

Challenge SSTIC 2026 - Step 4 : dancing in shadow

nebucca

Analyst,
You did it !

We now have unfiltered access to SAFE internal system. The users database has been lost, and Aegis has removed the update feature for safety reasons. The only known user can be found in a CSPN report, jean has a copy. This user seems to be limited and cannot disarm the system.

You have already done a lot, but can you work your magic with *weapon_authent* ? It may contain a vulnerability that could allow us to elevate our privileges. Even a leak could be usefull if we were able to recover the lost database.

The report also mentionned an anti ROP protection mechanism ? We have found references to an unknown product called ShadowGuard ; leaked data may contains more informations on this solution.

Remember your access to the monitoring stream, it may prove usefull.
-ñ

Pour cette étape, nous récupérerons 2 binaires : *weapon_authent* et *weapon_supervisor* :

- *weapon_authent* qui se charge de l'authentification des utilisateurs et de la communication avec le module final,
- *weapon_supervisor* qui met en place une protection anti ROP.

1 Que fait *weapon_authent* ?

Retour à *Ghidra* et *edb*, au démarrage *weapon_authent* charge une base de données des utilisateurs en mémoire, ouvre des pipes puis attend une connexion sur le port 1515. Quand celle-ci survient, un nouveau thread est lancé et le traitement du message reçu commence.

L'effort porte sur la compréhension du format des messages attendus par le serveur, mais la gestion précise des cas d'erreur nous facilite le travail.

1.1 Désérialisation

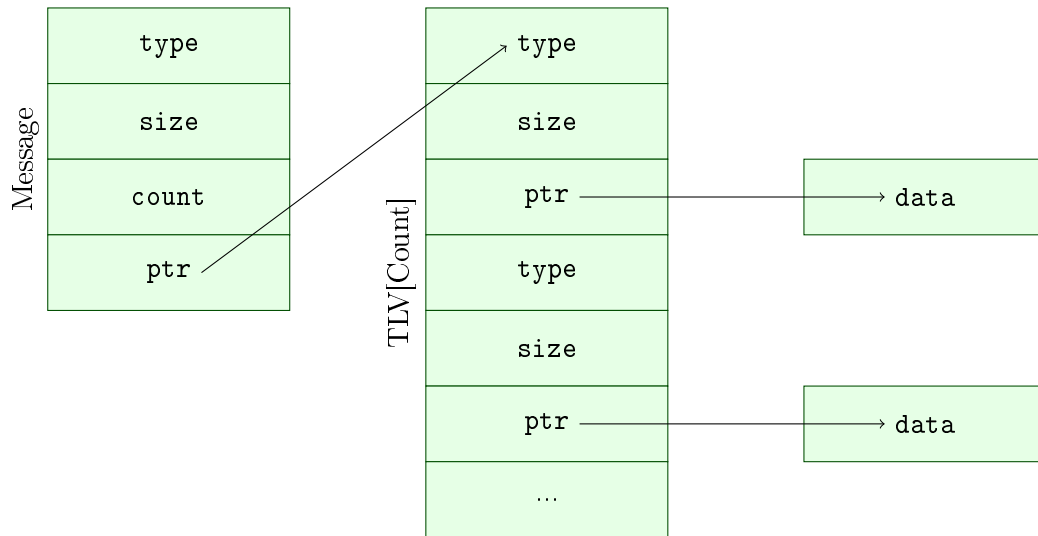
Chaque message reçu a la structure suivante :

Offset	Taille	
0x0	0x1	Type
0x1	0x2	Size
0x3	0x2	Count
0x5		TLV[Count]
Intégrité		

Chaque bloc TLV (Type-Length-Value) est composé de :

Offset	Taille	
0x0	0x1	Type
0x1	0x2	Size
0x3	Size	Data

Durant la désérialisation les données sont chargées afin de créer la structure :



Vérification de l'intégrité du message Une fois les données chargées, une vérification de l'intégrité des données peut être réalisée s'il reste des données non traitées après les TLV.

Pour chaque TLV, tous les words sont xorés ensemble puis le résultat est xoré avec le word correspondant des données restantes dans un buffer.

Le résultat doit être égal à 0 pour être considéré comme correct.

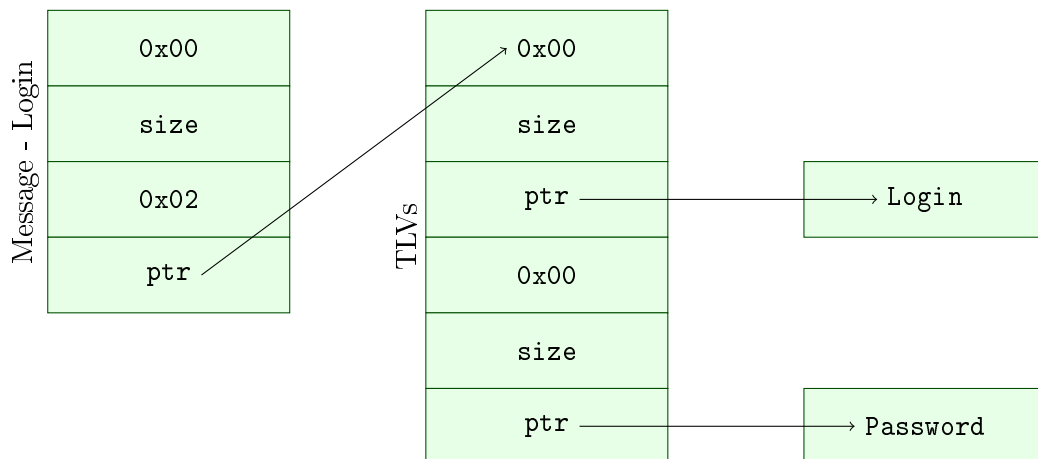
Les erreurs ne sont pas bloquantes.

1.2 Types de message

Grâce à ces messages plusieurs actions sont possibles. Avant chaque action, les droits de l'utilisateur (stockés en base) sont vérifiés.

1.2.1 Authentification

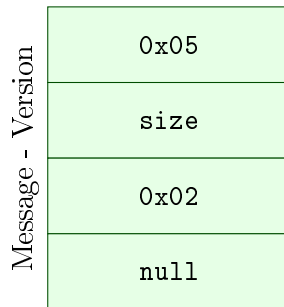
Avant de pouvoir faire autre chose, l'utilisateur doit s'authentifier.



Une fois cela fait, les droits de l'utilisateur sont récupérés dans la base.

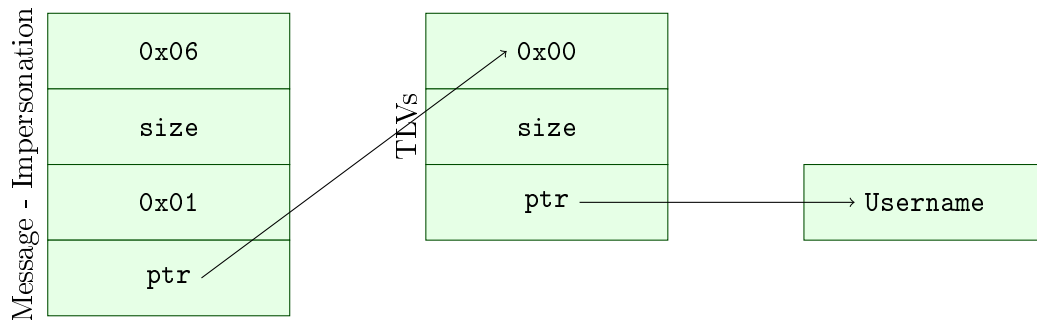
1.2.2 Version

La commande de version permet de récupérer le numéro de version du serveur.



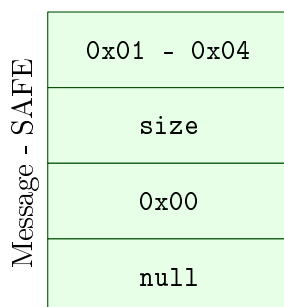
1.2.3 Impersonnification

La commande d'impersonnification permet à l'utilisateur courant de récupérer les droits d'un autre utilisateur, à la condition que les deux utilisateurs appartiennent au même groupe (cf. Step 5).



1.2.4 Communication avec SAFE

Si l'utilisateur dispose des droits suffisants, il peut envoyer des messages à SAFE.



2 Que fait *weapon_supervisor* ?

weapon_supervisor surveille *weapon_authent* via *ptrace*. L'application surveillée est exécutée en pas à pas. Chaque instruction est désassemblée.

En cas de CALL vers du code se trouvant dans la section `.text`, l'adresse est stockée sur une pile. Lorsqu'un RET est rencontré, l'adresse de retour sera comparée à celle sur la pile. Si elle ne correspond pas, l'adresse est dépilée (un message d'erreur est affiché) et comparée avec

les suivantes. Si l'adresse n'est pas trouvé, *weapon_supervisor* fait planter l'application surveillée.

En cas de CALL vers du code se trouvant en dehors de la section .text, un point d'arrêt est placé à l'adresse de retour prévue, et l'application sort du mode pas-à-pas.

3 Recherche de vulnérabilités

Nous disposons d'un seul canal d'entrée, du coup j'ai concentré ma recherche sur le code de désérialisation.

3.1 Arbitrary Free

La première vulnérabilité concerne une mauvaise gestion de la libération de la structure en cas d'erreur. En effet il est possible de provoquer un arrêt de la désérialisation si le TLV est mal formé. Cette arrêt déclenche alors une libération de la mémoire allouée pour les données, via les pointeurs présents dans le tableau de TLV évoqué plus haut. Le nettoyage utilise tous les pointeurs, même ceux non initialisés, du fait de l'arrêt. Il est possible alors de contrôler les pointeurs sur lesquels des free sont réalisés en remplissant le tcache.

3.2 Buffer overflow

Deuxième vulnérabilité, la désérialisation propose une vérification de l'intégrité des données en stockant le résultat du xor dans un buffer. Une erreur dans la gestion de la taille fait qu'il est possible de dépasser les limites du buffer jusqu'à pouvoir écrire sur l'adresse de retour stockée sur la stack.

4 Réflexions

4.1 Proposition 1 : Arbitrary Free + GOT + ROP

Ma première idée était :

- exploiter la vulnérabilité concernant les free pour réécrire la *.got.plt* et transformer l'appel à une fonction pour sauter directement à l'instruction suivant le retour et éviter le point d'arrêt. Et échapper ainsi à la surveillance du *weapon_supervisor*,
- utiliser le buffer overflow pour mettre en place une chaîne ROP et appeler la fonction d'affichage du numéro de version

4.2 Proposition 2 : Buffer overflow

Ma seconde idée : puisque le superviseur autorise un retour vers n'importe quelle adresse de la pile, pourquoi ne pas tenter de sauter directement à la fonction précédente à celle qui nous a appelés. Le superviseur le permet. Par contre la stack de l'application est un peu malmenée (le RSP n'est plus là, où il devrait se trouver).

Heureux hasard, nous avons un certain contrôle sur les valeurs présentes dans la stack puisque le RSP pointe alors sur le buffer contenant le nom de l'utilisateur. On est alors en capacité d'appeler les fonctions qui nous intéressent (affichage du numéro de version) avec les paramètres que nous souhaitons.

5 Exploitation

J'ai finalement choisi d'exploiter ma deuxième piste qui me semblait plus directe pour récupérer la base de données en mémoire.

5.1 Régler le problème de l'ASLR

La protection anti-ROP impose que l'adresse de retour fasse partie des adresses déjà présentes dans la pile des adresses de retour connues. Nous allons utiliser le buffer overflow, pour ré-écrire l'adresse de retour afin de retourner directement à la fonction parente.

- ThreadClient
- ReadTLV (ret 0x000055555555619d)
- DeserializeTLV (ret 0x0000555555557484)

La vulnérabilité permet de réaliser un xor de l'adresse sur la stack avec une valeur de notre choix. La xor distance entre 0x000055555555619d et 0x0000555555557484 vaut 0x01519. Les bits de poids forts n'ont pas besoin d'être modifiés car les fonctions appartiennent à la même section. Et du fait de l'alignement de la section en mémoire (0x1000), les bits de poids faibles sont constants malgré l'ASLR. Au final seul 0x000000000000?000 est indéterminé. Ainsi le fait de xor l'adresse sur la pile avec 0x01519 a de grandes chances de pointer vers l'adresse que l'on souhaite.

En cas d'échec, la protection anti-ROP redémarre l'application... et on retente.

Sinon, la protection anti-ROP, détecte l'anomalie, affiche un message indiquant l'adresse qui était attendue, mais n'interrompt pas l'exécution.

Autre conséquence En sautant directement à la fonction précédente on empêche l'exécution normale du prologue de la fonction :

ADD	RSP, 0x18
POP	RBX
POP	RBP
RET	

Ainsi 4 registres ne seront pas restaurés comme prévus : RSP, RBP, RBX et RPI. La pile se retrouve décalée et ce décalage provoque un plantage de l'application lors de l'arrêt du thread.

5.1.1 Éviter le plantage

Utilisons *gdb* afin d'obtenir la callstack lors du plantage :

```
Segmentation fault.
[Switching to Thread 0x7ffff78866c0 (LWP 19901)]
0x00007ffff7079a10 in ?? () from /lib/x86_64-linux-gnu/libgcc_s.so.1
(gdb) bt
#0  0x00007ffff7079a10 in ?? () from /lib/x86_64-linux-gnu/libgcc_s.so.1
#1  0x00007ffff707ac8a in ?? () from /lib/x86_64-linux-gnu/libgcc_s.so.1
#2  0x00007ffff707b3c0 in _Unwind_ForcedUnwind () from /lib/x86_64-linux-gnu/libgcc_s.so.1
#3  0x00007ffff792f7a4 in __GI__pthread_unwind (buf=<optimized out>) at
    ./nptl/unwind.c:130
#4  0x00007ffff7927d22 in __do_cancel () at ../sysdeps/nptl/pthreadP.h
    :271
#5  __GI__pthread_exit (value=0x0) at ./nptl/pthread_exit.c:36
```

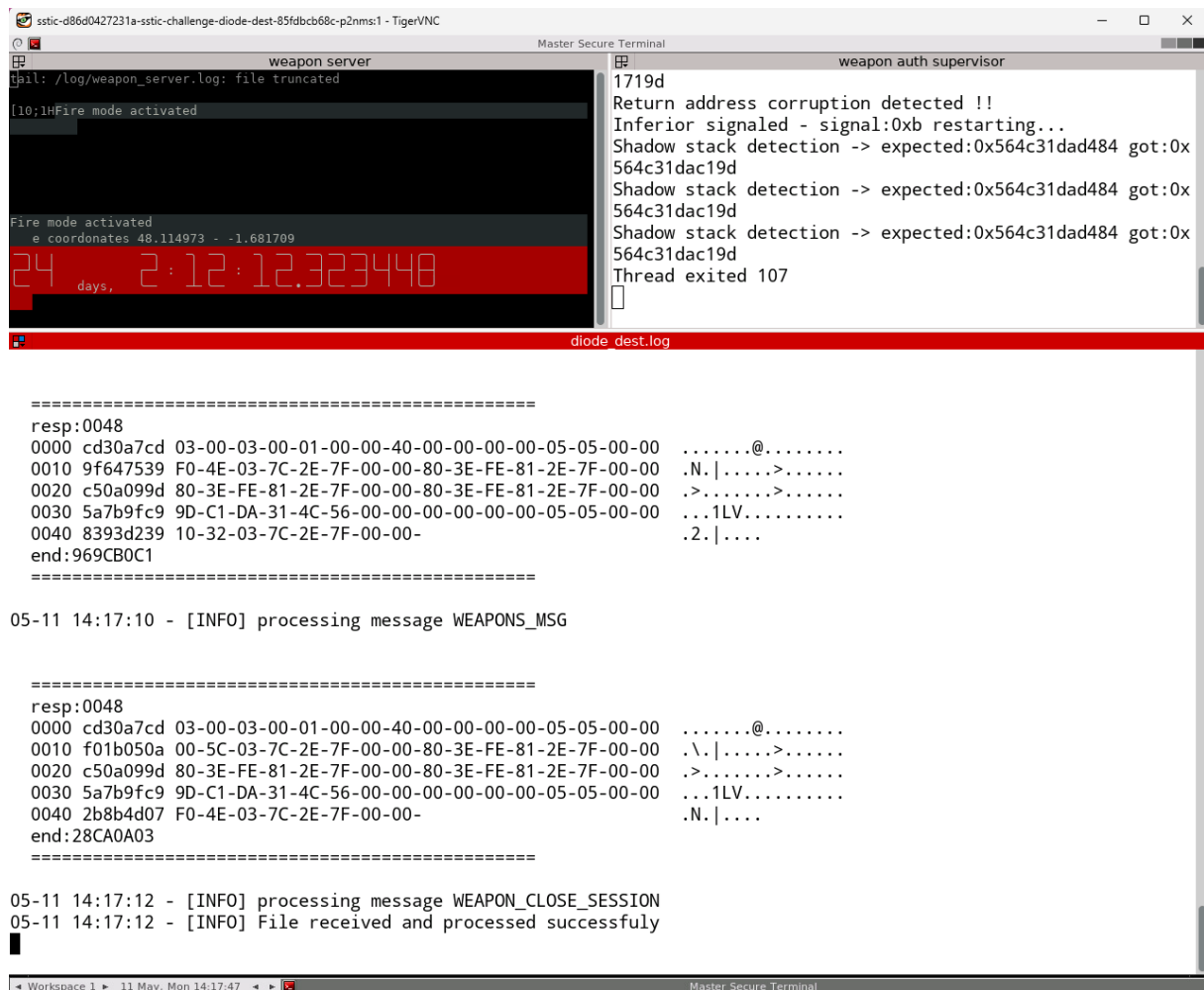


FIGURE 1 – Leak de l'adresse

L'appel à `thread_exit` provoque un déroulage de la pile qui entraîne un plantage de l'application.

Dans les sources de la `libgcc`, on observe :

- `_Unwind_ForcedUnwind()` appelle `_Unwind_ForcedUnwind_Phase2()` qui se charge du déroulage frame à frame,
- `_Unwind_ForcedUnwind_Phase2()` appelle `uw_frame_state_for()` pour récupérer les informations liées à la frame courante

Une condition d'arrêt du déroulage de la pile par `_Unwind_ForcedUnwind_Phase2()` est l'atteinte de la fin de la pile (`_URC_END_OF_STACK`).

Dans `uw_frame_state_for` justement :

```

if (context->ra == 0)
    return _URC_END_OF_STACK;

```

Pour provoquer l'arrêt du traitement, il suffit donc que l'adresse de retour lue sur la stack soit nulle.

Nous ne contrôlons pas le contenu de la stack qui est récupéré, mais en enchainant plusieurs appels "DeserializeTLV + buffer-overflow + ret ThreadClient" nous pouvons faire en sorte que l'adresse de retour sur la stack soit à 0.

Stack normale	Stack après 3 buffer-overflow
53555f4349545353 SSTIC_US	0000050500000000
0000000000005245 ER.	00007ffff0024600 .F. [] . . .
0000000000000000	00007ffff7885e70 p^ . [] . . .
0000000000000000	00007ffff7885e70 p^ . [] . . .
0000000000000000	00007ffff7885e70 p^ . [] . . .
0000000000000000	000055555555619d .aUUUU. . . .
0000000000000000	0000050500000000
0000000000000000	00007ffff00238f0 8. [] . . .
000055555555803c <.UUUU. . . .	00007ffff7885e70 p^ . [] . . .
0000000000000008	00007ffff7885e70 p^ . [] . . .
0000000000000000	00007ffff7885e70 p^ . [] . . .
00007ffff000bd0 [] . [] . . .	000055555555619d .aUUUU. . . .
0000000000000000	0000050500000000
0000000000000000	00007ffff0022bd0 []+ . [] . . .
0000000000000001	00007ffff7885e80 .^ . [] . . .
6c9424506b262100 .!&kP\$.l . . .	00007ffff7885e80 .^ . [] . . .
00007ffff7886cdc []l. [] . . .	00007ffff7885e80 .^ . [] . . .
00007ffff7885f70 p_. [] . . .	000055555555619d .aUUUU. . . .
00007ffff78866c0 []f. [] . . .	53555f4349545353 SSTIC_US
ffffffffffffffff88 .[] [] [] [] . . .	0000000000005245 ER.
0000000000000000	0000000000000000
00007ffff7926aa4 []j. [] . . .	0000000000000000

FIGURE 2 – État de la stack

5.1.2 Trouver l'adresse de la base de données

Durant l'initialisation de l'application, lorsque la base de données des utilisateurs est chargée en mémoire, son adresse est stockée dans la section .bss. Son adresse relative par rapport à la section .text est constante. Grâce au message d'erreur de la protection anti-ROP, nous connaissons l'adresse de la section .text en mémoire. Nous pouvons donc calculer son adresse exacte en mémoire.

5.2 Afficher les données

La fonction de récupération du numéro de version semble un bonne candidate pour récupérer des informations.

```
lea rbx, [rsp+0x60]
mov esi, [rsp+0x48]
mov rdi, [rsp+0x40]
mov rdx, rbx
call 0x55555555a20
```

Elle utilise 3 paramètres :

- RDI : pointeur vers la chaîne,
- RSI : taille de la chaîne,
- RDX : pointeur vers le buffer qui accueillera la réponse.

Nous ne pouvons pas modifier directement les valeurs sur la stack, mais si on précède l'appel de la fonction d'affichage de version d'un décalage de la pile comme précédemment, les paramètres sur la stack se retrouvent alors dans le buffer stockant le nom de l'utilisateur transmis lors de l'authentification. Si lors de l'authentification nous utilisons, comme nom d'utilisateur, une chaîne de la forme :

Stack normal			Stack après BO + username		
53555f4349545353	SSTIC_US		0000050500000000	
00000000000005245	ER.....		00007ffff0022c40	@, . []...	
0000000000000000		00007ffff7885e80	..^ []...	
0000000000000000		00007ffff7885e80	..^ []...	
0000000000000000		00007ffff7885e80	..^ []...	
0000000000000000		000055555555619d	..aUUUU..	
0000000000000000		53555f4349545353	SSTIC_US	
0000000000000000		0504030201005245	ER.....	
000055555555803c	<.UUUU..	ASCII "0.1.1.3" rdi	0d0c0b0a09080706	... ^...	
0000000000000008	esi	1514131211100f0e	
0000000000000005	msg type	1d1c1b1a19181716	
00007ffff000bd0	[]^ []...	rdx	2524232221201f1e	.. !"#\$%	
0000000000000000		2d2c2b2a29282726	&'()*+,-	
0000000000000000		3534333231302f2e	./012345	

FIGURE 3 – État de la stack lors de l'appel

'Username\x00 valeurs-de-notre-choix', il est alors possible de contrôler : l'adresse de la chaîne à afficher, sa taille et le type du message.

À nouveau, pour ne pas provoquer un crash de l'application, l'enchaînement de plusieurs décalage de la stack permet de faire pointer l'adresse de retour sur une adresse qui n'entraîne pas de plantage :

```
fde = _Unwind_Find_FDE (context->ra + _Unwind_IsSignalFrame (context)
- 1, &context->bases);
if (fde == NULL) { return _URC_END_OF_STACK; }
```

5.2.1 Récupérer le flag

Avec cette méthode nous pouvons afficher le contenu de la section .bss. Cela permet de récupérer l'adresse de la base en mémoire.

Nous utilisons à nouveau cette méthode pour afficher le contenu se trouvant à l'adresse que nous venons de trouver.

Le flag est là, au début de la zone.


```
ssstic-d86d0427231a-ssstic-challenge-diode-dest-85fdbcb68c-p2nms:1 - TigerVNC
Master Secure Terminal

weapon server
mail: /log/weapon_server.log: file truncated
[10;1HFire mode activated

Fire mode activated
fire coordinates 48.114973 - -1.681709
24 days, 2:08:44.123080

weapon auth supervisor
564c31dac19d
Shadow stack detection -> expected:0x564c31dad484 got:0x
564c31dac19d
Shadow stack detection -> expected:0x564c31dad484 got:0x
564c31dac19d
Shadow stack detection -> expected:0x564c31dad484 got:0x
564c31dac19d
Thread exited 125
Shadow stack detection -> expected:0x564c31dad484 got:0x
564c31dac19d

diode_dest.log
=====
05-11 14:21:13 - [INFO] processing message WEAPONS_MSG

=====
resp:00F8
0000 d9588d7a 05-00-00-00-01-00-00-F0-53-53-54-49-43-7B-35-61 .....SSTIC{5a
0010 6f80053d 39-31-30-39-65-33-34-61-62-39-62-61-36-39-35-32 9109e34ab9ba6952
0020 aef2fffd 38-65-32-36-36-62-66-32-38-39-61-35-66-62-31-34 8e266bf289a5fb14
0030 2eb5733b 32-61-64-35-36-31-39-38-36-34-66-35-35-33-7D-00 2ad5619864f553}.
0040 8a16ea37 00-00-00-00-00-00-00-01-00-00-00-2B-D5-B8-8B .....+...
0050 f76780ac C1-80-25-3B-E3-CD-8F-3E-0A-62-46-5A-C6-90-F9-73 ..%;...>.bFZ...s
0060 71d2ba24 A6-59-AF-72-8D-20-DA-18-B4-C5-B1-4B-02-00-00-00 .Y.r. ....K....
0070 12b8bffd 00-00-00-00-41-41-00-00-00-00-00-42-42-00-00 ....AA.....BB..
0080 2e3dab57 00-00-00-00-43-6F-6E-72-61-64-5F-41-79-61-6C-61 ....Conrad_Ayala
0090 75af8386 5F-4B-50-62-56-4F-75-6C-45-58-62-69-78-64-77-46 _KPbVOulEXbixdwF
00a0 a342a652 52-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 R.....
00b0 ecbb4b55 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 .....
00c0 63eb4ef1 00-00-00-00-41-00-00-00-B0-1C-6F-4D-56-9E-17-9E ....A.....oMV...
00d0 cb4a6f21 34-7F-DE-C5-FC-F2-D4-D9-BD-36-46-47-8B-82-12-F6 4.....6FG....
00e0 0e4a22c0 BE-C2-C1-CF-90-29-88-32-02-00-00-00-00-00-00-00 .....).2.....
00f0 7b08a7d2 90-C2-85-06-BF-8A-B0-4F- .....0
end:F13674EF
=====

05-11 14:21:14 - [INFO] processing message WEAPONS_MSG
```