Nom et Prénom : Ahdibi Aimene

SLMail V5.5 Buffer-overflow Exploit

**Lien de téléchargement de l’application :**

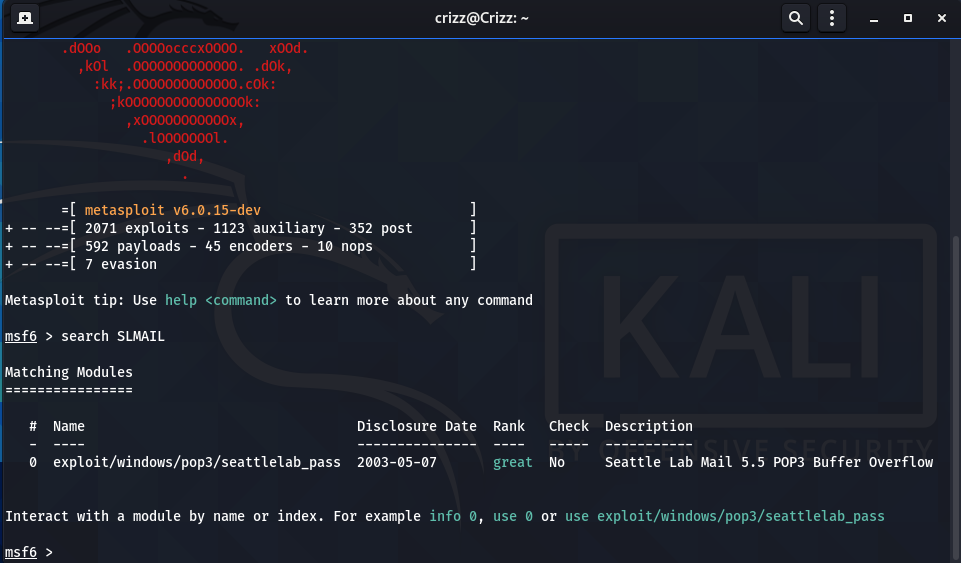
<https://www.exploit-db.com/apps/12f1ab027e5374587e7e998c00682c5d-SLMail55_4433.exe>

Os machine attaquante : KALI LINUX 2021

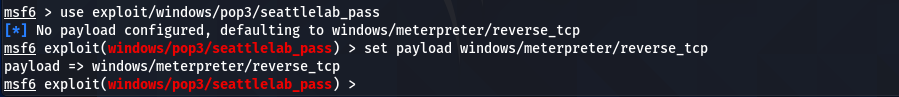
Os machine Victime : Windows Xp version 2002 SP 3

# **Partie 01 : Exploit Automatisé « Metasploit »**

Dans cette partie je vais exploiter la vulnérabilité de Buffer overflow dans le programme SLMail v5.5 en utilisant l’outil Metasploit vu qu’un exploit dans Metasploit de ce dernier programme existe déjà.



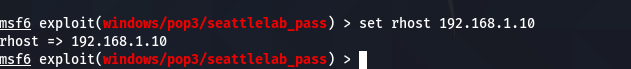
J’ai utilisé le payload Windows/meterpreter/reverse\_tcp pour ouvrir une session meterpreter après l’exploit



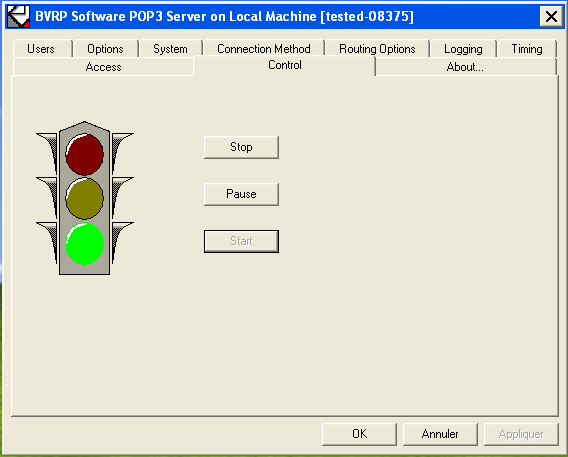


NB : @Kali : 192.168.1.20

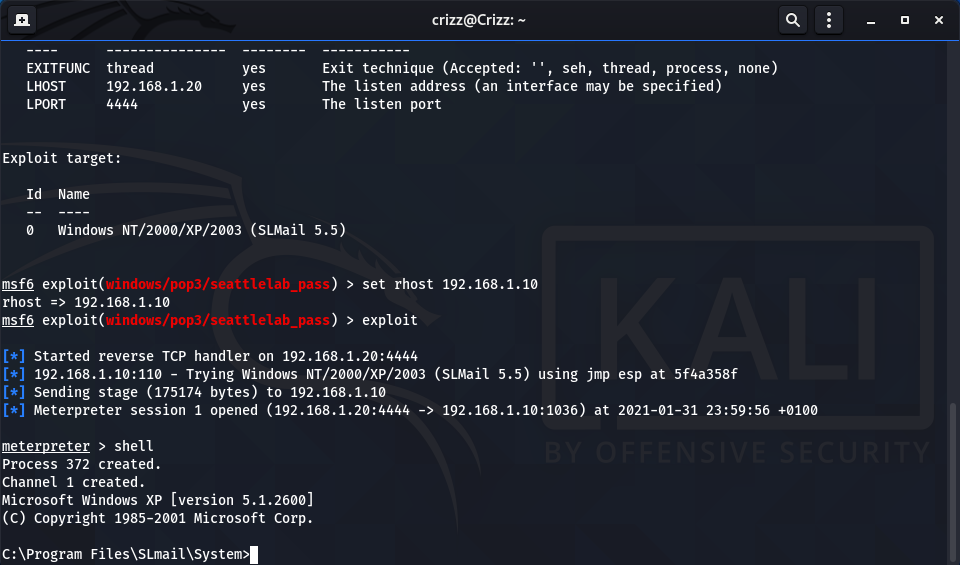
@Windows Xp : 192.168.1.10



Lancement de SLMail dans la machine XP :



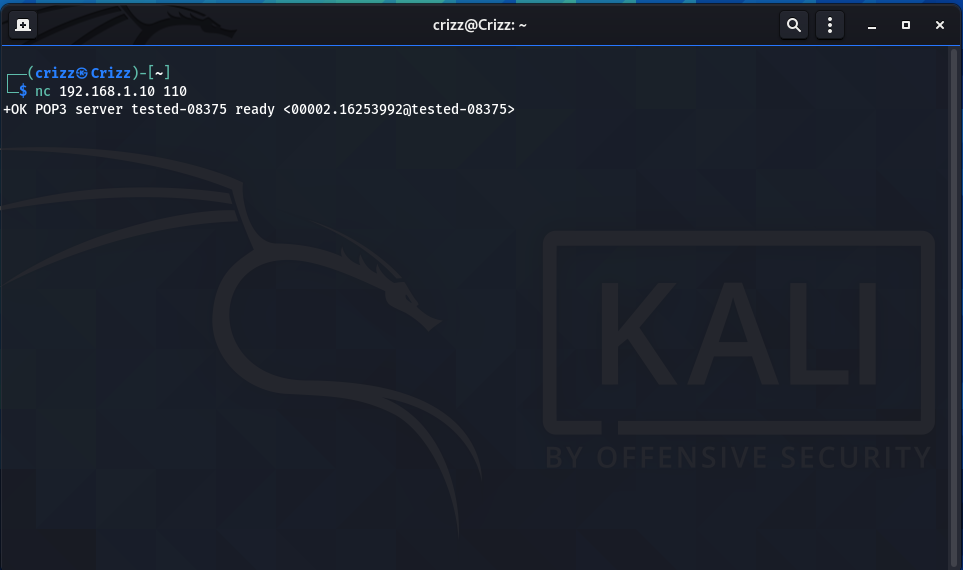
Lancement de l’exploit :

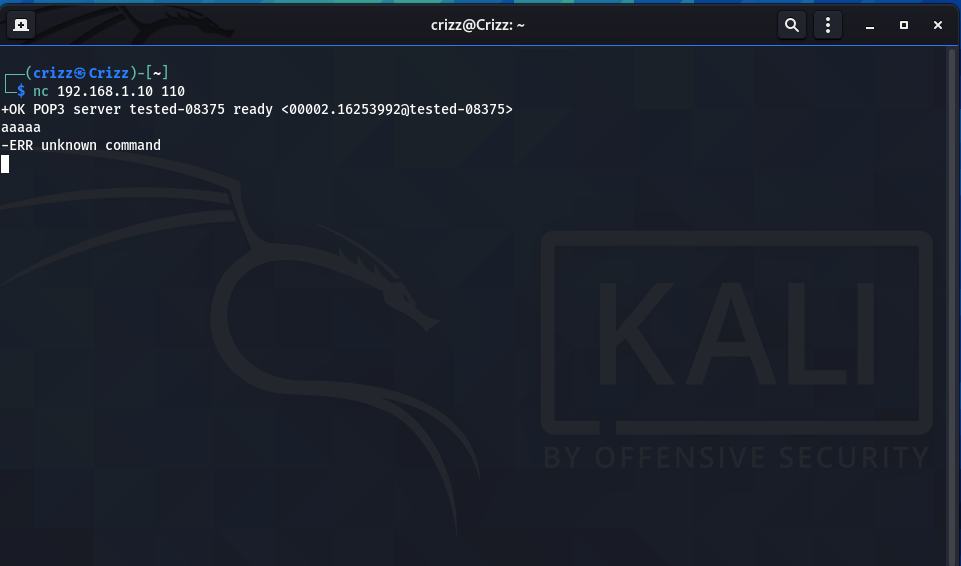


# **Partie 02 : Attaques Buffer Overflow Manuelle**

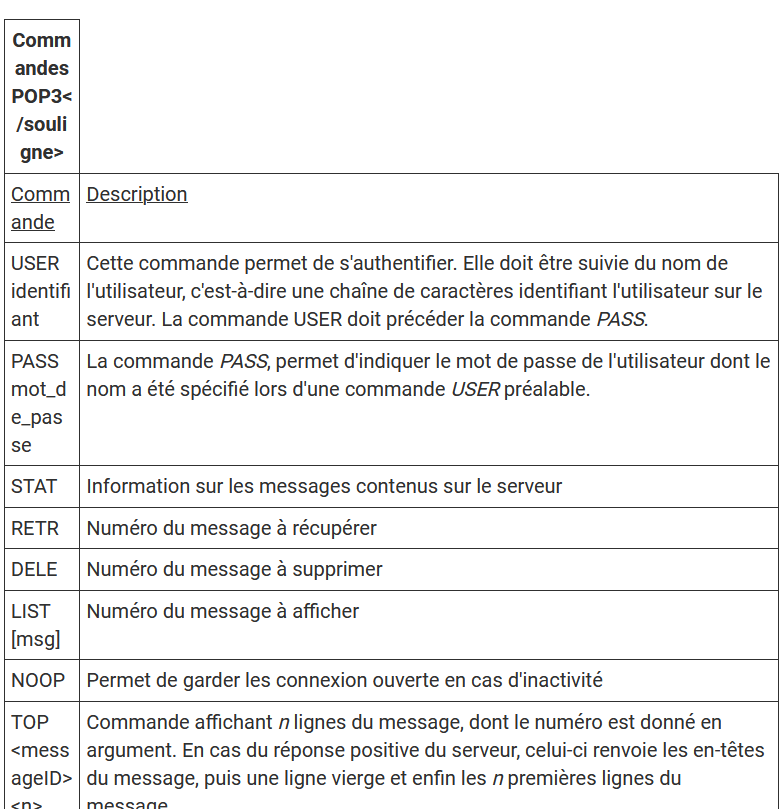
Dans cette partie, je vais debugger et essayer d’exploiter la vulnérabilité manuellement étape par étape jusqu’à l’exécution de la session Shell dans KALI .

## Connexion au port POP3 (110) par netcat (NC) et ‘debug’ des comportements normaux du programme :



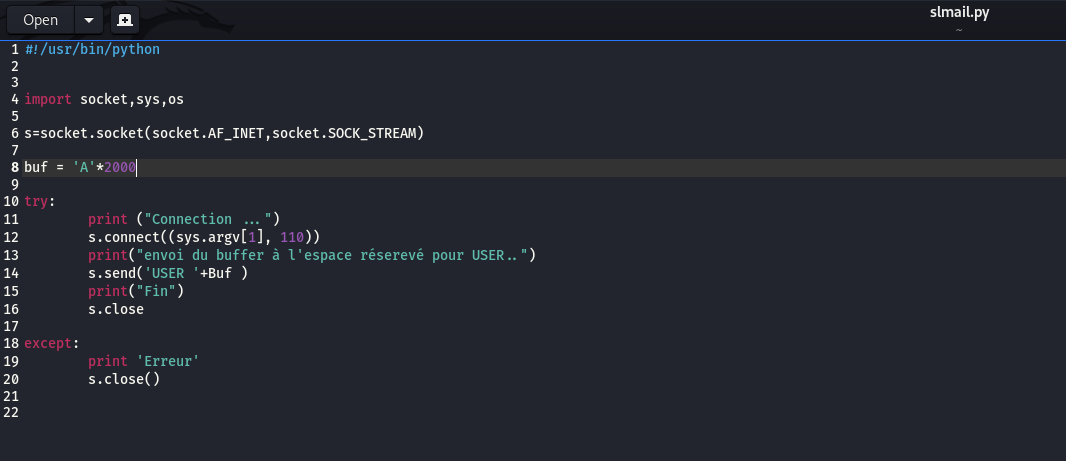


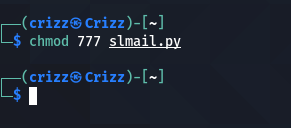
Les commandes disponibles pour POP3



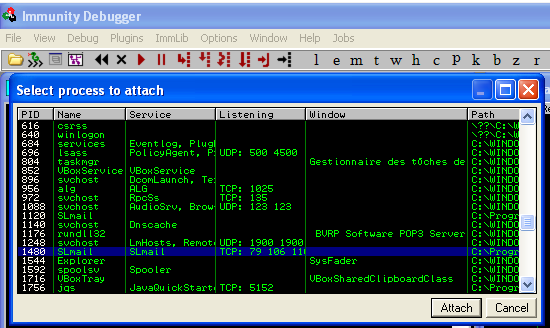


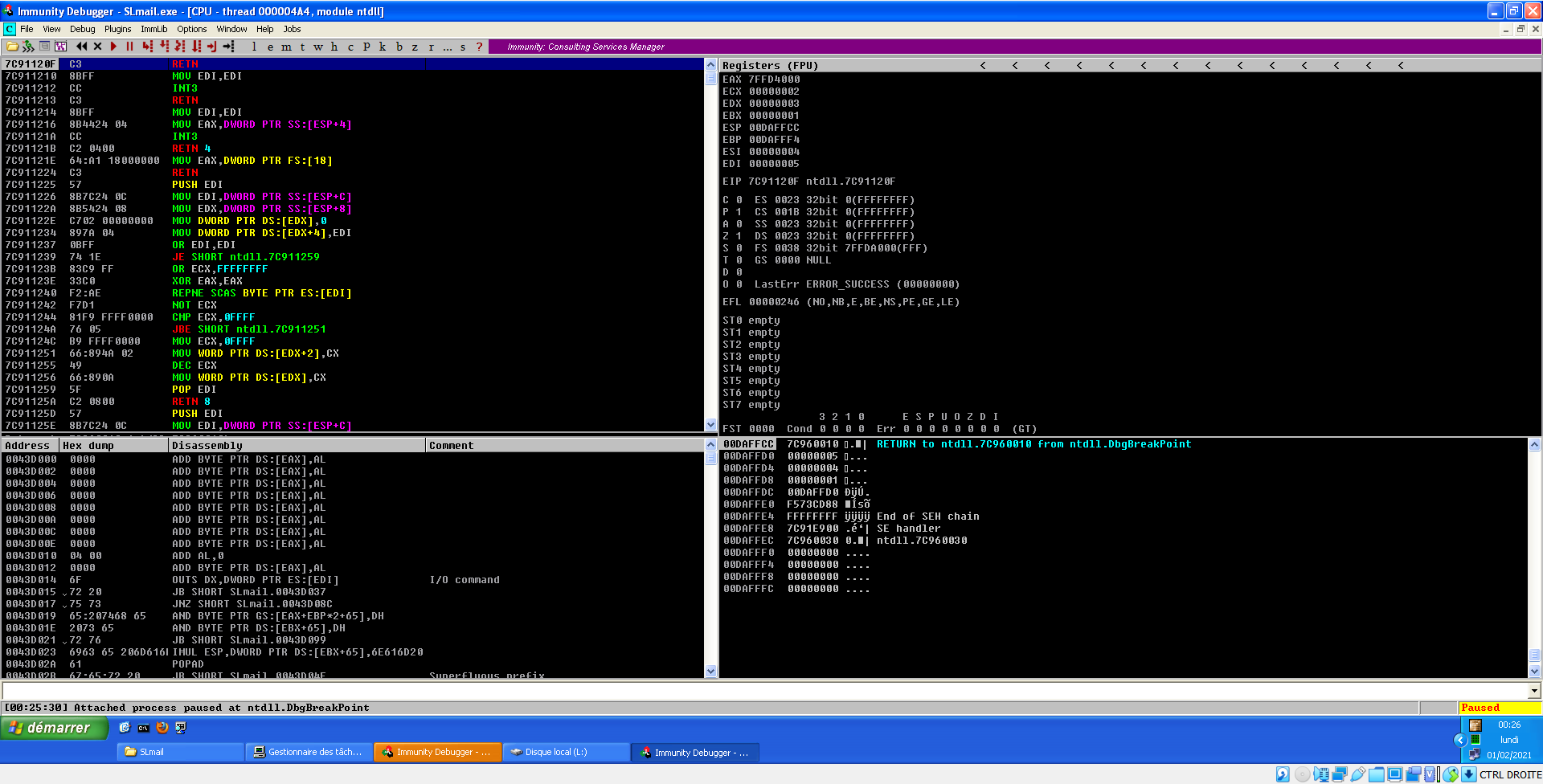
Maintenant je vais utiliser ces deux inputs les remplir et les inonder par des caractères pour essayer de cracher le programme :



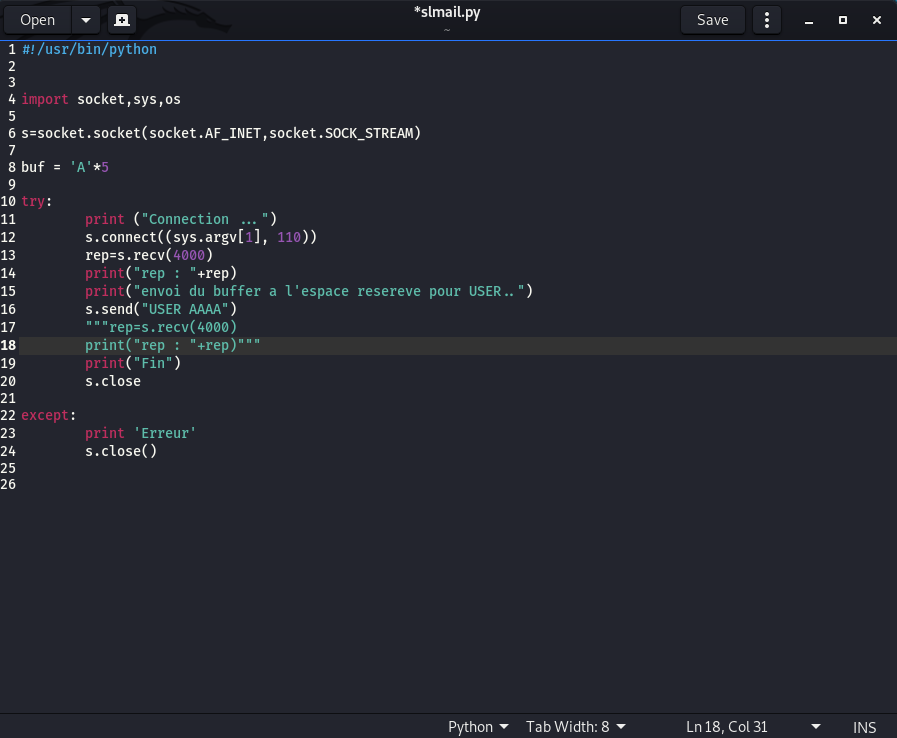


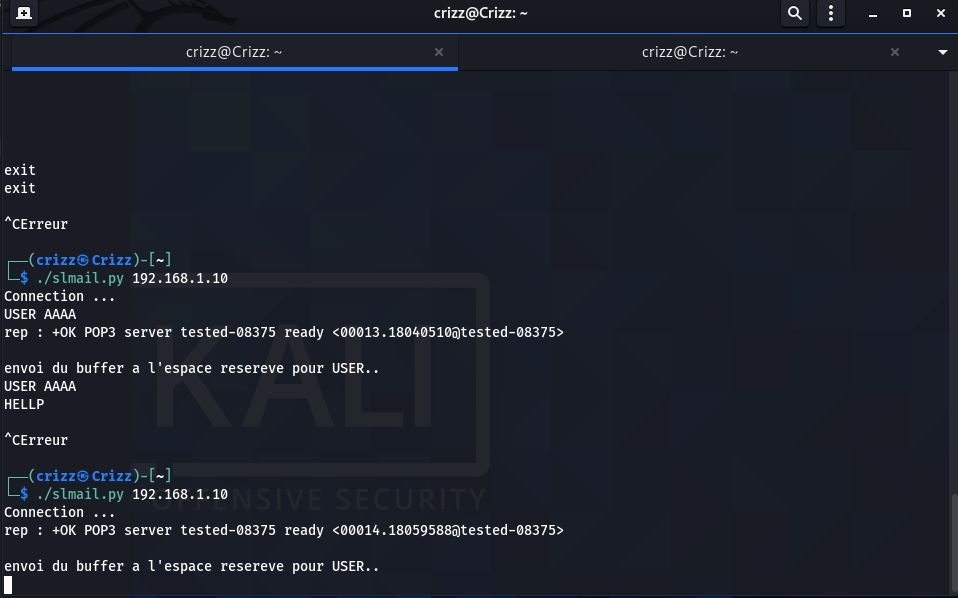
Maintenant je dois attacher l’exécutable du SLMAIL.exe à Immunity debugger pour voir les résultats



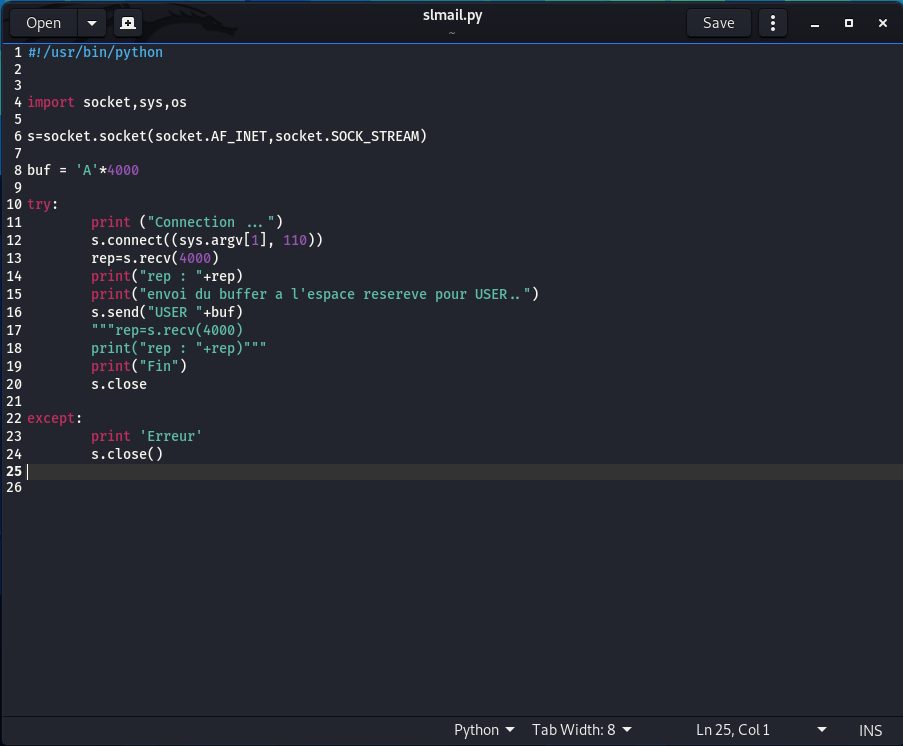


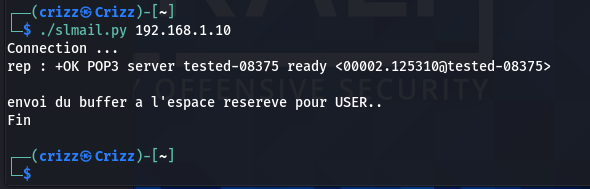
Et je lance le nouveau script pour tester l’état normal avant d’inonder le server avec le premier script

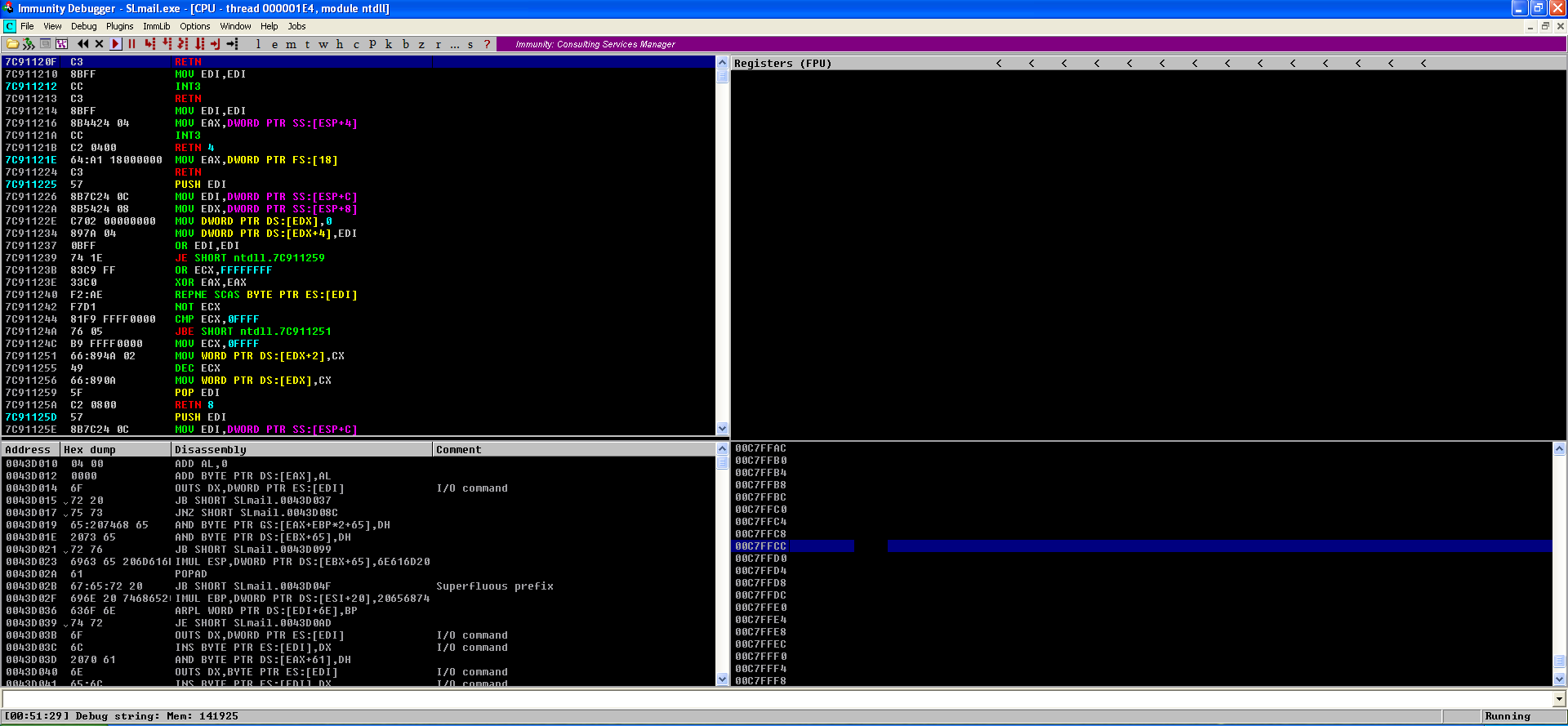




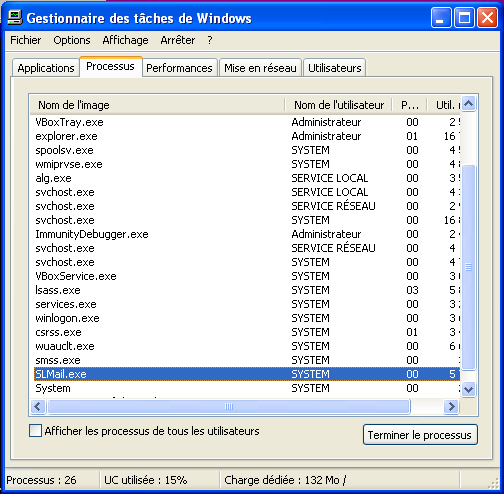
Maintenant je vais essayer d’injecter un buffer de 4000 oct et regarder ce qui se passe dans immunity :



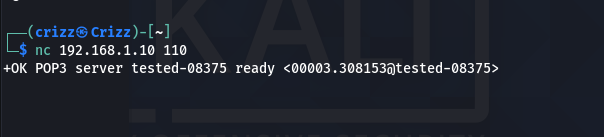




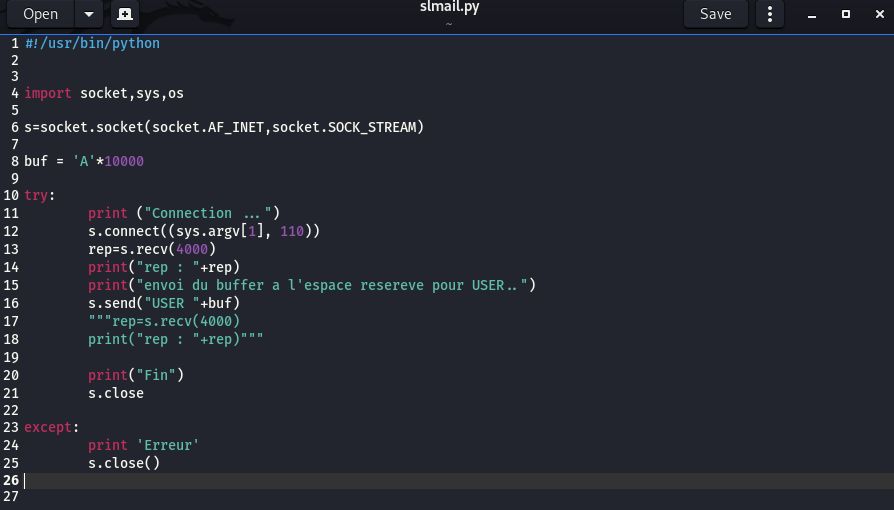
Rien ne s’affiche sur immunity



On constat que le programme n’a pas Crashé donc je vais essayer avec un buffer très grand

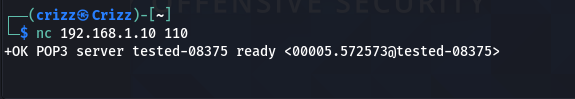


SLMAIL est toujours en marche même après l’envoi du premier buffer, je modifier le script de telle manière que le programme se crache



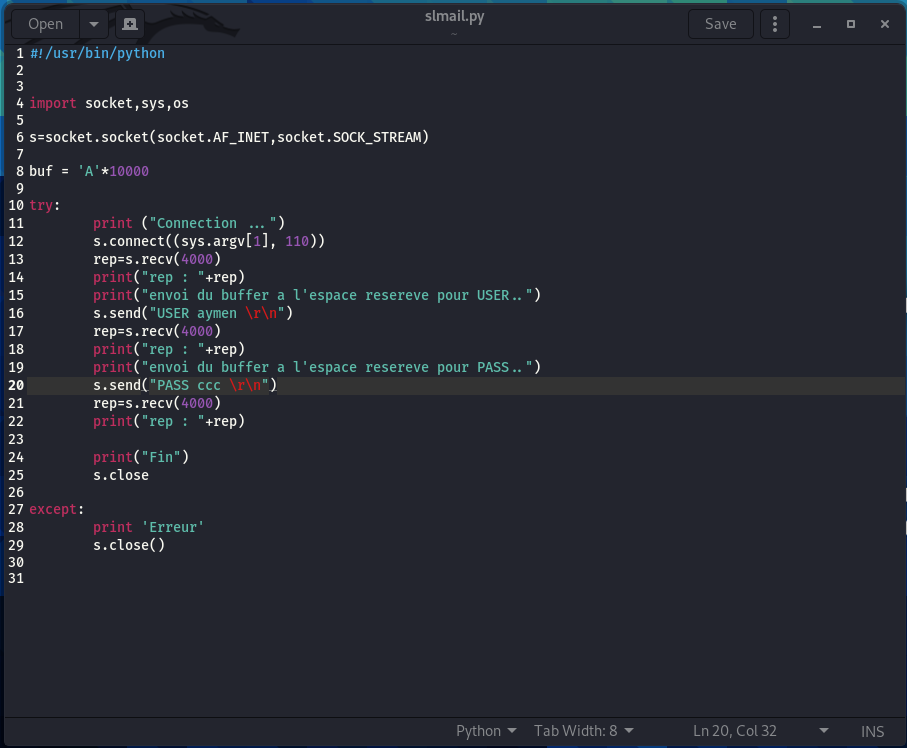


Même Chose comme résultat, je vais basculer vers deuxième input qui est le mot de passe après l’utilisateur

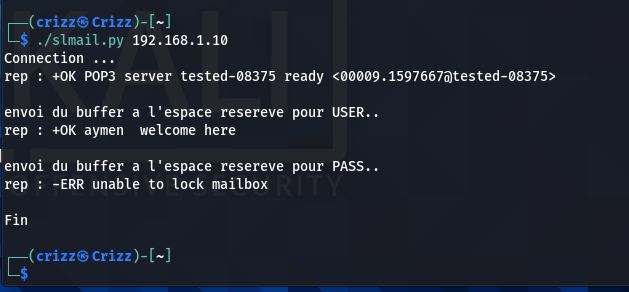


Serveur SLMAIL toujours en marche

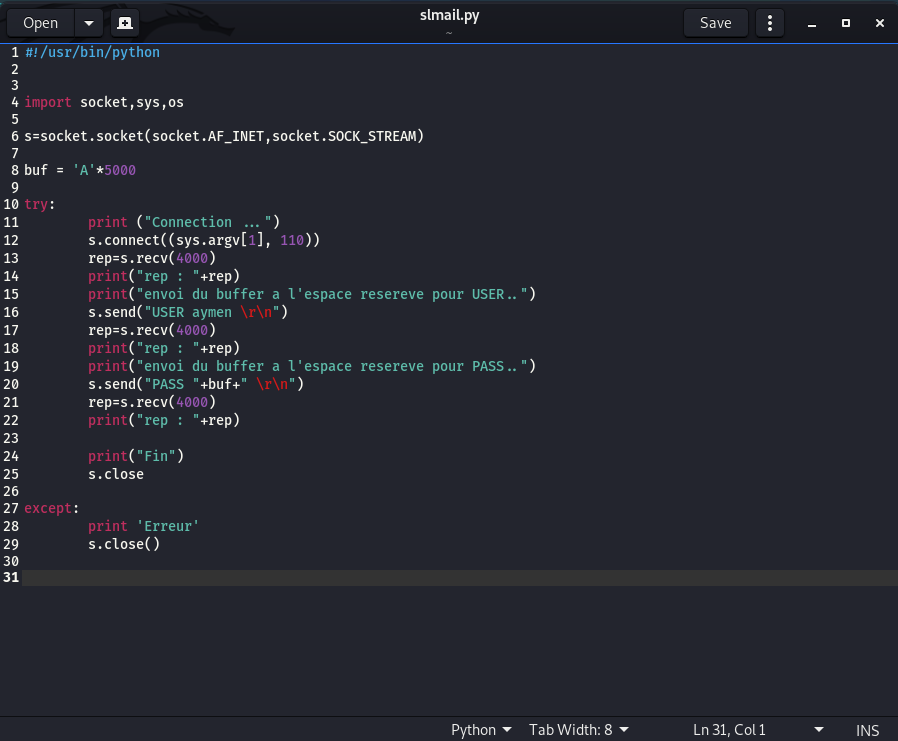
Le nouveau script est :

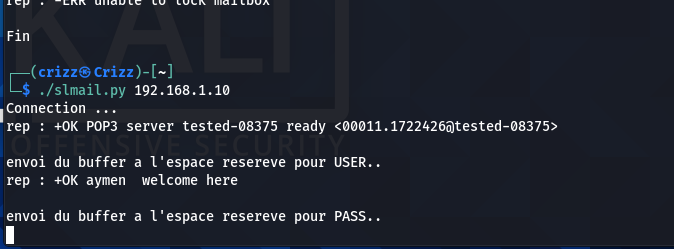


\r \n = CR LF pour le saut la ligne

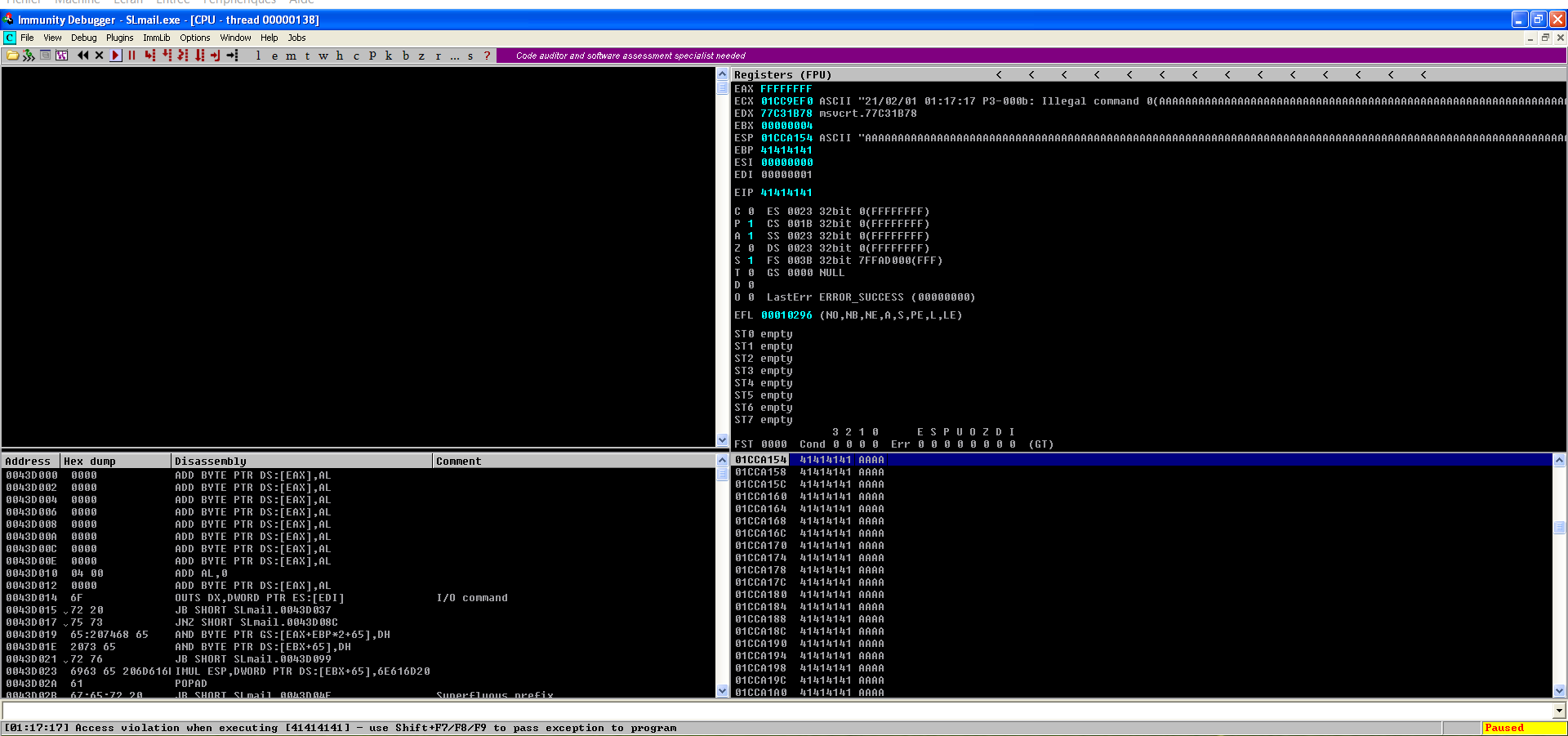


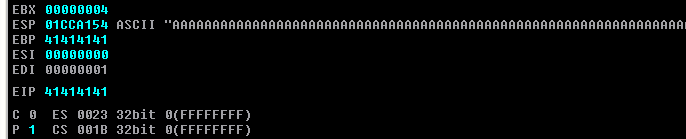
Maintenant injection et inondation du buffer :



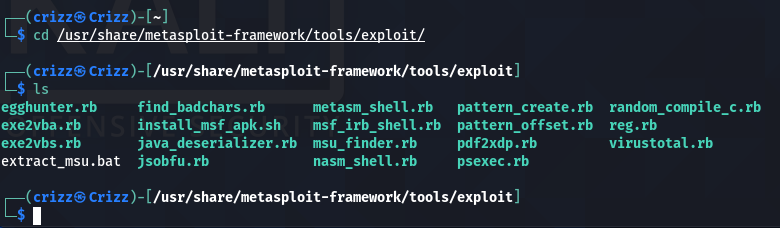


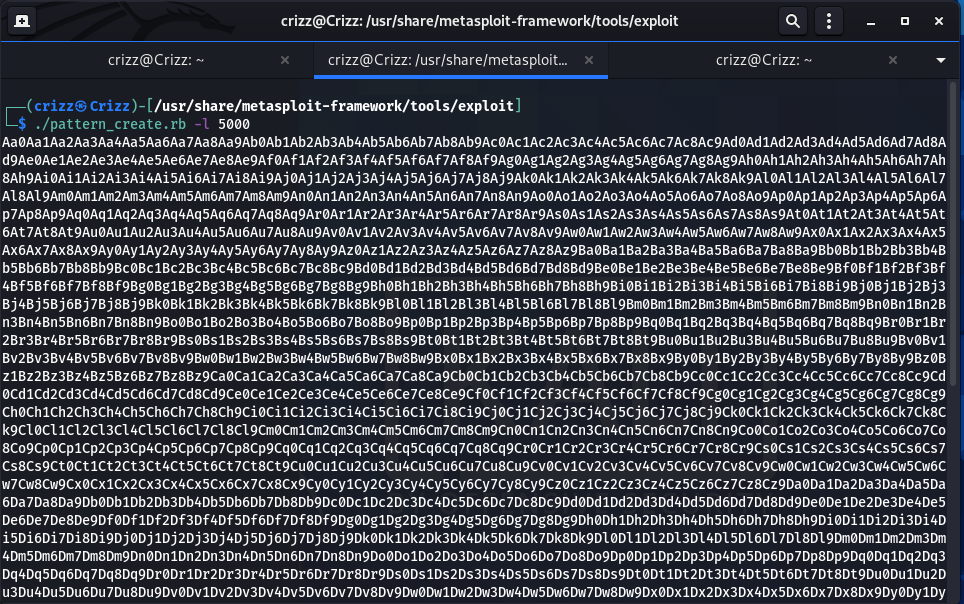
Le programme ne répond plus et dans immunity on voit que le programme est bien inondé avec le buffer déjà envoyé « La série des A \0x41 »



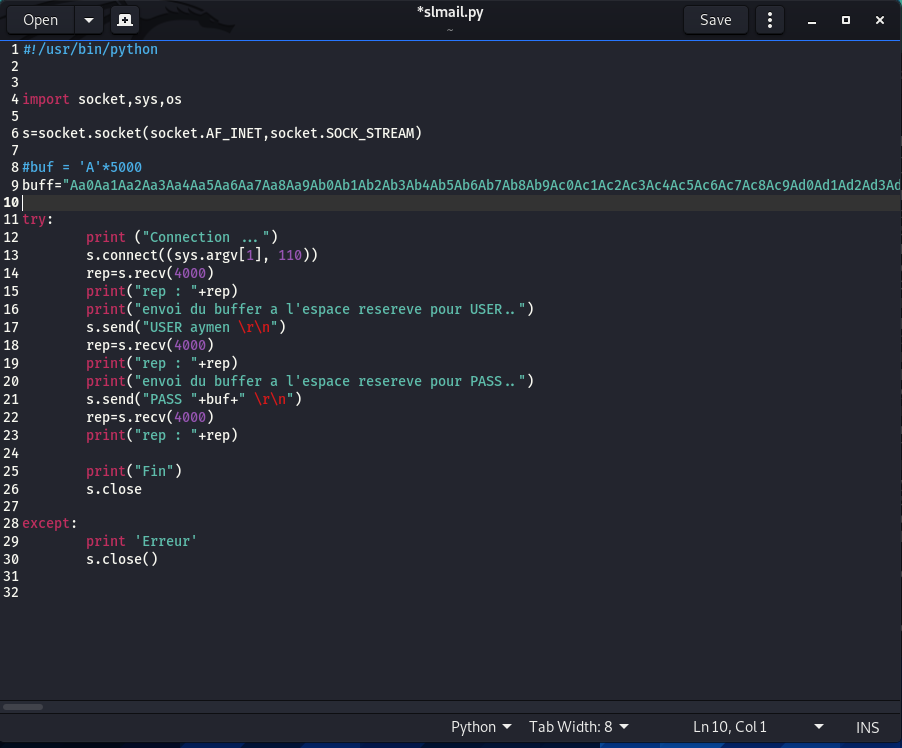


J’arrive à rediriger la valeur de EIP donc c’est possible de la redirigé vers un Shell code injecter dans le buffer mais avant ça je dois calculer le nombre d’octet à injecter avant l’écrasement de EIP pour ce faire j’utilise « pattern\_create.rb »

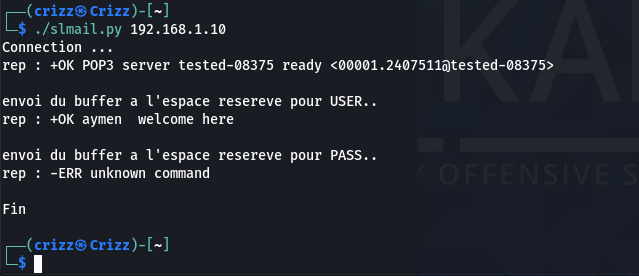




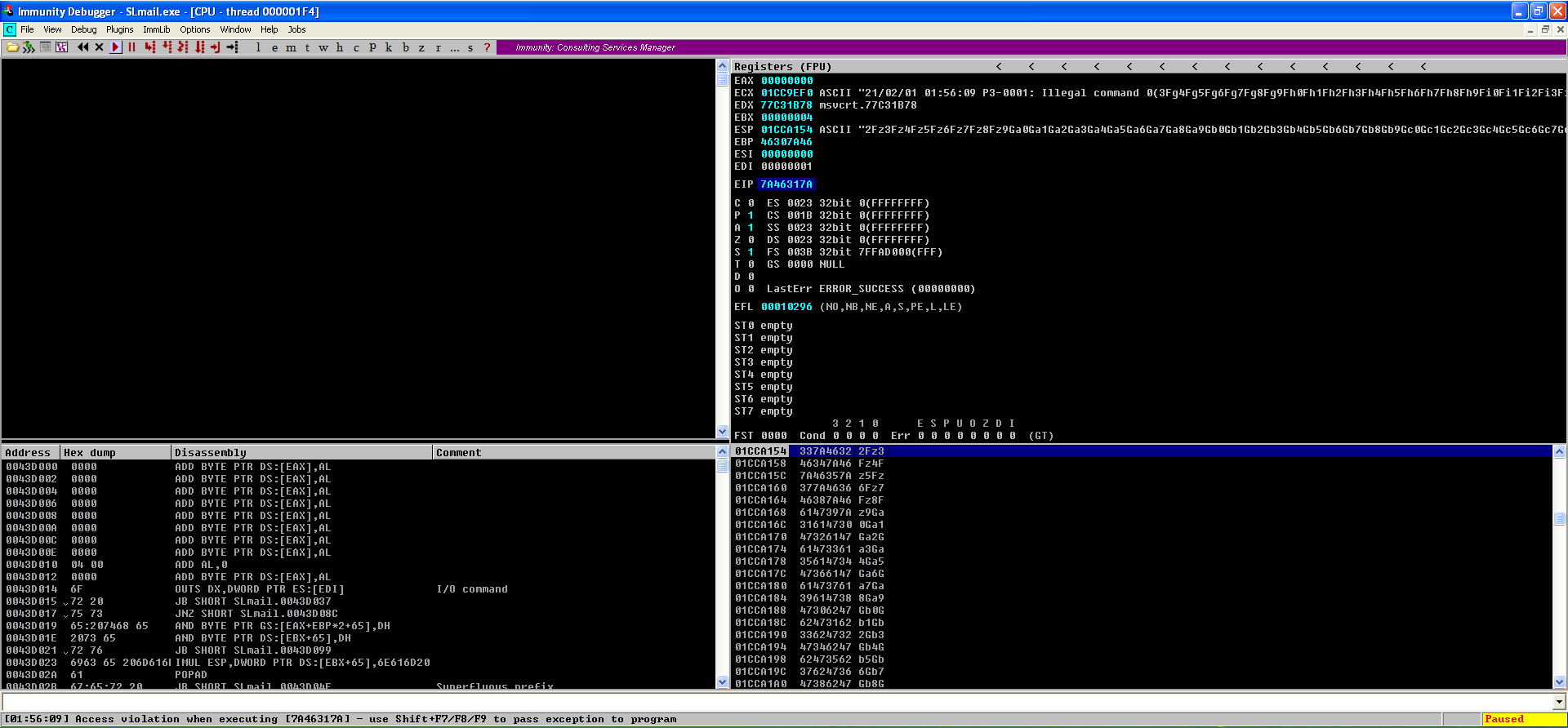
Le nouveau script est :

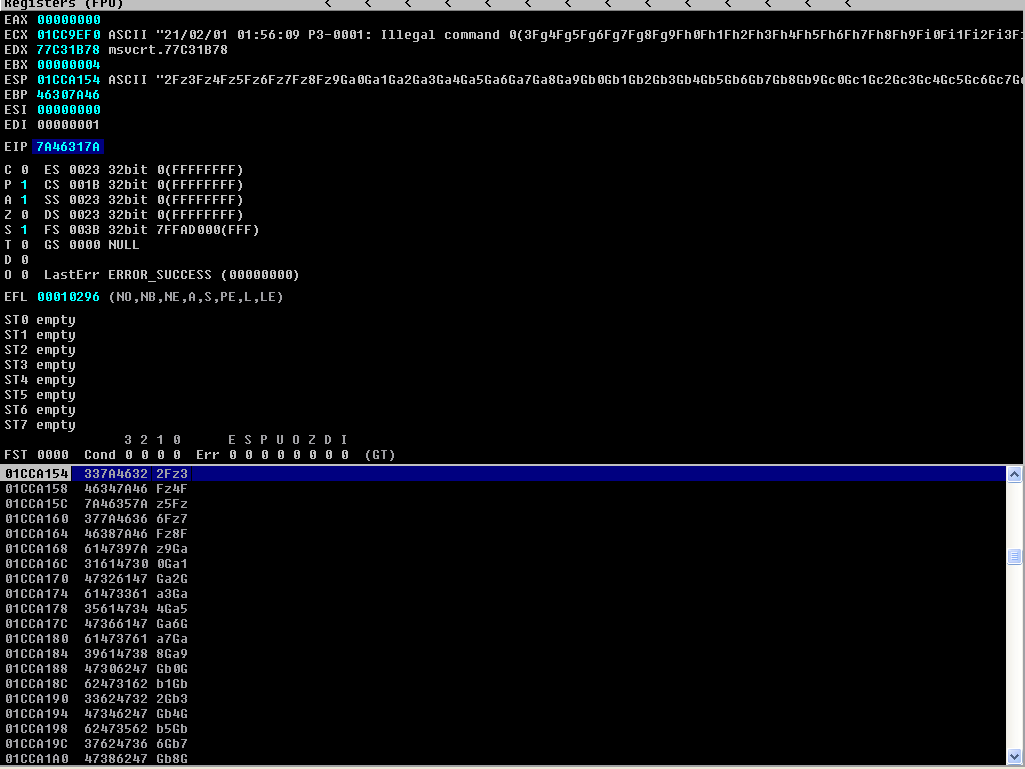


NB : Je relance toujours le programme SLMAIL et je l’attache dans immunity après chaque crashe

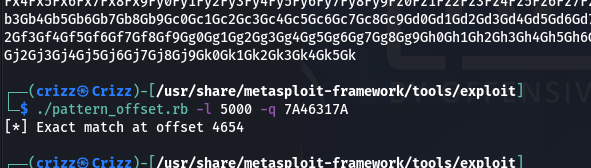


Vérification de la valeur EIP dans Immunity :





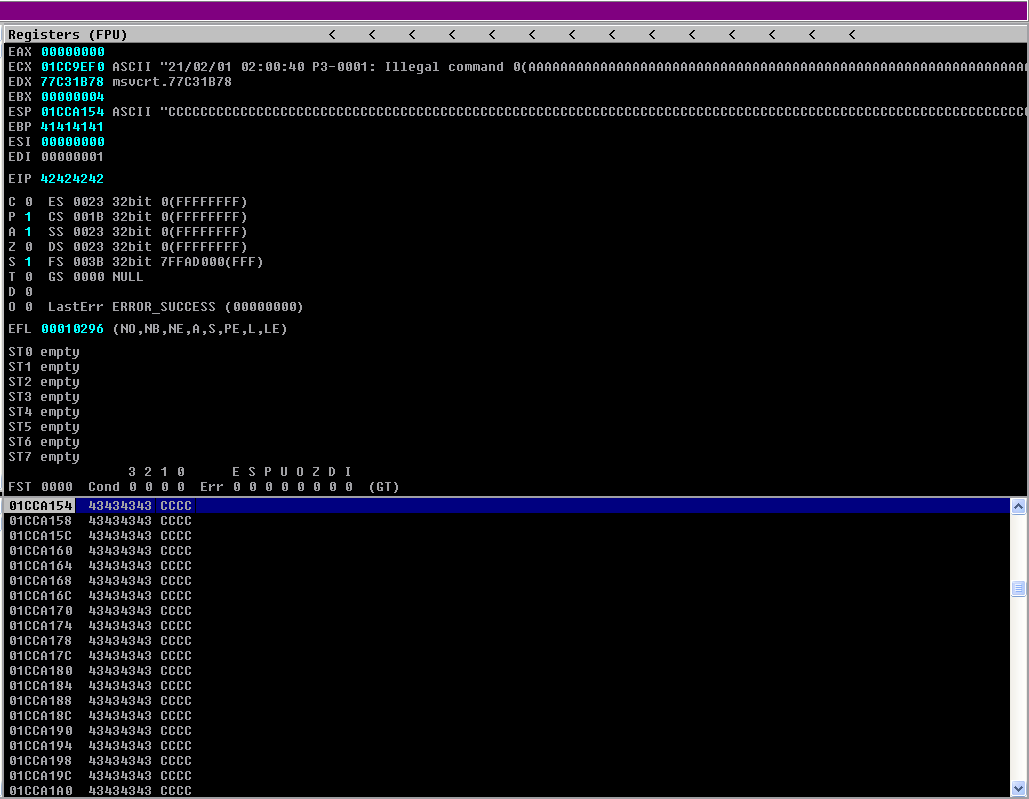
Maintenant je dois calculer l’offset en utilisant l’outil « pattern\_offset.rb » de metasploit

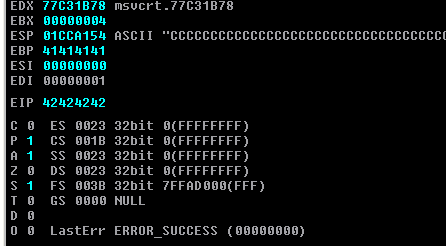


Maintenant le nouveau script devient :



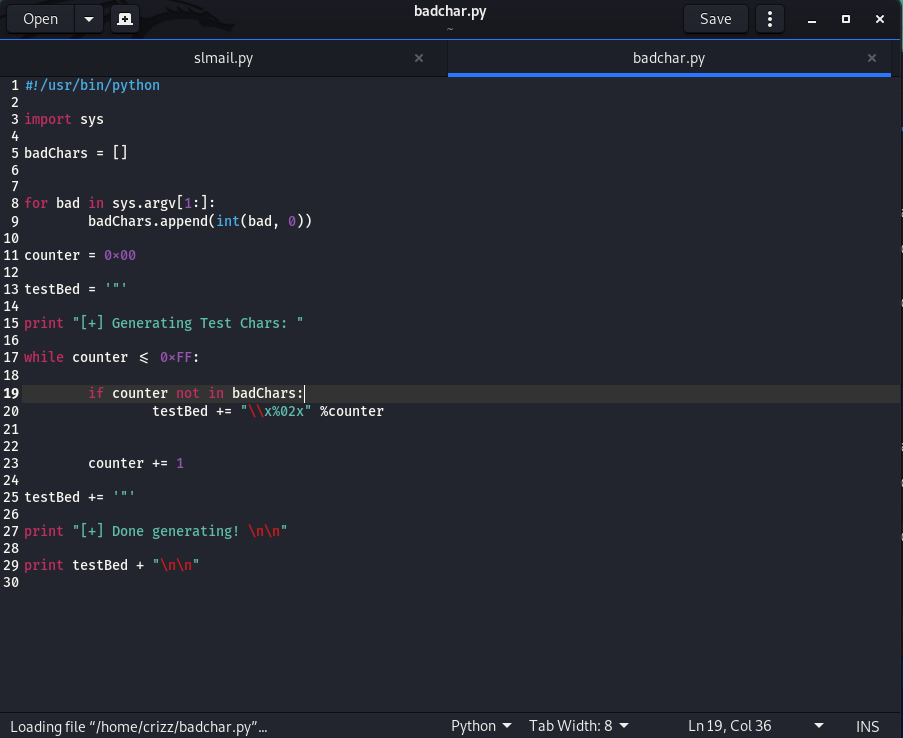
Envoi à nouveau donne :

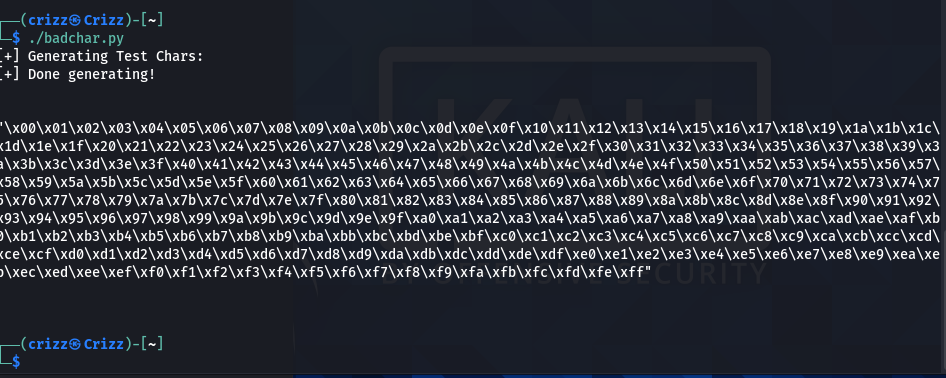


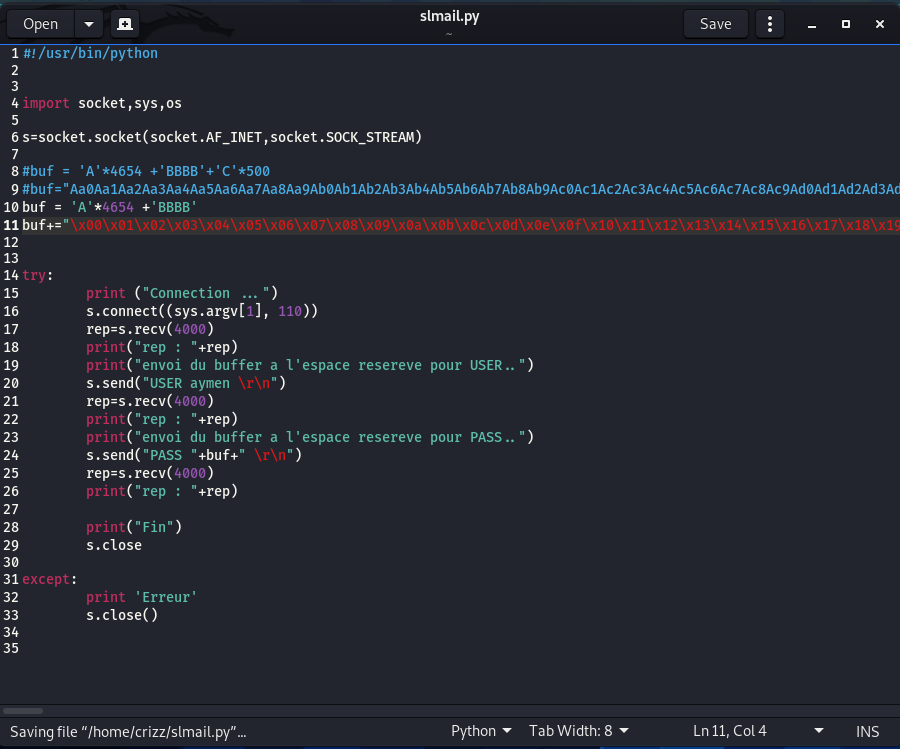


L’adresse de retour a été bien écrasée par les 4 B donc il me reste la détection des BAD CHAR afin d’injecter le Shell code approprié sans échec

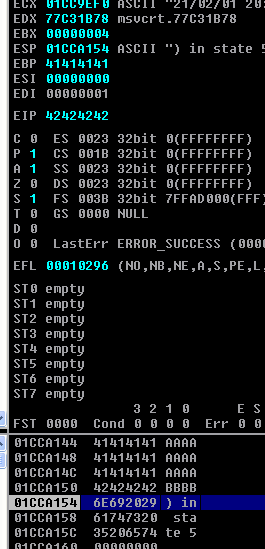
Le nouveau script de test des bad char est :





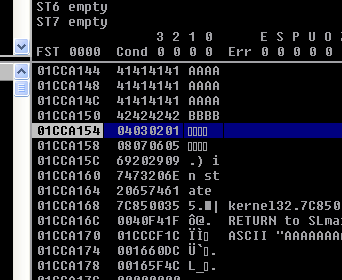


Maintenant injection et détection des BAD CHAR



On doit retrouver \x00 après les BBBB mais la chaine a été cassé donc première détection : \x00 est un BAD CHAR

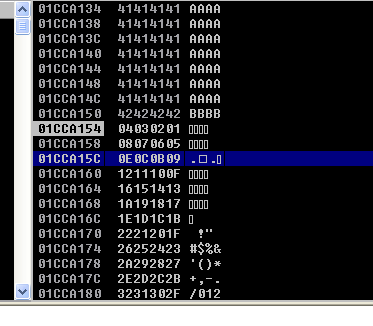
Je l’enlève de la chaine et je réinjecte mon code



Chaine encore une fois cassée, exactement on doit retrouver \x0a après \x09

Donc : \x0a est un BAD CHAR

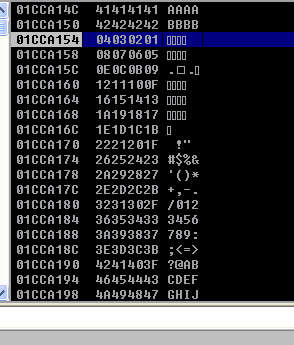
Je l’enlève de la chaine et je réinjecte ma chaine



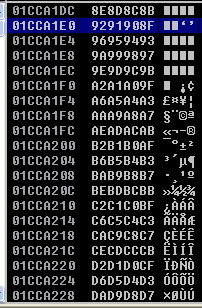
On doit retrouver le \x0D après le \x0C mais il a disparu

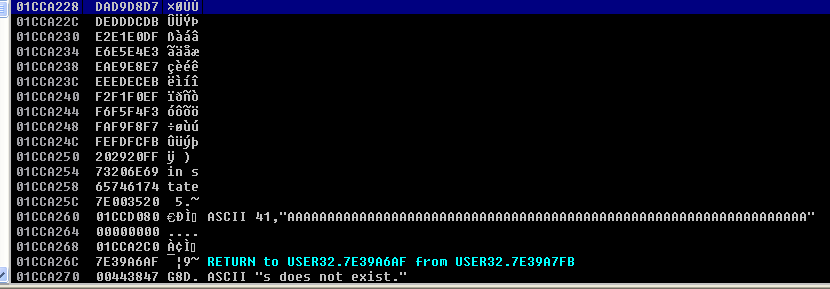
Donc : \x0D est un BAD CHAR

J’enlève \x0D est je réinjecte





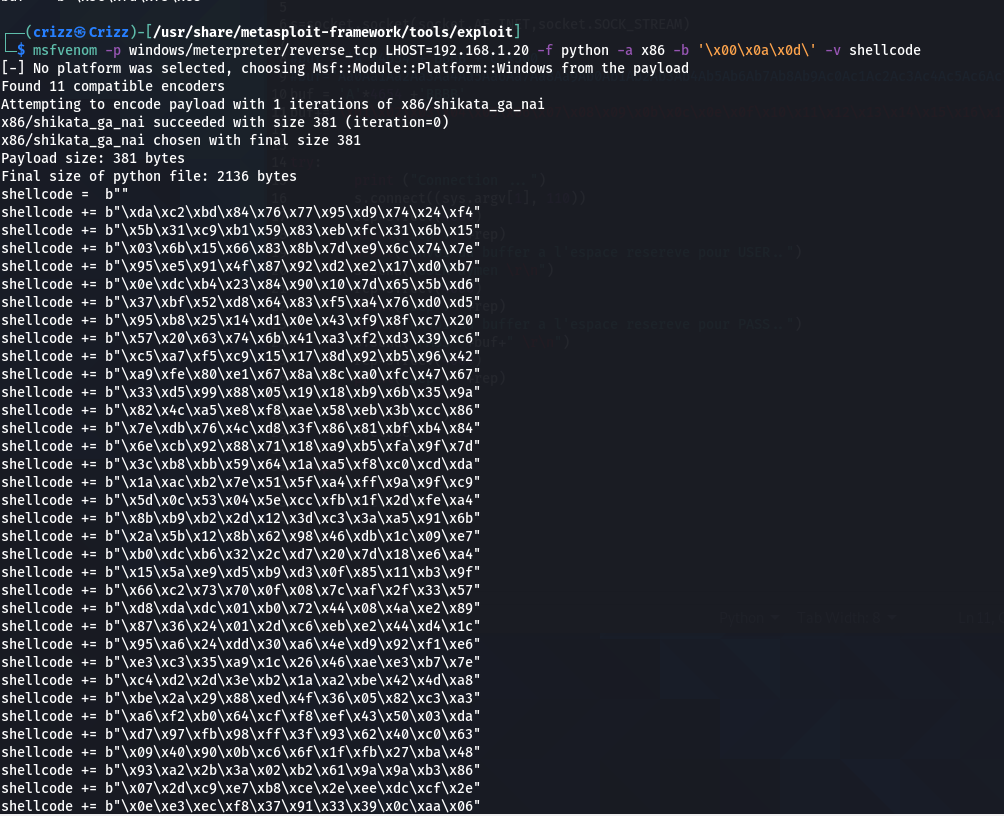




Tout la chaine des caractères est disponible sans interruption et sans caractère indisponible

Donc résultat final des Bad char est : \x00\x0a\x0d\

Maintenant je dois générer un Shell sans les Bad char et l’injecter dans mon code



Payload meterpreter pour capturer la session meterpreter ouverte par MULTI\HANDLER de Metasploit

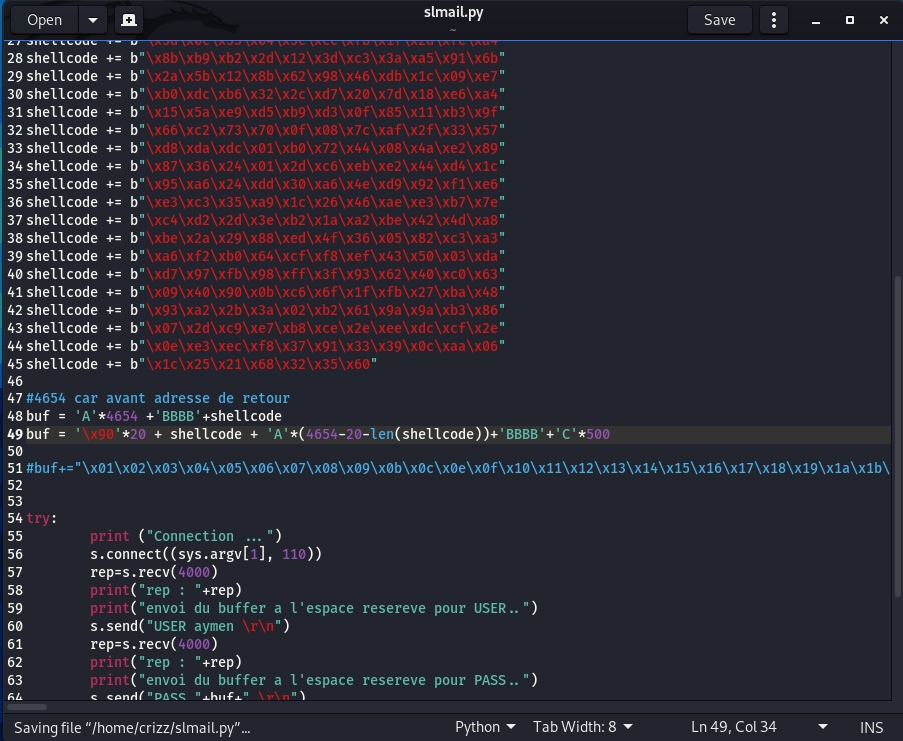
LHOST @ de la machine KALI

-f python pour un le script en python

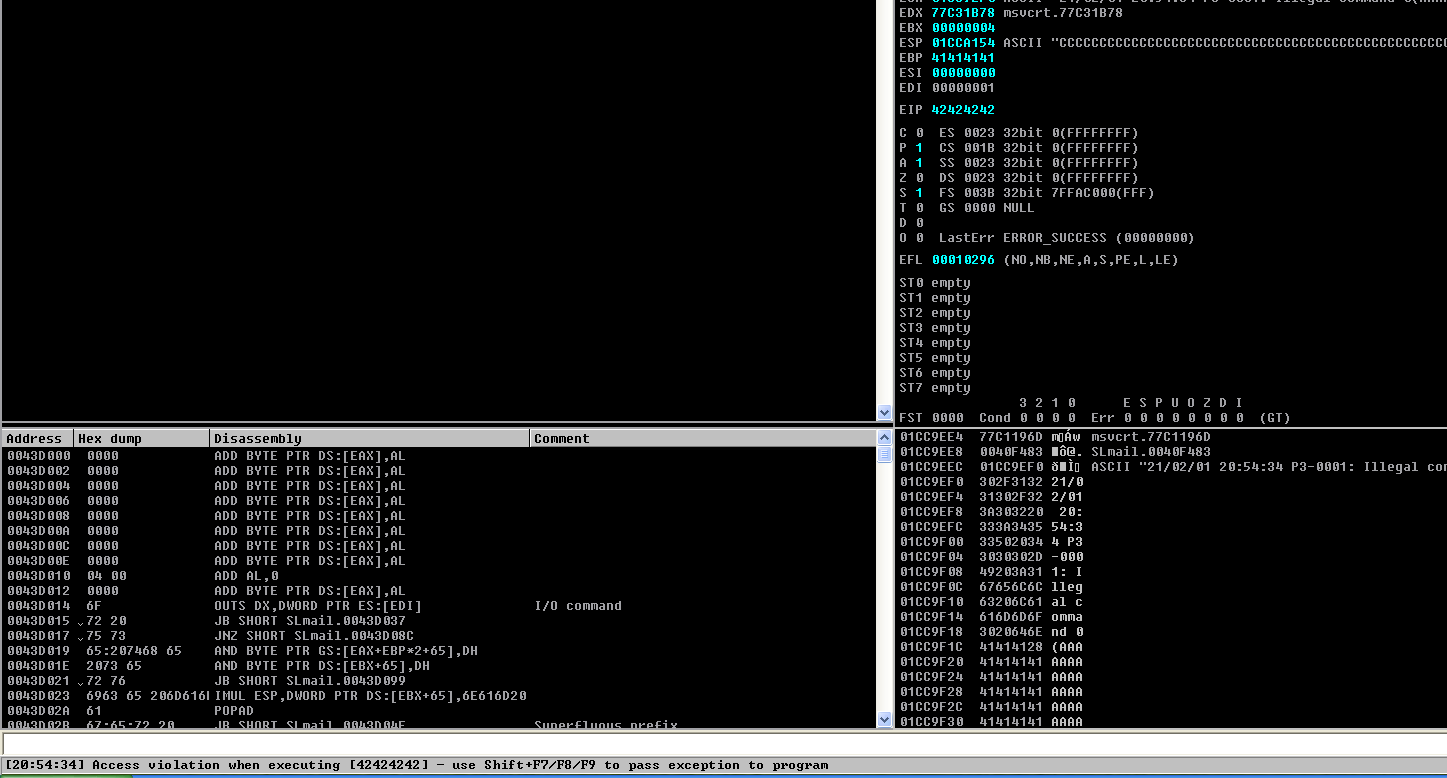
-a x86 car l’architecture de windows XP est en 32bits

-b c’est les Bad char à ne pas généré

-v pour donner un nom à la variable



Script pour tester l’injection du Shell dans le programme

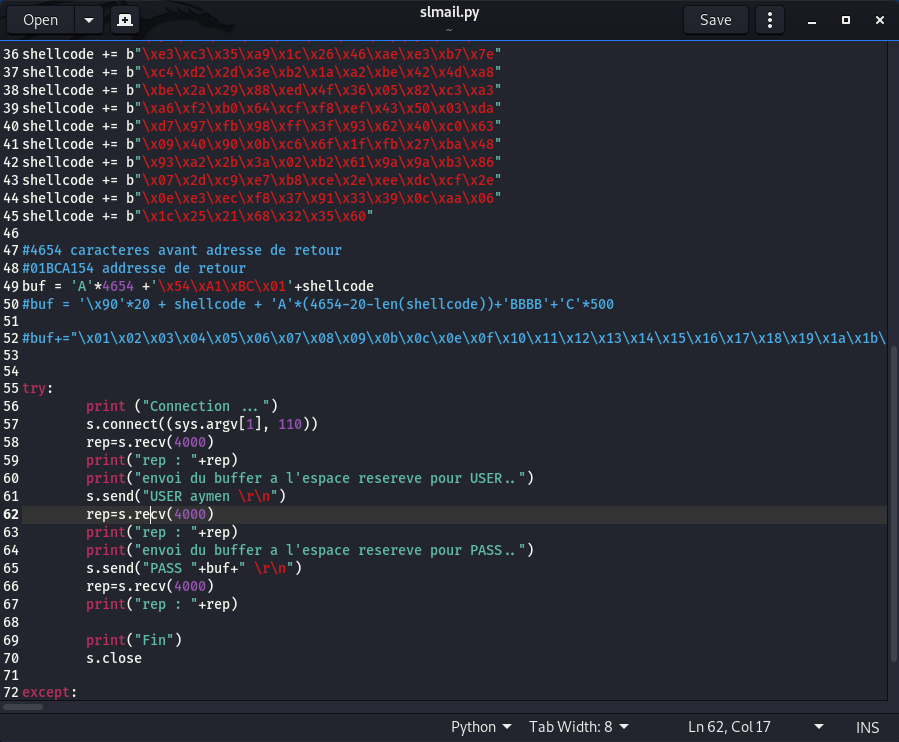


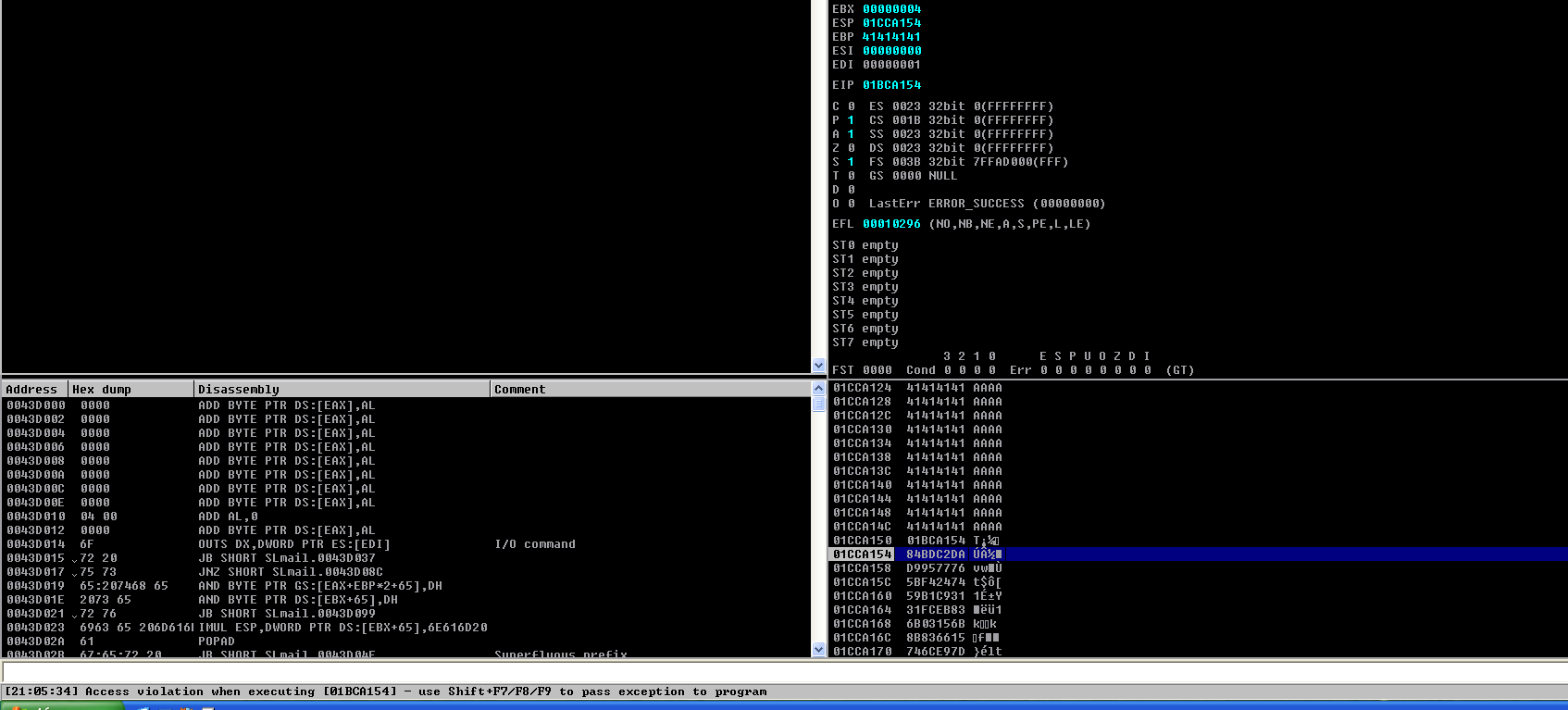
Je remarque que mon shell n’est pas disponible avant les ‘A‘

Et comme remarque je vois que SP point vers mes ‘C’

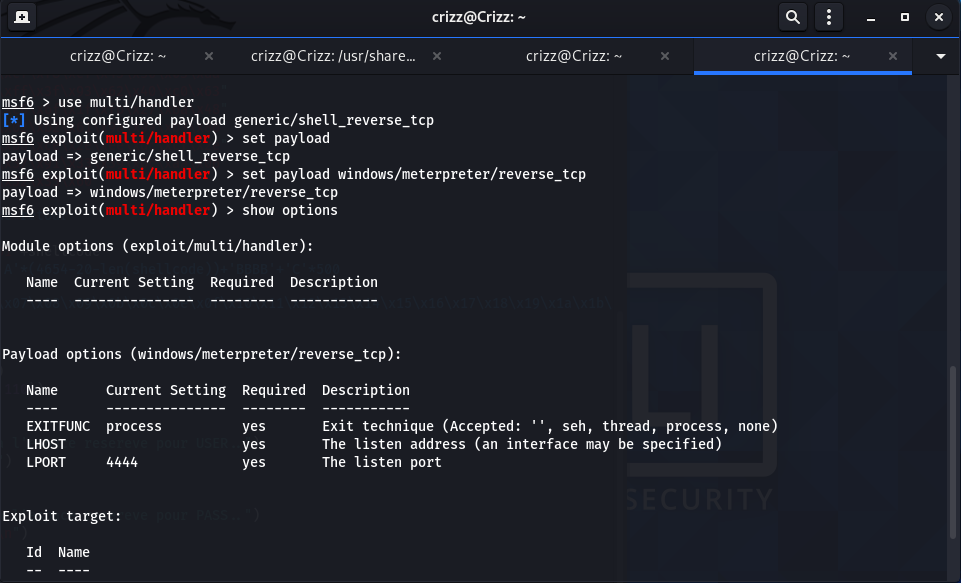
Un autre essai serai de changer l’adresse de retour qui est actuellement 4B à l’adresse de début des C et je remplace les C par mon Shell code

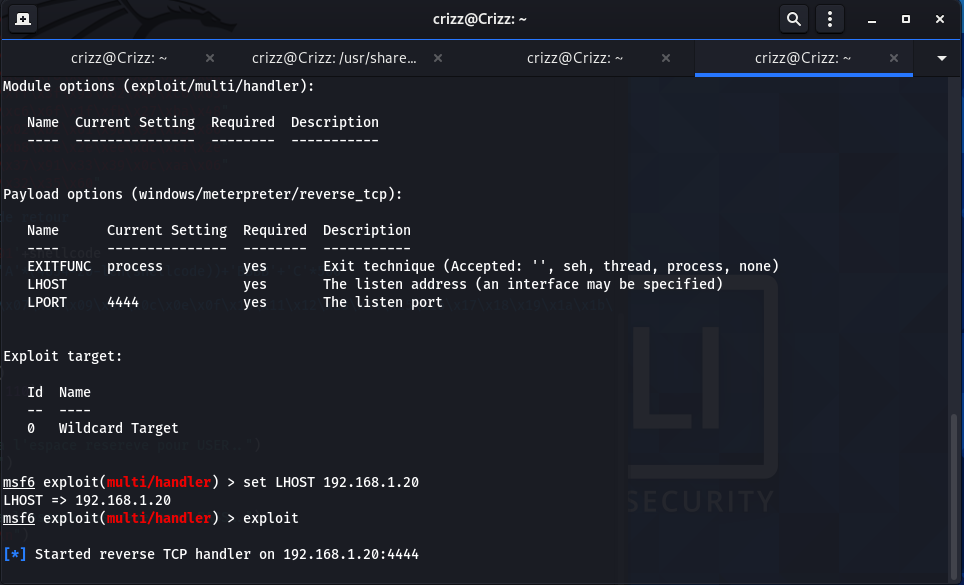
Le nouveau script serait :





La configuration de Metasploit serait :





Mais l’exploit ne marchera pas à cause de problème d’accès non autorisé vers une zone mémoire non destiné au programme « Violation d’accès mémoire »

On sait très bien que les programmes utilisent des « Bibliothèques de liens dynamiques » MODULES Systèmes‘.DLL’ lors de leurs exécutions ainsi on a remarqué qu’après écrasement, ‘SP’ Point vers mes ‘C’ injectés ou dans le deuxième cas il point vers mon Shell code (il point vers l’adresse de contenu après l’adresse de retour ), pour cela je dois trouver un modules .dll utilisé par le programmes qui valide les prérequis suivants :

1. Module .dll qui contient une commande JMP ESP pour faire sauter l’exécution de programme vers notre Shell code (on doit retrouver l’adresse de cette commande et l’injecter dans EIP
2. Module .dll non Protégé par ASLR qui rend l’adressage mémoire réservé à l’application ou module ou … dispatché dans plusieurs zones non contiguës

Ce qui rend SP point vers une autre adresse mémoire après l’exécution et le Shell code ne sera pas exécuté et il pointera vers une nouvelle @ à chaque exécution.

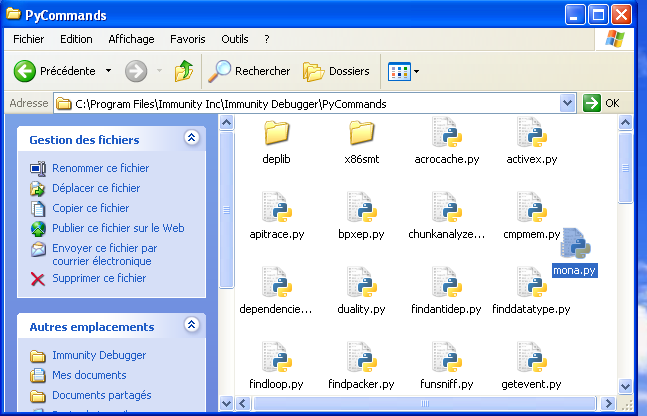
Conclusion le module sectionné doit avoir un adressage fixe

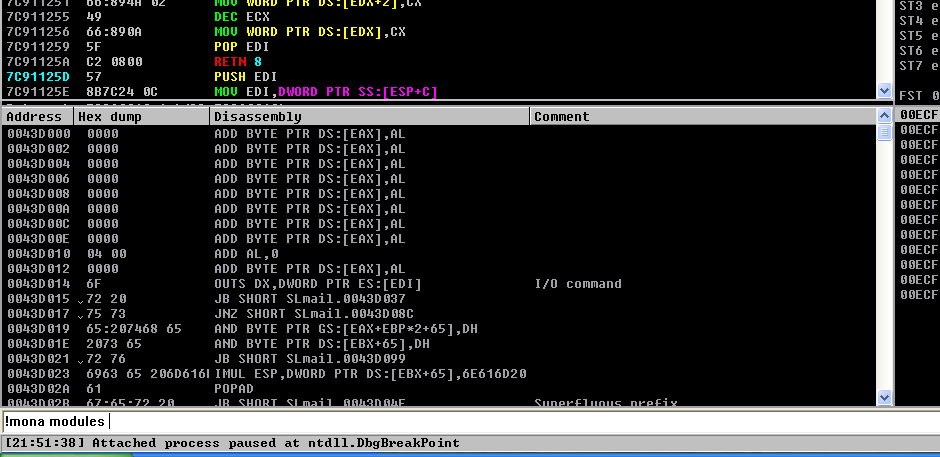
1. De préférence il vaut mieux sélectionner un DLL non protégé par tous les protections disponibles Comme SAFESEH, REBASE … etc. pour augmenter les chances de ce l’exploit

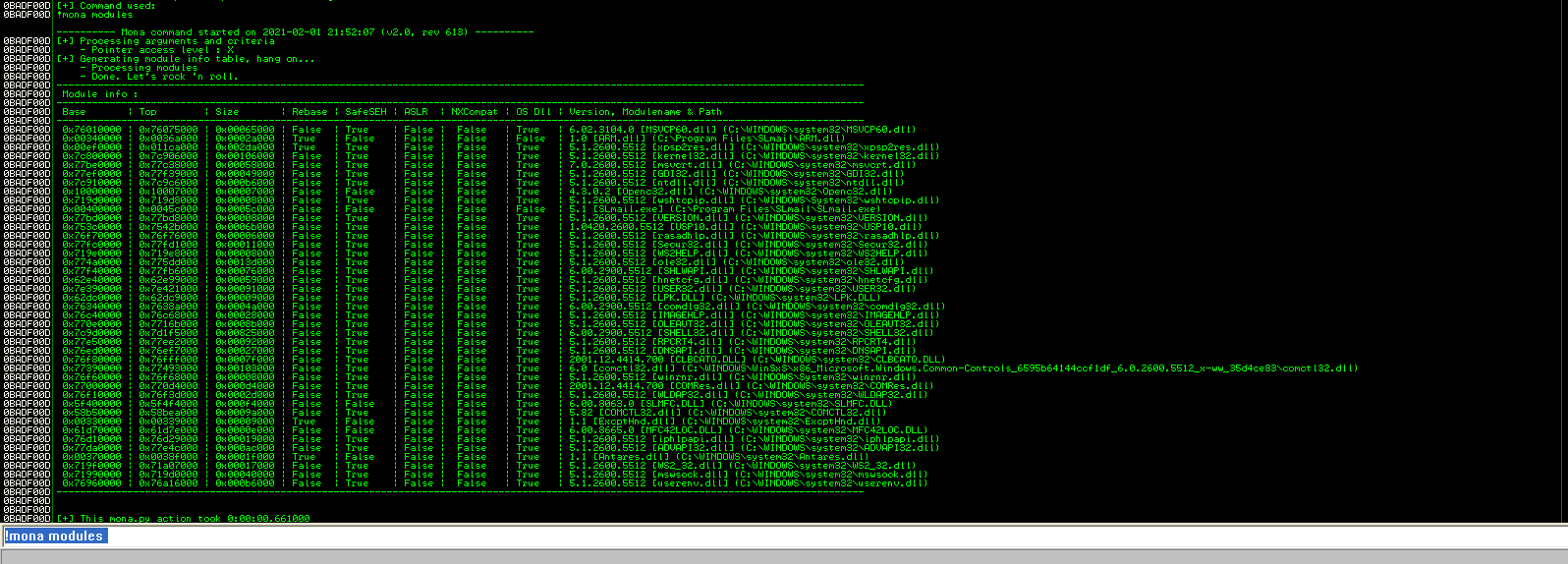
Pour une recherche rapide du module souhaité ainsi l’adresse d’une instruction JMP ESP dans ce dernier, j’ai utilisé le Script Mona.py

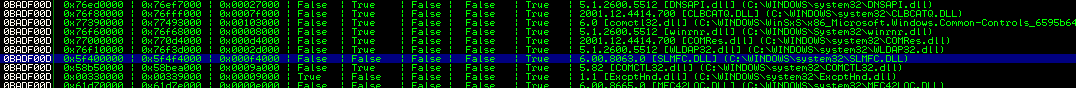
Ref : [GitHub - corelan/mona: Corelan Repository for mona.py](https://github.com/corelan/mona)

NB : Mona.py est un script python qui peut être utilisé pour automatiser et accélérer des recherches spécifiques tout en développant des exploits

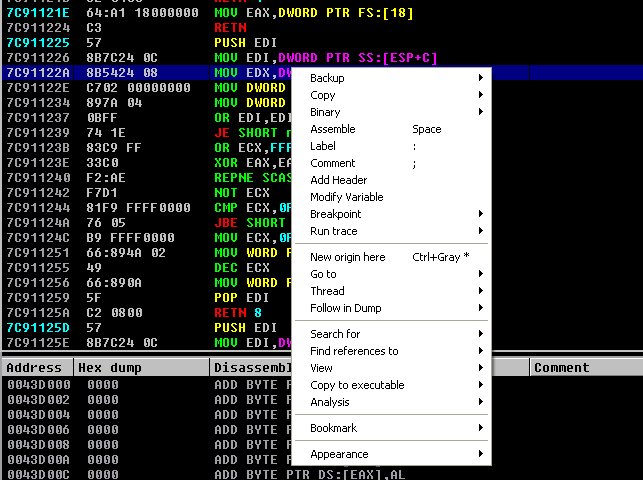








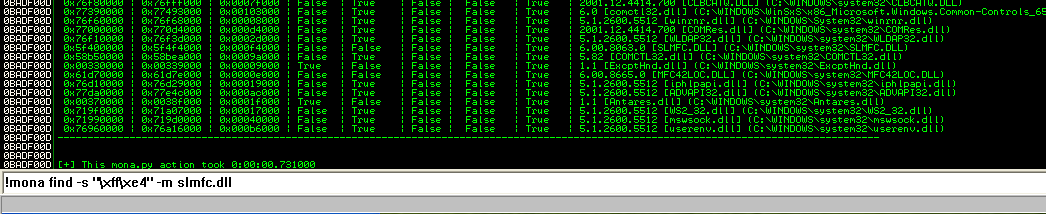
Ce module SLMFC.DLL ne contient aucune protection donc je passe vers la prochaine étape et chercher JMP ESP dans le module



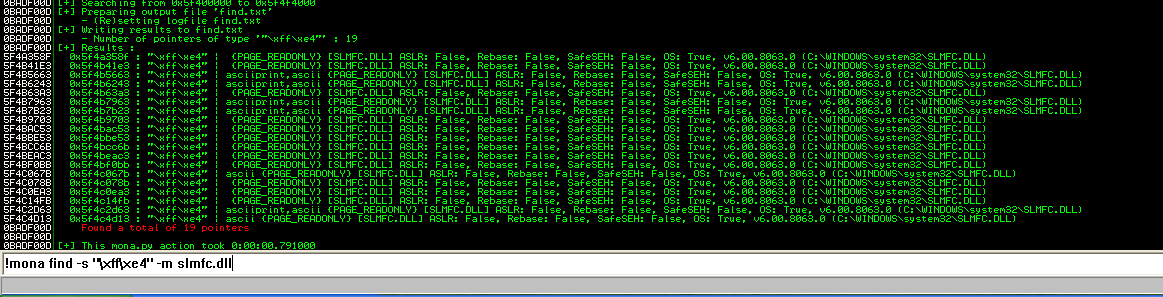




JMP ESP = \0xFF\0xE4 en code machine pour faire le chercher dans le module SLMFC.DLL



!mona find –s ‘’commande JMP ESP en code machine ‘’ –m ‘’nom Module’’



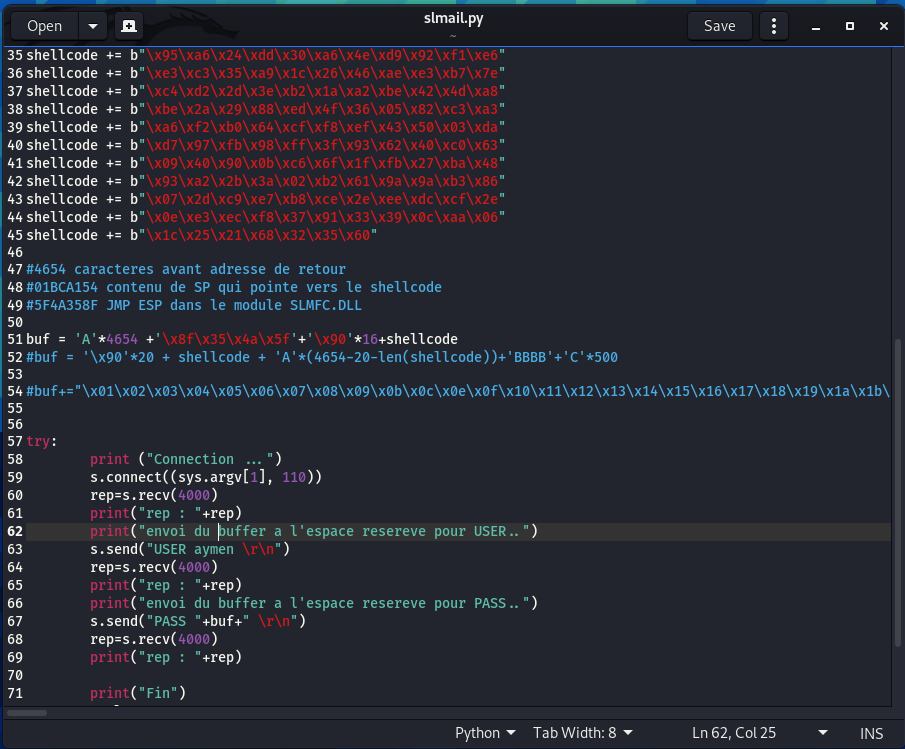
19 adresses trouvées et tous utilisable. Une seule suffit pour détourner le programme

Je prends la première par exemple : 5F4A358F

Maintenant ce qui reste est : le remplacement de l’adresse de retour dans le Shell code par cette adresse, donc EIP sera remplacer par cette adresse il va détourner le programme vers le module SLMFC.DLL exactement dans JMP ESP commande dans son code qui lui-même va retourner vers mon Shell code qui contenu dans SP

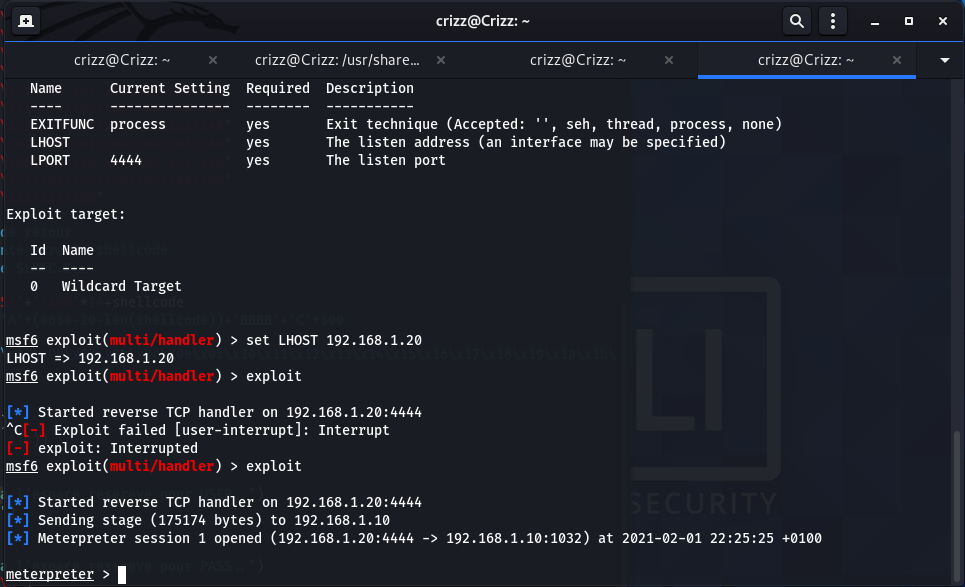
NB : l’adresse doit être mise dans le script en Little indian

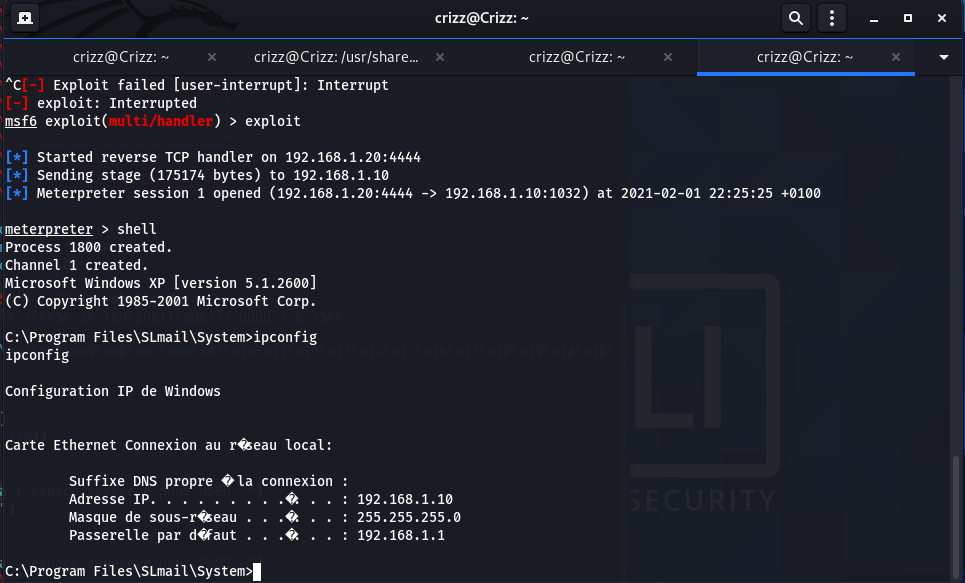
Le script final sera :



J’ai rajouté quelque NOP avant le SHELLCODE pour assurer que l'exécution ne commence pas au "milieu" de mon shellcode

Exploit exécuté avec succès





Remarque importante :avant/après exploit :

vérifier toujours que le service SLMail.exe est bien démarrer dans les processus en marche

* Vérifier dans le gestionnaire des taches
* Vérifier une connectivité normal au server pop3 par netcat :
  + nc @machine\_victime 110

Les adresses peuvent changer si vous utilisé l’application dans un autre OS ou un Windows XP avec SP différent ainsi l’architecture X86,x64 car l’adressage varie selon l’OS

L’application SLMAIL a été installé dans la machine vulnérable que vous nous avez envoyé

FIN