

Solution

du

Défi

organisé dans le cadre du

Symposium sur la Sécurité des Technologies de l'Information et de la Communication



Chapitre 1 - Introduction

L'aventure débute lorsqu'un chevalier errant découvre, à l'issue d'une longue et périlleuse quête, un légendaire et ancien PCAP (Parangon Contenant Abituellement des Perles).

Quelle ne fût pas la surprise de notre preux compagnon lorsqu'il découvrit la véritable nature de ce trésor ! A la place des perles promises, celui-ci ne trouva qu'un ramassis de morceaux de TCP (Très Commun Parangon), qu'il ne savait comment rassembler.

Invokant alors, à l'aide d'un parchemin de puissance, l'avatar terrestre du légendaire Requin des Câbles, il pût alors requérir son assistance, et réassembler son trésor, produisant alors un merveilleux ZIP (Zèbre Inspirant la Puissance). Celui-ci, dès son apparition, envoya notre héros directement dans une contrée mystérieuse...



Chapitre 2 – Les gens du Village

La contrée dans laquelle se retrouva notre brave héros était peuplée d'étranges personnages. Se rendant en premier lieu à l'auberge du Cactus, haut lieu de débauche des pays de l'Ouest, il pût aborder plusieurs héros légendaires peuplant les histoires contées à travers tout le continent :

- Sir Renzo
- Le Chevalier de la Serpillère et son fidèle écuyer
- E. F. Perseus, le pourfendeur de l'AES
- L'Ange Polyglotte
- Le Seigneur du Grenier
- Et de nombreux autres personnages dont l'énumération serait inutile et fastidieuse

Après une partie de mots-croisés bien méritée, notre chevalier rejoignit le nord du village afin de rencontrer un groupe d'autochtones désirant lui remettre des quêtes.

Un triumvirat accueillit notre héros, et chacun d'entre eux l'envoya à la recherche d'une clé permettant de quitter le village :

- Le premier homme souhaite retrouver un fantôme ;
- Le second a caché sa clé dans du matériel Texan ;
- Au troisième, proposant des activités indécentes, notre héros ne répondit pas.

La quête du Fantôme

Une capture de champ électromagnétique fût remise à notre brave, dans le but de l'étudier pour comprendre ce que le fantôme avait à dire.

Par chance, la besace du preux chevalier était bien remplie, et un outil magique fabriqué par un grand magicien du pays des fromages à trous lui permit immédiatement d'entendre la voix du fantôme. Si cet outil ne lui avait pas été disponible, il aurait pu se tourner vers l'acquisition d'un Chopchop auprès d'un bon armurier...

Les communications suivantes étaient particulièrement intéressantes aux oreilles de notre héros :

2016-02-29T13:51:12.692232 Keyboard Data 0x00000000	-	[2016/02/27 - 23
2016-02-29T13:51:12.692232 Keyboard Data 0x00000010	-	:15] sstic2016-s
2016-02-29T13:51:12.692232 Keyboard Data 0x00000020	-	tagel-solution.z
2016-02-29T13:51:12.692232 Keyboard Data 0x00000030	-	ip - Saisir mot
2016-02-29T13:51:12.692232 Keyboard Data 0x00000040	-	de passe
2016-02-29T13:51:12.693916 Keyboard Data 0x00000000	-	C
2016-02-29T13:51:12.695411 Keyboard Data 0x00000000	-	y
2016-02-29T13:51:12.696929 Keyboard Data 0x00000000	-	b
2016-02-29T13:51:12.698457 Keyboard Data 0x00000000	-	3
2016-02-29T13:51:12.699976 Keyboard Data 0x00000000	-	r
2016-02-29T13:51:12.701498 Keyboard Data 0x00000000	-	s
2016-02-29T13:51:12.703031 Keyboard Data 0x00000000	-	S
2016-02-29T13:51:12.704543 Keyboard Data 0x00000000	-	T
2016-02-29T13:51:12.706244 Keyboard Data 0x00000000	-	I
2016-02-29T13:51:12.707734 Keyboard Data 0x00000000	-	C
2016-02-29T13:51:12.709257 Keyboard Data 0x00000000	-	_
2016-02-29T13:51:12.710772 Keyboard Data 0x00000000	-	2
2016-02-29T13:51:12.712301 Keyboard Data 0x00000000	-	0
2016-02-29T13:51:12.713826 Keyboard Data 0x00000000	-	1
2016-02-29T13:51:12.715348 Keyboard Data 0x00000000	-	6

Il s'agit vraisemblablement d'une incantation permettant l'ouverture d'un Parangon.

Celui-ci est également transmis par le fantôme :

2016-02-29T13:51:13.046098 File Size Download 'C:\Users\sstic\Documents\Challenge SSTIC		
2016\Stage 1\sstic2016-stage1-solution.zip' (234 bytes)		
2016-02-29T13:51:13.049173 0x00000000	00 00 00 00 00 00 00 00 50 4B 03 04 0A 00 09 00 -	
..... PK.....		
2016-02-29T13:51:13.049173 0x00000010	00 00 38 76 5D 48 E1 1B DF 65 2C 00 00 00 20 00 -	
..8v]H.. .e,... .		
2016-02-29T13:51:13.049173 0x00000020	00 00 0C 00 1C 00 73 6F 6C 75 74 69 6F 6E 2E 74 -	
.....so lution.t		
2016-02-29T13:51:13.049173 0x00000030	78 74 55 54 09 00 03 7B 4C D4 56 A5 4C D4 56 75 -	
xtUT...{ L.V.L.Vu		
2016-02-29T13:51:13.049173 0x00000040	78 0B 00 01 04 E8 03 00 00 04 E8 03 00 00 03 6F -	
x..... O		
2016-02-29T13:51:13.049173 0x00000050	8E DF 08 F1 6C A5 14 EE 5C B3 4B A5 CE 77 DE 38 -	
....l... \.K..w.8		

2016-02-29T13:51:13.049173 0x00000060	97 A5 EE 25 E7 62 60 E0 39 75 6B 7E BE 57 7B 4A -
...%.b`. 9uk~.W{J	
2016-02-29T13:51:13.049173 0x00000070	30 DB 07 70 14 D5 DB C1 30 27 50 4B 07 08 E1 1B -
0..p.... 0'PK....	
2016-02-29T13:51:13.049173 0x00000080	DF 65 2C 00 00 00 20 00 00 00 50 4B 01 02 1E 03 -
.e,... . ..PK....	
2016-02-29T13:51:13.049173 0x00000090	0A 00 09 00 00 00 38 76 5D 48 E1 1B DF 65 2C 00 -
.....8v]H...e,.	
2016-02-29T13:51:13.049173 0x000000a0	00 00 20 00 00 00 0C 00 18 00 00 00 00 00 01 00 -
..	
2016-02-29T13:51:13.049173 0x000000b0	00 00 A4 81 00 00 00 00 73 6F 6C 75 74 69 6F 6E -
..... solution	
2016-02-29T13:51:13.049173 0x000000c0	2E 74 78 74 55 54 05 00 03 7B 4C D4 56 75 78 0B -
.txtUT.. .{L.Vux.	
2016-02-29T13:51:13.049173 0x000000d0	00 01 04 E8 03 00 00 04 E8 03 00 00 50 4B 05 06 -
.....PK..	
2016-02-29T13:51:13.049173 0x000000e0	00 00 00 00 01 00 01 00 52 00 00 00 82 00 00 00 -
..... R.....	
2016-02-29T13:51:13.049173 0x000000f0	00 00 -
..	

Une fois procédé à son extraction, son ouverture ne pose pas de problème connaissant la formule « Cyb3rSSTIC_2016 ». La clé apparaît finalement :

368BEC1CC7CC70C2245030934301C15

La quête Texane

Les populations Texane vivant sur l'autre rive de la grande mer des Atlantes disposent d'objets technologiques avancés fournis par la guilde des magiciens locale, dont le fameux Tambour Incongru, modèle 83, dans sa variante avancée (aussi nommé TIS3+).

Un chevalier digne de ce nom dispose de solides alliances avec toutes sortes de créatures. L'une d'entre elles est la sorcière Ida, mais notre ami décida de ne pas faire appel à elle immédiatement, et utilisa plutôt l'un des nombreux outils de la guilde les mages, accessibles par invocation directe à distance en tous points du globe.

Cela lui permit d'obtenir la formule originale qui avait permis la création du Tambour Incongru remis par le villageois.

Ladite formule était rédigée dans un langage magique basique, que notre héros pu déchiffrer seul. Ses maigres connaissances de magie des arcanes lui permirent de reconnaître

la formule du Cycle des Rameaux Canoniques de la 32^e lune (aussi appelé, parfois, par sa version courte, CRC32). Le sésame à présenter à l'objet devait, après transformation par la formule, correspondre à la rune C49ABZ57.

Notre chevalier, dont la force n'a d'égale que la détermination, décida alors de tenter toutes les combinaisons possibles lui permettant d'arriver au résultat escompté, ce qui lui donna rapidement (car il est rapide !) la solution 8959590Z.

Le Tambour Incongru libéra donc sa clé, concluant la quête.



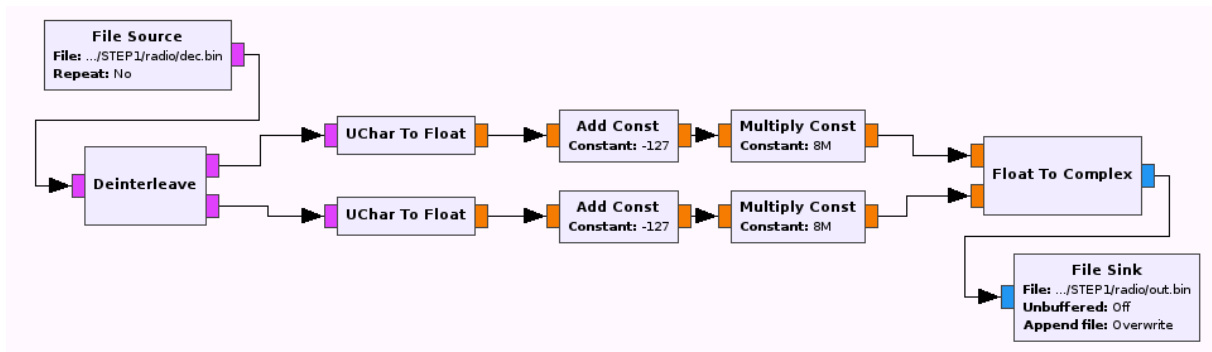
57D9F82B49C1EB3993CB82D26E37F69C

La quête du Golem Supérieur Magistral (GSM)

Repensant alors aux propositions indécentes du villageois précédemment rencontré, le chevalier se dit qu'il avait bien quelques minutes devant lui pour aller s'amuser un peu !

L'homme à la grosse antenne remis à notre héros un document dont les marques indiquaient qu'il s'agissait d'une quête impliquant un GSM.

Après décompression et modification runique, le document était prêt à être analysé !



Dans sa grande besace, le chevalier a toujours à sa disposition un Gros Réducteur pour Golem Supérieurs Magistraux (soit GRGSM), lui permettant aisément de venir à bout de ce genre de créatures.

Un premier passage par le grand Requin des Câbles lui révéla la vraie nature du Golem ! Celui-ci agissait principalement sur le 4^e plan Supérieur des Dryades Cochonnes et Cachées dans un Hêtre (SDCCH4). Cette information en main, le Requin des Câbles ne fit qu'une bouchée du golem, libérant la clé.

▼ TP-User-Data

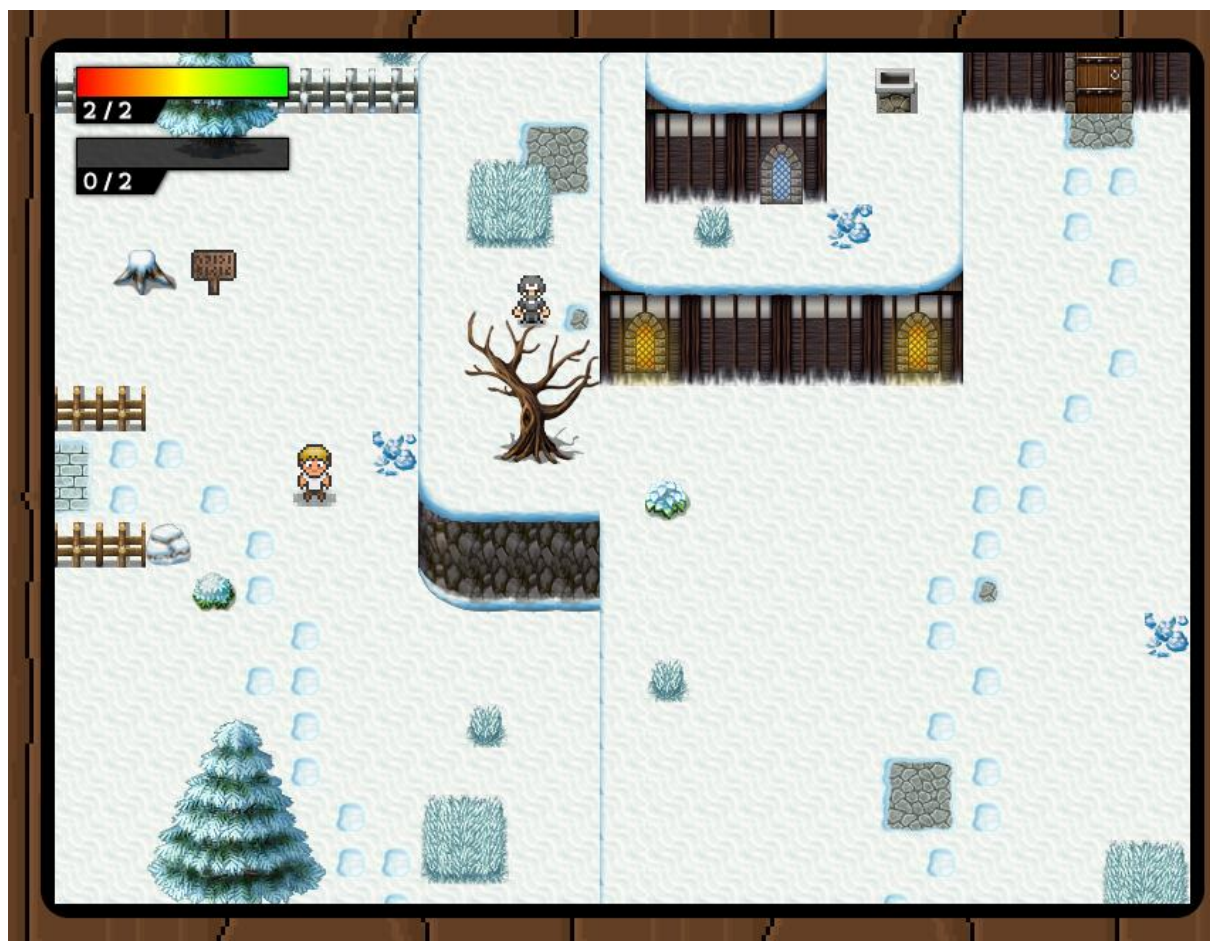
SMS text: Bonjour, votre cle est 1ac3d8c409e656380a06f6f2c6de6b4a

1AC3D8C409E656380A06F6F2C6DE6B4A

En possession des deux clés simples et de la double, notre héros fringuant pris la route de l'Est après que le garde lui ait laissé le passage...



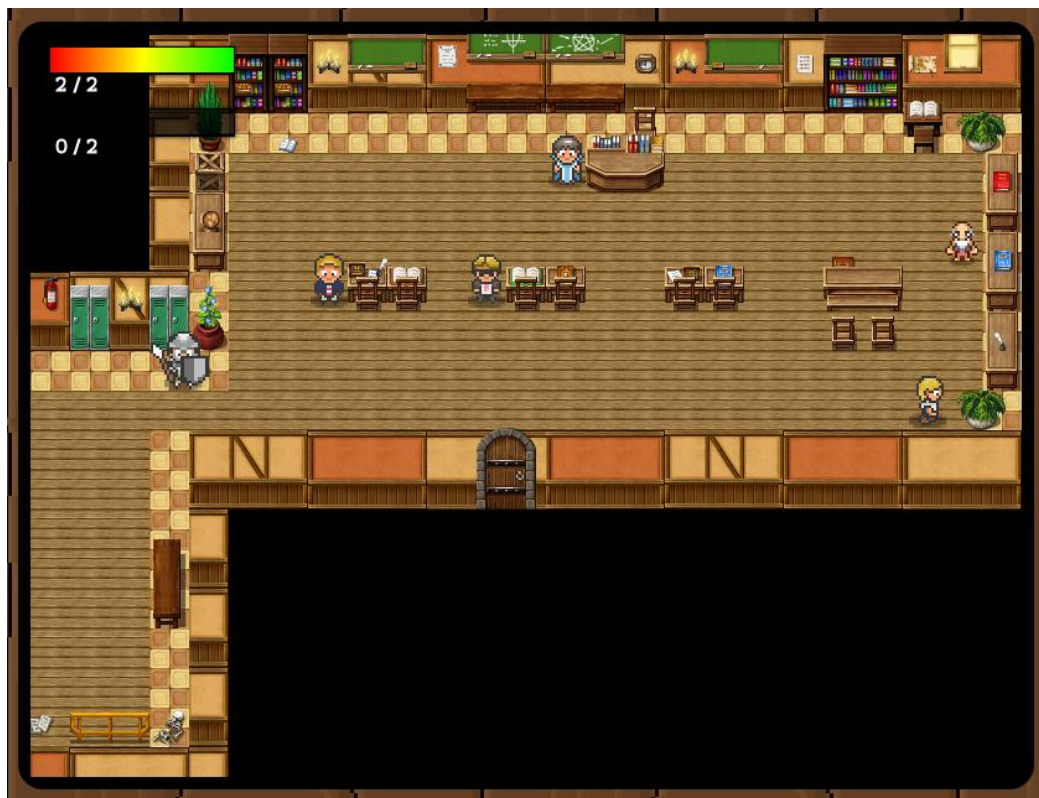
Chapitre 3 – Les neiges enneigées



Notre héros poursuit sa route dans les terres enneigées de l'Ouest, et ne tarda pas à trouver une demeure peuplée de quelques villageois.

Passé la discussion avec un lanceur d'alerte, les trois villageois restant proposèrent chacun une quête, dans le but de récupérer à nouveaux des clés permettant à notre chevalier de continuer sa route.

Mais une étrange plante verte attira rapidement l'attention du chevalier, de par sa chevelure étincelante...



Une rapide négociation avec elle libère alors un passage secret, sous forme d'un escalier, menant directement dans une nouvelle zone, sans aucun besoin de réaliser les quêtes proposées !



*Oh oh oh ! Il ne s'agissait là que d'un calembour ! Nulle magie ne permet d'outrepasser l'un
des gardes de ce monde !*

Reprenons donc nos quêtes...

Deux clés étaient nécessaires pour passer le garde, mais trois quêtes furent disponibles :

- Le combat avec un ELF (Elementaire Légendaire Faramineux) géant ;
- L'extraction et l'analyse minutieuse d'une table de caractères runiques ;
- L'affrontement avec un EFI (Elementaire Foncièrement Imbuvable).

Notre chevalier n'ayant aucune velléité à affronter une créature imbuvable, il choisit rapidement les deux premières missions proposées...

La quête de l'ELF géant

L'ELF géant fut fourni à notre héros enfermé dans une lampe magique.

Malheureusement, le plan actuel (nous nous trouvons actuellement dans la 4^e galaxie Elementaire du Xylophone Termitophage, aussi nommée EXT4) ne permettant pas la libération d'un ELF aussi imposant, notre chevalier utilisa son parchemin de téléportation majeure afin de se rendre dans la galaxie du Xylophone Faramineux Stoïque (parfois nommée XFS dans les cartes spatiales), et la lampe libéra alors son hôte.

Il fallait maintenant étudier l'ELF, afin de savoir quel mot runique prononcer pour qu'il libère la clé qu'il renfermait. Notre vaillant héros appela alors la sorcière IDA pour lui demander son aide.

Celle-ci, une fois arrivée, ne put que constater que l'élémentaire était bien trop imposant pour être directement étudié sous sa forme libérée. Elle le renferma alors dans sa lampe, et en extrait sa version diminuée, bien plus facile à observer.

La sorcière commença à rédiger une carte sur un parchemin, afin d'établir la correspondance entre les diverses parties du corps de l'ELF libéré et celles de sa variante atrophiée.

Table

**Correspondance
des emplacements**

0x1000	0x2e7affef6000
0x2000	0x352845ab0000
0x3000	0x352845ab3000
0x4000	0x42021da90000
0x5000	0x42021da9c000
0x6000	0x43abdb4a0000
0x7000	0x49e7e5410000
0x8000	0x49e7e541b000
0xa000	0x4a1706820000
0xb000	0x4a170682e000
0xc000	0x51466a42e000
0xd000	0x59eb440c0000
0xe000	0x59eb440c4000
0xf000	0x5a4815650000
0x10000	0x6f4b0e000000
0x11000	0x6f4b0e0f0000
0x12000	0x99a380500000
0x13000	0x99a380570000
0x14000	0x10f338cf0000
0x15000	0x10f338cf7000
0x16000	0x2a7ee24a0000
0x17000	0x2a7ee24aa000
0x18000	0x2affffff0000



Parthénien mis à jour 2012, www.3000parthénien.info

Dès lors, l'étude des mécanismes de défense de l'ELF peut être effectuée directement sur la version atrophiée. La majeure partie du mot runique à trouver est rapidement identifiée, mais deux vérifications sont plus complexes que les autres.

29..7ED1.....D54EC08C89341BEC

La première utilise une variation de la pile, positionnant celle-ci de telle sorte qu'elle termine par 03, et saute ensuite dans des instructions nulles en fonction de la valeur de l'octet 1. Ces instructions ont pour effet d'incrémenter une zone de 0x3 à chaque instruction. La valeur finale devant être de 0xb5, la seule valeur multiple de 3 correspondant est 0x1b5, ce qui implique 0x1b5/3, soit 0x77. La valeur finale doit donc être 0x100-0x77, soit 0x89.

La seconde vérification consiste à résoudre la formule magique $\cos(x) = x$, x étant un flottant sur 10 octets. 6 octets étant fournis, il suffit alors de trouver les 4 octets manquants, ce qui se résout aisément par force brute. La valeur à trouver est alors 0x998CD6D4.

29897ED1D4D68C99D54EC08C89341BEC

une fois fournie cette entrée à l'ELF, celui-ci libère alors sa clé :

Ǝ574B514667JF6AƁ2D83047BƁ871A54JF5

La quête de la table de caractères

une table de caractères runiques est remise à notre héros, ainsi qu'un emplacement dans lequel écrire une série de runes. Si la bonne série est entrée, la tablette le fait savoir, et la clé peut être dérivée de la série.

A l'aide de la sorcière, celui-ci pu rapidement identifier qu'une formule magique spécifique était associée à chaque rune. Pour la plupart des runes, il s'agissait d'associer un nombre entre 0 et 63 à la rune, et d'affecter ce nombre à la position de la rune. Mais l'une d'entre elles cachait une formule bien plus conséquente.

Répondant aux règles édifiées par les mages TrueType, il fallut alors que notre héros se plonge dans les anciens écrits d'A-doh-bee le Grand, afin de se forger un outil spécial permettant de décoder la formule.

une série d'équations lui apparut alors, permettant de définir quelle rune devait se trouver à quel emplacement. Voici quelques exemples de telles équations :

$$19 == \text{RS}[2]$$

$$5 == (\text{RS}[16] - 2))$$

$$90 == (320/64 * (64/64 * (7 + (2 + \text{RS}[9]))))$$

$$1970 == (320/64 * ((256/64 * (3 + (256/64 * ((\text{RS}[20] - 7) - 1)))) - 2))$$

Ces 4 équations permettent respectivement de trouver les runes pour les emplacements 2, 16, 9 et 20.

La résolution donne rapidement la bonne série de runes, produisant la suite de caractères suivante :

4ft/0AdJNj#BFW5ch+vVGw

Celle-ci ressemble aux anciennes incantations basiques de 64^e niveau (aussi appelées base64), se décodant en une clé :

E1FB7FD007493631C1156E5C87EBD51B

La quête de l'EFI

Soudain pris d'un terrible remord, notre chevalier décida tout de même de s'occuper de l'EFI, ne voulant pas laisser de quête inachevée.

L'EFI était particulier, sa composition n'étant pas semblable à celle des autres créatures dites de la branche x86. La lecture de ses entrailles se faisait toutefois sans difficulté.

Passée la récupération d'une clé, sa première action est le déchiffrement basique par sortilège XOR d'une zone de données. Celle-ci correspond à la GUID (Glorieuse et Unique Incantation Difficile) d'un sortilège de décompression.

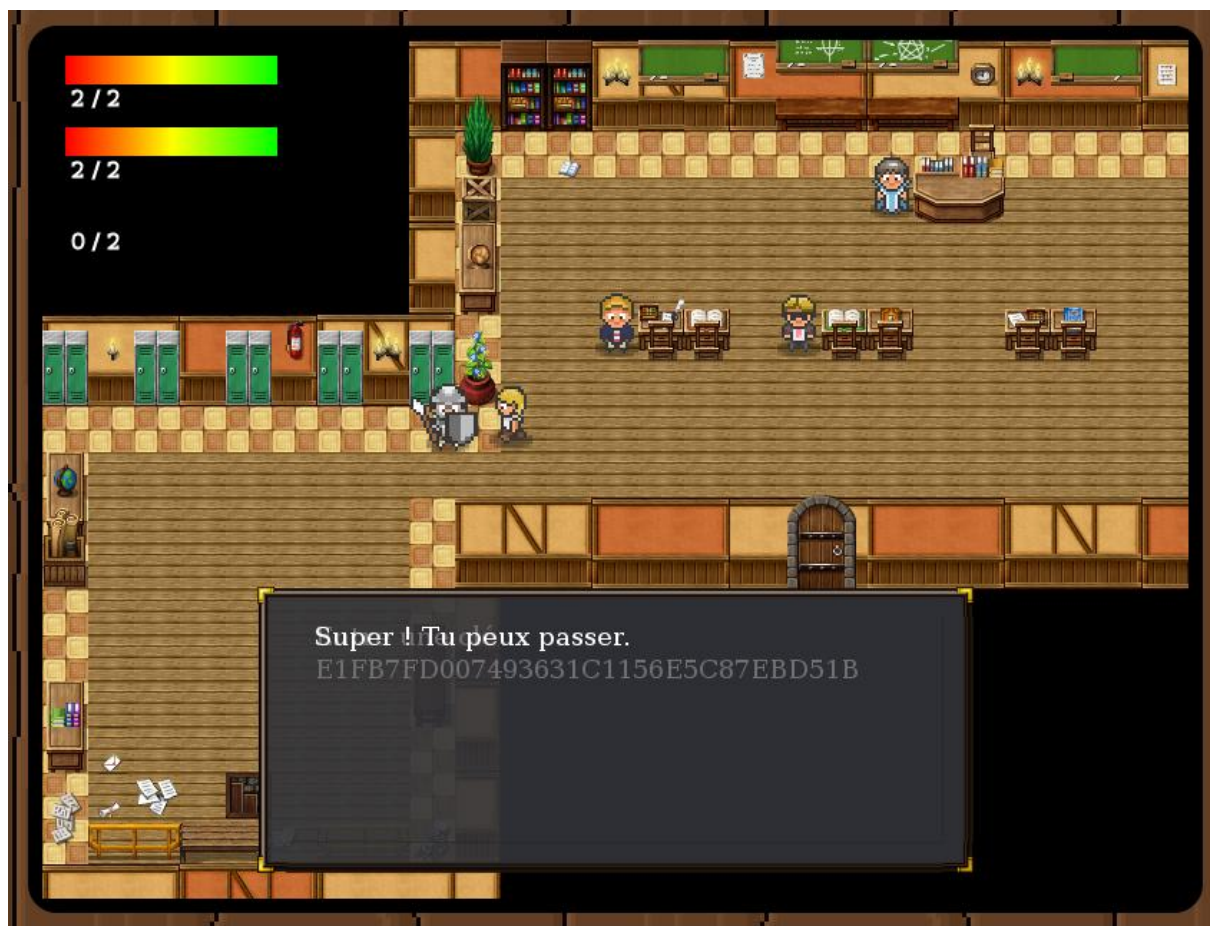
L'application de ce sortilège à une seconde zone permet l'extraction d'un indice capital :

secret data: cb41dcb1d89746705a7fe998f11acce7

La donnée secrète est ensuite récupérée, et subit les affres d'une boucle nonifiante rotationnelle par indice relatif (autrement dit, un ROT et un ROL dépendant de l'indice par octet). Une fois cette opération terminée, la clé tant attendue apparaît :

347D8C72720D6EC7A501583BEOBCCCCOC

Avec 3 clés en sa possession, notre héros peut alors passer le garde, qui n'en attendait que deux...



Chapitre 4 – La Cave du Temps

L'escalier derrière le garde descendait vers une étrange cave abritant des technologies du futur. Finis les chevaliers, épées, dragons et autres sortilèges, place à la magie des ordinateurs, leurs processeurs et les langages assembleur exotiques, Windows et ses subtilités, et les ingénieurs docteurs en XSS !

N.B. Le lecteur déçu du changement de style comprendra bien vite que certaines solutions ne pourraient être facilement décrites en restant dans le champ lexical de la chevalerie. Le retour à une période plus médiévale se fera dans le chapitre 5.



Cette fois-ci, 4 épreuves sont proposées, 2 d'entre elles libérant directement une double clé permettant de passer au niveau suivant :

- Un Driver Windows et son client userland (« usb ») ;
- Un screensaver Windows et une vidéo (« video ») ;
- Un ELF IA-64 (« strange », clé double) ;
- Un binaire Windows x64 (« ring », clé double).

Le chemin le plus rapide vers le niveau suivant passe par les deux épreuves donnant une clé simple (« usb » et « video »).

USB

Les fichiers fournis pour ce challenge sont :

- userSSTIC.exe : un PE 64 bits
- img.bz2 : une image disque compressée

Une analyse rapide du code nous montre qu'un driver est embarqué, et chargé sur le système à l'exécution. Ensuite, le contenu du répertoire « %SystemDrive%\SSTIC\ » est listé, chaque fichier est lu, et le contenu du fichier ainsi que son hash MD5 sont copiés dans une structure. Un thread est également chargé de détecter l'insertion d'un périphérique type disque, la lettre de ce nouveau étant sauvegardée via `ZwSetInformationProcess(ProcessDefaultHardErrorMode)`.

Curieusement, aucune IOCTL n'est effectuée vers le driver, la communication passant par un autre canal. Le binaire userland attend un changement de valeur à l'adresse 0x7FFE02C8. Cette adresse fait partie de la structure `KUSER_SHARED_DATA`, mappée en userland en 0x7FFE0000. Le champ en 0x2C8 est le `SystemExpirationDate`, et est utilisé par le driver pour passer au userland un pointeur vers une zone mémoire partagée entre user et kernelland.

Une première analyse du driver nous montre que les actions suivantes sont effectuées :

- Récupération de la lettre du disque via `NtQueryInformationProcess(ProcessDefaultHardErrorMode)` ;
- Création de la zone mémoire partagée via MDL, et écriture de l'adresse de cette zone dans le champ `SystemExpirationDate` de la structure `KUSER_SHARED_DATA` (mappée en 0xFFFFF78000000000) ;
- Récupération du contenu de chaque fichier lu par le userland, via un système de mutex pour que le processus userland et le driver sachent quand effectuer de nouvelles lectures/écritures. La structure contenant le contenu de chaque fichier est chiffrée en RC4 avec une clé générée aléatoirement. Une structure contenant la clé et le contenu chiffré est ensuite allouée et ajoutée à une liste chaînée ;

- Une fois tous les fichiers lus, l'intégralité de la liste est chiffrée en RC6 (clé en dur : « 551C2016B00B5F00 »), puis écrite sur le disque dont la lettre a été précédemment récupérée.

Le mécanisme de sélection des blocks dans lesquels écrire est assez long à analyser. Une observation de l'image disque montre que les partitions ne sont pas contiguës, et que des données à forte entropie se trouvent entre elles. Enfin, le code du driver nous indique que l'écriture sur le disque commencera au premier bloc composée uniquement de zéros.

La dernière difficulté est de trouver une implémentation de RC6 fonctionnelle, l'auteur ayant en plus confondu RC5 et RC6 lors de sa résolution...

La solution a donc été de « ripper » la fonction de déchiffrement, et de la réutiliser dans du code C avec quelques ajouts pour rendre compatible la convention d'appel avec Linux ☺

Dès lors, il est possible de lire les fichiers chiffrés sur le disque, et de rapidement trouver un ZIP :

```
$ ./rc5 img
```

```
$ ./rc4.py out.bin
```

```
$ file outfile*
```

```
outfile.0: ASCII text
```

```
outfile.1: ASCII text, with very long lines
```

```
outfile.2: PDF document, version 1.5
```

```
outfile.3: JPEG image data, Exif standard: [TIFF image data, big-endian,
```

```
direntries=7, orientation=upper-left, xresolution=98, yresolution=106,
```

```
resolutionunit=2, software=Adobe Photoshop CS5 Windows, datetime=2011:06:13
```

```
13:36:51], baseline, precision 8, 640x360, frames 3
```

```
outfile.4: Zip archive data, at least v2.0 to extract
```

```
$ cat outfile.0
```

```
password for the zip file : !WooYouAreSuchAnAwesomeGuy!
```

```
$ unzip outfile.4
```

```
Archive:  outfile.4
```

```
[outfile.4] 4.jpg password:
```

```
  inflating: 4.jpg
```

```
  extracting: key
```

```
$ xxd key
```

```
00000000: 0928 bde1 e3ed 8969 8632 dbff 4a23 1138
```


La clé finale est alors :

0928bde1e3ed89698632dbff4a231138

Video

Deux fichiers sont fournis pour ce challenge :

- Airlhes_screensaver_setup.exe
- Airlhes_CYBER_SECRET_possible_exfiltration.mp4

Dans la vidéo, on peut voir l'arrière d'un PC sur lequel tourne un screensaver, qui projette différentes couleurs sur le mur derrière lui.

L'exécutable est un installeur qui se charge d'installer le screensaver « Airlhes_screensaver.scr » sur la machine. Celui-ci est packé avec UPX.

Une fois dépacké, l'analyse nous montre les points suivants :

- Le contenu de la clé de registre
HKCU\Software\Airlhes\ScreenSaver\config\trajectories est récupéré et stocké précédée d'un Dword correspondant à sa taille ;
- Son contenu est ensuite utilisé pour calculer les codes de couleur à afficher à l'écran.

Le code est assez pénible à analyser, la partie intéressante étant noyée autour de nombreuses routines toutes sales étant inutiles pour comprendre le challenge et des boucles de timing pour gérer le temps d'affichage à l'écran.

Pour chaque byte de la clé de registre, des opérations sont effectuées, et finalement une valeur est extraire d'un tableau de 8 entrées (en 0x405020) xorées avec la constante 0x4FB78EB3.

Une fois déxorées, on obtient les valeurs : 0xFF0000, 0x00FF00, 0xFFFF00, 0x0000FF, 0xFF00FF, 0x00FFFF, 0xFFFFFFFF, 0x000000, correspondant aux couleurs Rouge, Vert, Jaune, Bleu, Violet, Bleu Clair, Blanc et Noir.

On constate ensuite que pour chaque byte d'entrée, trois couleurs sont produites en sortie. L'algorithme est le suivant :


```

Xor_byte = in_byte ^ key[i]

offset_color_1 = (Xor_byte%7 + offset_previous_color + 1)&7

color1 = colors[offset_color_1]

offset_color_2 = ((Xor_byte/7)%7 + offset_color_1 + 1)&7

color2 = colors[offset_color_2]

offset_color_3 = ((Xor_byte/49)%7 + offset_color_2 + 1)&7

color3 = colors[offset_color_3]

offset_previous_color = offset_color_3

```

La clé utilisée pour Xorer la valeur de la clé de registre est présente en 0x405040.

La variable `offset_previous_color` est initialisée à 6.

Il suffit alors de récupérer la séquence complète des couleurs de la vidéo, et d'inverser l'algorithme précédent.

Bien que la récupération des couleurs puisse être automatisée, l'auteur a jugé plus rapide de les récupérer à la main.

```
$ python decode.py
```

```
1800000078da0b1634172bdac558f9cb6e6257f3be7b5b003250077e
```

Les 4 premiers octets correspondant bien à la taille du reste de la chaîne, le décodage semble correct.

Le début de la chaîne ressemble étrangement à une en-tête `zlib`.

```

>>>
zlib.decompress("78da0b1634172bdac558f9cb6e6257f3be7b5b003250077e".decode("
hex")).encode("hex")

'5311371672ba0179fa3e918a83bedeb4'

```

La clé finale est alors :

5311371672ba0179fa3e918a83bedeb4

Strange

Deux fichiers sont fournis pour le challenge Strange :

- Un ELF compilé pour architecture Itanium

- Un fichier « 196 », contenant des données pour l'instant inconnues

Débuter dans ce challenge nécessite de lire un peu de documentation concernant l'architecture Itanium ! Quelques points intéressants :

- De très nombreux registres (128 « généraux », 128 contenant des flottants, etc.)
- Les fonctions récupèrent leurs arguments dans r32, r33, r34, etc.
- Les arguments passés aux sous-fonctions passent par les registres commençant après les registres utilisés localement (si une fonction utilise des registres jusqu'à r51, elle passe les arguments aux sous fonctions en commençant à r52)

De là, il est déjà possible d'identifier la fonction main en 0x40000000019C700.

Celle-ci effectue les actions suivantes :

- Ouvre le fichier « 196 » en lecture binaire ;
- Alloue un buffer de 526880 octets ;
- Ouvre le fichier passé en argument en lecture ;
- Vérifie via quelques lectures que celui-ci est une image au format PGM Ascii en niveaux de gris, de 12800 pixels ;
- Lit l'image ligne par ligne (l'image fait 640x20 pixels) et vérifie que chaque pixel est soit à 0, soit à 255 (image en noir et blanc) ;
- Entre dans une boucle de 32 itérations qui va :
 - o Lire 526880 octets du fichier « 196 » ;
 - o Lire l'image d'entrée par carrés de 20x20 pixels dans les 3200 premiers octets du buffer ;
 - o Effectue quelques vérifications basiques sur le carré de 20x20 pixels dans une première fonction ;
 - o Appelle une seconde fonction prenant en paramètre le buffer ;
 - Cette fonction appelle 161 sous fonctions : les 160 premières semblent faire la même opération sur des parties différentes du buffer ;
 - La dernière fonction semble mixer les résultats des précédentes.
 - o Les 4 valeurs retournées par la fonction doivent être inférieures à 0.15.

De là, plusieurs observations :

- L'image d'entrée contient 32 carrés de 20x20 pixels => cela correspond à la longueur de la clé, et chaque carré pourrait contenir un caractère ;
- Les 160 sous-fonctions de la fonction 2 effectuent la somme d'un produit de matrices, la première matrice étant le carré de 20x20 de l'image d'entrée, et la seconde étant contenue à l'offset $(3200 + k \cdot 3240 + 24)$ du gros buffer, pour chaque fonction k. De cette somme sont ensuite calculées 2 valeurs :
 - o $1/(1-\exp(x))$
 - o $(1/(1-\exp(x))) * (1 - (1/(1-\exp(x))))$

Quelques recherches sur ces valeurs nous amènent à des implémentations de réseaux de neurones pour la reconnaissance de caractères, ce qui semble confirmer notre première hypothèse ! Il ne reste qu'à comprendre le format du fichier 196 pour extraire l'ensemble des paramètres, et trouver les caractères à passer au réseau de neurones.

Un point reste à éclaircir, les 3200 premiers octets lus dans le fichier « 196 » à chaque itération de la boucle sont ensuite écrasés par le carré de 20x20. En extrayant ce contenu et en interprétant les valeurs comme des flottants, il est possible de redessiner des images de 20x20, qui cachent en fait les différents caractères (chiffres de 0 à 9) qu'il est possible de passer en entrée du réseau de neurones.

[illegible]

```

00000000000000000000000000
000000001111110000000000
00000111111111000000
00000110000011100000
000000000000001110000
000000000000000110000
000000000000000110000
000000000000000110000
000000000000000110000
0000000000000001100000
000000000000011100000
000000000000011000000
000000000000110000000
0000000000110000000000
0000000001100000000000
0000000011000000000000
0000000110000000000000
00000111111111110000
000001111111111110000
0000000000000000000000

```

```

00000000000000000000000000
000000011111100000000000
000001111111110000000000
000001000001110000000000
0000000000000011000000
0000000000000011000000
0000000000000011000000
0000000000000011000000
0000000000000011000000
0000000111111000000000
0000000111111000000000
0000000000000011100000
0000000000000000110000
0000000000000000110000
0000000000000000110000
0000000000000000110000
0000100000000011100000
0001111111111100000000
0000001111110000000000
0000000000000000000000

```


Une implémentation d'un tel réseau de neurones en Python est rapidement trouvée, et une fois les bons paramètres extraits du fichier, la solution tombe :

```
$ python parse_196.py 196
```

```
...
```

```
23425038472508287335772085544035
```

Ring

```
#sse_obfuscation #rage
```


Chapitre 5 – The End

De retour dans un monde plus normal, notre héros accomplit en fin sa quête...



Le monarque siégeant en haut de la pièce lui remet l'ultime parchemin, contenant la formule suivante :

Ŧ01p1 p'4qe3553 341p :

8P6d5j9Py88HUGHfGSKsJvqA@ffgbp.bet

Il s'agissait là de l'ancien dialecte du 13^e Rang des Orcs Terrifiants (ROT13), dont la traduction dans une langue plus moderne serait :

Ŧ01cl l'4dr3553 m4ll :

8L6q5w9Hl88UHTUsTfXfWidH@sstic.org