Sequitur-based Inference and Analysis Framework for Malicious System Behavior

Robert Luh[1,3], Gregor Schramm[1], Markus Wagner[2] and Sebastian Schrittwieser[1]
[1]*Josef Ressel Center TARGET, St. Pölten University of Applied Sciences, Matthias Corvinus St. 15, St. Pölten, Austria*
[2]*Institute for Creative Media Technologies, St. Pölten UAS, Matthias Corvinus St. 15, St. Pölten, Austria*
[3]*Faculty of Technology, De Montfort University, Leicester, U.K.*

Keywords: Malware Analysis, System Behavior, Attribute Grammar, Knowledge Generation.

Abstract: Targeted attacks on IT systems are a rising threat against the confidentiality of sensitive data and the availability of critical systems. With the emergence of Advanced Persistent Threats (APTs), it has become more important than ever to fully understand the particulars of such attacks. Grammar inference offers a powerful foundation for the automated extraction of behavioral patterns from sequential system traces.
In order to facilitate the interpretation and analysis of APTs, we present a grammar inference system based on Sequitur, a greedy compression algorithm that constructs a context-free grammar (CFG) from string-based input data. Next to recursive rule extraction, we expanded the procedure through automated assessment routines capable of dealing with multiple input sources and types. This enables the identification of relevant patterns in sequential corpora of arbitrary quantity and size. On the formal side, we extended the CFG with attributes that help depict the extracted (malicious) actions in a comprehensive fashion. The tool's output is automatically mapped to the grammar for further parsing and discovery-focused pattern visualization.

1 INTRODUCTION

IT systems are threatened by a growing number of different cyber-attacks. With the emergence of Advanced Persistent Threats (APTs), the focus shifted from off-the-shelf malware to attacks that are tailored to one specific entity. These targeted threats are driven by different motivations and often cause significantly more damage thanks to their focus on espionage or high-profile sabotage typically conducted by dedicated groups within organized crime, industry, or nation state intelligence.

APTs are increasingly affecting less prominent targets as well. In 2013 alone, "economic espionage and theft of trade secrets cost the American economy more than \$19 billion" (Munsey, 2013). 60% of espionage attacks now target small and medium businesses whereas each reported data breach exposes over a million identities on average (Symantec, 2015). The retail, healthcare, and finance sectors find themselves in the crosshairs most often.

While APTs use malware like most conventional attacks, their level of complexity and sophistication is usually much higher. This is problematic especially since defensive measures offered by security vendors often utilize the same detection approaches that have been used for years. The major drawback of these primarily signature-based systems is that the binary patterns required for detection are unlikely to exist at the time of attack, as most APTs are tailored to one specific target, making them likely to utilize zero-day exploits (Bilge and Dumitras, 2012; Sood and Enbody, 2013). In addition, meta- and polymorphic techniques are employed to throw off signature-based systems while the multi-stage nature of APTs makes it generally difficult to interpret findings individually (Filiol et al., 2007; Luh et al., 2016a).

This increased complexity makes it necessary to explore novel techniques for threat intelligence and malicious activity detection on multiple layers. Behavior-based approaches are a promising means to identify illegal actions. No matter the stealth mechanisms employed, the attacker will sooner or later execute his or her action on target – be it data theft, sabotage, or other fraudulent activity. Behavior patters and anomalies signifying a deviation from a known baseline can then be used to detect the threat.

However, both pattern and anomaly detection systems usually suffer from a lack of semantic interpretation; the so-called *semantic gap*, the hard-tobridge difference in syntactic event information and actual attack semantics, remains an issue. Also, patterns are often manually assigned to represent analyst knowledge, while anomaly detection systems do not usually attempt to explain the identified deviations. This makes potential victims vulnerable to unknown attacks and does little to further the exploration of meaning behind the actions of malicious actors.

Successfully discovering potentially malicious system behavior boils down to three major problem domains: the automated generation of patterns that contribute to detecting and understanding complex multi-stage attacks, attack semantics, and the holistic view on targeted attacks and their many properties. Arguably, a powerful formal definition of malicious behavior is the foundation for all of these aspects.

In this paper, we propose an IT system behavior inference and classification methodology based on the Sequitur algorithm (Nevill-Manning and Witten, 1997) and formalized through a context-free grammar (CFG) extended by semantic attributes (attribute grammar). The approach combines a condensed formal definition with the generation of knowledge linked to the information security and malware analysis domains: Instead of manually defining the many terminals and production rules that the description of a behavior trace would require, we automate the process through an extension of Sequitur that is fully capable of determining and evaluating significant rules. This eliminates the analysts' need to come up with fixed patterns describing harmful or benign behavior.

Specifically, we contribute by:

- Defining an attribute grammar capable of depicting sequential behavior while retaining information about triggering process and parameters,
- Developing a grammar inference process based on the Sequitur algorithm for arbitrary system event traces,
- Expanding this approach to a knowledge discovery system supporting automated evaluation and extraction of potentially interesting patterns usable in further interpretation or visualization efforts.

The remainder of this paper is structured as follows: In Section 1.1, similar works in the area of security-related inference are discussed. In Section 2, the specifics of our input event data, the developed attribute grammar, and the Sequitur algorithm are explained. Grammar inference and data analysis is detailed in Section 3. Our implementation and a full example (Section 4) as well as evaluated applications of the approach (Section 5) conclude the paper.

1.1 Related Work

In light of the large number of operating systems and programming languages currently available, a universal means of abstraction and classification of malicious behavior into a more generic representation is paramount. (Jacob et al., 2009) present a detection system based on attribute grammars, where syntactic rules describe possible combinations of operations constituting certain behavior, while semantic rules control the data flow between operations and assign general meaning to a sequence. The authors' system is intended as formal foundation for developing robust intrusion and malware detection automata. On the modeling side, (Filiol et al., 2007) propose a generalized model for malware detection which considers both sequence-based and behavior-based detection. An evaluation methodology for behavioral engines of existing products is proposed.

In general, the discovery of program behavior is key to understanding benign and malicious programs. (Zhao et al., 2010) present a semi-automatic graph grammar approach to retrieving the hierarchical structure of an application's activity. This is achieved by mining recurring behavioral patterns from execution traces. The inferred graph grammar and a syntactic parse tree visually represent reused structures found.

On the more traditional anomaly detection side, (Creech and Hu, 2014) introduce a host-based detection method that uses discontiguous system call patterns. The authors use a context-free grammar to describe (but not infer) benign and malicious call traces. Several decision engines were tested and compared in the paper, making it a good starting point for the selection of learning algorithms applicable to system call sequences.

In a patent submitted by (Eiland et al., 2012), the authors describe an intrusion masquerade detection system that includes a grammar inference engine based on Minimum Description Length (MDL) compression. The compression algorithm is applied to sets of input data to build user-specific grammars. The use of intrusion masquerade is ultimately based on the determined distance between template and observed algorithmic minimum sufficient statistic.

Visualization is a predominant theme in this field. With GrammarViz, (Senin et al., 2014) introduce a grammar mining and visualization tool based on CFG induction. While GrammarViz does not specifically consider attributes or malicious software scenarios in general, it describes a practical approach to manually analyzing time series data. (Senin et al., 2015) expand on the concept of algorithmic incompressibility for anomaly detection and present practical examples.

2 PRELIMINARIES

In this chapter, we introduce the type of system event data used as the foundation of semantic pattern analysis, the formal definition of this information as part of an attribute grammar, as well as the Sequitur compression algorithm we use to determine interesting patterns.

2.1 Event Data

The proposed system is based on so-called event traces, which are typically defined as descriptions of operating system kernel behavior invoked by applications and, by extension, a legitimate or illegitimate user. More often than not, these events are abstractions of raw system and API calls that yield information about the general behavior of a sample (Wagner et al., 2015). Raw calls may include wrapper functions (e.g. `CreateProcess`) that offer a simple interface to the application programmer, or native system calls (e.g. `NtCreateProcess`) that represent the underlying OS or kernel support functions. In the context of our system, event data is collected directly from the Windows kernel. We employ a driver-based monitoring agent (Marschalek et al., 2015). designed to collect and forward a number of events to a database server. This gives us unimpeded access to events depicting operations related to process and thread control, image loads, file management, registry modification, network socket interaction, and more. For example, a shell event that creates a new text file on a system may be simply denoted as a triple `explorer.exe,file-create,document.txt`. Additional information captured in the background includes various process and thread ID information required to uniquely identify an event within a system session.

2.2 Attribute Grammars

On the formal side, our system uses a context-free grammar extended by attributes, known as attribute grammar (Aho et al., 1986). This decision followed a comprehensive review of several grammars and languages, including graph grammars, state transition graphs based on NLC (Rozenberg, 1997), trace languages, and the aforementioned attribute grammars. The reason for our choice was grounded in the fact that semantically interesting connections between system events are often expressed by their parameters; parameters, that can be aptly modeled by the attributes of a context-free grammar. Performance and the availability of parsing tools also factored into the decision.

In order to enable the conversion of any kind of trace into an applicable ruleset for behavioral classification, it is necessary to formally define relevant (malicious) actions through distinct patterns that can be integrated into the grammatical hierarchy. We do not manually map system activity to concrete events but use inference to automatically determine likely rules. The derived patterns and, by extension, the full grammar, can be defined as follows:

Let $AG = (G,A,R,V)$ be an attribute grammar, where:

- $G = (N,T,P,S)$ is a context-free grammar
  - N... Set of non-terminal symbols (variables)
  - T... Set of terminal symbols (alphabet)
  - P... Production rules
  - S... Start symbol
- A is a finite set of attributes
- R is a finite set of attribution rules (semantic rules)
- V is a finite set of values assigned to an attribute

Every symbol $X \in (N \cup T)$ is assigned a finite set of attributes $A(X)$. The attribute $a \in A(X)$ is denoted $X.a$. Every attribute $a \in A(X)$ also has a set of values $V(X.a)$. Typically, an attribute a of symbol $X \in (N \cup T)$ that is e.g. assigned the value "0" is denominated as $X.a = 0$.

Our methodology uses attributes to store parameters of system events, such as the names of particular files that are being accessed or IP addresses that are being contacted in the course of a network operation. Attributes are also used to retain the connection to the invoking process of an event. In above example, these attributes would be the name of the file being created (namely `document.txt`) and the name of the process triggering the operation (e.g. `explorer.exe`).

Formally, the result is an attribute grammar $AG = (G,A,R,V)$, where $a_1 \in A(X)$ is the attribute $trigger\_name$ and $a_2 \in A(X)$ is defined as $element\_name$. The value $v_i \in V(X.a_1)$ identifies the actual name of the observed process responsible for triggering the individual event $X \in (N \cup T)$. Value $v_j \in V(X.a_2)$ denotes the process or file system element the process interacted with.

Raw system events captured by our monitoring agent are processed by the adapted Sequitur algorithm, which infers a full grammar in accordance to above definitions. This grammar is able to depict an arbitrary number of input traces instead of only single files (see Section 3.2 for details), thereby enabling further parsing and semantic analysis.

2.3 Sequitur Algorithm

Sequitur is a greedy compression algorithm that creates a hierarchical structure from a sequence of discrete symbols by recursively replacing repeated phrases with a grammatical rule (Nevill-Manning and Witten, 1997). The result is a representation of the original sequence, which effectively results in the creation of a context-free grammar. The algorithm creates this representation through two essential properties, which are called *rule utility* and *bigram uniqueness*. Rule utility checks if a rule occurs at least twice in the grammar, while bigram uniqueness observes if a bigram occurs only once. A bigram in this context describes two adjacent symbols or terms. Assuming we have a string `abcdbcabcd`, the first bigram would be `ab`, followed by a second bigram `bc`, and so forth. See Table 1 for a complete example of the process.

Table 1: Operation of Sequitur after (Nevill-Manning and Witten, 1997). Property application is *highlighted*.

Sym	String	Grammar	Remarks
1	a	S → a	
2	ab	S → ab	
3	abc	S → abc	
4	abcd	S → abcd	
5	abcdb	S → abcdb	
6	abcdbc	S → abcdbc	bc appears 2x
		S → aAdA	*bigram uniq.*
		A → bc	
7	abcdbca	S → aAdAa	
		A → bc	
8	abcdbcab	S → aAdAab	
		A → bc	
9	abcdbcabc	S → aAdAabc	bc reappears
		A → bc	
		S → aAdAaA	*bigram uniq.*
		A → bc	aA appears 2x
		S → BdAB	*bigram uniq.*
		A → bc	
		B → aA	
10	abcdbcabcd	S → BdABd	Bd appears 2x
		A → bc	
		B → aA	
		S → CAC	*bigram uniq.*
		A → bc	B used only 1x
		B → aA	
		C → Bd	
		S → CAC	*rule utility*
		A → bc	
		C → aAd	

The system introduced in this paper also evaluates the inferred grammar in addition to applying the Sequitur algorithm and automatically highlights rules describing potentially relevant behavior.

3 INFERENCE AND ANALYSIS PROCESS

3.1 Preprocessing

Before Sequitur can be used on log files, behavioral traces or other, sequential reports describing the activity of malicious programs, the traces need to be reduced to their core components. In this normalization stage, we have the choice to either strip away all attributes or to retain them in an abstracted fashion as part of the set of terminals. As we want to construct a full, semantics-aware attribute grammar, most information is typically kept. We only reduce volatile information such as (user) IDs, memory addresses, and registry paths to a more manageable set of terminals. Names of known system processes and libraries are not modified in any way while unknown binaries and modules (which are possibly randomly named) are represented by extension-aware placeholders (e.g. `1.txt` or `2.exe`).

In order to compare the impact of different levels of detail and granularity, we defined a total of three input formats. An example input and output scenario is discussed in Section 4.2.

Verbose – This trace format uses full, attribute-enabled events as individual words of the corpus. In verbose mode, the input data is transformed into the following format: `triggering-process,operation,element-name`, which translates to $v_i \in V(X.a_1)$,$t_x \in T$,$v_j \in V(X.a_2)$. For example, a specific file creation operation triggered by the known `explorer.exe` process would be preprocessed into the following textual input format: `explorer.exe,file-create,1.txt`.

Reduced – In this preprocessing mode, we omit attribute a_2 to generate a quick view of the high-level activity exhibited by the processes under scrutiny. Here, v_j is not processed, resulting in a reduced format of `triggering-process,operation`, depicted as e.g. `explorer.exe,file-create`.

Granular – The goal in granular mode is to investigate operations not as single word, but as elementary components. Each of the elements processed in verbose mode is treated by Sequitur as one terminal of the bigram. To maintain a level of separation between event triplets, a forth item denoting the start of a new event is prepended before each v_i. This results in the following input (items delimited by semicolon): `<start>;triggering-process;operation;element-name`.

3.2 Rule Extraction

Since Sequitur only takes a single input file per default, the algorithm had to be adapted to regard traces individually while retaining all information of origin. This way, grammar inference can be applied to several files at once without simply concatenating the input into a single, non-attributable compound trace. Specifically, we altered Sequitur to be capable of constructing rules across file boundaries denoted by a unique separator, which is ignored by the inference engine. This ultimately enables comparative analyses of larger, disconnected data sets that do not necessarily share repeating behavior within a single trace, which, under normal circumstances, is required for the inference process to trigger.

The main stages of the rule extraction process are the following:

Lexical Analysis – In this initial step, each unique terminal $t \in T$ is assigned a corresponding symbol, called a token. This numerical representation is used to streamline the process by reducing the processing complexity of string-only comparisons. Each new terminal is additionally stored in a translation (symbol) table for later reference.

Grammatical Inference – After the lexical analysis process the Sequitur algorithm is applied to generate an execution trace grammar consisting of tokenized terminal symbols. The first rule $p \in P$ of each grammar is the start rule, or 'zero rule', which depicts the full grammar of the compressed input data. Every line thereafter contains the following extracted information:

- Rule – The rule consists of a left-side rule name (variable), which is sequentially numbered, as well as right-side variables and terminals. The non-terminals are, again, references to finer-grained rules while the terminals represent the actual system events. In line with the definition of CFGs, there is only one single variable on the left side of a rule.
- Resolved rule – In order to provide a detailed view on individual rules, we recursively resolve each sequence of non-terminals $n \in N$ to their base terminals $t \in T$.

3.3 Rule Evaluation

As part of the evaluation process, the final grammar is parsed to determine how many times a specific derivation occurs in each of the investigated input files. Semantically interesting patterns include specific sequences that e.g. occur exactly once in each input trace, making them potential common denominators for a class of malicious behavior. Parsed information includes:

- File Rule (FR) Count – This number shows how many times a rule occurs in the current derivation of the input file.
- Grammar Rule (GR) Count – The overall count across all supplied input files is specified here. For a single trace, this number is identical to the FR count.
- Prevalence Count – This value specifies the number of input files a particular derivation has been found in. The result is displayed as x/y (x in y), where x is the number of files the pattern is prevalent and y is the overall count of individual input files.
- Match Flag – The extraction of interesting rules is facilitated by determining rules that are identical in occurrence and number across all of the processed input files, indicated by a Boolean flag.
- Rule Length – this value defines the overall number of items seen in the entire derivation (i.e. the resolved rule). Multiples of the input file count y are likely to represent recursively compressed rules.
- Rule Density – this support metric facilitates anomaly detection by calculating the ratio between inferred rules and single terminals that are present in rule zero.

The various counts calculated always include references to the original input files, which help retain each pattern's connection to its semantic source. In Section 4.2, we show an exemplary scenario for a 'verbose' (see Section 3.1) input set.

3.4 Rule Transformation

In order to transform the newly inferred rules into an attributed grammar as defined in Section 2.2, a set mechanism is required. In the initial version of our tool, we map each operation to an attribute-enhanced terminal while rule identifiers are transformed into descriptive variables: Specifically, each rule is dubbed in accordance to its semantic nature. For example, a rule describing a `process-create` operation followed by a `file-delete` operation is transformed into the descriptive variable CREATE-PROC_DELETE-FILE. A rule that describes the loading of two image files is dubbed LOAD2-IMG.

The full naming schema NS is currently defined as follows:

- $NS = (O, E, MO, ME, L)$, where
  - Operation O = {CREA, MOD, START, LOAD, KILL, DEL, CONN}
  - Event type E = {PROC, THR, IMG, FILE, REG, NET}
  - Operation mapping rules MO = {
    CREA $\rightarrow$ create,
    MOD $\rightarrow$ modify | change | edit,
    START $\rightarrow$ start | spawn,
    LOAD $\rightarrow$ load,
    KILL $\rightarrow$ kill | stop | terminate,
    DEL $\rightarrow$ delete,
    CONN $\rightarrow$ connect
    }
  - Event mapping rules ME = {PROC $\rightarrow$ process, THR $\rightarrow$ thread, IMG $\rightarrow$ image, FILE $\rightarrow$ file, REG $\rightarrow$ registry, NET $\rightarrow$ network}
- and labeling rules L, where
  - $(O_1||$ "-" $||E_1||$"-", ..., $O_n||$ "-" $||E_n)$
  - If $O_n == O_{n+1}$ then $O_n||$ "2"

The triggering process and element name are then transformed into the attributes tp and en. Recursive variable descriptors are supported – above naming schema always considers the fully resolved rule. See Section 4.2 for several examples of automatically determined variables.

Future versions of the tool will replace the current mapping with a true semantic descriptor that identifies specific attacker actions or objectives. While a manual assignment of such variables is already possible (Dornhackl et al., 2014), it is not feasible in larger analysis scenarios. The automation of the process is an import research challenge to come.

4 IMPLEMENTATION

4.1 System Overview

Our grammar inference and evaluation tool is based on the Sequitur application developed by Eibe Frank[1]. All core and extended functionality (such as rule resolving, multi-file capabilities, and statistical assessments) has been fully implemented in Java. The data used as basis for the analysis process is collected using a kernel driver agent deployed on 10 actively used and malware-free Windows 7 machines within a corporate environment. An additional virtual Windows instance is used for dynamically analyzing malicious software. The collected events are stored and processed on a dedicated Postgres database server that generates verbose or reduced traces (see Section 3.1) of specific processes or even entire system sessions. These traces are ultimately used as input for the Sequitur tool. See Figure 1 for a process overview.

[1]https://github.com/craignm/sequitur/tree/master/java

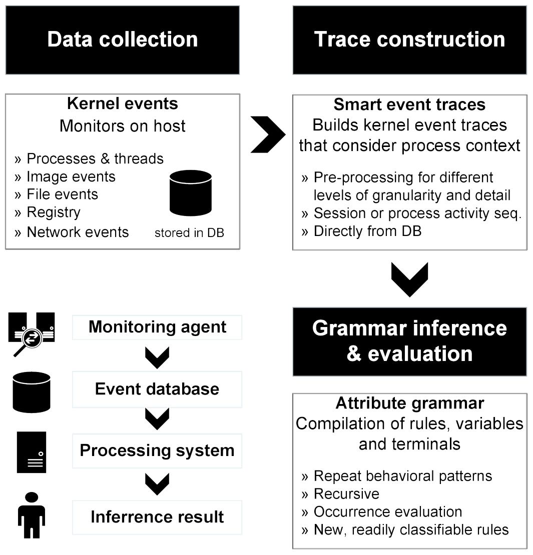


Figure 1: Overview of the implemented system.

In practical scenarios, it might be prudent to use clustering algorithms to pre-classify traces that might share common behavior. While these algorithms typically do not yield insight into event semantics, this intermediate step helps an analyst to select sequences that e.g. belong to a similar class of malware or describe a comparable attack stage. In such a scenario, our inference tool can be used to specifically extract behavioral patterns for a particular use case. In our initial tests, we used Malheur (Rieck et al., 2011) for this very purpose.

4.2 Example

With or without preselection, our grammar inference and evaluation tool will generate variables and production rules for a dynamically growing number of terminals and attributes. Below example demonstrates the use of our tool for two simplified 'verbose' input files generated from aforementioned kernel event traces.

```
Input file 1: Verbose mode. Delimiter: newline.
    explorer.exe,file-create,1.exe
    explorer.exe,process-start,1.exe
    1.exe,image-load,kernel32.dll
    1.exe,image-load,advapi32.dll
    1.exe,registry-modify,hklm/software/microsoft
    1.exe,registry-modify,hklm/software/microsoft
    1.exe,process-create,cmd.exe
    cmd.exe,process-create,net.exe
    1.exe,registry-create,machine/system
    1.exe,registry-modify,hklm/software/microsoft
    1.exe,registry-modify,hklm/software/microsoft
    cmd.exe,process-kill,net.exe
    1.exe,thread-terminate,thread
    explorer.exe,file-delete,1.exe
```

The second input file has been determined by Malheur to be similar, however the commonalities are yet unclear. This is where our pattern evaluation extension comes in.

```
Input file 2: Verbose mode. Delimiter: newline.
    explorer.exe,file-create,1.exe
    explorer.exe,process-start,1.exe
    1.exe,thread-create,thread
    1.exe,image-load,kernel32.dll
    1.exe,image-load,advapi32.dll
    1.exe,image-load,ws2_32.dll
    1.exe,registry-modify,hklm/software/microsoft
    1.exe,registry-modify,hklm/software/microsoft
    1.exe,process-create,cmd.exe
    cmd.exe,process-create,net.exe
    cmd.exe,process-kill,net.exe
    1.exe,thread-terminate,thread
    explorer.exe,file-delete,1.exe
```

Sequitur now infers the following rules and evaluates the frequency, prevalence and similarity of the input (zero rule, resolved rules and the output for input file 2 have been removed for legibility):

```
Output: Rule; File rule count; Grammar rule count;
      Prevalence count; Match flag; Rule length
1 -> explorer.exe,file-create,1.exe explorer.exe,
     process-start,1.exe; 1; 2; 2/2; true; 2
2 -> 1.exe,load-image,kernel32.dll 1.exe,load-
     image,advapi32.dll; 1; 2; 2/2; true; 2
3 -> 4 1.exe,process-create,cmd.exe cmd.exe,
     process-create,net.exe; 1; 2; 2/2; true; 4
4 -> 1.exe,registry-modify,hklm/software/microsoft
      1.exe,registry-modify,hklm/software/microsoft
     ; 2; 3; 2/2; false; 2
5 -> cmd.exe,process-kill,net.exe 1.exe,thread-
     terminate,thread explorer.exe,delete-file,1.
     exe; 1; 2; 2/2; true; 3
```

In our example, the tool has successfully extracted rules that describe behavior observed in both input files. The output is transformed into an attribute grammar as described in Section 2.2. Since semantics is a major factor of rule construction, we assign variables based on the nature of the inferred event. Specifically, above example can be formalized into a grammar as follows:

Let $AG_1 = (G_1, A, R, V)$ be an inferred CFG extended by attributes, where:

- $G_1 = (N, T, P, S)$, and where:
  - N = {CREA-FILE_START-PROC; LOAD2-IMG; MOD-REGCREA2-PROC; MOD2-REG; KILL-PROC_KILL-THR_DEL-FILE}
  - T = {
    file-create.tp, en = explorer.exe, 1.exe;
    process-start.tp, en = explorer.exe, 1.exe;
    image-load.tp, en = 1.exe, kernel32.dll;
    image-load.tp, en = 1.exe, advapi32.dll;
    process-create.tp, en = 1.exe, cmd.exe;
    process-create.tp, en = 1.exe, net.exe;
    registry-modify.tp, en = 1.exe, hklm/sw/ms;
    process-kill.tp, en = cmd.exe, net.exe;
    thread-terminate.tp, en = 1.exe, thread;
    file-delete.tp, en = explorer.exe, 1.exe
    }
  - P = {
    ZERO-RULE $\rightarrow$ CREA-FILE_START-PROC LOAD2-IMG MOD-REG_CREA2-PROC registry-create.tp, en = 1.exe, hklm/sys MOD2-REG KILL-PROC_KILL-THR_DEL-FILE;
    CREA-FILE_START-PROC $\rightarrow$ file-create.tp, en = explorer.exe, 1.exe process-start.tp, en = explorer.exe, 1.exe;
    LOAD2-IMG $\rightarrow$ image-load.tp, en = 1.exe, kernel32.dll image-load.tp, en = 1.exe, advapi32.dll;
    MOD-REG_CREA2-PROC $\rightarrow$ MOD2-REG process-create.tp, en = 1.exe, cmd.exe process-create.tp, en = cmd.exe, net.exe;
    MOD2-REG $\rightarrow$ registry-modify.tp, en = 1.exe, hklm/sw/ms registry-modify.tp, en = 1.exe, hklm/sw/ms;
    KILL-PROC_KILL-THR_DEL-FILE $\rightarrow$ process-kill.tp, en = cmd.exe, net.exe thread-terminate.tp, en = 1.exe, thread file-delete.tp, en = explorer.exe, 1.exe
    }
  - S = {ZERO-RULE}
- A = {tp; en}
- R is described as part of the preprocessing stage and defines which portion of the data translates into triggering process tp (v_i), operation (t_x), and element en (v_j).
- V = {explorer.exe; 1.exe; kernel32.dll; advapi32.dll; cmd.exe; net.exe; machine/software/microsoft; thread}

Above attribute grammar has been generated automatically and can now be used as the foundation for further (attribute-based) parsing efforts. The inferred variables, if stored, can be used as new behavioral templates for comparable input data sets. The next Section discusses practical applications of this approach.

5 PRACTICAL APPLICATIONS AND EVALUATION

The introduced system has a wide variety of applications. Ranging from preliminary knowledge extraction in malware analysis scenarios to understanding more complex attacks, the adapted inference methodology is versatile in both terms of input data as well as practical benefit. Below, we introduce and evaluate some of its applications and discuss future and ongoing work.

5.1 Preparatory Data Reduction

5.1.1 Concept

In many malware and APT attack stage analysis scenarios, analysts are forced to deal with huge amounts of data. Be it kernel events, raw system calls or even assembler-level information, abstraction and reduction of input data is essential to decrease the complexity of many an analysis task. Our solution provides the means through its easily adaptable prepocessing mechanism (see Section 3.1) and the grammar inference system itself. By using the Sequitur approach, it is possible to reduce the input corpus to only relevant bigrams, instead of working with the full, unfiltered set of event or code snippet unigrams. The grammar transformation mechanism (see Section 3.4) also enables us to work with an automatically generated placeholder variable $n \in N$ instead of several compound terminals.

5.1.2 Evaluation

Current efforts of the team include the pre-abstraction of behavior graph data subsequently used for edit distance calculations (Luh et al., 2017). Minimizing the amount of data to process drastically reduces the computation requirements of expensive graph transformation operations. Specifically, we evaluated several days' worth of benign system events monitored by our kernel driver, collecting 10k, 100k, and 200k sequential events across 525 uniquely named processes running on 10 clean Windows 7 machines. Under normal circumstances, this data would have to be assessed in its entirety, as it is used for creating baseline templates utilized in behavior deviation analysis. Thanks to our Sequitur-enabled data reduction, we can focus on event sequences (rules) that are representative for specific processes, significantly speeding up all subsequent, potentially exponential complexity graph operations.

In our largest exemplary dataset of 200k Windows kernel events, we reduced the number of events to 13,275 (-93.4%), which effectively cut the processing time for both graph template generation and graph transformation calculations by >97%, saving a total of 162 minutes by removing a large number of excess events not required for the analysis. This bumped the graph-based anomaly detection process significantly closer to real-time capabilities for smaller corpora. Performance evaluation showed a maximum memory utilization of around 3.6 GiB, with a total processing time of 22.8 minutes on a 64-bit Intel Core i7-4* workstation equipped with 16 GiB of RAM.

The overall process has been determined as scaling at quadratic time $O(n^2)$, putting it in line with many basic sorting algorithms. Sequitur without any evaluation and rule dissemination is known to operate in linear time $O(n)$ (Nevill-Manning and Witten, 1997), providing room for future optimization.

5.2 Anomaly Detection

5.2.1 Concept

In our above preprocessing example, we use grammar inference to determine interesting repeating patterns that are representative of the corpus under investigation. However, the reverse is also a viable scenario: By focusing attention on patterns that do not excessively reoccur, our approach can be used to identify anomalies in a sequence or set of sequences. Parts of the trace that are not replaced by variables during rule construction (i.e. the remaining terminals in between) represent unique events that, in such a scenario, are of particular interest. Rule density (see 3.3) is also important in scenarios where stable behavior is expected: the higher the share of terminals, the higher the overall entropy, and, by extension, the likelihood of anomalous behavior. All anomaly detection efforts can be aided by visualization tools such as GrammarViz (Senin et al., 2014) and our own ongoing research introduced in Section 5.3 below.

5.2.2 Evaluation

The Sequitur tool is not limited to system events but can be used with a wide range of sequential input data formats. We specifically evaluated an APT anomaly detection scenario on a set of temperature, speed, and photoelectric sensor data generated by a Siemens Simatic industrial control system (ICS) within a testbed environment. We assessed 12 full production runs in total, whereas one of the runs was maliciously altered by illegally interfering with the

Table 2: Extracted and evaluated rules for ICS sensor data traces with low rule density ($\leq 40\%$) and prevalence count (≤ 2). Each rule describes an anomaly not typically seen in other input data. FR...file rule, GR...grammar rule.

File	Rule	FR count	GR count	Prevalence	Length
Ben-2	3→139 139	2	2	1/12	16
Ben-2	4→140 140	3	3	1/12	4
Ben-2	11→1-0-1-0-1-1-1-1-1-0-1-0-32 1-0-1-0-1-1-1-1-1-0-1-0-32	2	4	2/12	2
Ben-2	12→1-0-1-0-1-1-1-1-1-0-1-0-40 1-0-1-0-1-1-1-1-1-0-1-0-40	2	2	1/12	2
Ben-2	23→1-0-1-0-1-1-1-1-0-0-1-1-56 1-0-1-0-1-1-1-1-0-0-1-1-56	2	2	1/12	2
Ben-2	52→53 53	3	5	2/12	4
Ben-2	69→1-0-1-1-1-1-1-1-0-0-2-5-59 1-0-1-1-1-1-1-1-0-0-2-5-59	2	2	1/12	2
Ben-2	75→0-0-1-1-1-1-1-1-1-0-2-5-52 0-0-1-1-1-1-1-1-1-0-2-5-52	2	4	2/12	2
Ben-2	76→152 152	3	6	2/12	4
Ben-2	98→1-0-1-1-1-0-1-1-0-0-2-8-60 1-0-1-1-1-0-1-1-0-0-2-8-60	2	2	1/12	2
Ben-2	102→0-0-1-1-1-1-0-1-1-0-2-8-54 0-0-1-1-1-1-0-1-1-0-2-8-54	2	2	1/12	2
Ben-2	139→4 4	2	2	1/12	8
Ben-2	140→0-0-0-0-0-0-1-1-1-0-0-0-20 0-0-0-0-0-0-1-1-1-0-0-0-20	2	2	1/12	2
Mal-1	52→1-0-1-0-1-1-1-1-0-0-1-4-56 1-0-1-0-1-1-1-1-0-0-1-4-56	2	4	2/12	2
Mal-1	57→1-0-1-1-1-0-1-1-0-0-2-4-60 1-0-1-1-1-0-1-1-0-0-2-4-60	2	2	1/12	2
Mal-1	72→1-0-1-0-1-1-1-1-0-0-1-6-52 1-0-1-0-1-1-1-1-0-0-1-6-52	2	2	1/12	2
Mal-1	81→82 82	2	2	1/12	64
Mal-1	82→83 83	3	3	1/12	32
Mal-1	83→157 157	3	3	1/12	16
Mal-1	84→85 85	3	3	1/12	4
Mal-1	85→1-0-1-0-1-0-1-1-1-0-2-7-60 1-0-1-0-1-0-1-1-1-0-2-7-60	3	3	1/12	2
Mal-1	86→1-0-1-1-1-1-1-1-1-0-2-7-59 1-0-1-1-1-1-1-1-1-0-2-7-59	2	2	1/12	2
Mal-1	157→84 84	2	2	1/12	8

rotation. This caused a number of atypical sensor readings that are nigh impossible to spot manually.

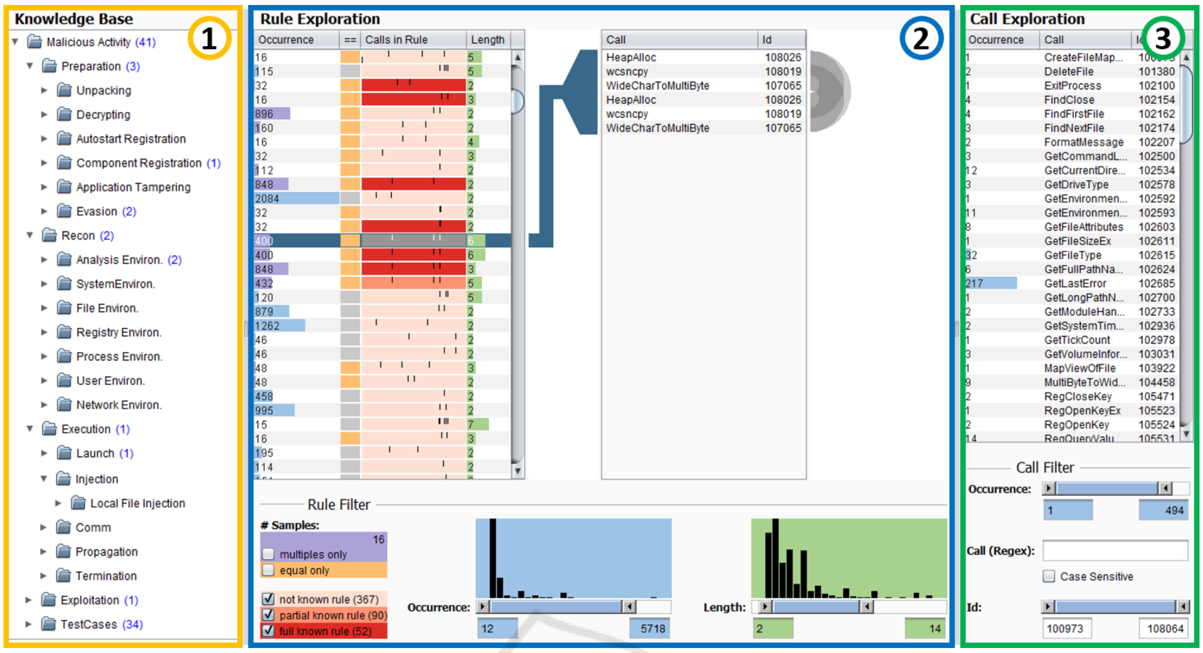
The full evaluated grammar for a total of 34,000 observed events was constructed almost instantaneously. Sequitur inferred a total of 2,155 rules (sans zero rules), resulting in a 93.7% data compression rate. In stage one, anomaly detection was conducted by assessing rules with a low rule density value. By that metric alone, it was already possible to single out the anomalous trace. With a terminal-to-rule ratio (TRR) of 62.8% (rule density of 37.2%), the malicious sample contained less uniform behavior patterns than the remainder of 11 traces with a mean ratio of 57.3%. Only two benign traces came close to that number, exceeding a TRR of 60%. The comparatively small margin is due to the fact that, in this scenario, anomalous data did not cause sensor spikes but rather triggered a slow, continuous change in behavior.

Further analysis of the possibly deviating behavior was (and is typically) required to solidify the initial verdict. To this end, we used our evaluation system (Section 3.3) to filter rules that are present in only a minority of files and that have a prevalence count of ≤ 2 out of 12. Armed with the preselection based on rule density, we particularly focused on the 3 traces with a TRR of $>60\%$. In benign trace 1 (not pictured), only one rule was determined as unique, effectively disqualifying the candidate. Benign trace 2 ("Ben-2" in Table 2) contained 13 rules that were not seen in the majority of the remaining corpus, whereas the malicious trace ("Mal-1") contained 10 nearly unique rules. A direct comparison of the two remaining anomalous candidates highlighted one particularly interesting, recursively compressed block per trace, which resolved into 16 (benign) and 64 (malicious) terminals, respectively. Patterns with a higher average length are particularly interesting as they identify larger, uninterrupted sequences unique for the dataset under scrutiny.

A domain expert is now able to investigate further and determine the individual events t that describe illegal sensor readings, thereby indicating an attack on the production line.

See Table 2 for a direct comparison of the two deviating behavior traces. Rule 3 of "Ben-2", which resolves into 8 terminal pairs as inferred by rule 140 (a rare, but valid sensor state), contributes most to the trace's analysis verdict. For "Mal-1", the same applies to rule 81 (32 iterations of rule 85), effectively identifying the anomalous sensor state.

The extracted, semantically relevant rules can now be formalized and stored for future parsing runs. While this can be done textually, a visuals-assisted solution promises even better results. In the next section, we introduce a practical approach to discovering new knowledge and assisting with anomaly detection efforts through a dedicated visual event analysis tool based on our Sequitur output.



Knowledge Base

- Malicious Activity (41)
  - Preparation (3)
    - Unpacking
    - Decrypting
    - Autostart Registration
    - Component Registration (1)
    - Application Tampering
    - Evasion (2)
  - Recon (2)
    - Analysis Environ. (2)
    - SystemEnviron.
    - File Environ.
    - Registry Environ.
    - Process Environ.
    - User Environ.
    - Network Environ.
  - Execution (1)
    - Launch (1)
    - Injection
      - Local File Injection
    - Comm
    - Propagation
    - Termination
- Exploitation (1)
- TestCases (34)

Rule Exploration

Occurrence	==	Calls in Rule	Length
16	🟧		5
115	⬜		5
32	🟧	🟥	2
16	🟧	🟥	3
896	🟧		2
160	🟧		2
16	🟧		4
32	🟧		3
112	🟧		2
848	🟧	🟥	2
2084	⬜		2
32	🟧		2
32	🟧	🟥	2
400	🟧		6
400	🟧	🟥	6
848	🟧	🟥	3
432	🟧	🟥	5
120	⬜		5
879	⬜		2
1262	⬜		2
46	⬜		2
46	⬜		2
48	🟧		3
48	🟧		2
458	⬜		2
995	⬜		2
15	⬜		7
16	🟧		3
195	⬜		2
114	⬜		2

Call	Id
HeapAlloc	108026
wcsncpy	108019
WideCharToMultiByte	107065
HeapAlloc	108026
wcsncpy	108019
WideCharToMultiByte	107065

Rule Filter

# Samples: 16

- [ ] multiples only
- [ ] equal only
- [✓] not known rule (367)
- [✓] partial known rule (90)
- [✓] full known rule (52)

Occurrence: 12 – 5718

Length: 2 – 14

Call Exploration

Occurrence	Call	Id
1	CreateFileMap...	[illegible]
2	DeleteFile	101380
1	ExitProcess	102100
4	FindClose	102154
4	FindFirstFile	102162
3	FindNextFile	102174
2	FormatMessage	102207
3	GetCommandL...	102500
12	GetCurrentDire...	102534
3	GetDriveType	102578
1	GetEnvironmen...	102592
11	GetEnvironmen...	102593
8	GetFileAttributes	102603
1	GetFileSizeEx	102611
32	GetFileType	102615
6	GetFullPathNa...	102624
217	GetLastError	102685
1	GetLongPathN...	102700
2	GetModuleHan...	102733
2	GetSystemTim...	102936
1	GetTickCount	102978
3	GetVolumeInfor...	103031
1	MapViewOfFile	103922
9	MultiByteToWid...	104458
2	RegCloseKey	105471
1	RegOpenKeyEx	105523
2	RegOpenKey	105524
14	RegQueryValu...	105531

Call Filter

Occurrence: 1 – 494

Call (Regex):

- [ ] Case Sensitive

Id: 100973 – 108064

Figure 2: Illustration of the Knowledge-assisted Malware Analysis System (KAMAS) designed to support malware analysts in their work. This system contains a (1) 'Knowledge Base' for automated analysis and knowledge sharing between the analysts, a (2) 'Rule Exploration' area, and a (3) 'Call Exploration' area used to investigate individual events. Various filters at the bottom help to remove redundant data.

5.3 Visualization & Knowledge Discovery

One of the major uses of our tool is undoubtedly the extraction of new domain knowledge. Inferred patterns can be compiled into a permanent grammar used to detect similar behavior in unknown traces. This process can be supported by interactive visualizations to drastically improve usability. This area of research is typically referred to as visual analytics (VA).

5.3.1 Visual Analytics

Specifically, VA is "the science of analytical reasoning facilitated by interactive visual interfaces" (Thomas and Cook, 2005).

A major tenet of VA is that analytical reasoning is not a routine activity that can be automated completely (Wegner, 1997). Instead it depends heavily on the analyst's initiative and domain experience, which is exercised through interactive visual interfaces. Such interfaces, especially information visualizations, are high bandwidth gateways for the depiction of structures, patterns, and connections hidden in the data. Furthermore, visual analytics often involves automated analysis methods that perform various computations on potentially large volumes of data.

When analysts solve real world problems they typically have large volumes of complex and heterogeneous data at their disposal, as is evidenced by above application scenarios (see Sections 5.1 and 5.2). Externalization and storing of implicit knowledge will make it available as *explicit domain knowledge*, which is defined as knowledge that "represents the results of a computer-simulated cognitive process, such as perception, learning, association, and reasoning (...)" (Chen et al., 2009).

Through visualization, explicit knowledge can be used to graphically summarize and abstract a dataset. Put simply, it enables quicker and more precise analyses of complex input data such as the set of traces used in the above ICS example.

Using VA for security applications is a widely accepted practice. In (Wagner et al., 2014), the authors surveyed tools for behavior-based malware analysis in addition to visual representations best suited to various domain challenges. Through a data–users–tasks analysis (Miksch and Aigner, 2014), they ascertained that the parse tree of a cluster grammar (such as the one generated by the Sequitur algorithm), can be abstracted to a directed acyclic graph, where each node represents part of a sequence. This is where our VA prototype comes in.

5.3.2 Prototype Implementation

Based on above findings, we created an early prototype, KAMAS, that can be used to visualize and further assess the data generated by the Sequitur tool. Figure 2 depicts a screenshot of the KAMAS interface.

The ”knowledge base” (1) contains the explicit knowledge used for automated analysis. Newly extracted patterns can be stored in a database for later use, whereas existing ones serve as real-time filter that automatically highlights known patterns. In the ”rule exploration” area (2), the analyst can see the different rules (left side only) generated by Sequitur, including all information pertaining to its file and grammar count as well as its length. Selecting a specific rule unfolds the fully resolved rule for further study. Additionally, an arc diagram will be shown to highlight events that constitute a known sequence. The ”call exploration” area (3) lists all events contained in the loaded file(s). To cope with the potentially huge amount of data, the analyst has the ability to use different, regular-expression-enabled filters to locate data of interest. New rules discovered through visual analysis can be added into the knowledge base via a simple drag and drop action.

A full evaluation of KAMAS (including a comprehensive usability study) will be disseminated at a later date.

6 FUTURE WORK

One of the main areas of future improvement is undoubtedly the automated semantic interpretation of inferred variables, which, in the tool’s current iteration, are assigned based on the operations (terminals) that constitute the respective rule. In the future, this representation will be changed to include actual attacker goals and actions. Ultimately, we plan to link the process to the team’s previous work, which focuses on the development of a targeted attack ontology (Luh et al., 2016b). Automatically extracted events will be mapped to said ontology, thereby providing a complete view on likely attack scenarios induced by the events in question.

In terms of validation, future work will encompass a detailed proof of soundness for the attribute grammar specification used in the paper. Furthermore, we will test our behavioral engine against evaluation systems such as the one introduced by (Filiol et al., 2007). Specific applications such as the anomaly detection functionality discussed in Section 5.2 will also be evaluated in greater detail.

On the knowledge discovery side, it is planned to finalize development of the KAMAS visualization tool mentioned in Section 5. Specific functionality enabling further statistical assessment will be included to facilitate (malware) forensics, automated sample classification, and various intrusion detection scenarios coupled with a database of explicit domain knowledge.

In general, the cross-integration of visual analytics and knowledge discovery methods will be an integral part of our future research into the practical applications of the Sequitur approach.

7 CONCLUSION

In this paper, we presented a grammar inference system based on an adapted version of Nevill-Mannings’ Sequitur algorithm. Thanks to its versatile nature, the tool offers various benefits for the information security community, ranging from knowledge discovery in sequential system activity or malware execution traces, to applied anomaly detection and grammar-enabled behavior interpretation.

We have successfully tested the induction and analysis system with several classes of input data. When used to streamline input traces for other, computationally expensive processes, we have achieved a significant reduction in complexity by extracting representative variables that describe relevant patterns. Anomaly detection based on the rule density metric showed promising results in identifying deviating traces and their behavioral sequences in close to real time. With KAMAS, we additionally introduced a visual analytics platform that uses the generated data to assist analysts in extracting relevant rules.

All in all, the grammar inference tool can be used to quickly and accurately discover and highlight recurring patterns in sequential sets of arbitrary host and network activity, thereby aiding in bridging the semantic gap between captured event traces and attacker behavior.

ACKNOWLEDGEMENTS

The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged.

Furthermore, this work was supported by the Austrian Science Fund (FWF) via the KAVA-Time project (no. P25489).

REFERENCES

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers, Principles, Techniques*. Addison Wesley.

Bilge, L. and Dumitras, T. (2012). Before we knew it: an empirical study of zero-day attacks in the real world. In *Proc. of the 2012 ACM conference on Computer and communications security*, pages 833–844. ACM.

Chen, M., Ebert, D., Hagen, H., Laramee, R., Van Liere, R., Ma, K.-L., Ribarsky, W., Scheuermann, G., and Silver, D. (2009). Data, information, and knowledge in visualization. *Computer Graphics & Applications*, 29(1):12–19.

Creech, G. and Hu, J. (2014). A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns. *Computers, IEEE Transactions on*, 63(4):807–819.

Dornhackl, H., Kadletz, K., Luh, R., and Tavolato, P. (2014). Defining malicious behavior. In *Ninth International Conference on Availability Reliability and Security (ARES)*, pages 273–278. IEEE.

Eiland, E., Evans, S., Markham, T., and Impson, J. (2012). Mdl compress system and method for signature inference and masquerade intrusion detection. US Patent 8,327,443.

Filiol, E., Jacob, G., and Le Liard, M. (2007). Evaluation methodology and theoretical model for antiviral behavioural detection strategies. *Journal in Computer Virology*, 3(1):23–37.

Jacob, G., Debar, H., and Filiol, E. (2009). Malware behavioral detection by attribute-automata using abstraction from platform and language. In *International Workshop on Recent Advances in Intrusion Detection*, pages 81–100. Springer.

Luh, R., Marschalek, S., Kaiser, M., Janicke, H., and Schrittwieser, S. (2016a). Semantics-aware detection of targeted attacks: a survey. *Journal of Computer Virology and Hacking Techniques*, pages 1–39.

Luh, R., Schrittwieser, S., and Marschalek, S. (2016b). Taon: An ontology-based approach to mitigating targeted attacks. In *Proc. of the 18th Int. Conference on Information Integration and Web-based Applications & Services*. ACM.

Luh, R., Schrittwieser, S., Marschalek, S., and Janicke, H. (2017). Design of an Anomaly-based Threat Detection & Explication System In *Proc. of the 3rd Int. Conference on Information Systems Security & Privacy*. SCITEPRESS.

Marschalek, S., Luh, R., Kaiser, M., and Schrittwieser, S. (2015). Classifying malicious system behavior using event propagation trees. In *Proc. of the 17th Int. Conference on Information Integration and Web-based Applications & Services*. Association for Computational Linguistics.

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

Munsey, C. (2013). Economic Espionage: Competing For Trade By Stealing Industrial Secrets. Accessed 2015-09-15.

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

Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). *Automatic analysis of malware behavior using machine learning*. Journal of Computer Security.

Rozenberg, G. (1997). *Handbook of graph grammars and computing by graph transformation*, volume 1. World Scientific.

Senin, P., Lin, J., Wang, X., Oates, T., Gandhi, S., Boedihardjo, A. P., Chen, C., and Frankenstein, S. (2015). Time series anomaly discovery with grammar-based compression. In *EDBT*, pages 481–492.

Senin, P., Lin, J., Wang, X., Oates, T., Gandhi, S., Boedihardjo, A. P., Chen, C., Frankenstein, S., and Lerner, M. (2014). Grammarviz 2.0: a tool for grammar-based pattern discovery in time series. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 468–472. Springer.

Sood, A. K. and Enbody, R. J. (2013). Targeted cyberattacks: a superset of advanced persistent threats. *IEEE security & privacy*, (1):54–61.

Symantec (2015). Symantec Internet Security Threat Report Volume 20. *Whitepaper*.

Thomas, J. J. and Cook, K. A., editors (2005). *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE.

Wagner, M., Aigner, W., Rind, A., Dornhackl, H., Kadletz, K., Luh, R., and Tavolato, P. (2014). Problem characterization and abstraction for visual analytics in behavior-based malware pattern analysis. In Whitley, K., Engle, S., Harrison, L., Fischer, F., and Prigent, N., editors, *Proc. 11th Workshop on Visualization for Cyber Security, VizSec*, pages 9–16. ACM.

Wagner, M., Fischer, F., Luh, R., Haberson, A., Rind, A., Keim, D., Aigner, W., Borgo, R., Ganovelli, F., and Viola, I. (2015). A Survey of Visualization Systems for Malware Analysis. In *Eurographics Conference on Visualization*, pages 105–125. EuroGraphics.

Wegner, P. (1997). Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91.

Zhao, C., Kong, J., and Zhang, K. (2010). Program behavior discovery and verification: A graph grammar approach. *IEEE Transactions on software Engineering*, 36(3):431–448.