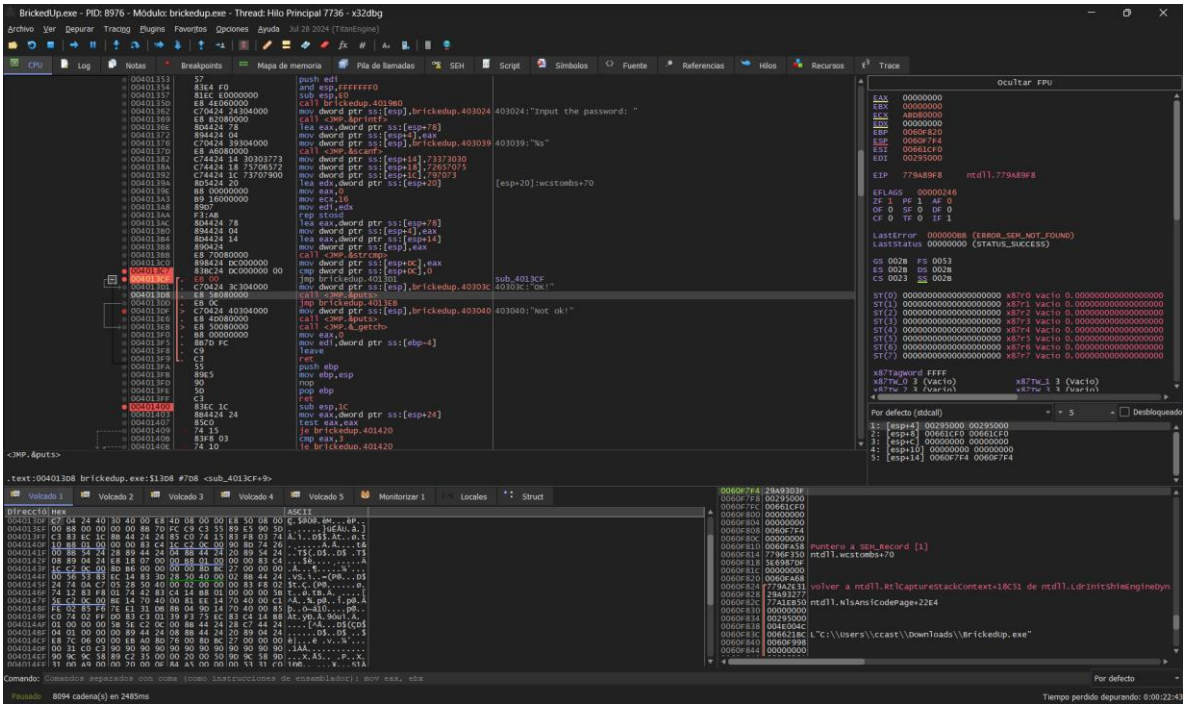


At memory address 40113C7, there is a comparison instruction (cmp) that checks whether the entered password matches the expected password. If the comparison is negative, a jump instruction (jne) is executed that takes the execution flow to address 4013DF, where the program terminates. This behavior indicates that the program allows access only if the correct password is provided.

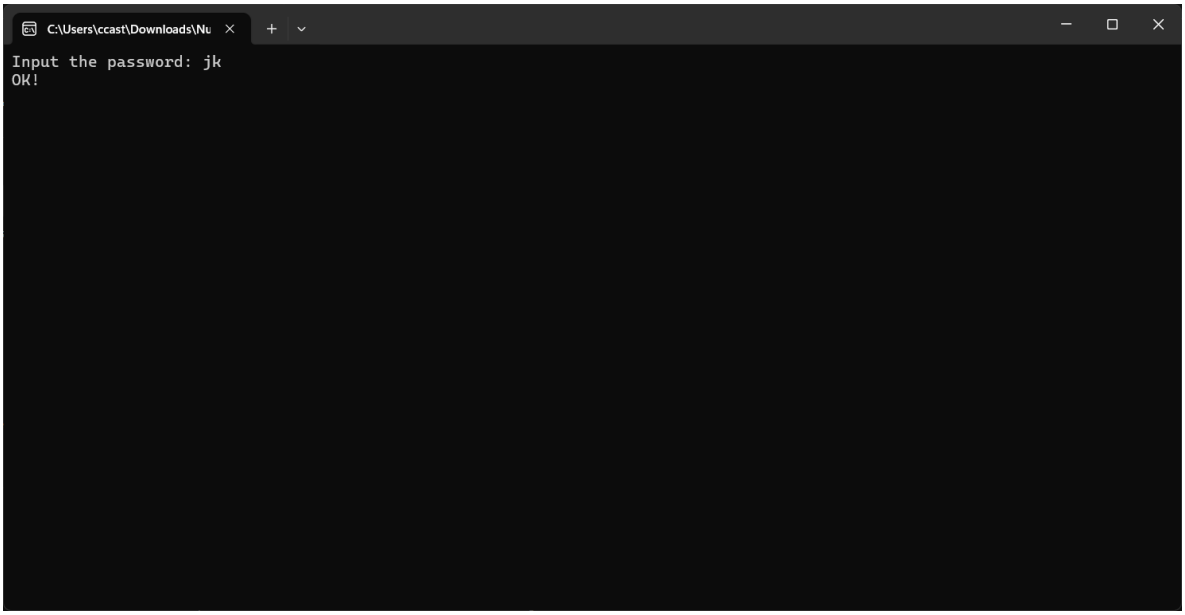


To bypass the password verification, the flow of the program was modified so that, regardless of the password entered, the user could access the system. This change was achieved by assembling the code so that it would jump directly to the success section without validating the user's input.





After implementing the modifications, the program was found to allow access regardless of the password entered. This result demonstrates a significant vulnerability in the implementation of the Bricked-Up authentication system.



Recommendation:

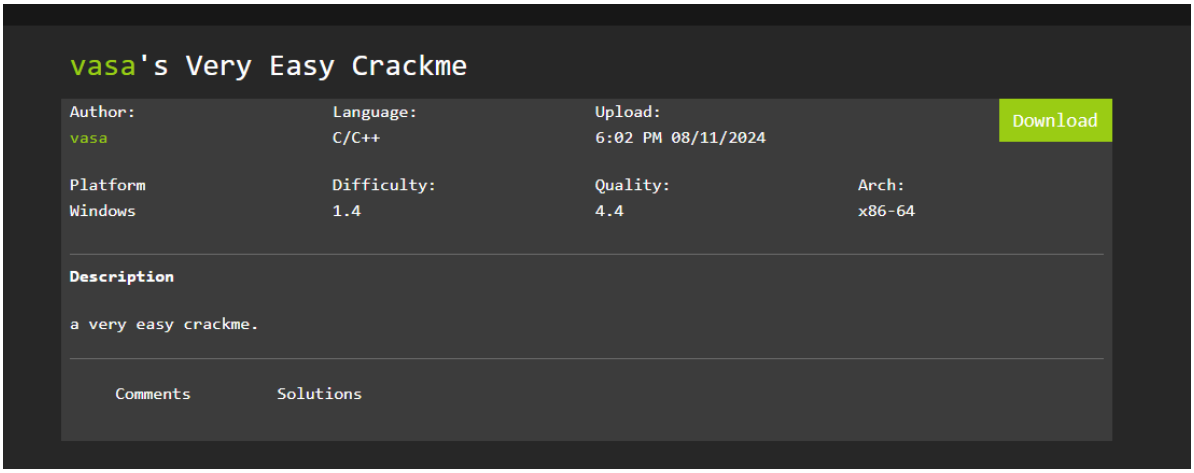
Who:	The system security team and developers of the programs.
Vector:	Remote.
Action:	*Item 1*: Implement robust input validations for password authentication. *Item 2*: Perform code reviews focused on authentication logic. *Item 3*: Establish unit tests that include test cases for authentication logic. *Item 4*: Document authentication logic and security mechanisms implemented.

Exploit Proof of Concept Vasa's Very Easy Crackme

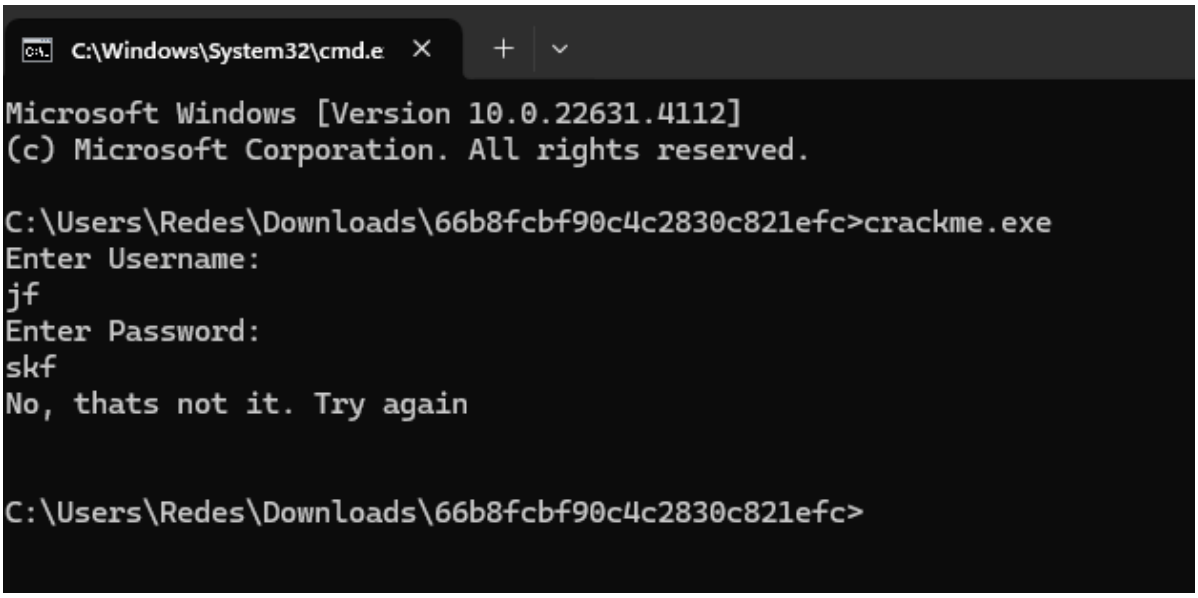
Crackmes

The Proof-of-Concept that follows highlights our ability to carry out remote code execution and take over the target system by exploiting the windows programs.

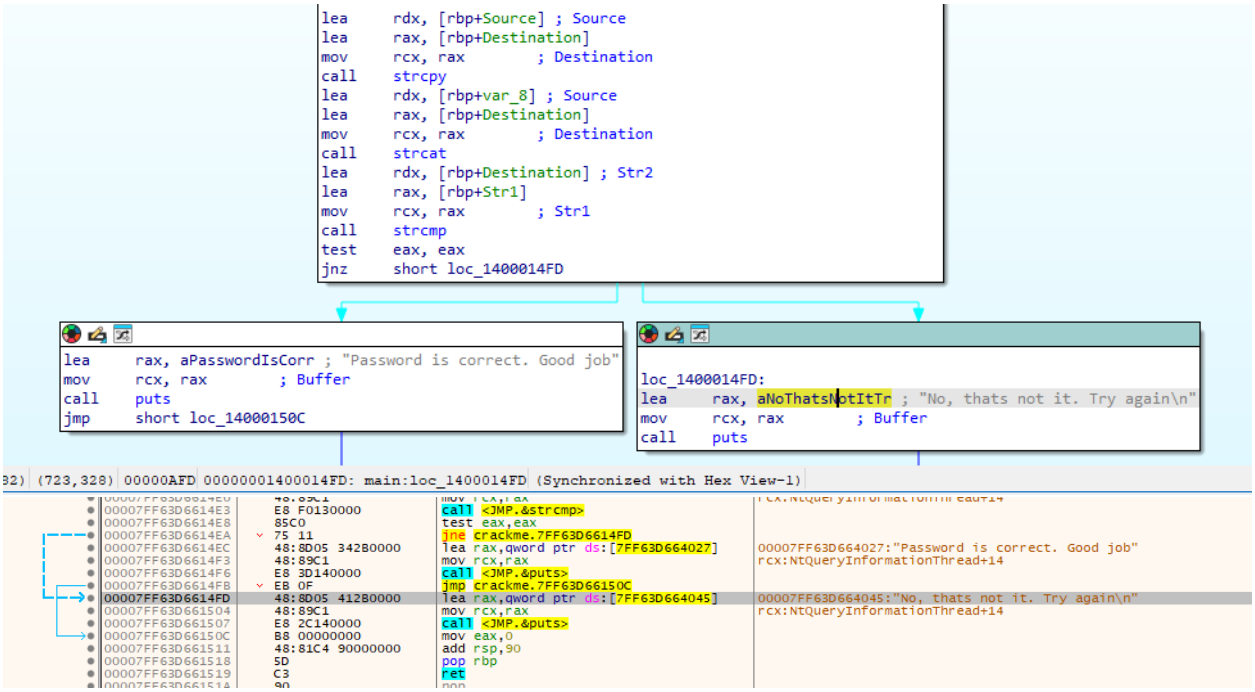
Through an analysis of Vasa's Very Easy Crackme program, an application designed for Windows and written in C/C++. The objective is to identify and exploit vulnerabilities in its authentication mechanism.



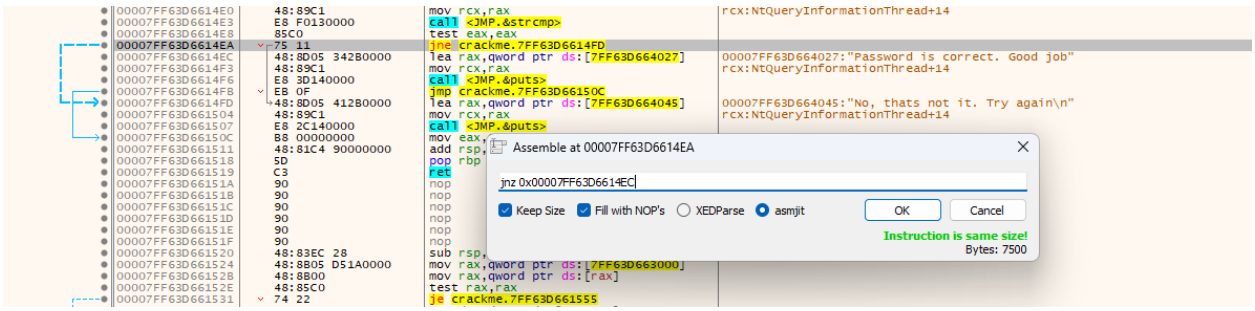
The crackmes.one platform was used to access the program and analyze its behavior. When running Vasa's Very Easy Crackme, the user is prompted to enter a username and password. If the credentials are incorrect, the program returns the message “No, that's not it. Try again” and exits. To understand the execution flow, the IDA disassembler was used.



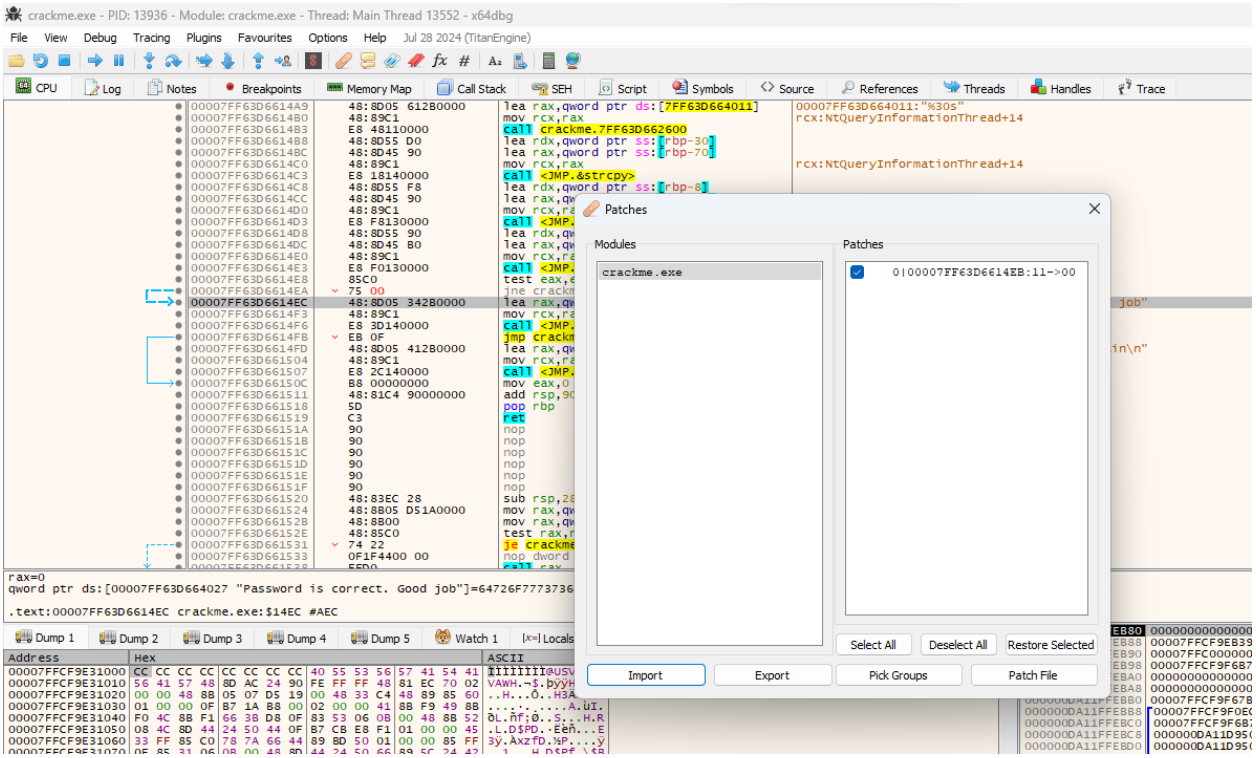
In IDA, the program flow was examined to identify relevant checkpoints. Subsequently, it was switched to x64dbg for further analysis. At this stage, the memory address 00007FF63D6614FD, where the error message was generated, was located. The instructions and conditional jumps were analyzed to determine how they could be manipulated to bypass authentication.



The hop prior to the execution of the error message was accessed, allowing the program flow to be modified. This involved writing code that altered the credential verification logic, allowing access without the need to provide valid information.



Once the code modifications were implemented, the solution was saved, and the program was executed. The result was successful: access to the system was obtained regardless of the credentials entered.



Recommendation:

Who:	The system security team and developers of the programs.
Vector:	Remote.

Action:	*Item 1*: Implement strong validations for user and password authentication. *Item 2*: Perform comprehensive security testing, focusing on the authentication flow. *Item 3*: Regularly review and audit code to identify and correct vulnerabilities. *Item 4*: Properly document authentication flow and critical security points in the code.