

On the Characterization of Attacks Families

Karen-Azurim García-Gamboa and Raúl Monroy

Computer Science Department
 Tecnológico de Monterrey, Campus Estado de México
 Carretera al lago de Guadalupe, Km 3.5, Atizapán, 52926, Mexico
 {kazurim,raulm}@itesm.mx

At a host level, an attack takes the form of a sequence of system calls. System calls, we call *low-level events*, are registered in audit logs. Our research aims at grouping host-level attacks into families having similar structure. In order to break into a system, an attacker follows a strategy, whose abstract structure consists of a sequence of high-level events. A *high-level event* is a user action, e.g. a file compilation.

We propose an approach which groups attacks into families and characterises the members of each family as a sequence of high-level events. Attacks are grouped in terms of common patterns of low-level events that share one another. To identify these patterns, we use pair-wise sequence alignment [3]. Attacks are characterised in terms of frequently occurring sub-sequences of low-level events; each these sub-sequences is used to form a high-level event. To identify repetitive sub-sequences, we use Sequitur [2].

Sequence alignment is the procedure of (pair-wise alignment) comparing two sequences by searching for a series of individual characters or character patterns that are in the same order in the sequences [3]. Let $A = a_1, a_2, \dots, a_m$ and $B = b_1, b_2, \dots, b_n$ be two sequences, with a_i ($i \in \{1, m\}$) and b_j ($j \in \{1, n\}$) over some alphabet Σ of size s , made out of low-level events [1]. Then, the sequence alignment algorithm finds the largest common symbol string, possibly with gaps inserted, that appears in both A and B . The length of this string we use as a measure of sequence similarity. Figure (1) shows an example sequence alignment, where a dash indicates a gap and a vertical line a match.

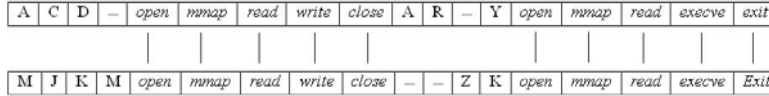


Fig. 1. Sequence alignment

Let s and s' be two low-level event sequences and let $align(s, s')$ denote sequence alignment application. Then, to measure the similarity between s and s' , written $s \sim s'$, we use:

$$s \sim s' \frac{(|align(s, s')| \times 100)}{\max(|s|, |s'|)} \quad (1)$$

where $|x|$ denotes the length of x , a sequence, and $\max(x, y)$ returns x if $x > y$ and y otherwise. We cluster s and s' into the same family whenever $s \sim s' > Th = 0.75$; Th was estimated by hand and was found to correctly distinguish a test set of attack families. Given a set of m sequences, the number of alignments to be done is given by C_2^m .

Sequitur is an algorithm that infers a hierarchical structure from a sequence of symbols by replacing repeated sub-sequences with a context-free grammatical rule that generates the sub-sequence, and continuing this process recursively, [2]. Let us now specify our context-free grammar, following [4]. Let Σ be the set of terminals containing unix commands; $N = \{n_1, n_2, \dots, n_k\}$ the set of nonterminals that are to be used as the left-hand side (lhs) of a production rule; and let S be the start symbol, $S \notin N \cup \Sigma$, called the *main production*. Then, a production rule is of the form $n_k \rightarrow x_1, \dots, x_n$, where x_1, \dots, x_n is the right-hand side (rhs) of the production rule, each $x_i \in N \cup \Sigma$ and where $n_k \in N \cup S$. Let $C \in \Sigma^*$ be a sequence of elements. On input C , Sequitur begins extracting elements from C and assigning it to the main production $S \rightarrow c_1, c_2$, in the order of appearance. Sequitur will process sequentially all the elements of C extracting adding them to S . Figure (2) shows the resultant grammar obtained from the Sequitur algorithm applied to the sequence example of figure (1).

$S \rightarrow DD$
$A \rightarrow \text{open mmap}$
$B \rightarrow A \text{ read}$
$C \rightarrow B \text{ write}$
$D \rightarrow C \text{ close}$

Fig. 2. Sequitur Algorithm results

The resultant grammar shows that the most repeated sequence is *open mmap read write close*. This sequence corresponds to a high-level event of manipulating a file. Based on experimental results of our approach, we conclude that we can construct attacks families based on their attack strategy.

References

1. Branck J. Szymanski B. Coull, S. and E. Breimer. Intrusion detection: A bioinformatics approach. *19th Annual Computer Security Applications Conference (ACSAC '03)*, page 24, 2003.
2. M. Latendresse. Masquerade detection via customized. *Springer LNCS, Lecture Notes in Computer Science*, 3548:141–159, 2005.
3. David W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, New York, 2001.
4. Román Posadas. Analysis of masquerade detectors performance under synthesized sessions. Master's thesis, Tecnológico de Monterrey, Campus Monterrey, 2006.