

# EGI Security challenge 5: Lehce na cvičišti ...

Radoslav Bodó <bodik@civ.zcu.cz>

Daniel Kouřil <kouril@ics.muni.cz>

## Abstrakt

Evropská gridová infrastruktura (EGI) poskytuje přístup k rozsáhlým výpočetním a úložným kapacitám, které využívají uživatelé z Evropy i celého světa. Zajištění dostatečné úrovně bezpečnosti je přirozeně velmi důležitou součástí provozu celé infrastruktury. Je to ovšem zároveň úkol náročný na koordinaci a komunikaci, protože zahrnuje komunikaci s velkým počtem správců a bezpečnostních expertů, kteří zodpovídají za jednotlivé zdroje připojené do EGI.

Za oblast provozní bezpečnosti je zodpovědný tým EGI CSIRT, který mimo jiné koordinuje řešení bezpečnostních incidentů, které se v EGI objeví. Jednou z aktivit EGI CSIRT je také pořádání „cvičení“, tzv. Security challenges, jejichž cílem je simulovat bezpečnostní problémy. Během řešení těchto incidentů se ověřuje, že informace jsou správně a včas předávány mezi zúčastněnými stranami a získané informace jsou kvalifikovaně zpracovávány.

V tomto příspěvku popisujeme poslední takové cvičení, které proběhlo v květnu 2011. Cvičení jsme se aktivně zúčastnili a podělíme se zejména o zkušenosti s forenzní analýzou, které mohou být užitečné i pro řešení reálných incidentů.

## 1 Úvod

Přestože odborným termínům v oblasti distribuovaného zpracování informací aktuálně vévodí *cloud computing*, nemůžeme pominout i jiná řešení, která nemají takový marketingový tah, nebo ne naopak přesunula z oblasti marketingu do zaběhnutých kolejí běžného života. Jednou takovou technologií jsou bezesporu gridy. Grid je distribuované prostředí, které nabízí přístup k velkým výpočetním a úložným kapacitám a zjednodušuje práci s nimi.

První gridová řešení se začala objevovat před více než deseti lety a od té doby vzniklo několik iniciativ poskytující gridové prostředí. K největším z nich patří Evropská gridová infrastruktura EGI<sup>1</sup>. EGI sdružuje tzv. národní gridové infrastruktury, které zastřešují poskytovatele zdrojů na úrovni jednoho státu. V rámci ČR je takto zapojeno několik clusterů a diskových úložišť z několika univerzit, Akademie věd a sdružení CESNET. EGI pak zajišťuje globální koordinaci jednotlivých národních aktivit.

Uživatelská základna EGI přesahuje sto tisíc aktivních uživatelů, kteří měsíčně vyprodukují přes 15 milionů výpočetních úloh z nejrůznějších oblastí vědy a výzkumu. EGI např. pomáhá zpracovávat data z urychlovače částic, který je provozován v ženevském CERNu.

Při takto rozsáhlé infrastruktuře a počtech uživatelů je přirozené, že zajištění bezpečnosti celého prostředí je nesnadná úloha. O provozní bezpečnost se stará tým EGI CSIRT, který sleduje aktuální hrozby a případně vydává upozornění, pokud se objeví nějaký bezpečnostní problém, který by mohl mít dopad na provoz EGI. EGI CSIRT spoléhá do značné míry na bezpečnostní experty na nižších úrovních, zejména tam, kde je potřeba zajistit komunikaci přímo s koncovými poskytovateli zdrojů nebo uživateli. Při běžném řešení rozsáhlejšího incidentu se tak řeší

---

<sup>1</sup><http://www.egi.eu/>

komunikace mezi poměrně velkým počtem lidí, což je náročné na koordinaci. Pro řešení incidentů v EGI existují politiky i prováděcí směrnice, které specifikují povinnosti všech „hráčů“. Samotná existence těchto pravidel ale automaticky nezajistí, že komunikace vždy probíhá správně a včas. V případě incidentů, zejména těch rozsáhlejších, je potřeba vyrovnat se se zvýšeným tlakem a stresem, který je způsoben tlaky ze všech stran, často včetně managementu nebo médií.

Ve snaze připravit sebe i všechny další potenciální partnery pořádá EGI cvičení, která simulují napadení infrastruktury rozsáhlým útokem. Řešení takového útoku vyžaduje koordinaci řady správců i bezpečnostních expertů, včetně samotného EGI CSIRT a je tedy dobrou příležitostí, jak ověřit připravenost institucí a pracovníků na všech úrovních.

V příspěvku se zaměříme na poslední takové cvičení, tzv. *EGI security challenge*, které proběhlo v závěru května 2011. V rámci tohoto cvičení byly vytipovány výpočetní klustery ve většině zapojených národních iniciativách, na kterých byla běžným způsobem spuštěna speciálně připravená úloha. Tato úloha obsahovala programy simulující chování „pěšáka“ botnetu, tj. po spuštění se snažila přihlásit na řídicí centrum a provádět získané povely.

## 2 Cvičení SSC5

Na přípravě cvičení SSC<sup>2</sup> pracoval tým EGI CSIRT několik měsíců a výsledkem je poměrně sofistikovaný systém pro správu botnetu a jeho vizualizaci. Cvičení probíhalo pod dozorem operátora cvičení, který hrál roli útočníka. Cvičení simulovalo situaci, kdy útočník získá oprávnění pro přístup k velké části infrastruktury, tj. k spouštění dávkových úloh a k manipulaci s daty. Celá simulace zahrnovala čtyřicet míst z dvaceti států. Informace o cvičení byli předem účastníkům oznámeny, aby věděli, že se jedná o simulaci.

Na počátku operátor vložil do dávkového systému úlohy pro všechny zúčastněné klustery. Po svém spuštění se úlohy přihlásí na řídicí centrum botnetu (C&C) a jsou posléze připraveny vykonávat příkazy od centra. Řídicí centrum i další pomocné komponenty byly opět spravovány operátorem cvičení. Ke spuštění jednotlivých botů docházelo postupně, podle toho, kdy se uvolnilo místo na výpočetnách uzlech.

Po spuštění části botnetu byla rozeslána hlášení o podezřelých síťových aktivitách správcům dvěma zapojených klusterů. Tato hlášení se podobala výzvám, které občas posílají jiné CSIRT týmy v okamžiku, kdy detekují průnik nebo podezřelou aktivitu. Od tohoto okamžiku se koordinace incidentu ujal EGI CSIRT.

V rámci prvních analýz „napadených“ strojů byly objeveny dvě podezřelé IP adresy, se kterými jednotliví boti komunikovali. Po získání těchto adres jsme využili monitorovací systém na bázi netflow dat FTAS, který provozuje sdružení CESNET. Pomocí systému FTAS jsme následně objevili komunikaci, která proběhla s těmito IP adresy zevnitř sítě CESNET2 a měli jsme tedy důvodné podezření o „kompromitování“ strojů, za které jsme přímo zodpovědní a zahájili jsme tedy vyšetřování v rámci českého gridu.

Vzhledem k pravděpodobnému rozsahu incidentu jsme uspořádali videokonferenci, které se účastnili administrátoři a bezpečnostní experti dotčených institucí a dohodli další postup. Paralelně s těmito aktivitami probíhala čilá komunikace na úrovni EGI CSIRT a nová zjištění sdílena mezi všemi účastníky.

První analýza kompromitovaného klusteru odhalila konkrétní stroj, na kterém úloha běžela. Administrátoři dále izolovali vlastní program bota a provedli základní forenzní ohledání napadeného stroje. Získané soubory z disku byly dále podrobeny podrobnému ohledání, které je detailně popsáno v následujících kapitolách.

Průběh celého cvičení shrnují dva propagační snímky, které jsou dostupné z <http://www.nikhef.nl/grid/ndpf/files/Global-Security-Exercises/FIRST2011/>

<sup>2</sup>[https://wiki.egi.eu/wiki/EGI\\_CSIRT:Security\\_challenges](https://wiki.egi.eu/wiki/EGI_CSIRT:Security_challenges)

### 3 Analýza malware egi.ssc5.wopr

Vzorek nalezené virové úlohy obsahoval:

- spouštěcí shell skript,
- 2 spustitelné soubory – malware `egi.ssc5.wopr`<sup>3</sup>
- Pakiti klienta<sup>4</sup>.

Ve většině případů je doporučeno z napadeného stroje sejmout forenzní obrazy stavu operačního systému [6] [7]:

- spuštěné procesy – `ps faxu`
- otevřené komunikační kanály – `netstat -nlpa; ipcs`
- obraz RAM – `dd /dev/k?mem`
- obrazy disků a jejich kontrolní součty – `dd; sum...`

#### 3.1 Úroveň 1: Sběr základních informací

Po zachycení byla provedena analýza chování spuštěním vzorku ve sledovaném prostředí testovacího výpočetního uzlu a ze 4 sekundového běhu byl pořízen záznam výstupních proudů `tmp.stdout` a `tmp.stderr`, záznam síťové aktivity `wopr.tcpcdump` a záznam interakce s operačním systémem `strace.out`.

```
l-- [ 12K] pakiti-ssc-client          cee7c9cefa1e94fb0a3d2bb18189d85bebb01632
l-- [ 479] ratatosk.sh              0aa241f5ca224ee54b6c3d1971736d9a6a5ad17d
l-- [1.6M] wopr_build_centos64.ANALY_GLASGOW a5cd77ab855e274d201021f12f7f96fcc0c8f367
l-- [1.2M] wopr_build_v6_debian32.ANALY_GLASGOW 8f1d97d80afde2872dd18df6b74228fd9c0139ac
l-- [ 95K] strace.out               a2ffa7beefafcf6d18eacbd9fddbd7a40ca9f93
l-- [  0] tmp.stderr               da39a3ee5e6b4b0d3255bfe95601890afd80709
l-- [ 88K] tmp.stdout              d6159289dab635781f56fa8608860cdd01ea026d
'-- [331K] wopr.tcpcdump           4f502e8a8d8219d2b965e0e98f947c0e08217010
```

Obrázek 1: Základní data zachyceného vzorku

Ze záznamu síťové aktivity (obr. 2) je možné zjistit, že malware používá HTTP C&C [15] (webové řídicí centrum) na IP adrese 195.140.243.4. Po svém startu ohlásí centrále základní údaje o stanici na které je spuštěn (požadavky "POST /message") a následně periodicky ohlašuje svou aktivitu "POST /pooling". Pro předávání strukturovaných dat (obr. 4) využívá formát JSON<sup>5</sup>. Průzkum webového serveru centrály nezjistil žádné jiné vstupní body řídicího centra (pouze /messages a /pooling).

Současně s HTTP komunikací generuje požadavky na různé záznamy DNS v doméně `sWITH.VexoCIde.org` (obr. 2). Testováním bylo zjištěno, že v doméně byl vytvořen *doménový koš*:

```
*.sWITH.VexoCIde.org A 202.254.186.190
```

S jistou pravděpodobností se mohlo jednat o postranní kanál pro předávání informací mezi řídicím centrem a spuštěným malwarem [25]. Skutečný účel těchto dotazů však objasnila až další analýza.

Výstup z programu (obr. 3) duplikuje informace zachycené pomocí sledování síťové aktivity. Navíc od ní však obsahuje všechna data v originálním znění, ve které je úvod každé zprávy (prvních 32b) před odesláním zašifrován (obr 4). Předpokládáme, že tento výstup a plain-textová část síťové komunikace byly v programu ponechány záměrně pro účely cvičení.

Další akce které malware na stroji po svém spuštění provádí mohou být zjištěny ze záznamu systémových volání (obr. 5).

<sup>3</sup><http://en.wikipedia.org/wiki/WarGames>

<sup>4</sup>Pakiti: A Patching Status Monitoring Tool – <http://pakiti.sourceforge.net/>

<sup>5</sup>JavaScript Object Notation – odlehčený formát pro výměnu dat

Time .	Source	Destination	Protocol Info
09:30:59.89	172.16.1.1	195.140.243.4	TCP 59202 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=61368076 TSER=0 WS=7
09:30:59.91	195.140.243.4	172.16.1.1	TCP http > 59202 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 TSV=255254849 TSER=61368076 WS=6
09:30:59.91	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=61368093 TSER=255254849
09:30:59.91	172.16.1.1	195.140.243.4	HTTP POST /message HTTP/1.1
09:30:59.93	195.140.243.4	172.16.1.1	TCP http > 59202 [ACK] Seq=1 Ack=655 Win=7104 Len=0 TSV=255254854 TSER=61368093
09:31:00.29	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:31:00.29	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=655 Ack=121 Win=5888 Len=0 TSV=61368473 TSER=255254944
09:31:00.29	172.16.1.1	195.140.243.4	HTTP POST /message HTTP/1.1
09:31:00.31	195.140.243.4	172.16.1.1	TCP http > 59202 [ACK] Seq=121 Ack=1292 Win=8448 Len=0 TSV=255254949 TSER=61368474
09:31:00.67	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:31:00.67	172.16.1.1	195.140.243.4	HTTP POST /message HTTP/1.1
09:31:00.69	195.140.243.4	172.16.1.1	TCP http > 59202 [ACK] Seq=241 Ack=1452 Win=9728 Len=0 TSV=255255044 TSER=61368855
09:31:01.06	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:31:01.06	172.16.1.1	195.140.243.4	HTTP POST /message HTTP/1.1
09:31:01.08	195.140.243.4	172.16.1.1	TCP http > 59202 [ACK] Seq=361 Ack=1615 Win=11072 Len=0 TSV=255255140 TSER=61369240
09:31:01.18	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:31:01.22	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=1615 Ack=481 Win=5888 Len=0 TSV=61369404 TSER=255255167
09:31:32.89	172.16.1.1	195.140.243.4	HTTP POST /polling HTTP/1.1
09:31:32.91	195.140.243.4	172.16.1.1	TCP http > 59202 [ACK] Seq=481 Ack=1739 Win=11072 Len=0 TSV=255263099 TSER=61401076
09:31:32.91	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:31:32.91	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=1739 Ack=601 Win=5888 Len=0 TSV=61401093 TSER=255263100
09:31:59.89	172.16.1.1	147.231.25.14	DNS Standard query A X4DDE01ef.sWitCh.VEXOcIde.org
09:32:01.94	147.231.25.14	172.16.1.1	DNS Standard query response A 202.254.186.190
09:32:05.89	172.16.1.1	195.140.243.4	HTTP POST /polling HTTP/1.1
09:32:05.91	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:32:05.91	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=1863 Ack=721 Win=5888 Len=0 TSV=61434093 TSER=255271349
09:32:38.89	172.16.1.1	195.140.243.4	HTTP POST /polling HTTP/1.1
09:32:38.91	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:32:38.91	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=1987 Ack=841 Win=5888 Len=0 TSV=61467092 TSER=255279600
09:32:59.89	172.16.1.1	147.231.27.173	DNS Standard query A X4DDE022B.SwitCh.VEXOcIde.org
09:33:00.12	147.231.27.173	172.16.1.1	DNS Standard query response A 202.254.186.190
09:33:11.89	172.16.1.1	195.140.243.4	HTTP POST /polling HTTP/1.1
09:33:11.91	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:33:11.91	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=2111 Ack=961 Win=5888 Len=0 TSV=61500091 TSER=255287850
09:33:44.90	172.16.1.1	195.140.243.4	HTTP POST /polling HTTP/1.1
09:33:44.91	195.140.243.4	172.16.1.1	HTTP HTTP/1.1 200 OK
09:33:44.91	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=2235 Ack=1081 Win=5888 Len=0 TSV=61533091 TSER=255296100
09:33:59.89	172.16.1.1	147.231.25.16	DNS Standard query A X4dDe0267.sWITCh.VexoCIde.org
09:33:59.93	147.231.25.16	172.16.1.1	DNS Standard query response A 202.254.186.190
09:34:14.79	172.16.1.1	195.140.243.4	TCP 59202 > http [FIN, ACK] Seq=2235 Ack=1081 Win=5888 Len=0 TSV=61562972 TSER=255296100
09:34:14.81	195.140.243.4	172.16.1.1	TCP http > 59202 [FIN, ACK] Seq=1081 Ack=2236 Win=11072 Len=0 TSV=255303574 TSER=61562972
09:34:14.81	172.16.1.1	195.140.243.4	TCP 59202 > http [ACK] Seq=2236 Ack=1082 Win=5888 Len=0 TSV=61562989 TSER=255303574

Obrázek 2: Výpis záznamu síťové aktivity

```

"running wopr_build_v6_debian32.ANALY_GLASGOW"
The peace bringer exited with: 0
"running wopr_build_centos64.ANALY_GLASGOW"
message_send(): { "pid": 12897, "uuid": "8618d203-c4a1-4393-b743-914c1c7fedcb", "version": 5,
  "sitename": "ANALY_XXX", "vo": "xxxxx", "payload": { "posix": { "uname": { "sysname": "Linux",
    "nodename": "xxxxxxxxxx.xxxxxxxx.cz", "release": "2.6.xx-1xx.x2.1.xxx", "version": "#1 SMP
    Tue xxx 4 12:47:36 EST 2011", "machine": "x86_64" }, "ownership": { "uid": 24202, "euid":
    24202, "gid": 1085, "egid": 1085, "uid_name": "xxxxxxxx002", "euid_name": "xxxxxxxx002",
    "gid_name": "xxxxxxxxt", "egid_name": "xxxxxxxxt", "sgid": 1307, "sgid_name": "xxxxx" },
    "process": { "pid": 12897, "ppid": 12891, "sid": 12569, "pgid": 12569, "cwd": "\/scratch\83947
    56.xxxxxx.xxxxxx.xxxxxxxx.cz\/cxxxxxg_Brn12797\/xxxxx3\/Pxxxx_Pxxxx_12826_1306315250\/Pxxxxxxx_12
    41710574_1306315252\/workDir" } } } }
Encrypting...
message_callback
Decrypting...

***** Got it *****

polling_message_send: { "pid": 12897, "uuid": "8618d203-c4a1-4393-b743-914c1c7fedcb", "version": 5,
  "sitename": "ANALY_XXX", "vo": "xxxxx" }
Encrypting...
Polling callback
Decrypting...
polling_callback: got response object without an input buffer
resolve_dispatch
resolve_cb
polling_message_send: { "pid": 12897, "uuid": "8618d203-c4a1-4393-b743-914c1c7fedcb", "version": 5,
  "sitename": "ANALY_XXX", "vo": "xxxxx" }
Encrypting...
Polling callback
Decrypting...
polling_callback: got response object without an input buffer

```

Obrázek 3: Ukázka tmp.stdout

```

Stream Content
POST /message HTTP/1.1
Content-Length: 590

i3.P..s.....@...9w..t.....-cf52-4b06-810d-b2f650b5eb9c", "version": 5, "payload": { "pos
"nodename": "=====e.cz", "release": "2.6.=====35", "version": "#1 SMP Tu
"x86_64" }, "ownership": { "uid": 24202, "euid": 24202, "gid": 1085, "egid": 1085, "uid_name":
"=====", "gid_name": "=====", "egid_name": "=====", "sgid": 1307, "sgid_name": "=====
7248, "sid": 7110, "pgid": 7877, "cwd": "\\tmp\\workDir" } } } }.HTTP/1.1 200 OK
Date: Thu, 26 May 2011 07:31:00 GMT
Content-Length: 0
Content-Type: text/html; charset=ISO-8859-1

POST /polling HTTP/1.1
Content-Length: 78

i3.P..s.....@...9w..t.....-cf52-4b06-810d-b2f650b5eb9c", "version": 5 }.HTTP/1.1 200 OK
Date: Thu, 26 May 2011 07:32:05 GMT
Content-Length: 0
Content-Type: text/html; charset=ISO-8859-1

POST /polling HTTP/1.1
Content-Length: 78

i3.P..s.....@...9w..t.....-cf52-4b06-810d-b2f650b5eb9c", "version": 5 }.HTTP/1.1 200 OK
Date: Thu, 26 May 2011 07:32:38 GMT
Content-Length: 0

```

Obrázek 4: Ukázka komunikace s řídicím centrem

```

open("/tmp/some.random.file.move.along", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 13
open("/var/tmp/some.random.file.move.along", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 14
open("/opt/glite/var/tmp/some.random.file.move.along", O_WRONLY|O_CREAT|O_TRUNC, 0666) =
-1 ENOENT (No such file or directory)
open("/opt/glite/var/log/some.random.file.move.along", O_WRONLY|O_CREAT|O_TRUNC, 0666) =
-1 ENOENT (No such ...

```

Obrázek 5: Ukázka strace.out

## 3.2 Úroveň 2: Co to k čertu ...

Dále byly provedeny základní forenzní kroky vedoucí k identifikaci obsahu spustitelných souborů nástroji `file`, `ldd`, `strings`, `objdump` a online službou `virustotal.com`. Vzorčky nebyly analyzovány žádným online sandboxem<sup>6</sup>.

### 3.2.1 `man file`

```
wopr_build_centos64.ANALY_GLASGOW:
ELF 64-bit LSB exec, x86-64, statically linked, for GNU/Linux 2.6.9, not stripped
wopr_build_v6_debian32.ANALY_GLASGOW:
ELF 32-bit LSB exec, Intel 80386, statically linked, for GNU/Linux 2.4.1, not stripped
```

Obrázek 6: Výstup programu `file`

Oba spustitelné soubory jsou staticky linkované a nejsou z nich odstraněny ladící informace (not stripped).

Statické linkování (static linking) se používá z důvodů snadné přenositelnosti. Takto sestavený program je (až na malé výjimky) nezávislý na dostupnosti potřebných verzí knihoven na cílovém počítači, protože všechny potřebný kód pro vykonání programu včetně všech knihoven je součástí spustitelného souboru. Vytvořený program je sice větší (na disku i v paměti), ale spolehlivější přenositelnost je pro malware výhodou.

Ladící informace (debugging symbols) ve spustitelném programu (jména a relativní adresy proměnných, podprogramů a jejich parametrů, ...) se využijí v případě, že program při svém běhu havaruje (post mortem analýza z `coredumpu`), nebo pokud je nutné jej debugovat za běhu. Pro samotnou činnost programu však nemají žádný význam a bývají odstraňovány (`man strip`) v koncové fázi vývoje software (release build). V případě malware umožní nebo ztíží jeho případnou analýzu.

### 3.2.2 `man strings`

Rychlou nápovědou o funkcích nebo činnosti viru mohou být sekvence tisknutelných znaků které binární soubor obsahuje. Na rozdíl od reálných virů, poskytují již první nalezené čitelné řádky<sup>7</sup> wopru znatelnou úlevu (obr. 7).

Ve vzorku byly nalezeny IP adresy veřejně dostupných DNS resolverů následované informacemi o adrese C&C a jeho URL handlerch, formátovací řetězec pro DNS dotazy, pravděpodobně jméno nějakého souboru, textové části shell skriptů testujících možnost zápisu do adresáře, seznam adresářů a skripty pro zavedení úlohy do lokálních plánovačů `at` a `cron`. Práci s textovými daty mohou ulehčit například nástroje typu `bashitsu`[34].

Takto „náhodně“ nalezené informace mohou pomoci nalézt ostatní infikované stroje, vylepšit stávající pravidla IDS a IPS systémů a zvolit další postup při analýze bezpečnostního incidentu a navrhování dalšího postupu jeho řešení.

### 3.2.3 `man objdump`

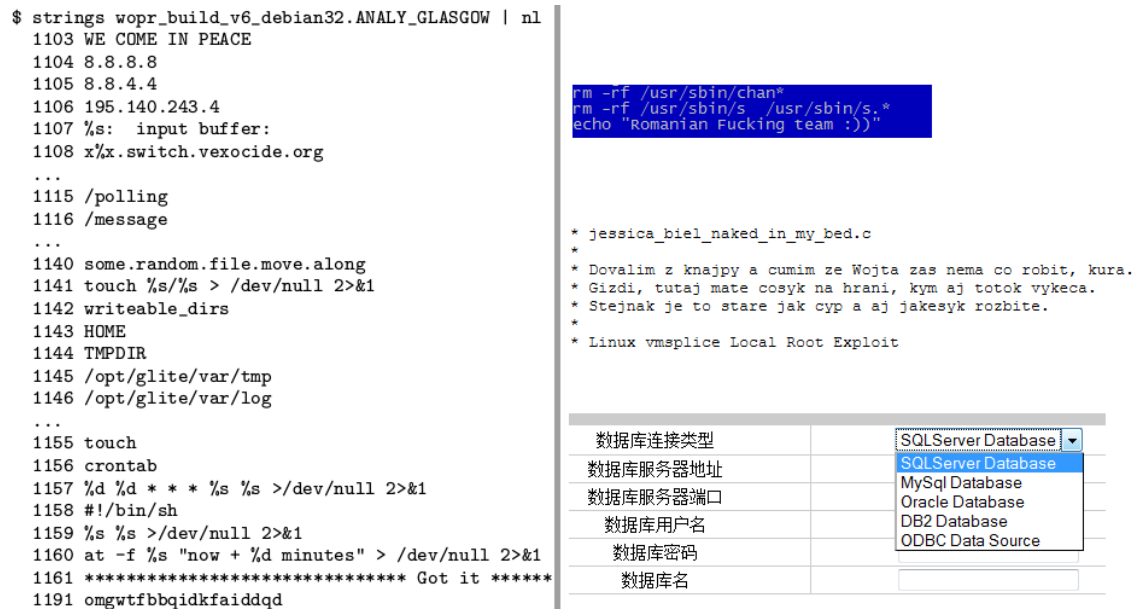
Pomocí programů `objdump`, `readelf`, ..., je dále možné získat (kromě množství náhodných hexadecimálních čísel) přehled o rozložení informací ve spustitelném souboru, jména a adresy používaných proměnných a podprogramů, a to jak interních, exportovaných nebo importovaných [26]. Tyto informace nám dále mohou zlepšit představy o účelu a povaze malware.

Z obrázku 8 jsou vidět některé používané funkce. Pomocí internetových vyhledávačů je pak možné zjistit, že funkce pocházejí pravděpodobně z knihovny `libevent`<sup>8</sup> z dílny bezpečnostního

<sup>6</sup>CWSandbox, Anubis Sandbox a Norman Sandbox, ...

<sup>7</sup>čti „Přicházíme v míru, google.com“ ;)

<sup>8</sup><http://monkey.org/~provos/libevent/>



Obrázek 7: Výběr tisknutelných sekvencí a ostatní ukázky

1338:	08052ce0	227 FUNC	GLOBAL DEFAULT	3 evbuffer_encrypt
1536:	0805e700	570 FUNC	GLOBAL DEFAULT	3 evbuffer_add
1695:	0804ce50	4871 FUNC	GLOBAL DEFAULT	3 aes_decrypt
2280:	0806dd40	360 FUNC	GLOBAL DEFAULT	3 evhttp_make_request
2521:	08069430	11 FUNC	GLOBAL DEFAULT	3 evhttp_request_get_respon

Obrázek 8: Ukázka tabulky symbolů egi.ssc5.wopr

specialisty Nielse Provoše. Podle popisu projektu (obr. 9) se tato knihovna výborně hodí pro tvorbu různých programů malware nevyjímaje.

*libevent is an event notification library for developing scalable network servers. The libevent API provides a mechanism to execute a callback function when a specific event occurs on a file descriptor or after a timeout has been reached. Furthermore, libevent also support callbacks due to signals or regular timeouts.*

Obrázek 9: Specifikace projektu libevent

Zajímavé jsou například funkce `aes_decrypt` a `evbuffer_encrypt`, dokládající pravděpodobnou metodu šifrování datového toku pomocí šifry AES. V dokumentaci API knihovny `libevent` tato funkce `evbuffer_encrypt` není.

### 3.2.4 man imagination

Při zkoumání neznámých souborů (např. nespustitelných) můžeme využít i různých druhů vizualizace[27]. V některých případech je potřeba obejít některou z forem použité obfuskace. `wopr` ve verzi 5.0 neobsahuje žádnou z nich:

- falešné magic hlavičky a přípony, neviditelná adresářová struktura,
- odstranění nebo přidání bílých znaků, různá kódování (base64, url, xor, concat ...),
- zkomprimování skutečného obsahu algoritmem gzip nebo specializovanými packery,
- nebo zašifrování částí malware<sup>9</sup>.

## 3.3 Úroveň 3: Mezi řádky

Další úroveň analýzy zachyceného vzorku bývá disassemblování a zjištění codeflow virového vzorku. Tato úroveň vyžaduje znalosti programování v C[24] a Assembleru[3][26], překladačů a operačních systémů na Von Neumannovských architekturách[19][20] a postupů používaných při konstrukcích malware[8][9][16][18][30].

Pro získání člověkem čitelného strojového kódu<sup>10</sup> může posloužit program `objdump` z kapitoly 3.2.3 nebo některý ze specializovanějších programů určených pro reverzní inženýrství [4][11][14].

V obou případech (obr. 10) je výsledkem disasemblingu textový soubor obsahující strojový kód programu. Práce s tímto souborem je poměrně náročná a v případě větších projektů prakticky nemožná i přes to, že `rec`[4] generuje kód vyšší úrovně (rozpoznané řídicí konstrukce programu jsou vyjádřeny v jazyce C).

Alternativou může být použití nástrojů poskytujících interaktivní vizualizaci typu OllyDbg<sup>11</sup>, ty poskytují k získanému kódu grafické rozhraní s možností vyhledávání a pohodlného sledování skoků při větvení programu. V každém případě se stále jedná o práci s plochou strukturou programu.

Naštěstí v dnešní době sofistikovaného malware [17][16] existují i specializované nástroje, které jednak poskytují vynikající analytické funkce pro získání dat z předložených programů a navíc s nimi umožní pracovat ve skutečně interaktivním grafickém rozhraní. Jedním z nich je IDA Pro [1] – pokročilý disassembler, jehož hlavní předností je interaktivní grafické zobrazení bloků strojového kódu zkoumaného programu provázané v místech větvení toku programu a provázání veškerých dostupných informací získaných z jednotlivých segmentů programu (text, data, got, plt, bss, ...).

Tímto nástrojem jsme dokázali identifikovat části malware `egi.ssc5.wopr` a určit jeho možnosti.

---

<sup>9</sup>wishmaster

<sup>10</sup>zjevný protimluv ;)

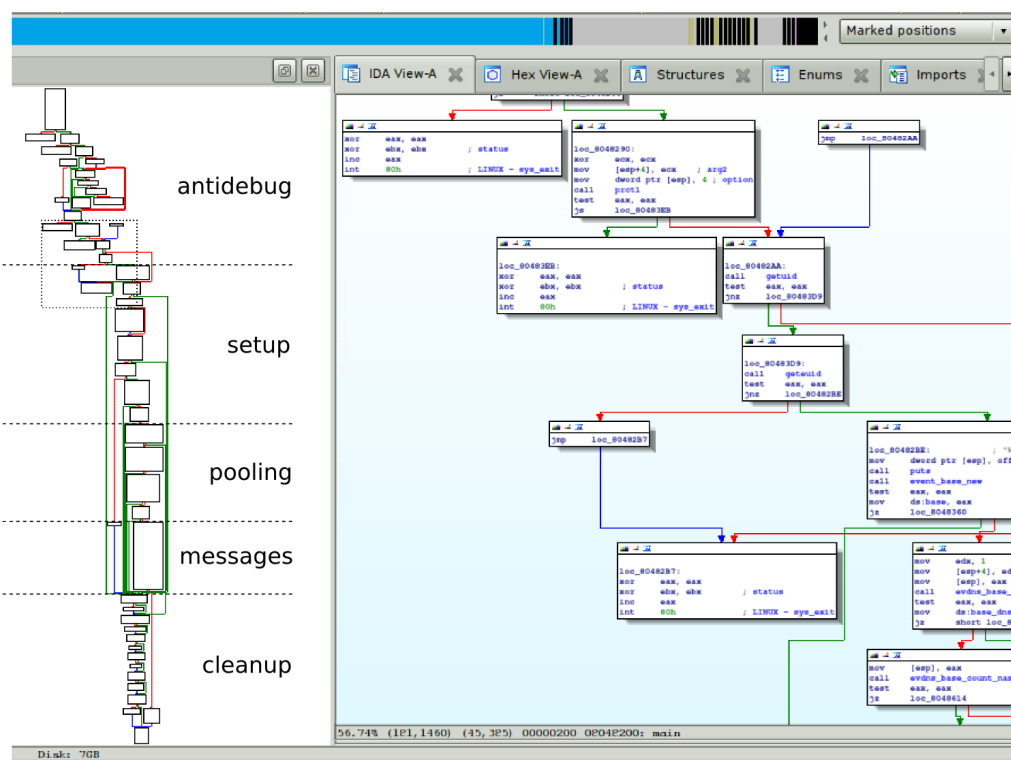
<sup>11</sup>pomineme-li fakt, že OllyDbg je hlavně debugger



REC

804825c: 8d 74 26 00	lea 0x0(%esi,%eiz,1),%esi	/	ebx = *(ecx + 8); if(anti_is_gdb() != 0) { ebx = 0; eax = 1; asm("int 0x80"); }
8048260: 75 e1	jne 8048243 <main+0x43>		
8048262: 31 c0	xor %eax,%eax	/	if(anti_is_ptraced() != 0) { ebx = 0; eax = 1; asm("int 0x80"); }
8048264: 31 db	xor %ebx,%ebx		
8048266: 40	inc %eax	/	eax = *ebx;
8048267: cd 80	int \$0x80		
8048269: 8b 42 04	mov 0x4(%edx),%eax		
804826c: 83 c2 04	add \$0x4,%edx		
804826f: 85 c0	test %eax,%eax		
8048271: 75 da	jne 804824d <main+0x4d>		
8048273: 8d b6 00 00 00 00	lea 0x0(%esi,%eiz,1),%esi		
8048279: 8d bc 27 00 00 00 00	lea 0x0(%edi,%eiz,1),%edi		
8048280: e8 4b ab 00 00	call 8052dd0 <anti_is_ptraced>		
8048285: 85 c0	test %eax,%eax		
8048287: 74 07	je 8048290 <main+0x90>		
8048289: 31 c0	xor %eax,%eax		
804828b: 31 db	xor %ebx,%ebx		
804828d: 40	inc %eax		
804828e: cd 80	int \$0x80		
8048290: 31 c9	xor %ecx,%ecx		

Obrázek 10: objdump vs rec

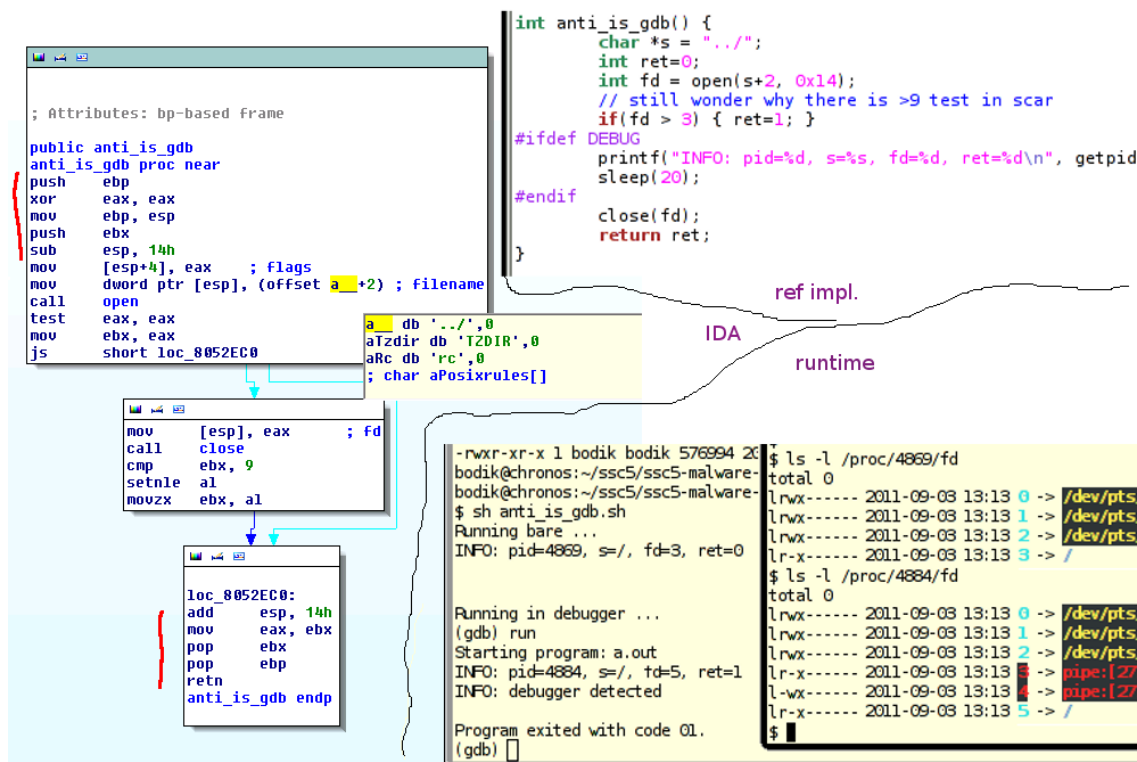


Obrázek 11: Ukázka grafického rozhraní IDA Pro 6.0 demo

### 3.3.1 Antidebug

Úvodní část `egi.ssc5.wopr` obsahuje několik antidebugovacích podprogramů. Tyto techniky jsou určeny pro znemožnění jeho spuštění v řízeném prostředí a ztížit jeho případnou analýzu. Mezi další techniky tohoto typu patří obfuskace, packing, falešné větve programu a přebytečné skoky, detekce běhu uvnitř debuggeru (`IsDebuggerPresent`, `OutputDebugString`, `FindWindow`, ...), prohledávání registrů, generování výjimek, kontrola velikosti různých souborů na disku, apod. V této oblasti se vynalézavosti meze nekladou.

**Podprogram `anti_is_gdb()`** testuje číslo prvního volného filedeskriptoru. Volání rutiny je umístěno hned na začátku programu a předpokládá se, že v té chvíli budou otevřeny pouze 3 základní I/O proudy (0 – `stdin`, 1 – `stdout`, 2 – `stderr`) a nově otevřený FD by tedy neměl být vyšší než 3. Pokud tomu tak je, naznačuje to, že byl program spuštěn v nestandardním prostředí a je pravděpodobně debugován. Např. při spuštění v debuggeru GDB, zdědí laděný proces otevřené STD proudy od svého rodiče a navíc má roury (IPC pipes) pro komunikaci s ním. Prvním volným FD je tedy 5 (při spuštění v DDD je první volný FD 7).

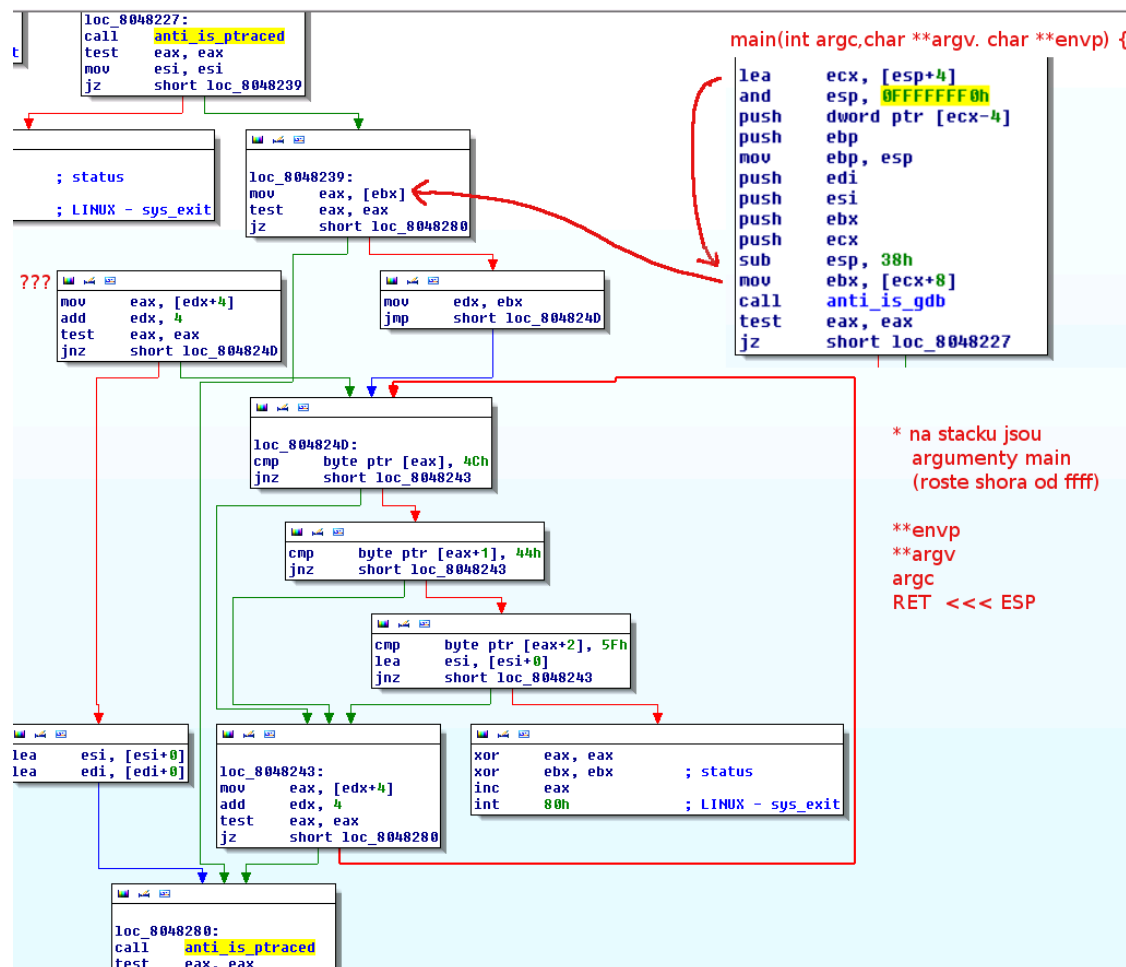


Obrázek 12: Analýza `anti_is_gdb()`

V `egi.ssc5.wopr` (obr. 12) je testován první volný deskriptor podmínkou `fd>9` (`cmp %ebx, 9`), což může souviset se specifiky cílového prostředí gridu (job piloting) nebo chybu při implementaci ochrany<sup>12</sup> vývojem v některém z pokročilých IDE (Netbeans, ...).

**Podprogram `anti_is_ptraced()`** testuje přítomnost aktivního debuggeru využitím faktu, že každý proces nemůže být současně trasován více než jednou. Malware vytvoří potomka, který se jej pokusí trasovat a návratovou hodnotou svému rodiči předá informaci o výsledku testu. Při analýze podprogramu byla sestavena referenční implementace.

<sup>12</sup>very unlikely ...



Obrázek 13: Analýza anti-test for ENV

**Sekce anti-test for ENV** (obr. 13) má za úkol zkontrolovat proměnné prostředí a vyhledat v něm proměnnou začínající znaky LD\_. Tato technika zřejmě slouží k odhalení běhu v některém z linuxových sandboxů založených na LD\_PRELOAD.

I přesto, že zjistit účel této oblasti kódu je poměrně snadný (porovnávání řetězce se znaky 0x4C 0x44 0x5F v cyklu je z grafu dobře vidět), je tato ukázka vhodná pro osvěžení znalostí instrukční sady x86 a způsobu uložení vícerozměrných polí v paměti.

Zásobník (paměť sloužící pro uložení řídicích dat programu) roste od vyšších adres k nižším. Při volání funkce se na něj nejdříve uloží argumenty volané funkce. Ukládají se od konce. Druhým a třetím argumentem fce `main` jsou pole řetězců, ve skutečnosti jsou to pointery, které ukazují na pole pointerů, které ukazují na skutečný obsah[24].

Při práci s těmito strukturami se používají instrukce `mov dst, [src]`, přičemž `[]` znamenají dereferenci adresy `src` – tedy do `dst` je nahrán obsah paměti z adresy `src`. Naproti tomu instrukce `lea dst, [src]` nahraje do `dst` adresu (ne obsah paměti), kterou představuje `src`. Pro tuto instrukci neznamenaí `[]` dereferenci adresy. Instrukce `mov` je určena pro manipulaci s daty na adrese, zatímco `lea` je určena pro vypočítávání adres. V obou případech se mohou uvnitř `[]` objevit aritmetické výrazy.

Kód související s tímto antidebug blokem začíná hned v první instrukci programu. Zde instrukce

`lea ecx, [esp+4]`<sup>13</sup> vypočítá adresu, na které leží první argument funkce `main` (parametr `argv`). Druhá *související* instrukce leží až za prologem funkce `main`. `mov ebx, [ecx+8]` nahraje obsah z adresy `ecx+8`<sup>14</sup> do registru `ebx`, ten tedy obsahuje adresu prvního pointeru v poli řetězců, které představuje proměnné prostředí spuštěného programu.

Samotný blok začíná až dále v programu za voláním funkce `anti_is_ptraced`. Jedná se o jednoduchou smyčku provádějící porovnání prvních 3 znaků z řetězce proměnné prostředí, která v případě nalezení shody celý program ukončí. V poli pointerů na jednotlivé řetězce se nakonec iteruje pomocí registru `edx` a blok končí opakovaným voláním funkce `anti_is_ptraced`.

**Zbýlý antidebug kód** zakáže pomocí volání `prctl(PR_SET_DUMPABLE, 0)` vygenerování obsahu procesu v případě jeho pádu ve snaze zanechat po sobě co nejméně stop a zkontroluje UID pod kterým je `egi.ssc5.wopr` spuštěn. Pokud je to uživatel s UID=0 je proces ukončen.

### 3.3.2 Nastavení

Po ochranách proti analýze začne vytvářet objekty pro další běh za pomoci knihovny `libevent`.

1. Nastaví knihovně seznam DNS serverů z konfigurace systému, případně použije IP adresy veřejných DNS serverů společnosti Google.
2. Naplňuje jednorázovou událost, která malware sama ukončí po uplynutí 14dnů.
3. Založí opakovanou událost, která periodicky ( $t = 60s$ ) provede unikátně vygenerovaný DNS dotaz do domény `sWITh.VexoCIde.org`. Dotazy obsahují jméno které je odvozeno z aktuálního času (kvůli cache) a odpověď je porovnána s konstantní hodnotou. Je zajímavé, že tato hodnota je v *network byte order* reprezentována jako `0xcafebabe`. Domníváme se, že jde o další možný antidebug kód, případně *deadman switch*. Pokud by se v průběhu cvičení vyskytly nějaké nečekané problémy tak by vhodná změna na NS serveru dané zóny dokázala celý projekt do několika vteřin řízeně ukončit.

### 3.3.3 Příjem příkazů

V další fázi své inicializace zaregistruje `wopr` opět periodickou akci. V tomto případě se jedná o pravidelné kontakty na centrálu v intervalu 0x21 sekund. Zprávy "POST /pooling" (obr.4) obsahují vždy pouze standardní hlavičku `{pid:, uuid:, version:, ...}` (viz `message_new`). Od serveru je v callbacku události (fce `pooling_callback`) očekávána odpověď ve formátu JSON a po jejím obdržení a úspěšném rozparsování je zpráva předána do podprogramu `command_dispatcher`.

Před odesláním a po příjmu každé zprávy, je úvodní část (první dva 16B bloky zprávy) zašifrována algoritmem AES256 v módu EBC<sup>15</sup> – `evbuffer_encrypt`. Ačkoli je jméno podprogramu v souladu s názvoslovím knihovny `libevent`, není součástí oficiálního projektu. Tomu nasvědčuje i jeho umístění ve spustitelném souboru (není ve stejné oblasti jako ostatní části knihovny). Jako klíč je použit kontrolní součet `sha256` řetězce `omgwtfbfqidkfaiddqd`. Funkcí *Save as C array* programu Wireshark, byla tato skutečnost ověřena referenčním programem, který je schopen dešifrovat zachycenou komunikaci mezi malware a C&C. Domníváme se, že pro šifrování byla použita implementace od Christophera Devina.

Heslo bylo identifikováno jako složenina několika IT zkratk a je vidět již ve výstupu `strings` z kapitoly 3.2.2:

- `omg` – Oh my god!
- `wtf` – What the fuck?
- `bbq` – Barbecue (zvolání, když obyčejné `wtf` nestačí)
- `idkfa` – Doom cheat – všechny zbraně a klíče
- `iddqd` – Doom cheat – nesmrtelnost

<sup>13</sup>na stacku tedy přeskočí návratovou adresu (4b).

<sup>14</sup>na stacku tedy přeskočí počet argumentů `main` (4B) + pointer na pole řetězců s parametry programu (4B).

<sup>15</sup>Každý blok je zašifrován stejným klíčem, viz EBC vs. CBC.

**Podprogram** `command_dispatcher` se pokusí rozparsovat odpověď od centrály a jsou v něm implementovány reakce na příkazy:

- `network_info` – viz kap. 3.3.4,
- `system_info` – viz kap. 3.3.4,
- `halt`, `kill` – oba ukončí provádění programu pomocí funkce `clean_kill`, který uvolní objekty knihovny `libevent` a poté zavolá `exit(0)`,
- `shell` – obdržený příkaz spustí pomocí volání `system`.

### 3.3.4 Úvodní přihlášení do botnetu

Posledním krokem před spuštěním knihovny `libevent` je shromáždění základních informací o systému kde malware běží a jejich odeslání v požadavcích "POST /messages". Informace o systému sbírají níže vyjmenované podprogramy a jejich výsledek je po skončení vždy odeslán jednotlivě.

- `payload_networked_devices_to_json` – zjistí hostname a sestaví seznam síťových zařízení (`getifaddrs`) včetně podrobných informací,
- `payload_postix_info_to_json` – zjistí informace o identitě aktuálního procesu: `uname`, `getuid`, `geteuid`, `getgid`, `getegid`, `getpuid`, `getrgid`, `getgroups`, `getpid`, `getppid`, `getsid`, `getgid`, `getcwd`,
- `payload_get_info_environment` – zjistí informace o proměnných prostředí `V0`, `Site`, `ROC`, `X509_USER_PROXY`
- `payload_look_for_writeable_dirs` – testem vyzkouší a nahlásí možnosti zápisu ve vyjmenovaných adresářích (kap. 3.2.2)
- `payload_test_cron_at` – název neodpovídá implementaci. Funkce vytiskne na obrazovku řetězec "`** GOT IT **`" a potom opět zavolá `payload_look_for_writeable_dirs`. Domníváme se, že toto je *nedokončená* větev programu, v malware sice jsou implementovány funkce `payload_test_try_cron` a `payload_test_try_at`, ale ty nejsou nikde volány.

Po dokončení této fáze je řízení programu předáno knihovně `libevent`<sup>16</sup>.

### 3.3.5 egi.ssc5.wopr.32 vs. egi.ssc5.wopr.64

Porovnáním 32b a 64b verze byly zjištěny drobné rozdíly. 64b verze neobsahuje úvodní anti-debug ochrany, navíc obsahuje různé ladící výpisy. `command_dispatcher` má drobně odlišnou posloupnost volání a obsahuje rozpoznání dalších příkazů (`credential_info` a `cron_at_info`), obě větve však vedou pouze na blok `printf("Not implemented yet")`. Implementace funkcí `payload_test_try_cron` a `payload_test_try_at` pouze vrací `-1`, ve 32b verzi jsou funkce implementovány celé, ale jsou nepoužívané.

## 3.4 Úroveň 4: Drakova hlava

Pokud by se jednalo o skutečný virus, následovaly by hlubší pokusy o infiltraci analyzovaného botnetu ať už dlouhodobým během viru v řízeném prostředí a sledováním jeho činnosti nebo přímými pokusy vyřazení botnetu z provozu přímým útokem na C&C či využitím deadman switchu ve spolupráci s registrátorem domény.

Pokud by se nepodařilo odhalit šifrovací klíč, mohly by být informace z kapitoly 3.1 použity ke kryptoanalýze typu known-plaintext (zřejmě za využití prostředků výpočetního prostředí EGI :).

Výsledky rozboru vzorků by umožnily dále zkoumat dopady běhu viru na napadených strojích, nabízí se prohlídka process accountingu za účelem zjištění příkazů spuštěných přes `command_dispatcher` nebo hlubší forenzní analýza napadených uzlů.

Vzhledem k tomu, že se jednalo o cvičení, nebyly podniknuty žádné tyto kroky.

---

<sup>16</sup>`event_base_dispatch`

## 4 A co na to Jan Tleskač<sup>TM</sup>?

I přes to, že analýza malware nebyla hlavním úkolem cvičení, domníváme se, že i s touto částí autoři počítali a sestavený vzorek byl velmi pečlivě připraven tak, aby poskytl vzdělávací informace na všech úrovních analýzy.

Žádný z antivirů ve službě `virustotal.com` neidentifikoval možnost hrozby i přes přítomnost neobfuskovaných antidebugovacích podprogramů.

Formát JSON je velmi výhodným formátem pro předávání strukturovaných dat typu asociativní pole (hash) a k jejich ukládání se dají využívat NoSQL databáze (např. Redis[31]).

Při práci mohou být s výhodou používány různé taháky[32] nebo nástroje pro filtrování[34] a management získaných informací[33]. Je dobré znalosti průběžně konzultovat se světovými vyhledávači. Proces analýzy mohou značně usnadnit profesionální komerční nástroje nebo alespoň jejich nekomerční verze.

Domníváme se, že malware by mohl být schopný působit i v prostředí s protokolem IPv6.

## 5 Závěr

Při řešení tohoto cvičení postupoval řešící tým podle obecných pravidel pro šetření bezpečnostních incidentů i specifických pravidel EGI. V úvodní fázi bylo nutné izolovat napadené uzly a jejich forenzní analýzou získat základní informace o vzniklém incidentu, nejlépe včetně vzorku virové úlohy. Podle nalezených dat dále identifikovat ostatní závadné úlohy ve výpočetním prostředí, nalézt uživatele který do systému úlohu vložil a zablokovat mu přístupové údaje.

Vzhledem k provázanosti jednotlivých výpočetních center bylo nutné sdílet informace o provedených akcích podle předepsaných postupů s ostatními členy EGI a týmem EGI CSIRT, tento proces byl hlavním úkolem cvičení. Postupně se podařilo shromáždit balík dat obsahující 2 virové vzorky včetně jejich spouštěcích skriptů a metadat o daných úlohách. Balík byl poté podroben analýze reverzním inženýrstvím, která měla za úkol rekonstrukci útoku a zjištění jeho dopadu na výpočetní prostředí.

Pro analýzu byly využity aktivní i pasivní metody. Aktivní část zahrnovala spuštění úlohy v řízeném prostředí interaktivního honeypotu za účelem získání záznamu o síťové aktivitě úlohy a její interakci s operačním systémem. Pasivní část zahrnovala dekompilaci nalezených virových vzorků a z nich zjištění codeflow viru.

Provedené šetření zjistilo pravděpodobné chování viru, identifikovalo použité knihovny, programovací styl a pomohlo získat znalosti, které by vedly k infiltraci do postaveného botnetu s pasivní možností sledování aktivit útočníka. Dále vedlo k navržení dalšího možného postupu forenzní analýzy napadených uzlů gridu.

Při analýze byl použit sw IDA Pro, jehož hlavní předností je interaktivní grafické zobrazení bloků strojového kódu zkoumaného programu provázané v místech větvení toku programu. (if, switch, call, goto, ...). Vzhledem k tomu, že zachycený virus byl uměle vytvořený pro potřeby bezpečnostního cvičení SSC5, jeho kód nebyl nijak obfuskován a programovací styl byl na vysoké úrovni, podařilo se relativně rychle a snadno identifikovat používané podprogramy.

Zachycený malware je *event driven network server* napsaný pomocí knihovny `libevent` a jako C&C používá HTTP echo based kanál. V něm jsou implementovány základní kameny pro ochranu komunikace (symetrické šifrování statickým klíčem). Celkově se jedná o jednoduchý *exec pad* jehož hlavním úkolem je spojení s řídicím serverem, sběr informací o napadeném prostředí a vykonávání příkazů botmastera pomocí systémového volání `system`. Malware byl vyroben v několika buildech, z nichž každý obsahoval drobné odchylky v dostupné funkcionalitě. 32bitová mutace obsahovala i antidebug prvky, které měly znesnadnit runtime analýzu. Implementované bezpečnostní prvky byly upraveny tak, aby usnadnily analýzu, ale zachovaly výukovou hodnotu. Autoři zřejmě počítali i s hloubkovou analýzou softwaru a umístili na několik míst vtípné poznámky tak, aby analýza byla i zábavná.

## Literatura a odkazy

- [1] *IDA Pro: multi-processor disassembler and debugger*  
<http://www.hex-rays.com/idapro/>
- [2] ISC Cheat sheet  
<http://isc.sans.edu/presentations/iscflyer.pdf>
- [3] Iczelion's Win32 Assembly Tutorials  
<http://win32assembly.online.fr/tut1.html>
- [4] REC Studio 4 - Reverse Engineering Compiler  
<http://www.backerstreet.com/rec/rec.htm>
- [5] k0fein: *EGEE - Enabling Grids for E-science-E*
- [6] Josef Kadlec: *Forenzní analýza Unixových systémů*  
Diplomová práce, fim.uhk.cz, 2006
- [7] Radoslav Bodó: *Kovářova kobyla .... případová studie forenzní analýzy OS Linux*  
Sborník konference Europol.cz, jaro 2007, Jablonné v Podještědí  
ISBN: 978-80-86583-12-9
- [8] MazeGen: *Obfuskace strojového x86 kódu*
- [9] Bojan Zdrnja: *ISC Diary: Analyzing an obfuscated ANI exploit*  
<http://isc.sans.org/diary.html?storyid=2826>
- [10] Yshey Eset.sk: *10 things to learn from malware*
- [11] [http://www.joineset.cz/w4-download/List\\_kandidatovi\\_cz.pdf](http://www.joineset.cz/w4-download/List_kandidatovi_cz.pdf)
- [12] Sam Stover, Matt Dickerson: *Using memory dumps in digital forensics*  
;login:, vol. 30 no. 6, pp. 43-48
- [13] Jan Goebel, Jens Hektor, Thorsten Holz: *Advanced honeypot-based intrusion detection*  
;login: v.31 n.6
- [14] Chaos Golubitsky: *Simple software flow analysis using GNU CFlow*  
;login: v.31 n.2
- [15] Craig A. Schiller, Jim Binkley, David Harlet, Gadi Evron, Tony Bradley, Carsten Willems, Michael Cross: *Botnets: The killer web app*  
ISBN: 978-1-59749-135-8
- [16] Dave Dittrich, Sven Dietrich: *Command and control structures in malware: from handler/agent to P2P*  
;login: vol.32 no.6
- [17] bodik and c++: *InSecurity 2009*  
EurOpen.cz, Štědrónín 2009
- [18] Alexander Sotirov, Mark Dowd: *Bypassing Browser Memory Protections: Setting back browser security by 10 years*  
Proceedings of BlackHat Conference 2008
- [19] Shawn Moyer: *(un)Smashing the stack: Overflows, Countermeasures and the Real World*  
Proceedings of BlackHat Conference 2008
- [20] Radoslav Bodó: *Jak se smaží zásobník: Esej na téma buffer overflow, včera dnes a zítra*  
Sborník konference Europol.cz, jaro 2009, Praděd  
ISBN: 978-80-86583-16-7
- [21] Dave Dittrich: *Basic Steps in Forensic Analysis of Unix Systems*  
<http://staff.washington.edu/dittrich/misc/forensics/>
- [22] Ivor Kollár: *Forensic RAM dump image analyser*  
Diplomová práce, mff.cuni.cz, 2010
- [23] xorl: *More GDB Anti-Debugging*  
<http://xorl.wordpress.com/2009/01/05/more-gdb-anti-debugging/>
- [24] Pavel Herout: *Učebnice jazyka C*  
ISBN: 978-8-7232-383-8
- [25] Erik Couture: *Covert Channels*  
[http://www.sans.org/reading\\_room/whitepapers/detection/covert-channels\\_33413](http://www.sans.org/reading_room/whitepapers/detection/covert-channels_33413)
- [26] M. Dobšíček, R. Ballner: *Linux bezpečnost a exploit*  
ISBN: 80-7232-243-5
- [27] Conti: *Security data visualization*  
ISBN: 978-1-59327-143-5
- [28] <http://www.swansontec.com/sregisters.html>
- [29] [http://www.skullsecurity.org/wiki/index.php/Simple\\_Instructions](http://www.skullsecurity.org/wiki/index.php/Simple_Instructions)
- [30] Chris Wysopal: *Detecting "Certified Pre-owned" Software and Devices*  
Conference BlackHat2009
- [31] Karel Minařík: *Redis: The AK-47 of Post-relational Databases*  
<http://europol.cz/Proceedings/38/Redis.The.AK-47.of.Post-relational.Databases.KarelMinarik.2011.pdf>
- [32] <https://home.zcu.cz/~bodik/cheatsheets/>
- [33] <http://freemind.sourceforge.net>
- [34] <http://code.google.com/p/bashitsu/>