DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

"Botnet Battlefield": A structured study of behavioral interference between different malware families.

Bishwa Hang Rai

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

"Botnet Battlefield": A structured study of behavioral interference between different malware families.

"Botnet Battlefield": Eine strukturierte Fallstudie über Verhaltnsinterferer zwischen verschiedenen Malware Familien.

Author:	Bishwa Hang Rai
Supervisor:	Prof. Dr. Alexander Pretschner
Advisor:	M.Sc. Tobias Wüchner
Submission Date:	February 15, 2016

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, February 15, 2016 Bishwa Hang Rai

Acknowledgments

Abstract

Malware of different families may not like each other. For example, different malware binaries might try to uninstall each other before infecting a system. This is an interesting case of 'environment-sensitive malware'. There are some anecdotal evidences (blogs). To the best of our knowledge, there is no prior research addressing this problem in a systematic way. In this research we systematically try explore the scene in the wild. We ran multiple malware samples from different families at the same time in the Anubis environment (a dynamic full-system-emulation-based malware analysis environment). We later analyzed the results of the emulation and detect such behaviors. We used clustering algorithm to cluster the malware based on its resources activities such as file,registry,section,syncs. With this apporach we could find 30 paris of malware that were indeed battling each other. :)

Contents

1 Introduction

Malware Dynamic Analysis and Static Analysis Anubis [Anu] Malware Family some blogs on battlefield our intension on finding one

2 Literature Review

Google for current works Clustering of malware Weka

3 Methedology

3.1 The Big Picture

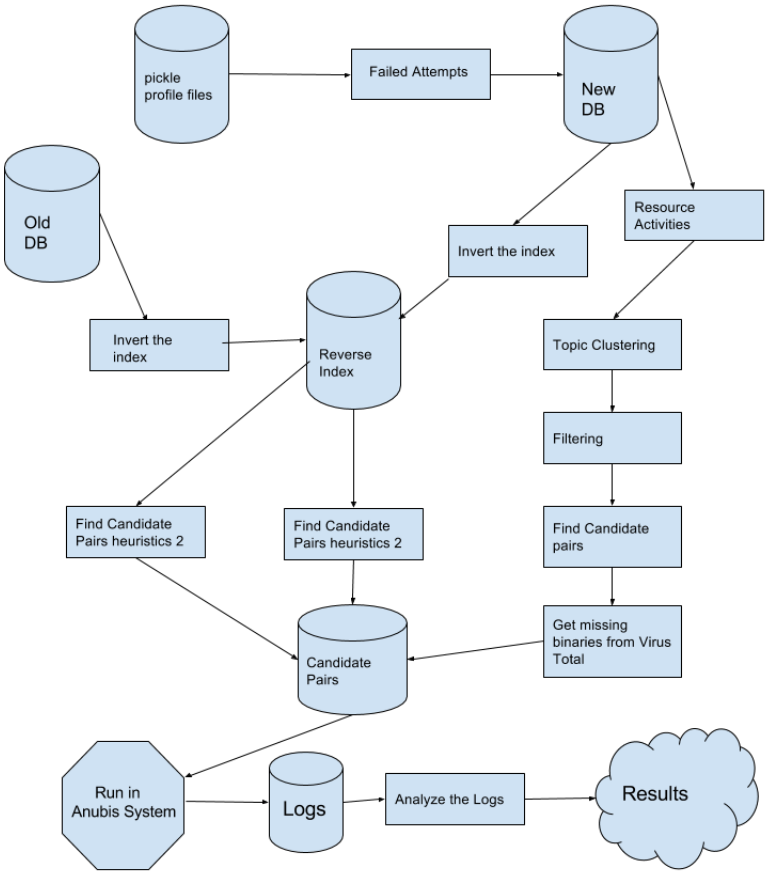

Figure 3.1: Illustration of how we planned to perform the experiment

Citation test [Lam94].

4 Implementation

4.1 Study of current Database

Our main resource for the research was direct access the millions of Malware samples from Anubis collected over the years. Not only was this resource our main strength but it was also a challenge to effectively and efficiently analyze those data for our research works.
We started with studying the Anubis system and current database. We primarily dealt with two database of Anubis backend, the **db_report** and **web_analysis**. The web analysis was the first entry of any sample malware submitted for analysis. It consisted of tables such as result, file, file_task. Each sample was given a unique file_id along with md5 and sha hashes. A new file_task *id* was created and the analysis of the sample would be done for different behavioral activities related to resources such as File, Registry, Mutex activities. These activities were saved in db_report database table. The resource activities of the malware in **db_report** database was associated with the **web_analysis** database by the constraint key *result_id* of *result* table in web_analysis database.

Listing 4.1: sql showing database structure to get file created activities of a malware

```
SELECT result_id FROM web_analysis.task join web_analysis.file_task using
    (task_id) join web_analysis.file using (file_id) WHERE task_id=
   result_id;
SELECT name from db_report.file_created join db_report.file_name using (
   file_name_id) where result_id ='12345';
```

We looked upon 3 resource type for the beginning. The resources type were *File, Registry, Mutex*. For each resource type we took into account their *create, read, delete* activities. The total number of malware sample that we had in our test database was **22154180**.
Our first step was to create a reverse index from the database so that we could get the list of malware that is associated with the resource activities. We started with writing a python script to do the task and save it in the file. However we found the first hurdle of our project. Since the number of malware were too large trying to save all their activities in data structure would cause our machine to run out of memory and hence

crash the system. To solve this we took the approach of map reduce.
We ran our script in batch 50K malware at a time and saved the reverse index of activity to the file with numbering. Around 420 files were created for each resource type. The result files were in the format where resource name and list of result ids were separated by commas (,) as delimiter. This was time consuming operation and it took almost 5 days for the script to finish. After the reverse index was generated in multiple numbered output, we sorted the file on resource name, alphabetically, and then joined the different parts with respect to the resource activity as key, and merged it into one single file. Since, this was a merge sort, we considered two sorted files at a time until single file was left, the reduce part was pretty fast (Big O $O(n * \log(n))$).

```
LANG=en_EN sort -t, -k 1,1 $file_name
LANG=en_EN join -t , -a1 -a2 $file_name
```

The snippet of resultant output:

```
Application Data,87623151,87079727,87034095
AutoHotkey,87623151,87079727,87034095
AutoScriptWriter,87623151,87079727,87034095
B:\mbr.exe,121858971
BIN,177509111,103858187
Base Images,189524063,184501719,87504631,86763863
Buttons,111448211
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/Adobe Reader
    8,178046895,174206059,183601891,89650247
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/_MEI1192/,161552035,116241803
```

4.2 Packer & Unpacker

In order to run the pair of malware together in side the Anubis environment we made a Win32 console application, named Unpacker, as Anubis VM was Windows XP based. The notion behind this Unpacker binary is like a dropper, where a single binary will unpack itself to create two binary and run each of them. We wrote a packer script that would add the candidate binary pair at the end of Unpacker binary and then further append the time delay and file size of each candidate binary as meta information. The unpacker binary when executed would read itself to fetch the meta information and then the candidate binary bytes.

We used a struct of 3 integer size to read and hold the meta information. The size of last three *unsigned int* byte had the binary pair sizes information and delay, know as meta information. The offset would be deduction of $3 * \textit{size of uint}$ from the total

Figure 4.1: [Packer structure] Structure of the Packer binary that would create the candidate pair and run them with delay.

size of itself. When the file size of packed binary was known, we used `fseek` function appropriately to start reading form the correct position until the exact size of binary were read. Then these bytes were saved to the new file pointer. Once the binary pairs were extracted by the unpacker new binary files were created in the Anubis environment and those were executed one after another with the delay of time as given in meta information. We used `windows.h` standard libraries *CreateProcess* and *Sleep* function for the purpose.

Listing 4.2: snippet of unpacker.c file

```
/* stuct for storing meta information */
typedef struct {
  unsigned int delay;
  unsigned int fsize1;
  unsigned int fsize2;
}meta_info;

/* reading the meta information and first binary */
rfp = fopen(argv[0], "rb");
wfp1 = fopen(fileName1, "wb");

fseek(rfp,0,SEEK_END);
size = ftell(rfp);
offset = size - sizeof(meta_info);
fseek(rfp, offset, SEEK_SET);
fread(&info, 1, sizeof(info), rfp);

/* calculate the unpackersize from the offset and files size. */
unpackersize = offset - (info.fsize1 + info.fsize2);
```

```
/* rewind back and to the point of the start of file1 */
fseek(rfp,0,SEEK_SET);
fseek(rfp, unpackersize, SEEK_SET);

nread_sofar = 0;
while (nread_sofar < info.fsize1) {
    nread = fread(buf, 1, min(info.fsize1 - nread_sofar, sizeof(buf)), rfp
        );
    nread_sofar += nread;
    fwrite(buf, 1, nread, wfp1);
}
fclose(wfp1);
```

4.3 Initial Experiment

From the reverse index of the resource activities of the Malware, we created a mapping between the created activities against the deleted or read activities. We created a list of malware based on the common resource name as the key, where a list of malware created the resource and another list of malware delete/read the same resource. We started looking for one to one interaction of malware to a single resource. Any resource type that was created by exactly single malware and deleted by exactly another single malware. We looked for a set (a, b), where malware 'a' creates the resource r, and malware 'b' deletes the same resource, r, and no other malware has create or delete activities on that resource r.
After finding such pairs, when we ran those malware pairs in the Anubis system using our unpacker, we found lots of dropper malware causing this interaction. Binary 'a' was a dropper that would create many binaries including the binary 'b', and binary 'b' actually read itself upon execution, which was recorded as read activity by Anubis. After running many other candidate pairs with different interaction and analyzing the results manually, not only were we able to understand the Anubis report in depth but also found an error on our approach.
Our notion behind checking the malware interaction was finding candidate pairs such that one malware 'a' creates some resource, 'r', and another malware 'b' tries to access or delete the same resource,'r'. But since Anubis ran each submitted binary in an isolated environment, there was no way that a malware 'b' would find the resource that was supposed to be created by malware 'a'. We should have been looking for failed attempt activities, where a malware unsuccessfully tried to access or delete a resource

created by another malware, but the current database that we had did not had record of such failed attempt. This lead for us to look for the alternatives were we could find log of such failed attempt activities.

4.4 Behavioral Profile

Previously, Bayer, Comparetti, Hlauschek, et al. used Anubis system for the dynamic analysis of malware to gets it execution traces [Bay+09]. They created a behavioral profile based on the execution traces of programs irrespective of order execution. It consisted a list of different operations operated on the different OS objects during the execution of binary. We had these behavioral profiles saved as python pickle file in our system and we used this to recreate new database according to our need.

Listing 4.3: Behvaioral Profile sample

```
op|file|'C:\\Program Files\\Common Files\\sumbh.exe'
 create:1
 open:1
 query:1
 query_file:1
 query_information:1
 write:1

op|registry|'HKLM\\SOFTWARE\\CLASSES\\CLSID\\{00021401-0000-0000-C
    000-000000000046}\\INPROCSERVER32'
 open:1
 query:1
 query_value(''):1
 query_value('InprocServer32'):0
 query_value('ThreadingModel'):1

op|section|'BaseNamedObjects\\MSCTF.MarshalInterface.FileMap.ELE.B.FLKMG'
 create:1
 map:1
 mem_read:1
 mem_write:1
```

4.5 Creation of Database

Recreating the database was one of the bottleneck in our project and consumed much time. The profile files were needed to be accessed via network file system, walking through list of directories and file to find the profile pickle of malware. Not all of the profile pickle were found. We found *16031518* out of *22154180* malware. We mapped the Windows Native and Windows Api call into three category modified, accessed, and delete. We took only 8 of the resource type into account while parsing the profile files; those were File, Registry, Sync, Section, Process, Mutex, Job, and Driver. We went through the most common used API calls and then mapped them into the broad three categories as described earlier.

Listing 4.4: Mapping of WinodowsNT and Windows API call

```
MODIFY_LIST = {
        file: ["create_named_pipe", "create_mailslot", "create", "rename",
             "lock", "set_information", "write", "unlock", "flush_buffer",
            "suspend", "map", "resume"],
        registry: ["create", "restore_key", "save_key", "map", "set_value",
             "set_information", "compress_key", "lock", "resume", "suspend"
            , "mem_write"],
        process: ["create", "set_information", "suspend", "resume", "unmap
            ", "map"],
        job: ["assign", "set_information"],
        driver: ["unload"],
        section: ["create", "map", "unmap", "mem_write", "extend", "
            suspend", "resume", "set_information", "release"],
        sync: ["create", "release", "map", "set_information", "mem_write"],

        service: ["create", "start", "control"]
        }

ACCESS_LIST = {
        file: ["query_file", "query", "open", "query_directory", "
            query_information", "read", "monitor_dir", "control", "
            device_control", "fs_control", "query_value"],
        registry: ["enumerate", "enumerate_value", "flush", "monitor_key",
             "open", "query", "query_value", "mem_read"],
        process: ["open", "query"],
        job: ["open", "query"],
```

```
        driver: ["load"],
        section: ["open", "query", "mem_read", "read", "query_file", "
            query_system"],
        sync: ["open", "query"],
        service: ["open"]
        }

DELETE_LIST = {
        file: ["delete", "open_truncate"],
        registry: ["delete", "delete_value"],
        process: ["delete"],
        job: ["delete"],
        driver: ["delete"],
        section: ["delete"],
        sync: ["delete"],
        service: ["delete"]
        }
```

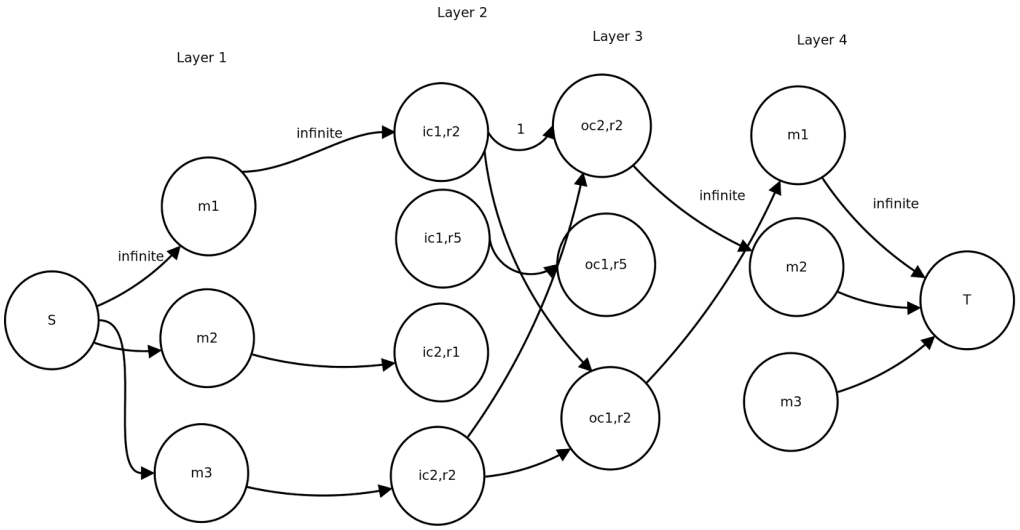

Figure 4.2: Max flow implementation to find out candidates with optimal number of resource

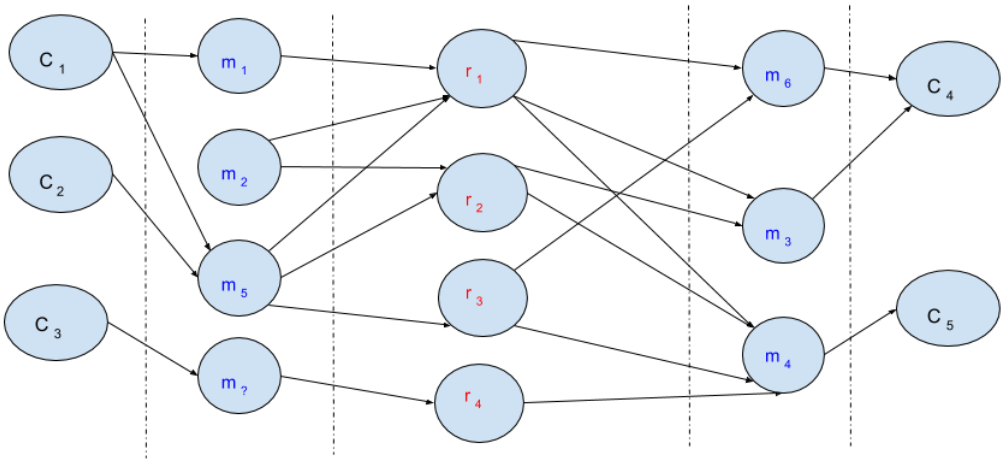

Figure 4.3: Chart showing the interconnection between Cluster topics, Malware, and Resources.

5 Future Work

6 Conclusion

Glossary

computer is a machine that....

Acronyms

TUM Technische Universität München.

List of Figures

List of Tables

Bibliography

[Anu] Anubis. *Anubis - Malware Analysis for Unknown Binaries*. `https://anubis.iseclab.org/`. Last Accessed: 2016-01-03.

[Bay+09] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda. "Scalable, Behavior-Based Malware Clustering." In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*. 2009.

[Lam94] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.