Malicious Behavior Patterns

Conference Paper · April 2014

DOI: 10.1109/SOSE.2014.52

4 authors, including:

Robert Luh
Fachhochschule Sankt Pölten
16 PUBLICATIONS **58** CITATIONS
SEE PROFILE

Paul Tavolato
Fachhochschule Sankt Pölten
17 PUBLICATIONS **82** CITATIONS
SEE PROFILE

2014 IEEE 8th International Symposium on Service Oriented System Engineering

Malicious Behavior Patterns

Hermann Dornhackl, Konstantin Kadletz, Robert Luh, Paul Tavolato
Institute of IT Security Research
University of Applied Sciences
St. Pölten, Austria
{hermann.dornhackl, konstantin.kadletz, robert.luh, paul.tavolato}@fhstp.ac.at

***Abstract*—This paper details a schema developed for defining malicious behavior in software. The presented approach enables malware analysts to identify and categorize malicious software through its high-level goals as well as down to the individual functions executed on operating system level. We demonstrate the practical application of the schema by mapping dynamically extracted system call patterns to a comprehensive hierarchy of malicious behavior.**

Keywords-malware; behavior pattern; formal grammar

I. INTRODUCTION

Malicious software (malware) is undoubtedly one of the biggest threats to the global IT infrastructure that exists today. It has become a common tool in digital theft, corporate and national espionage, spam distribution and attacks on infrastructure availability.

Malware detection commonly uses a signature-based approach: known malware is described by purely syntactic characteristics – mostly bit strings or simple patterns defined by regular expressions which are stored in a signature database. Signature-based detection has several shortcomings: Firstly, obfuscation techniques commonly utilize polymorphic or metamorphic mutation to generate an ever-growing number of malware variants that are different in appearance but functionally identical. This leads to bloated signature databases and, ultimately, to an overall slowdown of the detection process. Secondly, signature-based techniques only detect malware which has already been identified and analyzed; new species or hitherto unknown variants are often overlooked.

An alternative to signature-based detection is the so-called behavior-based approach. Here, the malicious behavior of software is analyzed during execution of a suspicious code sample. At the time of writing two anti-virus products claim to do real behavioral analysis: FireEye [1] and LastLine [2]. Both provide functions to gather some behavioral information during dynamic analysis. Afterwards, a simple set of rules is applied to the generated report in order to decide whether the sample's behavior is malicious or not. Behavioral analysis of suspicious code samples is, despite the disadvantage in performance, a promising approach to detecting and pre-classifying malware: a specific piece of malware is not characterized by its syntactic appearance but rather by its semantic functionality – in whatever disguise it might appear. In order to implement behavioral techniques it is necessary to develop a comprehensive model of malicious behavior and to compensate for the downsides of dynamic analysis.

In order to improve the decision process about a sample's maliciousness this paper concentrates on the definition of malicious behavior patterns based on a categorization by objectives. Traditionally, malware is categorized by class [3], vector, or propagation channel [4]. The general objective is usually not part of the categorization; while it is common to dub certain malware as e.g. 'banking Trojan', the actual goal or objective is rarely considered (e.g. sabotage or data theft). The one approach described in literature that comes close to considering objectives of attacks is the Attack-Tree-Approach introduced by Schneier [5] in 1999. Here, a top-down attack tree is built from a root node that represents one specific goal of an attacker. Subordinate nodes represent activities that could be used to achieve the goal attached to the root of the tree. Possible activities are rather diverse and can contain such different things as social engineering, or exploiting system vulnerabilities. The main application of such an attack tree is risk assessment: costs can be attached to each activity. This provides decision support for security managers.

Unlike Schneier's approach the method introduced in this paper concentrates on attacks by malicious software. The defining components of the described behavior patterns are system calls executed by the software sample under scrutiny. In order to identify a sample's goal, malicious behavior patterns need to be defined. This is where the behavior schema comes in.

Learning about a sample's high-level objective requires the use of dynamic analysis techniques, namely the execution of the malicious sample in a controlled environment. System activities such as file or registry operations are recorded and interpreted.

Specifically, this paper contributes by:

- Presenting a *malicious behavior schema* spanning across four levels of granularity, ranging from high-level goals to the individual system calls

Acknowledgment: The work was supported by KIRAS project 836264 funded by the Austrian Federal Ministry for Transport, Innovation and Technology.

978-1-4799-3616-8/14 $31.00 © 2014 IEEE
DOI 10.1109/SOSE.2014.52

- Introducing an *attributed grammar* developed for the automated parsing of call-level traces generated by an API hooking tool
- Implementing a scoring method to categorize threats as well as a way to separate malicious from benign software
- Presenting an automated call-level alternative to conventional dynamic analysis suites

For reasons of practical relevance the examples presented in this paper concentrate on malware for Microsoft Windows systems. It is, however, possible to adapt the entire schema to other operating systems or even entirely different problem domains.

A. Attack Types

In order to model high-level malware goals we need to distinguish between different attack types. Specific attacks can be separated into several categories depending on their ultimate goal as defined by the key concepts of information security [6]:

- *Confidentiality*: Confidentiality attacks aim at gathering sensitive information (user and/or system data).
- *Integrity*: Service or data integrity attacks alter information or system behavior.
- *Availability*: Availability attacks aim at disrupting the normal operation of a system.
- *Authenticity*: Authenticity attacks attempt to fool the user into believing that a piece of falsified data is genuine.

II. Modelling Malicious Behavior

We decided on a layered approach for defining malicious behavior. There are four levels, whereas each tier is a member of the category above. The level of detail increases with each tier – while malware goals and approaches are rather generic, tasks and activities are highly dynamic and specialized. The foundation of the model are single, sequential or recurring system calls (patterns) that comprise the individual tasks the malware performs in order to achieve the goal set by its author.

In the following we detail all levels of the malicious behavior schema and its underlying system call patterns.

A. Malicious Behavior Hierarchy

1) Goals

Goals describe the objective of the malware on a very high level. The goal categories listed below are not expected to change much over time, as they are largely independent from the technical implementation of the malicious application. General goals are:

- *Espionage*: In this category, we sum up all activities aiming at stealing data while keeping a low profile. Primary attack types: *Confidentiality, Integrity*.
- *Sabotage*: Sabotage equals the manipulation or right-out destruction of an IT system; subterfuge is usually a non-issue. Primary attack type: *Availability*.
- *Hijacking*: This goal describes the integration of target machines into botnets or the repurposing of an existing service. Primary attack type: *Availability*.
- *Scamming*: This category sums up all kinds of attacks aiming at coercing money or information from a human user. Primary attack type: *Confidentiality*, *Authenticity*.

2) Approach

An approach describes the general method employed by the malware to achieve its defined goal. Approaches within the respective goals are:

Espionage:

- *User data theft*: This espionage approach aims at stealing company data or personal information – it is one of the major drivers of (industrial) espionage. Attack type: *Confidentiality*.
- *System data theft*: Here, the malicious application collects system data such as account credentials or certificates for e.g. account spoofing. Attack type: *Confidentiality*.

Sabotage:

- *Destruction*: The malicious software aims at logically or physically destroying assets belonging to or protected by the IT system. Attack type: *Availability*.
- *Manipulation*: Manipulation of a system may cause erroneous behavior or alter system functionality. Attack type: *Integrity*, *Availability*.

Hijacking:

- *Repurpose*: Repurpose attacks change the inherent nature of a system or service in order to fulfil the attacker's needs. Attack type: *Integrity*, *Availability*.
- *Relay*: Attack relays are zombie machines used for single or distributed attacks (e.g. denial of service) against a specific target or are used as a proxy to forward messages or data. Attack type: *Availability*.

Scamming:

Note: This goal cannot be fully defined without considering human actions; it can, however, be approximated through the definition of the technical implementation of certain supportive functions.

- *Coercion*: Coercion-type approaches (e.g. phishing) try to convince the user to do the malware author's bidding by feigning benign behavior. Attack type: *Confidentiality*, *Authenticity*.
- *Blackmail*: Blackmail-type approaches (e.g. scareware) threaten/blackmail the user. Attack type: *Authenticity*.

3) Task

Tasks are parent categories for system activities (OS operations) and describe the general behavior of malware as it works towards achieving its goal. They are separated into five tiers, of which two are optional depending on the size and scope of the current category: *behavior modules*, *vectors*, *task groups* (optional), *intermediate tasks* (optional) and *elementary tasks* (see fig. 1). Tasks are helpful when it comes to methodically develop behavior patterns comprised of system calls; they can be considered components of a malware sample's lifecycle beginning with the preparation of its environment and ending with its ceasing of operation. The method of delivery of the actual payload as well as its attack vector are the central aspects of this schema category. The four main behavior modules (tier 1 task categories) are:

- *Preparation* - Preparation tasks prepare the system for the execution of the malicious software and include the initial activities of the sample itself (see fig. 1).
- *Reconnaissance* - Reconnaissance sums up all activities related to investigating (enumerating) the system prior to payload delivery.
- *Execution* - Execution tasks are related to the launch of the malware as well as its typical lifecycle outside the actual payload delivery. Means of injection, external communication and propagation are part of this category.
- *Exploitation* - Exploitation is split into two main categories: tampering and information disclosure. Tampering includes all kinds of behavior alteration and manipulation of files/services while information disclosure deals with various data theft scenarios.

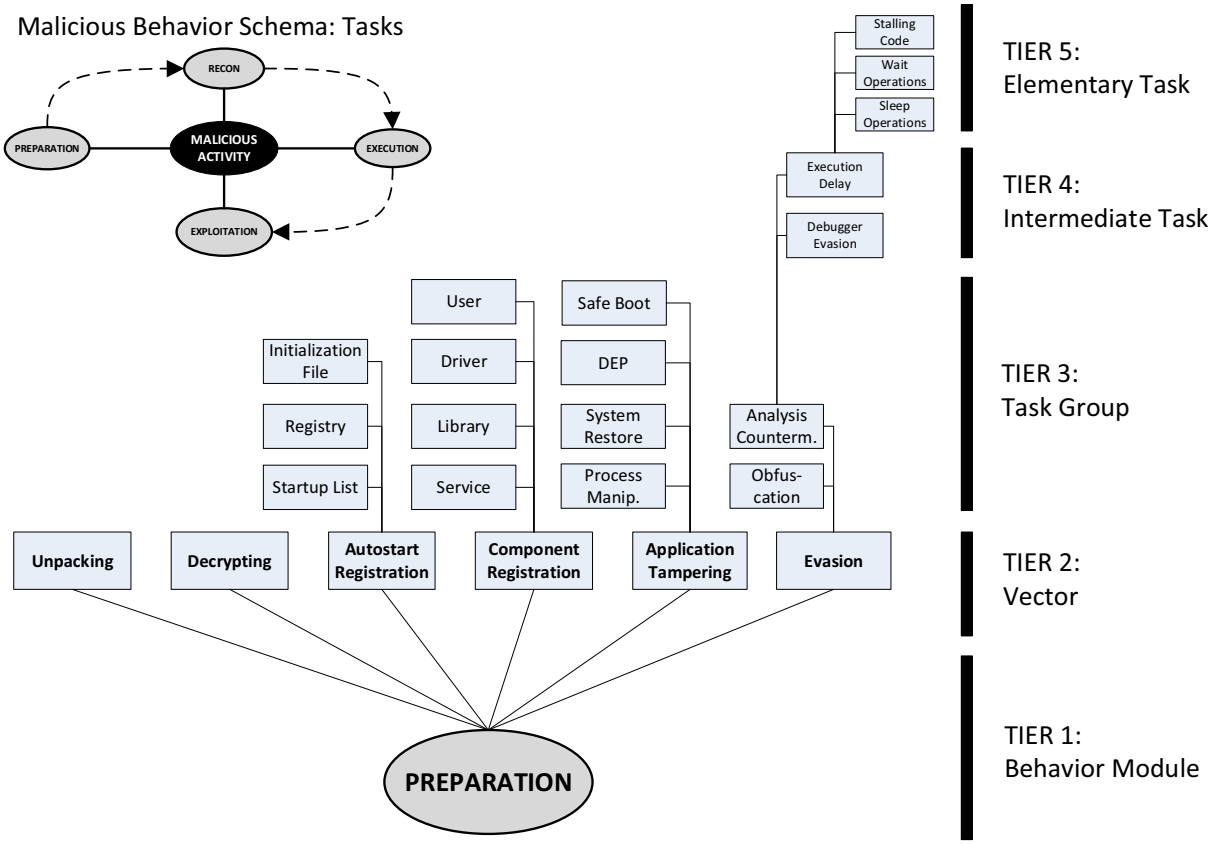

Fig. 1. Preparation Task Tiers

These modules represent the attacker's view on the classical physical security (DDDR) model [7]: Preparation is the counterpart of *deterrence*, reconnaissance is the attacker's equivalent for *detection*, execution takes place when a physical system would attempt to *delay* a previously detected intruder, and the actual exploitation mirrors a system's *response*.

Tasks are linked to the approaches and goals through a compound score (alignment and goal affinity) that is awarded to each pattern (see section C below).

4) System Activity

Programmatically, each elementary task is in fact one or several series of instructions to the underlying operating system. In this lowest level of the behavior model, one or more system calls (such as `NtCreateFile` or `HttpSendRequest`) are assigned to each task pattern. System calls usually contain one or more parameters as well as a return value. Both are relevant when it comes to determining the maliciousness of a call or call sequence. Fig. 2 depicts a sequence pattern that can be used to spawn a file.

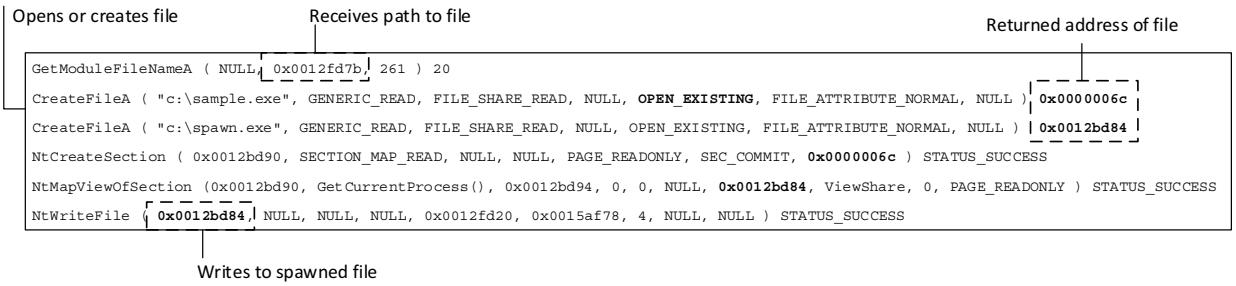

Fig. 2. File spawn (sequential pattern)

B. Formal Definition

In order to convert the schema from a simple model into an applicable rule set for malware classification it is necessary to formally define malicious behavior through distinct patterns that can be integrated into the hierarchy. After considering several formalisms, we decided to use *attributed grammars* [8] to define a language that contains these malicious behavior patterns. The grammar maps tasks to system activities (represented by patterns of system calls). The reason for this choice was the fact that semantically interesting connections between system calls are often expressed by their parameters; parameters, that can be aptly modelled by the attributes of an extended context-free grammar.

A similar way of using attribute grammars and automata to define abstract, language independent activities was described by Filiol et al [9]. They introduced a two-layer approach to malware detection: the first layer – *abstraction* – takes system calls from execution traces and maps them to language- and system-independent activities; the second layer – *detection* – tries to identify patterns with apparent malicious purpose. Only four such activities are mentioned, however (duplication, propagation, residency, overinfection tests); our paper fills the gap by introducing a comprehensive malware behavior schema armed with a unique scoring mechanism that, like the system call parameters, utilizes attributes to pass information towards the hierarchy root.

C. Pattern Classification (Scoring)

Each sample has a tendency towards one or more general goals, depending on tasks associated to the respective goal or approach; e.g. a sample may lean towards sabotage or hijacking, depending on how many tasks of the respective category it executes. There are two concepts we consider vital for classification and scoring of malicious behavior:

1) Alignment

We define the "alignment" of a sample as its tendency towards malicious or benign system activity. For example, debugger evasion is rarely used by conventional software (it therefore leans towards 'malicious') while the 'Communication' vector is not distinctively detrimental since benign remote installers often use such methods to download additional resources. Each level that is parent for another aggregates the mean alignment values of its children – up to its root. Using this method, we can calculate a total value for each vector/behavior module. The model supports derived percentages that are weighed according to their significance and/or frequency of occurrence.

2) Goal Affinity

Goal affinity defines how each vector and their tasks coincide with the malware author's respective goals as well as the approaches used to reach them. For example, certain 'Exploitation' tasks may primarily be used for sabotage purposes while 'Communication' tasks could hint at 'Relay' activity. Goal affinity is used to bridge high-level malware

categories (goal, approach) to low-level system tasks; it allows the analyst to quickly identify the likely objective of a sample. There is a fifth category which we dubbed 'self-serving'. It signifies that a respective task is primarily used to prepare the system for its own execution; self-serving activity is not required to coincide with a certain vector or payload (i.e. it is goal-independent). Each task (tier 5) is awarded an affinity score ranging from 'irrelevant' (0) to 'essential' (100). In short, these values are used to rank the likelihood of a certain system activity being part of a malware sample's general goal; for example, most 'analysis environment detection' tasks are usually not relevant for the overall goal but are considered self-serving. File environment information gathering, on the other hand, is likely to be utilized by 'Espionage'-centered malware.

The scores of each element of the higher level tiers are derived from the (manually defined) scores of the tier 5 activities. Derivation cannot be based on simple arithmetic formula but must be defined individually (as shown for the example in fig. 4). This is due to the fact that the score of an intermediate task is not only dependent on the individual scores of the elements of the subordinate level but must also encompass the semantics of the specific combination of these elements (two individually harmless tasks can be very harmful when co-occurring).

D. Application Examples

Using the formal grammar it is now possible to parse a system call trace (recorded by an API monitoring tool during execution of the sample) in search of malicious patterns. If the previously generated LR-parser (we used OX [10] for that purpose) returns a positive result, we can deduce that the traced sample displays the behavior modelled by the grammar.

Let us take a look at the result of the parsing process and how a specific pattern is scored:

1) Sample A

The malware sample under scrutiny is a recent (2013), randomly selected Wildlist [11] sample. We executed the sample inside both a shielded virtual and native Windows environment (XP SP3). The sample generated about 15 megabytes of API trace data [12] spread over 3 processes. For reasons of clarity and comprehensibility, we take a look at only a few of the patterns:

Analyzed sample (MD5): `9a66a1cf8cbf733a28a83db4727e70d0`

```
Line
91     kernel32.dll:   NtQuerySystemInformation( SystemBasicInformation,…)
4066   sample.exe:     CreateDirectoryA ("C:\DOCUME~1\mw\LOCALS~1\Temp\$inst",…)
4094   sample.exe:     SetFilePointer ( 0x0000006c,…)
4097   sample.exe:     ReadFile ( 0x0000006c,…)
4098   kernel32.dll:   NtReadFile ( 0x0000006c,…)
14062  sample.exe:     CreateFileA( "C:\DOCUME~1\mw\LOCALS~1\Temp\$inst\2.tmp",…)
                       Returns: 0x00000070
14074  sample.exe:     WriteFile( 0x00000070,…)
41521  sample.exe:     CreateFileA( "C:\Program Files\lpd\lpd\obleat.bat",…)
62173  shell32.dll     CreateProcessW( NULL, ""C:\Program Files\lpd\lpd\obleat.bat" ",…,
                       CREATE_DEFAULT_ERROR_MODE | CREATE_NEW_CONSOLE,…)
75701  USER32.dll:     CreateFileW( "C:\WINDOWS\System32\WScript.exe",…)
93559  shell32.dll     CreateProcessW( "C:\WINDOWS\System32\WScript.exe", ""C:\WINDOWS\
                       System32\WScript.exe" "C:\Program Files\lpd\lpd\mne_nada.vbs" ",…)
```

Fig. 3. Partial trace

Line 91 matches the pattern 'Reconnaissance/ SystemEnvironment/SystemInformation'. Since similar calls are made by benign software during their execution, the alignment rating of this particular activity is only slightly above neutral (55%). The creation of a temporary directory ('Execution/ Propagation/Spawn/DirectoryCreate') at line 4066) is similarly harmless. In the three following lines we observe file reading behavior (linked by the first parameter) followed by the creation of several temporary files ('Execution/Propagation/Self/ Installation'). The remaining lines show that the sample creates and subsequently executes two files which matches ('Execution/Propagation/Spawn/ FileCreate' and 'Execution/Launch/LocalExecution/ ProcessStart', respectively). While the creation of a file and the execution of a process generate only slightly elevated alignment scores, the parameter-linked, reoccurring combination of the two is considered a mid-range (75%) threat.

In the trace of the first additional spawn (`obleat.bat` – not shown in fig. 4 for space reasons) we see a large number of calls aiming at manipulating the `hosts` file which is responsible for local name resolution and ultimately determines the behavior of the system when connecting to an internet address (pattern match: Exploitation/Tampering/Configuration/Network/NameResolution'). The final spawn (WScript.exe and the VBS files it executes) contains the actual network connectivity functions such as `InternetConnectA` ('Execution/Communication/Internet/HTTP').

In comparison to analysis suites such as Anubis [13] and Joe Sandbox [14], our detection approach holds up well: neither of the commercial products was able to glean more meaningful information from the executed sample. In addition, only 38 of 49 (77.5%) of current anti-virus products registered with VirusTotal [15] were able to detect the Trojan even though Wildlist samples are well-known and publicly listed pieces of malware [11].

2) Sample B

The second code sample was taken from the 2012 DC3 basic malware challenge issued by the U.S. Department of Defense [16]. Early during its execution, the sample registers a new service. This kind of behavior is typical for the preparatory stages of malware execution ('Preparation/Component Registration/Service'). The sample then disables the Windows firewall so that it can communicate unimpeded with the outside world ('Exploitation/ Configuration/Network/Firewall'). Such behavior is distinctively malicious and generates a high (95%) alignment score. If the sample would do more than just attempt to ping a specific host like it does – e.g. transmit data from the local host to a remote system – it would meet the requirements of an 'information disclosure' payload.

While Anubis was unable to execute the sample, Joe Sandbox identified the same suspicious activity our approach detected, albeit with less detail and only rudimentary classification. Signature-based products fared worse: Only one of the anti-virus scanners recognized the sample as malware. Armed with these results we can now grade and visualize the matching task patterns (see fig. 4).

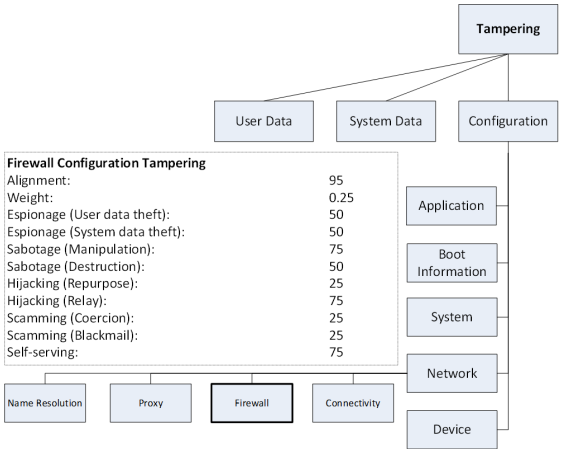

Fig. 4. Grading example

Each of these subscores is converted into an attribute and passed on to the next superordinate level. Every tier comes with its own weight and further aggregates the rating until we reach the root node and are able to derive the sample's final scores.

III. Conclusion

Defining system call patterns to model malicious behavior is effective especially for preparatory, reconnaissance and execution tasks. In general, tasks can be analyzed as long as they utilize system or API functions (as opposed to assembly instructions that can only be found using tedious static analysis). Unlike signature-based approaches the schema and its grammatical foundation can not only deal with already known malware, but is able to detect hitherto unknown variants of malware and even completely new specimen.

In comparison to other behavior-based approaches the malware behavior schema offers a more complex and thus more accurate definition of malware behavior patterns. These patterns are well-suited for pre-classification of malicious samples – provided the task patterns are of high enough quality. Up until now, each of the approximately 400 patterns was defined manually; automated pattern generation based on grammatical inference [17] is currently being developed.

References

[1] FireEye: http://www.fireeye.com

[2] LastLine: http://www.lastline.com

[3] M. Egele, T. Scholte, E. Kirda, C. Kruegel: "A survey on automated dynamic malware analysis techniques and tools", Technical University Vienna, SAP Research, Institut Eurecom ad University of California, 2010.

[4] P. Szor: "Virus research and defense"; Addison-Wesley, 2005.

[5] B. Schneier: "Attack trees", Dr. Dobb's Journal, Dec. 1999.

[6] NIST: "NIST 800-30 - Risk management guide for information technology systems", U.S. Department of Commerce, 2002.

[7] M. L. Garcia: "Design and Evaluation of Physical Protection Systems", Butterworth-Heinemann, 2007. pp. 1–11.

[8] A. V. Aho, R. Sethi, J. D. Ullman: "Compilers: Principles, Techniques, and Tools", Addison-Wesley, 2006.

[9] G. Jacob, H. Debar, E. Filiol: "Malware Behavioral Detection by Attribute-Automata Using Abstraction from Platform and Language", in: Recernt advances in intrusion detection, LNCS 5758, 2009, pp 81-100.

[10] K.M. Bischoff: "Ox: An Attribute-Grammar Compiling System based on Yacc, Lex, and C", Tutorial Introduction, 1993, ftp://ftp.cs.iastate.edu/pub/ox/.

[11] Bulletin: "The wildlist – viruses out in the wild", http://www.virusbtn.com/resources/wildlists/index (last access 12/09/2013), Virus Bulletin Ltd.

[12] R. Batra: API Monitor: http://www.rohitab.com/apimonitor.

[13] Anubis: http://anubis.iseclab.org.

[14] Joe Sandbox: http://www.joesecurity.org

[15] VirusTotal: https://www.virustotal.com.

[16] SANS Institute: "Case Study: 2012 DC3 digital forensic challenge basic malware analysis exercise", https://www.sans.org/reading-room/whitepapers/malicious/case-study-2012-dc3-digital-forensic-challenge-basic-malware-analysis-exercise-34330

[17] C. Gonzalez, M. Thomason: "Syntactic Pattern Recognition", Addison-Wesley, 1978.