Problem Characterization and Abstraction for Visual Analytics in Behavior-Based Malware Pattern Analysis

Markus Wagner, Wolfgang Aigner, Alexander Rind
Institute of Creative\Media/Technologies
St. Pölten Univ. of Applied Sciences, Austria
<first name>.<last name>@fhstp.ac.at

Hermann Dornhackl, Konstantin Kadletz, Robert Luh, Paul Tavolato
Institute of IT Security Research
St. Pölten Univ. of Applied Sciences, Austria
<first name>.<last name>@fhstp.ac.at

ABSTRACT

Behavior-based analysis of emerging malware families involves finding suspicious patterns in large collections of execution traces. This activity cannot be automated for previously unknown malware families and thus malware analysts would benefit greatly from integrating visual analytics methods in their process. However existing approaches are limited to fairly static representations of data and there is no systematic characterization and abstraction of this problem domain. Therefore we performed a systematic literature study, conducted a focus group as well as semi-structured interviews with 10 malware analysts to elicit a problem abstraction along the lines of data, users, and tasks. The requirements emerging from this work can serve as basis for future design proposals to visual analytics-supported malware pattern analysis.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*User-centered design, Evaluation/methodology*

Keywords

Evaluation, malicious software, malware analysis, problem characterization and abstraction, visual analytics

1. INTRODUCTION

Due to the increasing threats from malicious software (malware), monitoring of vulnerable systems will become increasingly important. This applies to networks, individual computers, as well as mobile devices (e.g. [36, 44, 11]). For this purpose, there are various approaches and techniques available to detect or to capture malicious software as presented in Sect. 2 and 5. To overcome the limitations of static signature-based approaches, behavior-based malware detection can be used [9]. Here, the dynamic behavior of software is analyzed by logging execution traces of potentially malicious code. On the one hand, such traces of dynamic software behavior can become very large very quickly, resulting in tens of thousands of lines to be analyzed. Furthermore large numbers of hitherto unknown malware samples emerge every day and need to be triaged. Because of that and the fact that manual analysis by domain experts is very cumbersome, automated data analysis methods are needed. In order to automate this process as much as possible, patterns of particular call sequences need to be specified and categorized as being potentially harmful or harmless. These patterns can then semi-automatically be detected and analyzed in context. On the other hand, this process cannot be automated completely as domain experts need to be in the loop to identify, correct, and disambiguate intermediate results. This combination of large amounts of data, complex data analysis needs, and the combination of automated data analysis with analytical reasoning by domain experts lends itself very well to the notion of visual analytics [34, 15].

In our work, we follow the paradigm of problem-oriented research, i.e., working with real users to solve their tasks [28]. To be able to support domain experts in their demanding task of detecting and refining patterns of malicious behavior in malware execution traces, it is imperative to perform a thorough problem characterization and abstraction as a first step. The main motivation is to analyze and characterize the data to be dealt with as well as user requirements and features needed for a malware detection system supported by visual analytics methods.

In particular, we follow the *nested model for visualization design and validation* as proposed by Munzner [23]. It is a unified approach that structures visualization design into four levels and combines these with appropriate validation methods to mitigate threats to validity at each level. Starting from the top, the levels are *domain problem and data characterization*, *operation and data type abstraction*, *visual encoding and interaction design*, and *algorithm design*. In the paper at hand, we specifically focus on the two top levels, problem characterization and abstraction. These two steps are crucial in reaching a common language and understanding between domain experts and visualization researchers, and the requirements gathered provide the basis against which design proposals can be judged [28]. Abstracting from the concrete domain vocabulary to the vocabulary of visualization also helps to match known visualization solutions from other domains to the problem at hand and vice versa. Moreover, disclosing problem characterization and abstractions for particular domain problems serves as valuable basis for researchers other than those who have conducted the research. Other researchers might also propose

VizSec'14, November 10 2014, Paris, France
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2826-5/14/11...$15.00
http://dx.doi.org/10.1145/2671491.2671498

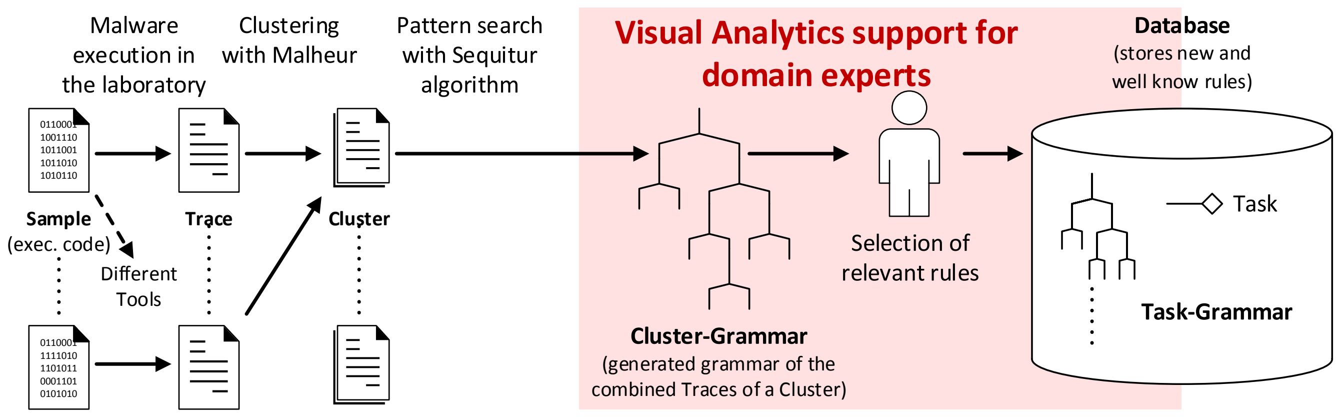
Malware execution in the laboratory
Clustering with Malheur
Pattern search with Sequitur algorithm
Visual Analytics support for domain experts
Database (stores new and well know rules)
Sample (exec. code)
Different Tools
Trace
Cluster
Selection of relevant rules
Task
Task-Grammar
Cluster-Grammar (generated grammar of the combined Traces of a Cluster)

Figure 1: Different stages of a behavior-based malware detection approach as identified in focus group sessions with IT-security experts. The red area shows the part to be supported with visual analytics methods.

different solutions based on this groundwork. As shown in the survey by Lam et al. [17], such contributions are still very rare. We aim for providing such a contribution in the domain of behavior-based malware pattern analysis.

This paper is organized as follows: Sect. 2 provides background knowledge about the work of our collaborators. In Sect. 3 we present related work in the fields of malware analysis and problem-oriented visual analytics research. Furthermore in Sect. 4 we describe the research methods for the results which we present in Sect. 5. At the end of the paper we summarize and abstract the domain problem of malware pattern analysis along the lines of data, users, and tasks for visual analytics-supported systems, which is followed by the conclusion and future work in Sect. 7.

2. BACKGROUND

Malware is undoubtedly one of today's biggest threats to the Confidentiality/Integrity/Availability (CIA) triangle of information security [33]. Malware has become a common tool in digital theft, corporate and national espionage, spam distribution and attacks on infrastructure availability.

Behavior-based Malware Recognition: Malicious behavior in software is identified through static or dynamic analysis [9]. On the one hand, when statically analyzing a possibly malicious software sample, the binary file is usually disassembled and dissected function by function. Dynamic analysis, on the other hand, focuses on the sample's behavior [9]: The file is executed on a test system and its activity is observed and recorded. Both approaches yield patterns or rules that are later used for detection and classification of malicious software.

Current malware detection/classification commonly uses a signature-based approach: Known malware is described by its syntactic characteristics – mostly bit strings or simple patterns (e.g., defined by regular expressions). Signature-based detection has several shortcomings [5]: 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. Secondly, signature-based techniques can only detect malware that has already been identified and analyzed; new species or hitherto unknown variants are generally overlooked. An alternative to signature-based detection is the so-called behavior-based approach. Here, a sample's activity is analyzed during execution using dynamic analysis techniques. Afterwards, a previously defined 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 dynamic behavior – in whatever disguise it might appear.

This paper focusses on the visualization of the pattern extraction and recognition process used in a research project on formal definition of malware behavior [8].

Pattern Extraction Process: The process of extracting relevant behavior patterns is split into several stages (see Fig. 1). Initially, the sample under scrutiny is executed inside an isolated, partially virtualized laboratory environment. Tools such as APImon[1] and Procmon[2] monitor all activities and generate a report (i.e. trace) of system and API calls sequentially invoked by the sample. These traces are then clustered using Malheur, an automated behavior analysis and classification tool developed by Rieck et al..[3] In the next step, all traces within a cluster are concatenated and processed using the Sequitur algorithm [24]. Originally developed for file compression, Sequitur automatically replaces frequent patterns with short symbols, effectively generating a context-free grammar in the process, referred to as cluster-grammar. A human analyst must then assess this grammar and extract rules describing potentially relevant behavior.

Here is where visualization comes in: The selection of relevant rules relies heavily on the accessibility and readability of the extracted information.

Once a relevant rule has been identified, it is stored in the pattern database and assigned to a task of the malicious behavior schema [8]. The grammar of the abstracted tasks therein, i.e. the task-grammar, is the foundation for the automated generation of parsers that are ultimately used to detect malicious behavior in newly submitted traces.

[1]`http://www.rohitab.com/apimonitor`, accessed July 07, 2014.
[2]`http://technet.microsoft.com/en-us/sysinternals/bb896645`, accessed July 07, 2014.
[3]`http://www.mlsec.org/malheur/`, accessed July 07, 2014.

3. RELATED WORK

Currently, problem-oriented research is underrepresented in visualization literature – in particular works on problem characterization and abstraction are rare [17, 28], even though these are essential for design and implementation of suitable visual analytics solutions. Therefore, we will first present some notable examples of problem characterization papers in other domains and then focus on visualization work for malware analysis. Visualization techniques from this area will be discussed in more detail in the literature research (Sect. 5.1).

Problem Characterization and Abstraction: Sedlmair et al. [26] studied the daily routines of automotive engineers and their tool support using interviews and task observation. After analysis along different collaboration settings, their main contributions are system requirements for multiple display environments. The MizBee design study [21], set in comparative genomics, starts with the characterization of questions asked in this problem domain as the first of four contributions. For this, they conducted interviews with two expert biologists, who work in the area. Tory et al. [35] conducted a field study in the domain of building design to investigate the current use of visualization in meetings. The design requirements for RelEx [27] were based on detailed characterization of data, tasks, and existing tools, which were obtained using literature research, contextual observation, semi-structured interviews, and a focus group. Fink et al. [10] articulated a set of design principles for visualization systems in cyber security but focused on high-resolution display working environments for security analysts. Additionally they presented prototypes according to these design principles. Goodall et al. [12] conducted contextual interviews to gain understanding of the intrusion detection workflow of security analysts. Additionally they proposed a three-phased model where tasks could be decoupled by necessary know-how. This way organizations get more flexibility in training new analysts. However, none of these studies tackled behavior-based malware pattern analysis.

Malware Analysis and Visualization: Lee et al. [20] made a good case for visualization in malware analysis, which they propose is needed to recognize and extract unseen malware patterns. The survey by Shiravi et al. [31] described and categorized 38 different network security visualization systems, which they divided into 5 different classes of use-cases. The definition of the classes was based on the behavior of the malicious activities, which could be detected with these tools. Some of the presented approaches also supported methods for interactive data exploration. In contrast to this survey, we are looking at the behavior-based malware analysis on system and API call level. Likewise, a part of Conti's book [7] is dedicated to malware analysis. However, all of the mentioned approaches are related to visualization of network traffic and not of malware execution traces. Approaches for malware detection in general are covered in the surveys of Egele et al. [9] and Bazrafshan et al. [3].

Albeit some work has been performed in visualization for malware detection, it can be recognized that so far, no work has been published on performing problem characterization and abstraction in the domain of malware analysis from a visual analytics perspective. To close this gap and provide a basis for further designers of visual analytics systems in this domain, we chose to investigate it using methods from human-computer interaction [19, 30] as described next.

4. METHOD

To ensure a knowledgeable characterization and abstraction of malware pattern analysis along the triangle of data, users, and tasks [22], we followed a threefold approach consisting of systematic literature research, a focus group [19] and semi-structured interviews with domain experts [19].

The methods applied in our research are related to cognitive task analysis (CTA) [39]. Our threefold approach includes the classification families 1 (*observations and interviews*) and 2 (*process tracing*) of Wei and Salvendy's classification of CTA methods [39]. For the *observations* and *process tracings* we are using example views during the interviews.

4.1 Literature Research

In the first step, we used different keyword combinations (e.g. malware, malicious software, visual analytics, visualization, time-oriented data, security, etc.). In the second step, we searched for the authors of the currently best matching papers and combined them with the currently best matching keywords of the previous research. Based on this search strategy, it was possible to find 26 different scientific publications on malware analysis for IT-security on local hosts. In order to refine our results, we investigated all the abstracts and conclusions and removed less appropriate papers. This lead to a number of 6 highly relevant papers and 5 tools that could be identified.

4.2 Focus Group

The focus group consisted of 7 people: 4 IT-security experts who are working in a research project on malware recognition via formal description of the behavior (Malware-Def team) and 3 visual analytics experts. The iterations of the focus group meetings were established on a monthly basis with a duration of approximately 3 hours (currently 4 sessions). The results of each meeting were documented by written notes. The main aim was to find out:

- Which datasets are useful and interesting to visualize?
- Which data structures will be processed?
- What are possible visualization scenarios?
- How could expert knowledge be extracted or included?

In addition to these basic questions we matched the domain-specific vocabulary and established mutual understanding. Further, we managed to create a bird's-eye view for a better understanding of the different analysis steps of behavior-based malicious software detection (see Fig. 1).

4.3 Interviews

Based on the results of the focus group, we developed interview guidelines for semi-structured interview sessions.

Study Design and Material: The interviews were divided into two parts: brainstorming and example view exploration. The first part was structured along the following main questions related to behavior-based malware detection:

- What is the workflow of analysis?
- Which tools are used?
- How is data collected and analyzed?
- Which data records are interesting?
- What will be done with the discovered patterns?
- Are the data records currently graphically evaluated?
- Are there any visualization tools available for this field?

After the brainstorming part each person was exposed to 6 visualization techniques (Arc Diagram [37], Multiple

View [13], OutFlow [41], Wordtree [38], Parallel Tag Cloud [6] and Pixel-Oriented Visualization [14]). We selected these visualization techniques after our preliminary problem understanding from the focus group sessions for being potentially applicable in the application domain and covering a broad range of different options for graphical representations. With these views we aimed to get answers for the following questions:

- Is this or a similar visualization method already known?
- Could you read information out of this visualization?
- Is this visualization method usable for your work?

The example views were printed on paper and the same order was used for each person. See our supplementary material for the complete interview guideline used (in German) as well as the shown example views.[4]

Participants: We selected a group of 10 IT-security experts to participate in the interview sessions. All the interviewed persons are working in the field of malware detection or in a related field of IT-security at three different Austrian companies. The interviewed group includes 2 female and 8 male participants (see Table 1 for further details). Four of the interview partners were also members of the focus group.

Person	Organization	Age	Knowledge	Gender	Education
P1	R1	50-59	expert	m	PhD
P2	R1	30-39	expert	m	MSc
P3	R1	20-29	expert	m	MSc
P4	R1	20-29	expert	m	MSc
P5	C1	20-29	expert	m	MSc
P6	C1	20-29	expert	m	MSc
P7	C2	20-29	expert	f	MSc
P8	R2	30-39	basic	f	MSc
P9	R2	30-39	basic	m	BSc
P10	R2	30-39	basic	m	MSc

Table 1: Data of the interviewed persons. All the persons of R1 are related to the focus group. (R := research group, C := company)

Procedure: Each interview was scheduled for approximately one hour and was documented by audio recording and notes. The results of the interview sessions were combined and evaluated for the result presentation in Sect. 5.

5. RESULT

In this section we present the results of the literature research followed by the results of the focus group and the semi-structured interviews.

5.1 Literature Research

In addition to the previously presented surveys in Sect. 3, we examined 5 existing approaches from 6 papers from the perspective of visual analytics (based on Shneiderman's *Visual Information Seeking Mantra* [32]).

Yao et al. [43] describe the design of an interactive framework for automated trust negotiation (*ATN*). With ATN, it is possible for the user to show credentials, policies and to analyze the relations of negotiated components. These sessions used a trust target graph (TTG) which is built up from the two negotiated ATN sessions. For the visualization of the ATN session, a node-link diagram was used.

[4]http://mc.fhstp.ac.at/supp/VizSec14

Willems et al. [40] describe *CWSandbox* and a combination of behavior-based malware detection with API hooking, and DLL injection. Furthermore Trinius et al. [36] created a parameterized abstraction of detailed behaviors of malicious software based on CWSandbox. For the visualization of the malware they used two different visualization methods. On the one hand they used treemaps and on the other hand they used thread graphs [42].

Shabtai et al. [29] report that visualization of raw data did not make sense for the experts. To improve results for the visualization they implemented an intelligent visualization interface called *VISITORS* which was an extension to the *KNAVE-II* tool applied earlier mostly in the medical domain. The tools contained the following features: temporal data abstraction, knowledge-based interpretation, summary of data, queries, visualization, and exploration of a large amount of time-oriented data. Furthermore the system includes a signal-concept visualization over time (using divided/stacked bar charts and index charts), a visualization for multiple concepts' association over time (using a sort of parallel coordinates), and indented lists.

Yee et al. [44] worked on reverse engineering of a binary executable by transforming a stream of bytes of a program into a sequence of machine instructions. The static and the dynamic debugger system interacted with a graph visualization system called *Mini-Graph* to visualize the analysis data of the targeted executable file. Furthermore it contained a tabular text visualization area which shows different data (e.g. address, hex, comments). This way the program flow of the targeted system could be reconstructed and it was easier to detect fragments of malicious instructions.

The *VERA* approach introduced by Quist and Librock [25] uses dynamic analysis to visualize the flow of a program. The visualization system uses a 2D representation of the data, which was transformed into a 3D space. They claim that a 2D representation is very useful for a quick initial analysis, but the 3D view provides a convincing view of the data by offering better introspection and zooming features for the code. With VERA, the authors provided a navigable tool to explore the data (panning, zooming and filtering). Based on a user study, they showed that the system is very helpful for them and for (non)experienced system users.

Summary

All the presented tools operated locally and use 2D visualizations to present the data. Only VERA uses 2D and 3D visualizations for the data representation. ATN, CWSandbox, and Mini-Graph visualize malware data using node-link diagrams. Furthermore, the CWSandbox tool also uses treemaps. Only VISITORS uses bar charts for the visualization and combines this with index charts and parallel coordinates. It can be seen that most utilized visualization techniques are node-link diagrams or graphs. Particularly, interactivity is rather restricted in the mentioned approaches. CWSandbox does not support any kind of interaction for data exploration. All the other tools provide basic interaction features, but only VERA and VISITORS are more elaborated. VERA supports interaction, zooming, filtering, and panning functionalities. The VISITORS tool supports zooming, filtering and details on demand functions. Thus, it became apparent that visual analytics methods are less common in currently available approaches for malware analysis.

5.2 Focus Group

In the first meeting, the visual analytics experts and the IT-security experts established a common understanding of their respective fields of work and their objectives. In subsequent meetings the technical vocabulary has been developed and clarified. Furthermore, we iteratively worked on discussing the four questions as formulated in Sect. 4. The data to work with is generated automatically (see Fig. 1 and Sect. 2). One possible visualization scenario consists of the Sequitur [24] file. It should be possible to select one call sequence to see if this is a known pattern or not. Furthermore it should be possible to store new patterns as expert knowledge in the system.

Rule	Count	Sequence
27 → 3311 3329	8	RegOpenKeyW RtlEnterCriticalSection Rt...
28 → 40 41	5	RegQueryValueExW RtlInitUnicodeString...
29 → RtlNtSt...	76	RtlNtStatusToDosError
...	...	...

Table 2: Example of a Sequitur cluster grammar file.

Table 2 shows an excerpt of a grammar as produced by the Sequitur algorithm: Column 1 contains the grammar rules with its left and right part separated by →. The right side very often consists of 2 non-terminals, but it is not limited and actually can go up to 10 or more. Numbers simplify non-terminals; terminals represent system and API calls and the names of the calls are used. Column 2 ("Count") gives the number of use of the rule within the derivation of all the traces of one cluster stringed together. Column 3 gives the terminal string that is finally derived by this rule (during derivation).

5.3 Interviews

To analyze the requirements for a future visual analytics tool, we conducted 10 semi-structured interviews as described in Sect. 4.

Brainstorming

What is the workflow of analysis? As a first result it became apparent that the workflow depends on whether you are working as an anti-virus manufacturer or as a malware analyst. From the view of a malware analyst, the best choice of action is to perform a combination of static and dynamic analysis. The basic approach is to execute malware samples in an isolated area on virtual hosts and/or on native hosts for analysis and pattern finding (testing on native host takes more time because reinstalling of the machine after the test is more time consuming). Regarding used operating systems, the participants reported that most of the time their malware samples will be executed on all currently used Windows operating systems (Windows XP - Windows 8.1) and both x86 and x64 architectures. This is done to determine the sample's target system. All the activities of the malware samples are logged by a wide range of tools.

Which tools are used? For report generation, the research group R1 primarily utilized APImon and Procmon. T3sim (proprietary software of IKARUS Security Software), Joe Sandbox[5] and FireEye[6] were occasionally used to complement specific analyses. Furthermore, they use Malheur to cluster reports generated by the other tools. P7 used APImon, Procmon, Cuckoo Sandbox,[7] IDAPro,[8] FireEye and Joe Sandbox. She emphasized that: *"IDAPro is the Swiss Army Knife of a malware analyst."* The members of C1 use IDAPro, Anubis (formerly TTAnalyze) [2] and some different sandbox tools that were not specifically named. Additionally, R2 reported to work with IDAPro and Procmon as well.

How is data collected and analyzed? Our interview partners explained that after the application of one of the tools, the generated files have to be evaluated by hand. This is a very labor intensive task because each file contains several thousand lines. Additionally, not only the execution of malware samples is important, but also the examination of the final state of the machine on which a malware sample was executed.

Which data records are interesting? The interviewees gave quite a number of different examples of interesting data records. Specifically, it was named that when examining the static part of the data, you are able to see signatures, hashes, and strings (parts of the program code which could be identified as a collection of ASCII symbols). It has also been mentioned, that the network communication of a program is very interesting: information like where does the program connect to?, upload?, download? can be answered by exploring this data. A further important finding is that it is possible that malware changes its activities at runtime. This means that malware could contain encrypted code which will be decrypted at runtime. In addition to the program activities, our interview partners mentioned that it is very interesting to see whether a program registers itself with one of the autostart lists of the operating system.

What will be done with the discovered patterns? R1 reported that all the patterns he finds will be compared with the currently stored patterns in the database. Furthermore the semantic of a found pattern needs to be manually associated to a number of predefined categories. All the other persons explained that they do not store found patterns, but only report them if there is enough time. According to our interview partners, there is no tool that allows for storing found patterns for future evaluations. Finally an interesting insight was that all interviewed experts had their very own approaches toward pattern recognition that made it difficult to consolidate them to a more generalized view.

Are the data records currently graphically and visually evaluated? The interview partners reported that there are some visualization tools available but they often did not fit their needs. One of the tools mentioned is Procdot[9] which is a visualization tool for the traces/data generated by Procmon. Additionally, IDAPro generates a call graph to visualize the program calls of the analyzed malware sample. The tool Anubis colors the results green or red in addition if they are malicious or not.

Example Views

Arc Diagram: The feedback of the interviewees implies that this visualization technique is quite conceivable for pattern recognition and tamper detection for system and API

[5] `http://www.joesecurity.org/joe-security-products`, accessed July 14, 2014.

[6] `http://www.fireeye.com/`, accessed July 14, 2014.

[7] `http://www.cuckoosandbox.org/`, accessed July 14, 2014.

[8] `https://www.hex-rays.com/products/ida/`, accessed July 14, 2014.

[9] `http://www.cert.at/downloads/software/procdot.html`, accessed July 14, 2014.

calls. Similarly, it has been mentioned to be well suited for grammar and database visualization. By means of the arc thickness and diameter of the circle, the frequency and the intensity of the connection could be shown. In addition, color differentiation is also very important and helpful to distinguish the malware and the system and API call types. Furthermore, it was stated that this visualization technique could be highly suitable for the visualization of temporal processing but also that it looks a bit unstructured. One possible problem identified could be the scalability of the visualization technique with the amounts of data to be dealt with in malware analysis.

Multiple Views: The example we've used consisted of an overview of the data using bar charts and text on the left and a detailed view of a selected data entry on the right. This visualization method had a very high recognition factor by displaying the details of the selected data on the right according to other approaches. The interviewed experts suggested that this method could be well suited to represent behavior scores on the left and the sample's inherent API and system calls, including frequency of occurrence on the right. Additionally this visualization method was said to be potentially well-suited for the visualization of the Sequitur results and for comparing several different samples on system and API call level (e.g. on the left side there could be the sum of the same system and API calls and on the right side, in the detail view, one could compare them).

OutFlow: OutFlow was found to be applicable to visualize various system and API calls which yield the same results. Malignant combinations could be highlighted using color and after each intermediate step an assessment of the malignancy of the samples in percent could be specified. For example there are several different combinations of system and API calls to add a program to a system's auto start list/area/directory. As an extension to recognize loops, the interview partners suggested back links. Furthermore, OutFlow was identified for opening up the possibility to recognize unnecessary or more importantly, obfuscated code trying to mask the true intent of the sample. This method could also be used to visualize different execution threads and their dependencies (e.g. file-handles).

Wordtree: When discussing Wordtree, our interview subjects suggested that using a color differentiation of various malware families would be very helpful. It was mentioned that the use of different font sizes to represent the frequency of concurrency is not as important because a uniquely occurring system or API call combination is sufficient to wreak considerable damage. Furthermore, this technique was said to be potentially helpful for a stepwise categorization by visualizing subdivisions in order to specify the focus of the executed sample (e.g. network focused, calculation focused). Additionally, it seems to be well-suited for the visualization of system and API call sequences and possibly for the database structure, too. A good expansion option would be to lead the individual strands back together to locate patterns with the same result.

Parallel Tag Cloud: Considering this method it was mentioned that it would be useful for the side-by-side comparison of various samples. It would be interesting to correlate system and API calls to specific malware families ore to search for calls which are used by different malware families. However, it would be important to put the focus on the connections between the nodes rather than on the text size. R2 also put forward a word of caution: *"It seems as if only the most common elements of the data to be compared are displayed - this could be misleading."*

Pixel-Oriented Visualization: On the one hand, many of the interviewed experts mentioned that this technique would be well-suited to show data from different samples for comparison and for comparisons of reports over time (e.g. the occurrence of different types of malware over a time period). Additionally this method could be used for the visualization of disk partitions or encrypted data in the samples. On the other hand, the technique was also critically viewed, as for example by R1: *"This visualization technique is ill-suited for the group's purpose. It rather seems to be handy for certain statistical evaluations."*

Combinations: Most of the interviewees indicated that a combination of multiple views, Arc Diagram, and Wordtree would be preferred, followed by OutFlow and pixel-oriented visualization. In addition P3, P4 and P6 suggested to rotate the Arc Diagram by 90°. The Parallel Tag Cloud, in turn, has been described as the least useful solution.

6. DATA–USERS–TASKS ANALYSIS

Above we have characterized the domain problem of behavior-based malware analysis using literature research, focus group meetings, and semi-structured interviews. Next, we summarize and abstract the domain problem using the Data–Users–Tasks design triangle [22]. This high-level framework is structured around three questions:

- What kinds of data are the users working with? (*data*)
- Who are the users of the VA solution(s)? (*users*)
- What are the (general) tasks of the users? (*tasks*)

Based on the answers to these questions, designers of VA methods can find or design appropriate visual representations of the data along with appropriate analysis and interaction methods to support a domain problem.

Data: In dynamic analysis malware analysts work with collections of traces, which are sequences of relevant system or API *calls*. In addition call parameters and return values of the calls can be exposed. However, they do not examine these traces directly because of the large data volume. Our collaborators uses the Sequitur algorithm [24] to generate context-free grammars from the clusters of traces, which they refer to as *cluster grammars* (Fig. 1). Each grammar describes the derivation of a terminal string (all traces of one cluster stringed together). Each node of the parse tree has a rule of the grammar associated with it and derives a sub-sequence of terminal symbols. Table 2 shows examples of such *rules* along with the number of occurrences of this rule in the parse tree and the terminal sub-sequence derived from this rule. Additionally, the distribution of rules over the traces in a cluster is available. For example a cluster of 20 malware samples might yield a total trace of 20,000 calls and a cluster grammar of 1,000 rules. These data are currently stored in a database together with metadata for system and API calls and a taxonomy of malware behaviors. Furthermore the database contains call sequences described by rules that have already been assigned to a certain malicious behavior, referred to as *task grammar*.

The system and API calls have a value on a *nominal* scale with a cardinality greater than 100. (Sub-)sequences of calls are *time-oriented data* on an *ordinal time scale* with *instants* as time primitives [1]. The parse tree for each cluster is a graph with rules associated to nodes (except termi-

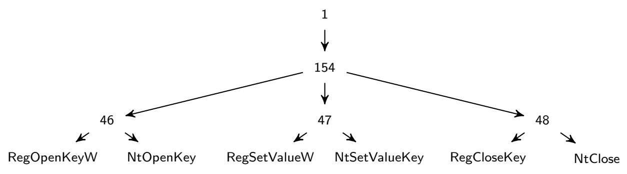

Figure 2: Simplified example of a parse tree.

nal nodes), the edges representing the expansion of the rule (Fig. 2). The parse tree of a cluster can be modeled as a *simple directed acyclic* graph [16] (i.e. a tree). The intermediate nodes are non-terminals of the grammar represented by a number and the end nodes are terminal symbols of the grammar represented by system calls (see Fig. 2). Of main interest is the number of reoccurrences of a non-terminal and their locations. Alternatively, the cluster grammar can be modeled as a *network* with rules as nodes and their composition as edges. The underlying graph is *simple, directed, acyclic,* and usually not planar [16]. Node attributes are primarily call sequences or single calls and quantitative data such as occurrence counts over traces. There are no edge attributes.

Users: Malware analysis is performed by domain experts, *malware analysts.* These users have a strong computing background – typically a university degree in computer science or IT security. They command background knowledge about system and API calls, malware vectors, and a particular intuition how harmless calls can combine to malicious behavior. The users are comfortable with combining a range of different tools such as command line, text editor, and adhoc developed software but have no dedicated experience with Visual Analytics solutions. Yet, they are willing to familiarize themselves with a new tool because they need to perform malware analysis often and for extended periods. However, malware analysis is a specialist activity, so there will be relatively few users.

Tasks: The primary task of malware analysts is to *select relevant rules from the cluster grammar, categorize them by a malicious behavior task, and store them with the task grammar* (i.e. database). Secondary tasks include the manual adaptation and fine-tuning of rules found in the cluster grammar, comparing rules from the cluster grammar to rules already existing in the task grammar, and creating new rules manually either directly from traces or from their background knowledge/literature. Finding relevant of rules is an ill-defined problem and depends on many factors, in particular occurrence count, distribution over traces, and background knowledge of the involved system and API calls.

Overall, malware analysis is *pattern discovery* [18], i.e. discovering relevant call sequences in traces. The primary task can be abstracted [4] as *producing* rules for the task grammar. For this, users must first *identify* rules by *exploring* the cluster grammars, *browsing* by particular occurrence counts, with *special focus* on system or API calls.

7. CONCLUSION AND FUTURE WORK

Based on the performed literature research, focus group meetings, and semi-structured interviews we formulated a problem characterization and analysis. The interviewees enumerated many tools for the different work steps depending on the focus of their work/research. Furthermore they analyze the collected data usually manually because the currently available tools do not cover all the needs of the interviewees. By means of the six presented example views it was possible to identify preferred visual representation combinations (e.g. multiple view + arc diagram + word tree).

Summarizing by means of data–users–tasks, we can abstract the parse tree of a cluster grammar as a simple directed acyclic graph with nominal data attributed to the nodes. The users of the future system will be malware analysts (domain experts). Furthermore the main tasks are to select different rules, categorize them by their task and store them in the database as well as manual adaption and/or tuning of found rules.

Unlike the existing work in IT-security, this problem characterization and abstraction focused on malware pattern analysis and constitutes a solid base for future work. It allows visual analytics designers to create and judge design proposals for future solutions and it helps to identify similarities to other domains and their visual analytics solutions. Finally, it also aids domain experts to reflect about their own work. While designing such visual analytics solutions domain experts should of course stay involved in a user-centered design process [30]. We intend to pursue this path in collaboration with the members of the focus group.

8. ACKNOWLEDGMENT

This work was supported by the Austrian Science Fund (FWF) via the KAVA-Time project no. P25489 and the Austrian Federal Ministry for Transport, Innovation and Technology via KIRAS project no. 836264. Many thanks to C. Niederer and M. Zeppelzauer for their feedback.

9. REFERENCES

[1] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data.* Springer, 2011.

[2] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A tool for analyzing malware. In *15th Ann. Conf. Europ. Inst. Computer Antivirus Research, EICAR*, 2006.

[3] Z. Bazrafshan, H. Hashemi, S. Fard, and A. Hamzeh. A survey on heuristic malware detection techniques. In *Conf. on Info. and Knowledge Technology*, pages 113–120, 2013.

[4] M. Brehmer and T. Munzner. A multi-level typology of abstract visualization tasks. *TVCG*, 19(12):2376–2385, 2013.

[5] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *India Software Eng. Conf.*, pages 5–14. ACM, 2008.

[6] C. Collins, F. Viegas, and M. Wattenberg. Parallel tag clouds to explore and analyze faceted text corpora. In *Symp. on Visual Analytics Science and Technology*, pages 91–98, 2009.

[7] G. Conti. *Security data visualization: graphical techniques for network analysis.* No Starch Press, 2007.

[8] H. Dornhackl, K. Kadletz, R. Luh, and P. Tavolato. Malicious behavior patterns. In *IEEE Int. Symp. on Service Oriented System Eng.*, pages 384–389, 2014.

[9] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comp. Surv.*, 44(2):6:1–6:42, 2008.

[10] G. Fink, C. North, A. Endert, and S. Rose. Visualizing cyber security: Usable workspaces. In *Int. Workshop on Vis. for Cyber Sec.*, pages 45–56, 2009.

[11] E. Gelenbe, G. Gorbil, D. Tzovaras, S. Liebergeld, D. Garcia, M. Baltatu, and G. Lyberopoulos. Security for smart mobile networks: The NEMESYS approach. In *IEEE Global High Tech Congr. on Electronics*, pages 63–69, 2013.

[12] J. R. Goodall, A. Komlodi, and W. G. Lutters. The work of intrusion detection: Rethinking the role of security analysts. In *Proc. of the 10th Americas Conf. on Info. Systems*, pages 1421–1427, NY, 2004.

[13] D. Gotz, H. Stavropoulos, J. Sun, and F. Wang. ICDA: A platform for intelligent care delivery analytics. *AMIA Annual Symp. Proceedings*, 2012:264–273, 2012.

[14] D. Keim. Designing pixel-oriented visualization techniques: theory and applications. *TVCG*, 6(1):59–78, 2000.

[15] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering the information age: solving problems with visual analytics.* Eurographics, 2010.

[16] A. Kerren, H. C. Purchase, and M. O. Ward, editors. *Multivariate Network Visualization.* LNCS 8380. Springer, Cham, 2014.

[17] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale. Empirical studies in information visualization: Seven scenarios. *TVCG*, 18(9):1520–1536, 2012.

[18] S. Laxman and P. S. Sastry. A survey of temporal data mining. *Sadhana*, 31(2):173–198, 2006.

[19] J. Lazar, J. H. Feng, and H. Hochheiser. *Research Methods in Human-Computer Interaction.* Wiley, 2010.

[20] D. Lee, I. S. Song, K. Kim, and J.-h. Jeong. A study on malicious codes pattern analysis using visualization. In *Int. Conf. on Info. Science and Applications*, pages 1–5, 2011.

[21] M. Meyer, T. Munzner, and H. Pfister. MizBee: A multiscale synteny browser. *TVCG*, 15(6):897–904, 2009.

[22] S. Miksch and W. Aigner. A matter of time: Applying a data-users-tasks design triangle to visual analytics of time-oriented data. *Computers & Graphics*, 38:286–290, 2014.

[23] T. Munzner. A nested model for visualization design and validation. *TVCG*, 15(6):921–928, 2009.

[24] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Int. Res.*, 7(1):67–82, 1997.

[25] D. Quist and L. Liebrock. Visualizing compiled executables for malware analysis. In *Int. Workshop on Vis. for Cyber Sec.*, pages 27–32, 2009.

[26] M. Sedlmair, D. Baur, S. Boring, P. Isenberg, M. Jurmu, and A. Butz. Requirements for a MDE system to support collaborative in-car communication diagnostics. In *Workshop on Beyond the Laboratory: Supporting Authentic Collaboration with Multiple Displays*, 2008.

[27] M. Sedlmair, A. Frank, T. Munzner, and A. Butz. RelEx: Visualization for actively changing overlay network specifications. *TVCG*, 18(12):2729–2738, 2012.

[28] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. *TVCG*, 18(12):2431–2440, 2012.

[29] A. Shabtai, D. Klimov, Y. Shahar, and Y. Elovici. An intelligent, interactive tool for exploration and visualization of time-oriented security data. In *Int. Workshop Vis. Comp. Sec.*, pages 15–22. ACM, 2006.

[30] H. Sharp, Y. Rogers, and J. Preece. *Interaction Design: Beyond Human-Computer Interaction.* John Wiley & Sons, 2nd edition, 2007.

[31] H. Shiravi, A. Shiravi, and A. Ghorbani. A survey of visualization systems for network security. *TVCG*, 18(8):1313–1329, 2012.

[32] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *IEEE Symp. on Visual Languages*, pages 336–343, 1996.

[33] G. Stoneburner, A. Y. Goguen, and A. Feringa. SP 800-30. risk management guide for information technology systems. Technical report, NIST, 2002.

[34] J. J. Thomas and K. A. Cook. *Illuminating the path: The research and development agenda for visual analytics.* IEEE Comp. Society Press, 2005.

[35] M. Tory and S. Staub-French. Qualitative analysis of visualization: A building design field study. In *Proc. Workshop BEyond Time and Errors: Novel evaLuation Methods for Information Visualization*, pages 7:1–7:8. ACM, 2008.

[36] P. Trinius, T. Holz, J. Gobel, and F. Freiling. Visual analysis of malware behavior using treemaps and thread graphs. In *Int. Workshop on Vis. for Cyber Sec.*, pages 33–38, 2009.

[37] M. Wattenberg. Arc diagrams: Visualizing structure in strings. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 110–116, 2002.

[38] M. Wattenberg and F. Viegas. The word tree, an interactive visual concordance. *TVCG*, 14(6):1221–1228, 2008.

[39] J. Wei and G. Salvendy. The cognitive task analysis methods for job and task design: review and reappraisal. *Behaviour & Information Technology*, 23(4):273–299, 2004.

[40] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using CWSandbox. *IEEE Sec. Privacy*, 5(2):32–39, 2007.

[41] K. Wongsuphasawat and D. Gotz. Outflow: Visualizing patient flow by symptoms and outcome. In *IEEE VisWeek Workshop on Visual Analytics in Healthcare, Providence, Rhode Island, USA*, 2011.

[42] Y. Xia, K. Fairbanks, and H. Owen. Visual analysis of program flow data with data propagation. In J. R. Goodall, G. Conti, and K.-L. Ma, editors, *Vis. for Comp. Sec.*, LNCS 5210, pages 26–35. Springer, 2008.

[43] D. Yao, M. Shin, R. Tamassia, and W. Winsborough. Visualization of automated trust negotiation. In *IEEE Workshop on Vis. for Comp. Sec.*, pages 65–74, 2005.

[44] C. L. Yee, L. L. Chuan, M. Ismail, and N. Zainal. A static and dynamic visual debugger for malware analysis. In *Asia-Pacific Conf. on Communications*, pages 765–769, 2012.