

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 Contents

Assessment Overview.....	3
Assessment Components	3
External Penetration Test	3
Finding Severity Ratings	4
Scope	4
Scope Exclusions	5
Client Allowances	5
Executive Summary	5
Attack Summary Bricked-Up	5
Attack Summary Vasa's Very Easy Crackme.....	6
Security Weaknesses.....	6
Inadequate Validation Protections	6
Revealing Error Codes	6
Lack of Code Encryption.....	7
External Penetration Test Findings	7
Exploit Proof of Concept Bricked-Up.....	7
Exploit Proof of Concept Vasa's Very Easy Crackme	11

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

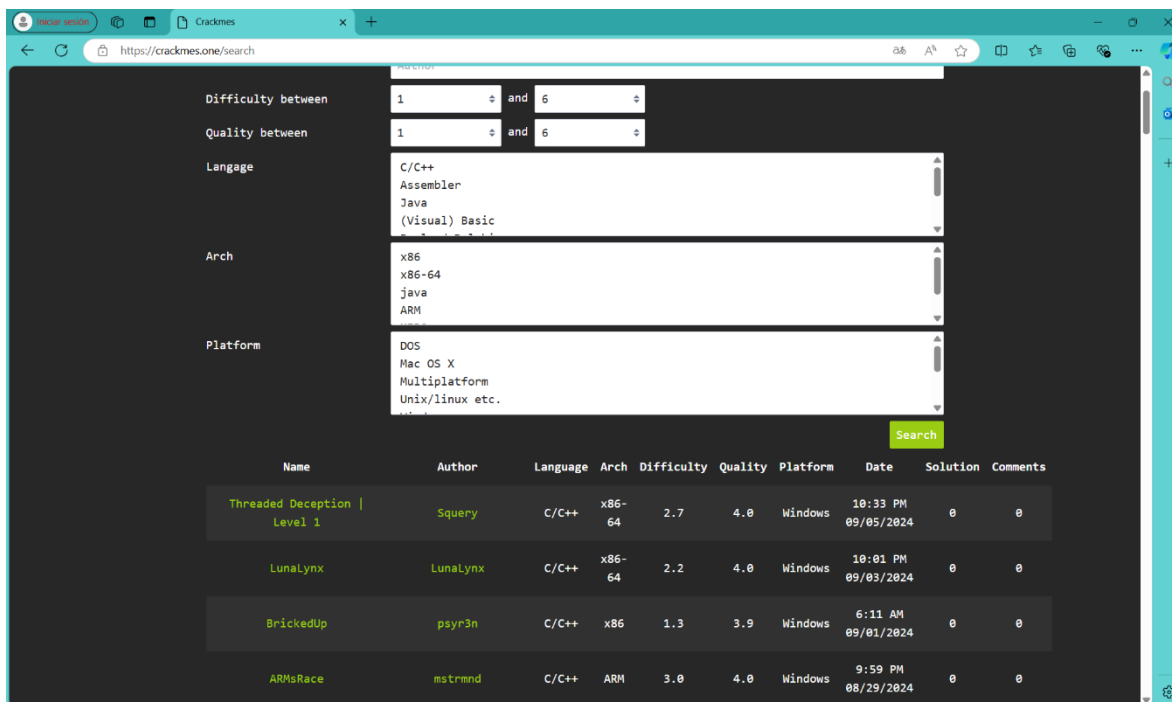
Description:	An analysis of Vasa's Very Easy Crackme and Bricked up program, which requests login credentials, was performed. After observing the error message, x64dbg and/or IDA were used to identify and modify the code at a specific address, avoiding the access verification. Finally, the modified program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

Exploit Proof of Concept Bricked-Up

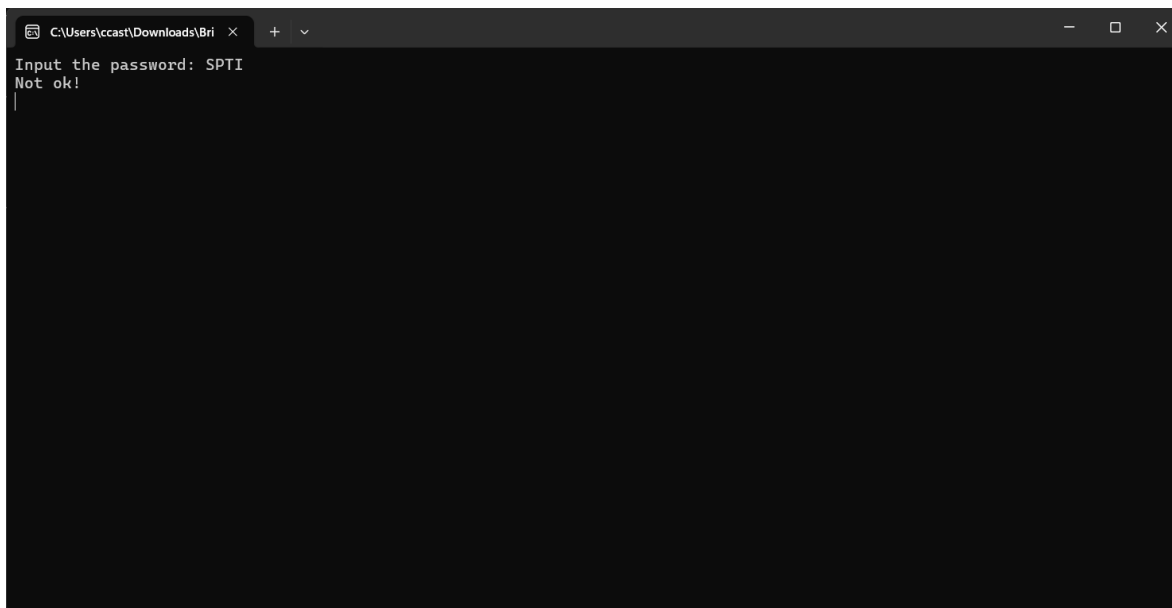
Crackmes

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.

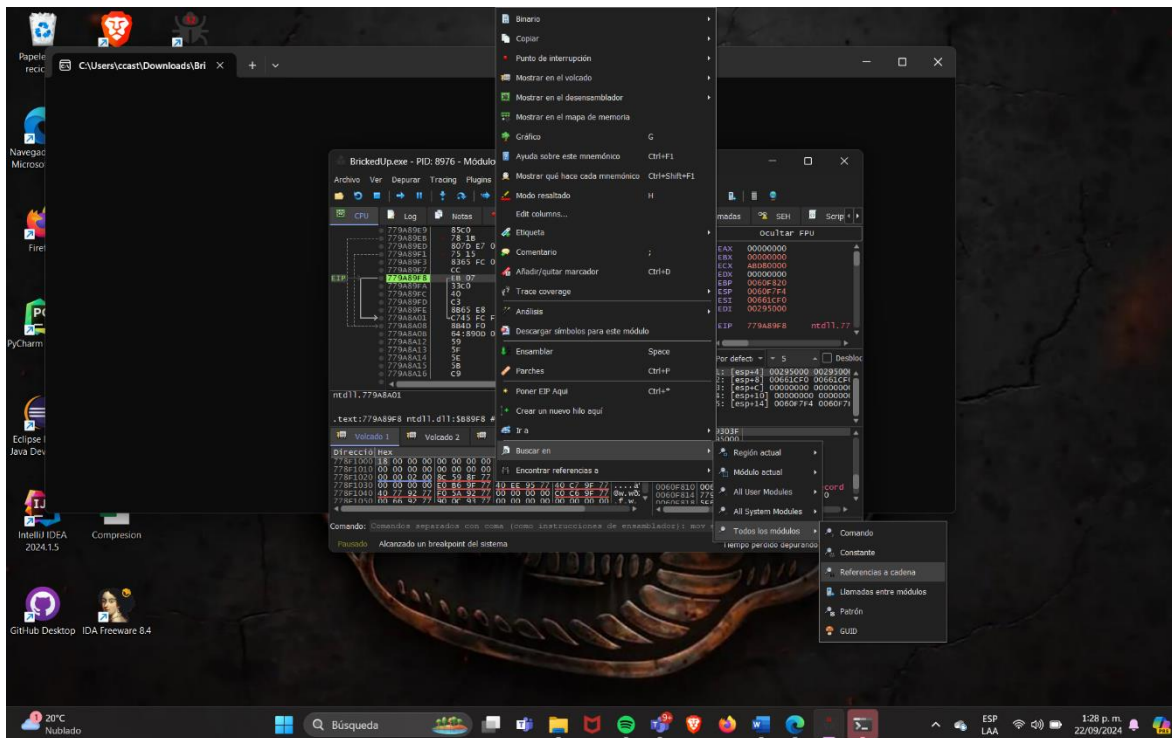
By means of an analysis of the Bricked-Up program, a Crackme designed for Windows and written in C/C++. The objective is to explore the vulnerabilities of this program by manipulating its authentication logic.



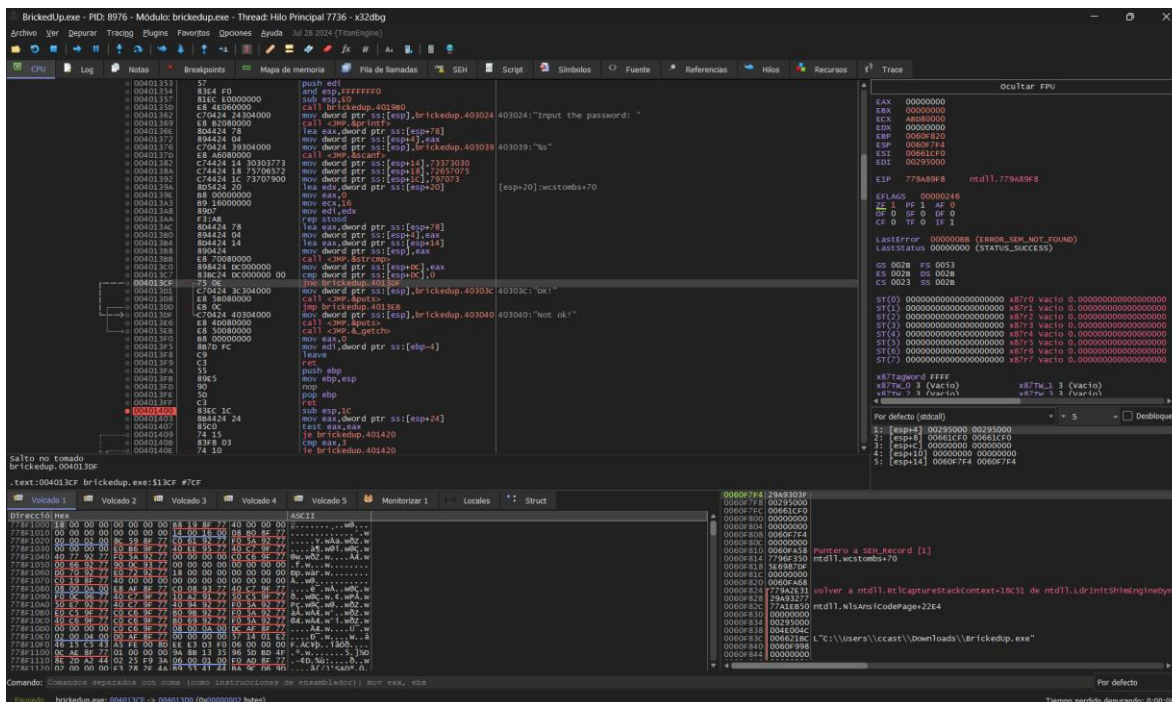
When starting Bricked Up, the user is prompted for a password, and if the password is incorrect, the message “Not ok!” is displayed and the program closes.



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.

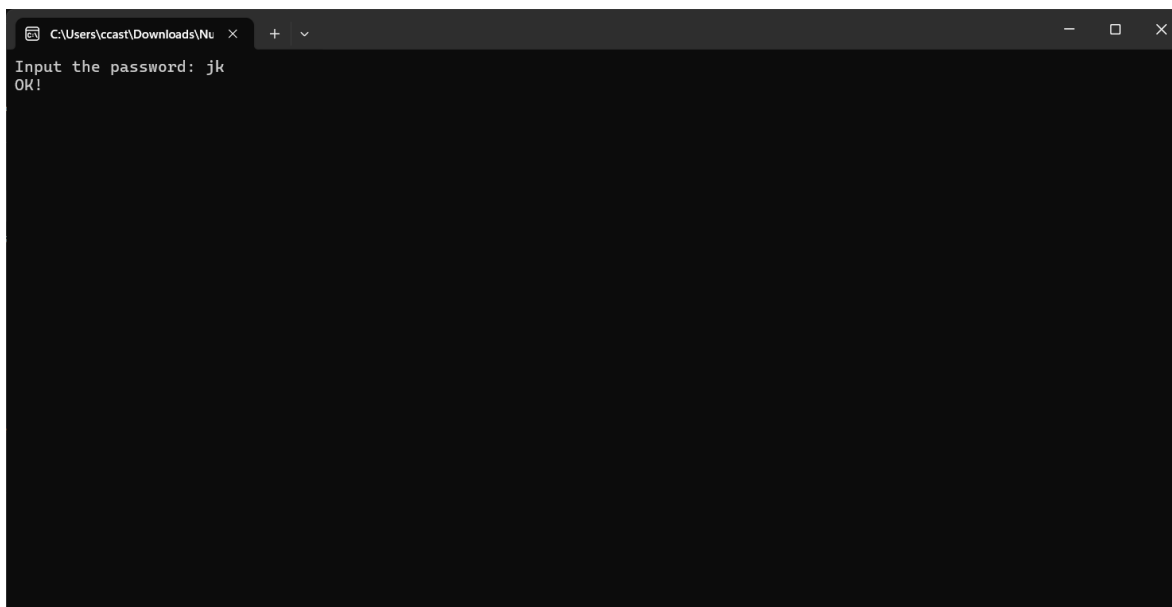


At memory address 40113C7, there is a comparison instruction (cmp) that checks whether the entered password matches the expected password. If the comparison is negative, a jump instruction (jne) is executed that takes the execution flow to address 4013DF, where the program terminates. This behavior indicates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



[illegible]

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:

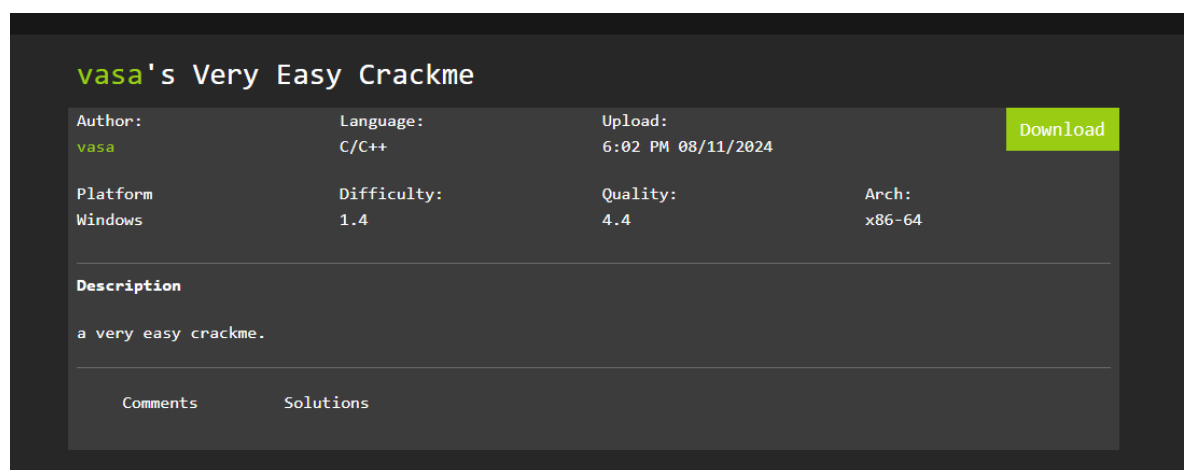
Who:	The system security team and developers of the programs.
Vector:	Remote.
Action:	<p><i>Item 1:</i> Implement robust input validations for password authentication.</p> <p><i>Item 2:</i> Perform code reviews focused on authentication logic.</p> <p><i>Item 3:</i> Establish unit tests that include test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

Exploit Proof of Concept Vasa's Very Easy Crackme

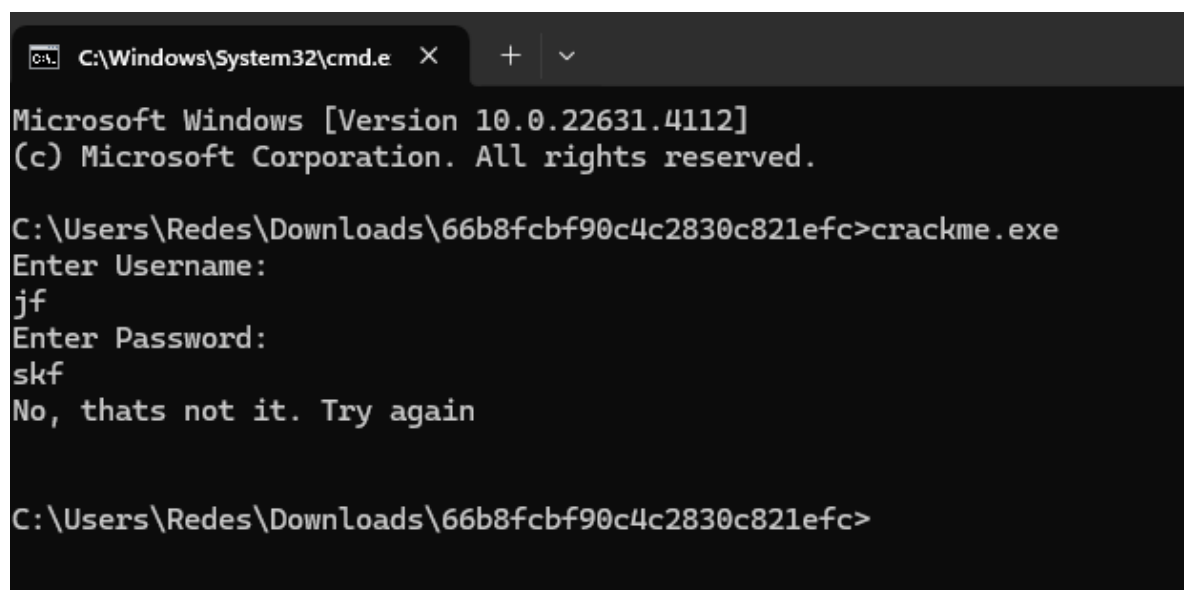
Crackmes

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.

```

lea rdx, [rbp+Source] ; Source
lea rax, [rbp+Destination]
mov rcx, rax ; Destination
call strcpy
lea rdx, [rbp+var_8] ; Source
lea rax, [rbp+Destination]
mov rcx, rax ; Destination
call strcat
lea rdx, [rbp+Destination] ; Str2
lea rax, [rbp+Str1]
mov rcx, rax ; Str1
call strcmp
test eax, eax
jnz short loc_1400014FD

lea rax, aPasswordIsCorr ; "Password is correct. Good job"
mov rcx, rax ; Buffer
call puts
jmp short loc_14000150C

loc_1400014FD:
lea rax, aNoThatIsNotItTr ; "No, thats not it. Try again\n"
mov rcx, rax ; Buffer
call puts

```

32) (723,328) 00000AFD 00000001400014FD: main:loc_1400014FD (Synchronized with Hex View-1)

00007FF63D6614E0	48:89C1	mov rcx,rax	rcx:NtQueryInformationThread+14
00007FF63D6614E3	E8:F0130000	call <JMP.&strcmp>	
00007FF63D6614E8	85C0	test eax, eax	
00007FF63D6614EA	75:11	jnz crackme.7FF63D6614FD	00007FF63D664027: "Password is correct. Good job"
00007FF63D6614EC	48:8D05 342B0000	lea rax, qword ptr ds:[7FF63D664027]	rcx:NtQueryInformationThread+14
00007FF63D6614F3	48:89C1	mov rcx, rax	
00007FF63D6614F6	E8:3D140000	call <JMP.&puts>	
00007FF63D6614FB	E8:0F	jmp crackme.7FF63D66150C	00007FF63D664045: "No, thats not it. Try again\n"
00007FF63D6614FD	48:8D05 412B0000	lea rax, qword ptr ds:[7FF63D664045]	rcx:NtQueryInformationThread+14
00007FF63D661504	48:89C1	mov rcx, rax	
00007FF63D661507	E8:2C140000	call <JMP.&puts>	
00007FF63D66150C	B8:00000000	mov eax, 0	
00007FF63D661511	48:81C4 90000000	add rsp, 90	
00007FF63D661518	5D	pop rbp	
00007FF63D661519	C3	ret	
00007FF63D66151A	48	ret	

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.

00007FF63D6614E0	48:89C1	mov rcx,rax	rcx:NtQueryInformationThread+14
00007FF63D6614E3	E8:F0130000	call <JMP.&strcmp>	
00007FF63D6614E8	85C0	test eax, eax	
00007FF63D6614EA	75:11	jnz crackme.7FF63D6614FD	00007FF63D664027: "Password is correct. Good job"
00007FF63D6614EC	48:8D05 342B0000	lea rax, qword ptr ds:[7FF63D664027]	rcx:NtQueryInformationThread+14
00007FF63D6614F3	48:89C1	mov rcx, rax	
00007FF63D6614F6	E8:3D140000	call <JMP.&puts>	
00007FF63D6614FB	E8:0F	jmp crackme.7FF63D66150C	00007FF63D664045: "No, thats not it. Try again\n"
00007FF63D6614FD	48:8D05 412B0000	lea rax, qword ptr ds:[7FF63D664045]	rcx:NtQueryInformationThread+14
00007FF63D661504	48:89C1	mov rcx, rax	
00007FF63D661507	E8:2C140000	call <JMP.&puts>	
00007FF63D66150C	B8:00000000	mov eax, 0	
00007FF63D661511	48:81C4 90000000	add rsp, 90	
00007FF63D661518	5D	pop rbp	
00007FF63D661519	C3	ret	
00007FF63D66151A	48	ret	
00007FF63D66151F	48:83EC 28	sub rsp, 28	
00007FF63D661520	48:8B05 D51A0000	mov rax, qword ptr ds:[7FF63D663000]	
00007FF63D661524	48:8B00	mov rax, qword ptr ds:[rax]	
00007FF63D66152E	48:85C0	test rax, rax	
00007FF63D661531	74:22	jz crackme.7FF63D661555	

Assemble at 00007FF63D6614EA

Keep Size ☒ Fill with NOP's ☐ XEDParse ☐ asmjit

OK Cancel

Instruction is same size! Bytes: 7500

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.

crackme.exe - PID: 13936 - Module: crackme.exe - Thread: Main Thread 13552 - x64dbg

File View Debug Tracing Plugins Favourites Options Help Jul 28 2024 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace

00007FF63D6614A9 48:8005 612B0000 lea rax,qword ptr ds:[7FF63D664011] 00007FF63D664011:"%s"
 00007FF63D6614B0 48:89C1 mov rcx,rax rcx:NtQueryInformationThread+14
 00007FF63D6614B3 E8 48110000 call crackme.7FF63D662600
 00007FF63D6614B8 48:8055 D0 lea rdx,qword ptr ss:[rbp-30]
 00007FF63D6614BC 48:8045 90 lea rax,qword ptr ss:[rbp-70]
 00007FF63D6614C0 48:89C1 mov rcx,rax rcx:NtQueryInformationThread+14
 00007FF63D6614C3 E8 18140000 call <JMP.&strncpy>
 00007FF63D6614C8 48:8055 F8 lea rdx,qword ptr ss:[rbp-8]
 00007FF63D6614CC 48:8045 90 lea rax,qword ptr ss:[rbp-8]
 00007FF63D6614D0 48:89C1 mov rcx,rax
 00007FF63D6614D3 E8 F8130000 call <JMP.>
 00007FF63D6614D8 48:8055 90 lea rdx,qword ptr ss:[rbp-8]
 00007FF63D6614DB 48:8045 80 lea rax,qword ptr ss:[rbp-8]
 00007FF63D6614E0 48:89C1 mov rcx,rax
 00007FF63D6614E3 E8 F0130000 call <JMP.>
 00007FF63D6614E8 85C0 test eax,eax
 00007FF63D6614EA 75 00 jne crackme.7FF63D6614F3
 00007FF63D6614EC 48:8005 342B0000 lea rax,qword ptr ds:[00007FF63D664027 "Password is correct. Good job"]=64726F7773736
 00007FF63D6614F3 48:89C1 mov rcx,rax
 00007FF63D6614F6 E8 3D140000 call 0F
 00007FF63D6614FB E8 0F000000 jmp crackme.7FF63D6614FD
 00007FF63D6614FD 48:8005 412B0000 lea rax,qword ptr ds:[00007FF63D661504
 00007FF63D661507 48:89C1 mov rcx,rax
 00007FF63D66150C E8 2C140000 call <JMP.>
 00007FF63D661510 8B 00000000 mov eax,0
 00007FF63D661511 48:81C4 90000000 add rsp,90
 00007FF63D66151B 5D pop rbp
 00007FF63D661519 C3 ret
 00007FF63D66151A 90 nop
 00007FF63D66151B 90 nop
 00007FF63D66151C 90 nop
 00007FF63D66151D 90 nop
 00007FF63D66151E 90 nop
 00007FF63D66151F 90 nop
 00007FF63D661520 48:83EC 28 sub rsp,28
 00007FF63D661524 48:8B05 D51A0000 mov rax,qword ptr ds:[00007FF63D66152B
 00007FF63D66152B 48:8B00 mov rax,qword ptr ds:[00007FF63D66152E
 00007FF63D66152E 48:85C0 test rax,rax
 00007FF63D661531 74 22 jle crackme.7FF63D661533
 00007FF63D661533 0F1F4400 00 nop dword ptr [rax+0]
 00007FF63D661534 C3 ret

rax=0
 qword ptr ds:[00007FF63D664027 "Password is correct. Good job"]=64726F7773736
 .text:00007FF63D6614EC crackme.exe:\$14EC #AEC

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals

Address	Hex	ASCII
00007FFCF9E31000	CC CC CC CC CC CC CC CC 40 55 53 56 57 41 54 41	iiiiiiiiuuuu
00007FFCF9E31010	56 41 57 48 80 AC 24 90 FE FF FF 48 81 EC 70 02	VAWH--\$.byyh
00007FFCF9E31020	00 00 48 88 05 07 D5 19 00 48 33 C4 48 89 85 60	..H...0..HSA
00007FFCF9E31030	01 00 00 0F 87 1A B8 00 02 00 00 41 88 F9 49 88A.U.I.
00007FFCF9E31040	F0 4C 88 F1 66 38 D8 0F 83 53 06 08 00 48 88 52	0L.Hf.0..S...H.R
00007FFCF9E31050	08 4C 8D 44 24 50 44 0F B7 CB E8 F1 01 00 00 45	.L.D3PD..Eeh...E
00007FFCF9E31060	33 FF 85 C0 78 7A 66 44 89 8D 50 01 00 00 85 FF	3y.AXZFD.WP...y
00007FFCF9E31070	0F 85 31 06 08 00 48 88 4A 74 7A C0 66 89 5F 74 47	1 H n6pP 48B

EB80 0000000000000000
 EB83 00007FFCF9EB39
 EB90 00007FFCF9EB39
 EB98 00007FFCF9EB39
 EB99 00007FFCF9EB39
 EB9A 0000000000000000
 EB9B 0000000000000000
 EB9C 00007FFCF9EB39
 EB9D 00007FFCF9EB39
 EB9E 0000000000000000
 EB9F 0000000000000000
 EBA0 0000000000000000
 EBA1 0000000000000000
 EBA2 0000000000000000
 EBA3 0000000000000000
 EBA4 0000000000000000
 EBA5 0000000000000000
 EBA6 0000000000000000
 EBA7 0000000000000000
 EBA8 0000000000000000
 EBA9 0000000000000000
 EBAA 0000000000000000
 EBAB 0000000000000000
 EBAC 0000000000000000
 EBAD 0000000000000000
 EBAE 0000000000000000
 EBAF 0000000000000000
 EBB0 0000000000000000
 EBB1 0000000000000000
 EBB2 0000000000000000
 EBB3 0000000000000000
 EBB4 0000000000000000
 EBB5 0000000000000000
 EBB6 0000000000000000
 EBB7 0000000000000000
 EBB8 0000000000000000
 EBB9 0000000000000000
 EBBA 0000000000000000
 EBBB 0000000000000000
 EBBC 0000000000000000
 EBBD 0000000000000000
 EBBE 0000000000000000
 EBBF 0000000000000000
 EBC0 0000000000000000
 EBC1 0000000000000000
 EBC2 0000000000000000
 EBC3 0000000000000000
 EBC4 0000000000000000
 EBC5 0000000000000000
 EBC6 0000000000000000
 EBC7 0000000000000000
 EBC8 0000000000000000
 EBC9 0000000000000000
 EBCA 0000000000000000
 EBCB 0000000000000000
 EBCD 0000000000000000
 EBCF 0000000000000000

C:\Users\Redes\Downloads\66b8fcfbf90c4c2830c821efc>SOLUCION.EXE
 Enter Username:
 DSDJF
 Enter Password:
 FKJ
 Password is correct. Good job
 C:\Users\Redes\Downloads\66b8fcfbf90c4c2830c821efc>
 C:\Users\Redes\Downloads\66b8fcfbf90c4c2830c821efc>|

Recommendation:

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

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