# Discovering Emergent Norms in Security Logs

Olgierd Pieczul*†, Simon N. Foley†
*Ireland Lab, IBM Software Group, Dublin, Ireland
†Computer Science Department, University College Cork, Ireland

***Abstract*—A model is presented that characterizes security logs as a collection of norms that reflect patterns of emergent behavior. An analysis technique for detecting behavioral norms based on these logs is described and evaluated. The application of behavioral norms is considered, including its use in system security evaluation and anomaly detection.**

## I. Introduction

Log data is a useful source of information when monitoring security violations and identifying incidents [14], [17], [15]. A security log records a sequence of relevant system and/or application events in chronological order. Conventional security (log) analytics systems tend to take an *event-centric* view of these logs whereby potential threats and security anomalies are defined, and searched for, in terms of relatively straightforward event interactions. For example, rudimentary tools search for an occurrence of a particular event, an event with a specific property, or some collection of events over a given period of time [6]. More advanced tools search for simple patterns or correlations of events that may represent potential threats [19]. Log analysis, in these cases, characterizes anomalies in terms of prescribed events and their properties.

This event-centric view of log analysis can require the specification of a large number of ad-hoc rules that may be difficult to comprehend and manage. Rather than thinking of anomalies in terms of low-level log-events, we are interested their characterization in terms of more abstract system artifacts. For example, characterizing a separation of duty anomaly in terms of transactions of log events. While transactions are effectively sequences of events, we argue that using behavioral abstractions such as transactions, workflows, and so forth, can facilitate comprehension of the information in a security log. This leads to a more *behavior-centric* view of security logs whereby anomaly analysis is characterized in terms of behavioral abstractions. For example, checking log data for conformance with business process workflows [23] or workflow security requirements [2], or matching a security log against an automaton [20], temporal logic specification [7] or n-gram profile [11] that represents acceptable behavior of the system.

A challenge to taking this behavior-centric approach is finding the right level of abstraction of log events and developing specifications that adequately represent anomaly-free behavior. Existing approaches require an a priori user prescription on event abstraction, for instance, ignoring all event attributes other than operation name [11]. Given an event abstraction, a specification of acceptable behavior may be user-defined [7], [9], [18]. However, the scale of the system may be such that it is difficult for a user to formulate a complete and efficacious specification of normal behavior. As a consequence, security specifications focus on those behaviors perceived to be critical, with an assumption that the other behaviors, known or unknown, are not significant. Often, it is these side behaviors that can lead to a security compromise of the system.

Behavior mining techniques can be used on system logs to automate the discovery of models that represent these potentially known and unknown normal behaviors [8], [11], [23]. However, the level of abstraction in these models is fixed and must be a priori specified, with the result that some side behaviors may not be adequately captured in the discovered model of acceptable behavior. Deciding one particular level of abstraction of the log data may conceal behavioral patterns of interest that emerge at other levels of abstraction. For example, considering transaction-id, in addition to the operation name, in events may help to reveal other repeating patterns of behaviors, such as transactions, in the log. Furthermore, the complexity of a large-scale system may be such that a full understanding of the event attributes and behaviors exhibited in a system log may not be known.

This paper proposes an approach to analyzing system logs for the purposes of discovering models that represent normal behavior. The primary contribution of the proposed approach is that it can be used to identify suitable behavior abstractions in the system log. These *behavioral norms* represent repeating patterns of behavior at different levels of abstraction that occur in the system logs. In following prescribed behavior/security controls, some norms may already be known, for example, a secure workflow of transactions implemented by the system. Other norms, for example, representing side-behaviors, may be unknown, in that that they have not been explicitly formulated but emerge as a consequence of normal system operation. Two experiments, described in this paper, were carried out which demonstrate the emergence of such norms.

The paper is structured as follows. Section 2 describes the proposed model of behavioral norms. Section 3 outlines how system logs can be searched for norms. Experiments that were used to evaluate the approach using simulated and real log sources are described in Section 4. The contribution of the work and related research is discussed in Section 5 and Section 6 concludes the paper.

## II. A Model of Behavioral Norms

### A. *Events and traces*

The following is specified using Z notation [21] and is syntax and type checked using Fuzz checker. An *event* represents an observation of some interaction with a system. This paper focusses on events drawn from system logs, e.g., `[Feb 24 08:15:04 wlan0: authenticated]` denotes an event from a Linux kernel log. Let *Event* denote the set of all possible events. While an event may be regarded as defined in terms of

```
 1  Feb 24 08:15:04 wlan0: authenticate with 38:60:77:7d:6c:4f
 2  Feb 24 08:15:04 wlan0: send auth to 38:60:77:7d:6c:4f
 3  Feb 24 08:15:04 wlan0: authenticated
 4  Feb 24 08:15:04 wlan0: associate with 38:60:77:7d:6c:4f
 5  Feb 24 08:15:04 wlan0: RX AssocResp from 38:60:77:7d:6c:4f
 6  Feb 24 08:15:04 wlan0: associated
 7  Feb 25 14:12:18 wlan0: authenticate with 38:60:77:7d:6c:4f
 8  Feb 25 14:12:18 wlan0: send auth to 38:60:77:7d:6c:4f
 9  Feb 25 14:12:18 wlan0: authenticated
10  Feb 25 14:12:18 wlan0: associate with 38:60:77:7d:6c:4f
11  Feb 25 14:12:18 wlan0: RX AssocResp from 38:60:77:7d:6c:4f
12  Feb 25 14:12:18 wlan0: associated
```

Fig. 1. Linux kernel event log $\langle k_1, k_2 \ldots, k_{12} \rangle$

a collection of attributes, we do not prescribe any particular attributes or structure on an event.

An *event equivalence* relation $\_ \sim \_ : Event \leftrightarrow Event$ defines classes of events that are considered to have common characteristics. Let $Eq\mathcal{E}$ define the set of all event equivalence relations. For example, kernel events [Feb 24 08:15:04 wlan0: send auth to 38:60:77:7d:6c:4f] and [Feb 24 08:15:04 wlan0: authenticated] are defined as (date) equivalent as they both occur on the same date. Alternatively, the events [Feb 25 14:12:18 wlan0: associate with 38:60:77:7d:6c:4f] and [Feb 24 08:15:04 wlan0: send auth to 38:60:77:7d:6c:4f] are defined as mac-equivalent as they refer to the same hardware device.

The different characteristics of an event can be described in terms of its *attributes*. For example, a kernel log event could be described in terms of attributes date, iface, action and device. Let *Attribute* define the set of all attributes. Given event $e : Event$ and set of attributes $A : \mathbb{P} Attribute$ then the *event projection* $e@A$ gives the event $e$ with attributes not in $A$ removed. For example, [Feb 24 08:15:04 wlan0: authenticated]@{iface,action} gives event [wlan0: authenticated]. Intuitively, event projection can be used to define event equivalence. Given set of attributes $A$ and events $e,f : Event$ then define $e \sim_A f \Leftrightarrow (e@A = f@A)$.

A *trace* is a sequence of events. Let *Trace* define the set of all traces.

$$Trace == \operatorname{seq} Event$$

For example, Figure 1 depicts a trace of kernel log events $\langle k_1, k_2 \ldots, k_{12} \rangle$.

For the purposes of this paper, system logs are used to build approximations of system behavior, in particular, sets of n-grams [11] are used to model acceptable system traces. This construction is represented as an equivalence relation, whereby $t \equiv s$ is interpreted to mean that the underlying approximate model considers trace $t$ to be *similar* to trace $s$ (n-gram match). If n-grams are the same length as the trace then similarity is defined by trace equality. If n-grams are of length one then, for the kernel log example, we have, for example, $\langle k_1, k_2, k_3 \rangle \equiv \langle k_2, k_3, k_1 \rangle$. N-gram based trace similarity is typically defined with respect to some threshold [11]. For the purposes of this paper the reader may take $t \equiv s$ to mean that traces $t$ and $s$ are similar to an acceptable predefined degree, without any loss of generality.

Given event projection operation $\_@\_$ then *trace projection* $t@A$ is defined as the projection (based on attributes in $A$) of events in $t$. For example, $\langle k_1, k_2 \rangle @ \{\text{action}\} = \langle \text{authenticate, send auth} \rangle$. This can be further generalized to projection over sets of traces, where given a set of traces $T$ then $T@A$ returns the set of traces $t@A$ where $t \in T$.

## B. Strands and Partitions

A *strand* is a trace of events that share a common characteristic. Let $Eq\mathcal{E}$ be the set of all possible equivalence relations over *Event*. Given an equivalence relation $\sim$ defined over events then define $Strand(\sim)$ to be the set of all possible traces of equivalent events. Note, $\mathbb{P}$ denotes a power set and $\operatorname{ran}(f)$ denotes range of the function $f$.

$$\begin{array}{|l}
Strand : Eq\mathcal{E} \rightarrow \mathbb{P}\, Trace \\
\hline
\forall \_ \sim \_ : Eq\mathcal{E} \bullet \\
\quad Strand(\_ \sim \_) = \{ t : Trace \mid (\forall\, e,f : \operatorname{ran}(t) \bullet e \sim f) \}
\end{array}$$

For example, if $\sim_{date}$ denotes date-equivalence of kernel log events then traces $\langle k_1, k_2, k_3 \rangle$ and $\langle k_2, k_4, k_6 \rangle$ from Figure 1 are members of $Strand(\sim_{date})$, while trace $\langle k_1, k_7 \rangle$ is not a member.

Any trace can be *partitioned* into a set of strands that preserve the event ordering in the original trace. Define function $prtn(\sim, t)$ to be the partitioning of the trace $t$ into a set of strands according to the event equivalence relation $\sim$. For example, partitioning the kernel log in Figure 1 according to date-equivalence generates exactly two strands $\langle k_1, .., k_6 \rangle$ and $\langle k_7, .., k_{12} \rangle$.

## C. Norms

Sets of event traces can be used to model system behavior [4]. We are interested in inferring behavioral models of systems from their logs. These models may contain repeating patterns of behavior. A behavioral *norm* is a set of traces that is considered to define a comparable behavioral pattern. For example, traces $\langle k_1, k_2 \rangle$ and $\langle k_7, k_8 \rangle$ from the kernel log in Figure 1 represent comparable authentication-related behavior and are (behaviorally) different to $\langle k_2, k_3 \rangle$.

A trace equivalence relation $\_ \approx \_ : Trace \leftrightarrow Trace$ defines classes of traces that have a comparable behavioral characteristic. Let $Eq\mathcal{T}$ define the set of all trace equivalence relations. Given a trace equivalence relation $\approx$, then $Norm(\approx)$ defines the set of all possible norms based on $\approx$.

$$\begin{array}{|l}
Norm : Eq\mathcal{T} \rightarrow \mathbb{P}(\mathbb{P}\, Trace) \\
\hline
\forall \_ \approx \_ : Eq\mathcal{T} \bullet \\
\quad Norm(\_ \approx \_) = \{ c : \mathbb{P}\, Trace \mid \forall\, t_1, t_2 : c \bullet t_1 \approx t_2 \}
\end{array}$$

For example, the set of traces $\{\langle k_1, k_2, k_3 \rangle, \langle k_7, k_8, k_9 \rangle\}$ could be regarded as an authentication norm drawn from Figure 1 and characterizing a fragment of authentication behavior during a wireless network connection. Trace projection can used to construct a definition for trace equivalence whereby, given traces $t, s$, a set of attributes $A$ and a trace projection function @, then $t \approx_A s \Leftrightarrow t@A \equiv s@A$. Note that trace equivalence ($\approx$) is different to trace similarity ($\equiv$), the latter defining an approximate test for trace equality. In the kernel log example and with trace equality as similarity, the authentication norm is based on the trace equivalence $\approx_{\{\texttt{action}\}}$ and $\langle k_1, k_2, k_3\rangle \approx_{\{\texttt{action}\}} \langle k_7, k_8, k_9\rangle$. Under this same interpretation, $\{\langle k_1, k_2, k_3\rangle, \langle k_4, k_5, k_6\rangle\}$ is not an authentication norm.

A set of traces is partitioned by trace equivalence into a set of norms. Define $prtn(\approx, T)$ to be the partitioning of the set of traces $T$ into a set of norms based on the trace equivalence relation $\approx$. In this paper norms are defined in terms of the application of a trace-equivalence relation to a (projected) strand partition of a log. Formally, given a log $l$, an event equivalence relation $\sim$ and a strand equivalence relation $\approx$ then the set of norms is defined as $prtn(\approx, prtn(\sim, l))$.

## D. Modeling HTTP logs

Figure 2 depicts an HTTP log of events $\langle h_1, \ldots, h_{12}\rangle$ generated, for example, by a web-based order processing system. Each line defines an HTTP request event described using the Common Log Format [16] with attributes: remote host, RFC931 authenticated user, [date], "request", status and bytes. For ease of presentation in this example, trace similarity ($\equiv$) is defined by equality and we omit some attributes that are not relevant to the discussion.

*1) Strands for HTTP events:* The behavioral patterns in the HTTP request log can be modeled in many different ways. By event date-equivalence, the log is partitioned into the two strands $\langle h_1, .., h_7\rangle$ and $\langle h_8, .., h_{12}\rangle$, reflecting the common practice of "rolling" logs on a daily basis. An alternative view might consider requests from the same user to be equivalent, partitioning the log into three strands involving frank, alice and lucy.

Suppose that the path portion of an HTTP request event is defined in terms of the attributes method, action and item. For example, in Figure 2 the event $h_1$ is an order (=$h_1$@ {action}) for item 4c4712 (=$h_1$@{item}). In this case an event equivalence relation $\sim_{\{\texttt{item}\}}$, defined in terms of item-equality, groups events together as actions carried out on an order; this partitions the HTTP log into the three distinct strands depicted in Figure 3. Note that for ease of presentation and when no ambiguity can arise attributes may be omitted from a log; in this case Figure 3 is based on the equivalence relation $\sim_{\{\texttt{item}\}}$ defined over httpLog@{user,date,method,action,item}. An alternative view that partitions the log into strands of operations carried out by a given user on a given item is depicted in Figure 4.

```
{⟨frank [05/Nov/2012:09:11:26] PUT /order/4c4712,
  lucy  [05/Nov/2012:16:30:16] GET /order/4c4712,
  lucy  [05/Nov/2012:16:32:32] PUT /invoice/4c4712,
  frank [05/Nov/2012:17:47:33] GET /invoice/4c4712⟩,
 ⟨alice [05/Nov/2012:13:18:46] PUT /order/1d261e,
  lucy  [05/Nov/2012:17:46:06] GET /order/1d261e,
  lucy  [05/Nov/2012:17:48:35] PUT /invoice/1d261e,
  alice [06/Nov/2012:09:58:48] GET /invoice/1d261e⟩,
 ⟨frank [06/Nov/2012:09:10:07] PUT /order/61ec0c,
  lucy  [06/Nov/2012:14:34:31] GET /order/61ec0c,
  lucy  [06/Nov/2012:14:47:20] PUT /invoice/61ec0c,
  frank [06/Nov/2012:16:01:45] GET /invoice/61ec0c⟩}
```

Fig. 3. Strands from httpLog partitioned by attribute item

```
{⟨frank PUT /order/4c4712, frank GET /invoice/4c4712⟩,
 ⟨frank PUT /order/61ec0c, frank GET /invoice/61ec0c⟩,
 ⟨alice PUT /order/1d261e, alice GET /invoice/1d261e⟩,
 ⟨lucy  GET /order/4c4712, lucy  PUT /invoice/4c4712⟩,
 ⟨lucy  GET /order/1d261e, lucy  PUT /invoice/1d261e⟩,
 ⟨lucy  GET /order/61ec0c, lucy  PUT /invoice/61ec0c⟩}
```

Fig. 4. Strands $prtn(\sim_{\{\texttt{user,item}\}}, \text{httpLog})$ projected through $\{\texttt{usr}, \texttt{method}, \texttt{action}, \texttt{item}\}$

*2) Norms for HTTP traces:* The three strands depicted in Figure 3 give a projection of $prtn(\sim_{\{\texttt{item}\}}, \text{httpLog})$ that reflect an item-centric view of the HTTP log. Within each item strand there is a common pattern of behavior, specifically, item ordering actions followed by item invoicing actions. This repeating pattern in order processing can be characterized in terms of the norm:

```
{⟨PUT /order/4c4712, GET /order/4c4712,
   PUT /invoice/4c4712, GET /invoice/4c4712⟩,
 ⟨PUT /order/1d261e, GET /order/1d261e,
   PUT /invoice/1d261e, GET /invoice/1d261e⟩,
 ⟨PUT /order/61ec0c, GET /order/61ec0c,
   PUT /invoice/61ec0c, GET /invoice/61ec0c⟩}
```

based on the trace-equivalence $\approx_{\{\texttt{method,action}\}}$ over the projected strands defined by $prtn(\sim_{\{\texttt{item}\}}$, httpLog)@{method,action,item}. Intuitively, the above norm represents a transaction-style behavior pattern in the events of the HTTP log. The norm is a collection of strands that have an equivalent behavior pattern (according to $\approx_{\{\texttt{method,action}\}}$), each carried out on an identified target (in this case, item).

Intuitively, strands define sequences of equivalent events, while the strands that make up a norm define a (repeated) equivalent behavior pattern. Different event and trace equivalences result in different norms, reflecting different kinds of patterns of behavior within the system. For example, Figure 4 depicts a partition of the log into strands that define the actions of a given user on an given item. The strand ⟨frank PUT/order/4c4712, frank GET/invoice/4c4712⟩ from this partition represents frank ordering and invoice-processing item 4c4712. Across these strands is a repeating user-order-invoice behavior norm that can be identified in terms of strands that have equivalent method and action attributes, that is, by the (strand) trace equivalence relation $\approx_{\{\texttt{method,action}\}}$. In this case, the strand partition of the log (given in Figure 4) is partitioned into a customer norm and a merchant norm, as depicted in Figure 5. A customer (norm) puts orders and gets invoices while the merchant (norm) gets orders and puts invoices. These norms also suggest user-roles whereby Frank and Alice are customers and Lucy has a merchant role.

```
{{⟨frank PUT /order/4c4712, frank GET /invoice/4c4712⟩,
  ⟨frank PUT /order/61ec0c, frank GET /invoice/61ec0c⟩,
  ⟨alice PUT /order/1d261e, alice GET /invoice/1d261e⟩},
 {⟨lucy GET /order/4c4712, lucy  PUT /invoice/4c4712⟩,
  ⟨lucy GET /order/1d261e, lucy  PUT /invoice/1d261e⟩,
  ⟨lucy GET /order/61ec0c, lucy  PUT /invoice/61ec0c⟩}}
```

Fig. 5. $prtn(\approx_{\{\texttt{method,action}\}}, prtn(\sim_{\{\texttt{user,item}\}}, \text{httpLog})$ projected through $\{\texttt{usr}, \texttt{method}, \texttt{action}, \texttt{item}\}$

```
 1  10.20.3.11 - frank [05/Nov/2012:09:11:26] "PUT /order/4c4712 HTTP/1.1" 200 1724
 2  10.43.9.1  - alice [05/Nov/2012:13:18:46] "PUT /order/1d261e HTTP/1.1" 200 4354
 3  10.1.12.1  - lucy  [05/Nov/2012:16:30:16] "GET /order/4c4712 HTTP/1.1" 200 6356
 4  10.1.12.1  - lucy  [05/Nov/2012:16:32:32] "PUT /invoice/4c4712 HTTP/1.1" 200 2326
 5  10.1.12.1  - lucy  [05/Nov/2012:17:46:06] "GET /order/1d261e HTTP/1.1" 200 8320
 6  10.20.3.11 - frank [05/Nov/2012:17:47:33] "GET /invoice/4c4712 HTTP/1.1" 200 2925
 7  10.1.12.1  - lucy  [05/Nov/2012:17:48:35] "PUT /invoice/1d261e HTTP/1.1" 200 221
 8  10.20.3.11 - frank [06/Nov/2012:09:10:07] "PUT /order/61ec0c HTTP/1.1" 200 3327
 9  10.76.13.8 - alice [06/Nov/2012:09:58:48] "GET /invoice/1d261e HTTP/1.1" 200 6366
10  10.1.12.2  - lucy  [06/Nov/2012:14:34:31] "GET /order/61ec0c HTTP/1.1" 200 2727
11  10.1.12.2  - lucy  [06/Nov/2012:14:47:20] "PUT /invoice/61ec0c HTTP/1.1" 200 9326
12  10.20.3.11 - frank [06/Nov/2012:16:01:45] "GET /invoice/61ec0c HTTP/1.1" 200 332
```

Fig. 2. httpLog trace of HTTP requests $\langle h_1, \ldots, h_{12} \rangle$

## III. In Search of Norm

A norm identifies common sequences of *operations* carried out relative to a *target*. Let $O$ and $T$ denote sets of attributes intended to represent the operation and the target of events, respectively. Event equivalence $\sim_T$ distinguishes targets while strand equivalence $\approx_O$ distinguishes operations. For example, in Figure 5, order-invoice operations are carried out relative to user-item targets. We are interested in identifying likely target and operation attributes (event and strand equivalence relations) that generate norms that provide meaningful characterizations of a system's behavior. Considering a norm defined as a date operation on a method target, that is, $prtn(\approx_{\{\mathtt{date}\}}, prtn(\sim_{\{\mathtt{method}\}}, \mathsf{httpLog}))$, does not reveal anything interesting about the httpLog. However, the norms in Figure 5 do reveal potentially interesting customer and merchant related norms.

Given operation and target attribute sets $O$ and $T$, respectively, then $l@(O \cup T)$ gives the view of interest of the log $l$. In this case, the norms of log $l$, defined as

$$\mathcal{N}_T^O(l) = prtn(\approx_O, prtn(\sim_T, l))@O$$

are intended to represent patterns of operations on targets. Note that events not in $O \cup T$ are considered superfluous while targets are not explicitly included in the final set of norms since their existence is implicit in the strands they relate to. These norms $\mathcal{N}_T^O(l)$ are computed on the basis of a single *learning* log $l$. The effectiveness of using $\mathcal{N}_T^O(l)$ as a representative model of the given system's behavioral norms is determined by comparison with the norms generated by a further *test* log $t$ of valid interactions of the system.

Let $\mathcal{M}_T^O(l,t)$ define an operation that compares the norms of $\mathcal{N}_T^O(l)$ and $\mathcal{N}_T^O(t)$ and returns a measure in [0..1] that indicates their degree of similarity to each other, whereby a higher value indicates a greater degree of similarity. Intuitively, $\mathcal{M}_T^O(l,t)$ gives a measure of the false negatives when one treats $\mathcal{N}_T^O(l)$ as a model of the behavioral norms in the system. A measure of the false positives for $\mathcal{N}_T^O(l)$ as a model of behavioral norms is given by $\mathcal{M}_T^O(l,c)$, where $c$ is a *control* log of the system, that is a log with sequence perturbations that are known not to occur in the system. Section III-A outlines our current encoding of $\mathcal{M}_T^O(l,t)$ based on n-grams.

### A. N-gram based Norm Similarity

N-grams are used to provide approximate models of system behavior [11]. An observed trace of the system is encoded as a set of n-grams of size $n$. The objective is to find a suitable n-gram size such that this set of n-grams can be used to test, to some degree of accuracy, the validity of any execution trace of the system. Forrest [11] describes a strategy for building this set from a log trace and uses system test and control logs to indicate accuracy. The simplicity of this approach makes it attractive for initial implementation, where we focus on testing for norms existence.

This n-gram model provides (approximate) trace matching and can be used to implement the trace similarity relation ($\equiv$). This is generalized to norms, whereby a set of strands (a norm) is encoded as a set of n-grams that provide an (approximate) method to test a strand for membership of a norm. A variation of the Jaccard coefficient [5] is used to provide a measure of the similarity between two sets (norms) of n-grams. This gives the size of the intersection of the sets divided by the size of the union of the sets. Let $\mathcal{J}(n,m)$ denote this measure between norms $n$ and $m$.

Given a log $l$ and a test trace $t$ then the average of the best of the Jaccard coefficients from the log norms in $\mathcal{N}_T^O(l)$ to the test norms in $\mathcal{N}_T^O(t)$ defines a similarity measure between the sets of norms. Thus, given operator and target attributes $O$ and $T$; log and test traces $l$ and $t$, and an underlying n-gram model then define:

$$\mathcal{M}_T^O(l,t) = \sum_{n \in \mathcal{N}_T^O(l)} \left( \frac{\max_{m \in \mathcal{N}_T^O(t)} \mathcal{J}(n,m)}{\mid \mathcal{N}_T^O(l) \mid} \right)$$

The same calculation is used to define the similarity $\mathcal{M}_T^O(l,c)$ between the log trace $t$ and the control trace $c$.

### B. Implementation

Given traces $l$ (log), $t$ (test) and $c$ (control), then a norm search finds operation attributes $O$, target attributes $T$ and an n-gram model ($\equiv$) resulting in the best values for $\mathcal{M}_T^O(l,t)$ and $\mathcal{M}_T^O(l,c)$. A prototype of this search has been implemented. The objective of this paper is to demonstrate the existence and potential utility of norms. While the current implementation is quite effective for demonstrating this for the moderately-sized logs and attribute sets described in the next section, we have not focussed on search efficiency. In principle, the search is exponential in its parameters. However, many techniques exist for dealing with these kinds of problems and developing a scalable norm search is a topic for future research.

## IV. Evaluation

Two experiments were carried out in order to evaluate whether norms emerge from system logs. Section IV-A describes the emergence of norms from the logs of a simulated system. Section IV-B documents norms that emerged in our study of a complex enterprise-grade application system.

## *A. Norms in a simulated system*

A system that simulated the execution of the HTTP example in Section II-D was developed. It was extended to include additional actions like cart, payment, dispatch, return, etc. The objective of this experiment was to demonstrate that norms could be discovered in logs of systems that were fabricated deliberately to have repeating patterns within their apparently random-looking behavior. In particular, that the norm search would find the sets of operation $O$ and target $T$ attributes as expected.

The system was built as a collection of concurrent users, engaging HTTP events. A simple control-flow model was used to specify user behavior scenarios in terms of repeating sequences of events to be carried out across (random) items. During simulation, a user repeatedly selects, at random, execution sequences from the choice defined by the scenario. This control-flow model was used to generate system traces by (randomly) interleaving the user behavior scenarios.

The simulation comprised of 500 executions by 25 users, each selecting executions at random from the choice of 30 sequences defined by the user-scenario. Running the simulation twice generated different learning $l$ and test $t$ traces, each containing approximately 50,000 events. A similar sized control log $c$ was generated by re-running the simulation, but making random perturbations to the sequences to generate invalid scenario behaviors. The search considered n-grams of 3, 5 and 7 in length.

While the event attributes included user,method,action and item, it should be noted that in this experiment no information was provided to the search algorithm that might a priori suggest candidate operation and target attributes. Each event also included a transaction identifier (trans) that tied the event to a unique execution sequence within a user behavior scenario.

Table I provides a selection of operation and target attributes, along with $\mathcal{M}_T^O(l,t)$ (false positive) and $\mathcal{M}_T^O(l,c)$ (false negative) measures computed for given n-gram sizes $n$ and based on the simulated log traces $l, t$ and $c$.

| operation $O$ | target $T$ | $n$ | $\mathcal{M}_T^O(l,t)$ | $\mathcal{M}_T^O(l,c)$ |
|---|---|---|---|---|
| trans | user | 3 | 0.14 | 0.45 |
| item | method, user | 7 | 0.18 | 0.47 |
| trans | item | 3 | 0.18 | 0.46 |
| method, action | user | 5 | 0.28 | 0.00 |
| method, user | trans | 3 | 0.32 | 0.01 |
| method, action | item, user | 3 | 0.75 | 0.02 |
| method, action | trans | 7 | 0.78 | 0.03 |
| method, action | trans | 5 | 0.81 | 0.00 |
| method, action | trans | 3 | 0.85 | 0.02 |

TABLE I. SOME NORMS IN THE SIMULATED SYSTEM

These results suggest, as expected, that the repeating pattern of method−action operations carried out on transactions (trans) is a good norm; providing a high-degree of similarity between the learning and test logs (few false positives) and a low-degree of similarity between the learning and control logs (few false negatives). While attribute trans was intentionally constructed to provide a unique transaction identifier for the user-scenario, it is interesting to note that the search also suggests the pair user,item as a reasonable set of target attributes. On further investigation it turned out that in the simulation logs, it was more likely that the execution scenarios of different users involved different items, that is, there were relatively few instances of order-invoice transactions involving multiple users. This could be regarded as an unexpected norm that emerged as a consequence of the simulation design. Note that presented similarities are mertics that are relevant to establishing best attrbiutes for norm model and they have no relation with similarity levels in possible application of norms in anomaly detection.

Recall that a norm (set of strands) is implemented as a set of n-grams which, by its nature, provides an approximation for the behavior pattern. It is instructive to consider the effect that the n-gram approximation model (encoding trace similarity) has on the number of norms identified in $\mathcal{N}_T^O(l)$, for the given $O$ and $T$. Figure 6 compares the numbers of norms discovered for different degrees of trace similarity. With a degree of similarity of 1 traces must exactly match and as consequence, there are a large number of distinct norms, many with very similar but strictly different behavior. With a degree of similarity of 0 then traces of the same events, but differently ordered, are considered matched and as a result there is effectively one norm, matching all possible behaviors.

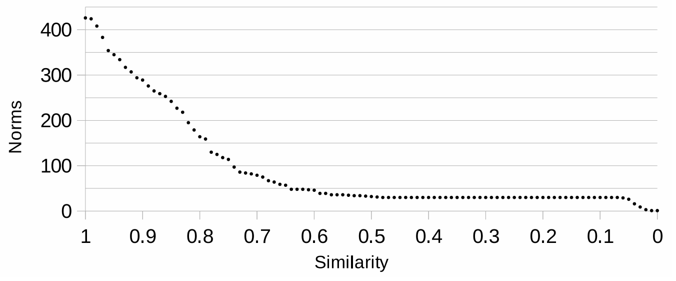

Fig. 6. Number of norms for different n-gram approximations (simulated system)

Our goal is to find a similarity level that results in n-gram approximation that is useful to build norms. Such level of similarity should allow some scope of difference between traces while still consider them as being behavioraly equivalent.

Inspecting Figure 6 reveals that 30 distinct norms are identified for a substantial range, between 0.5 and 0.05, of the degree of trace similarity. Recall that in configuring the control-flow model, logs were generated by users repeatedly selecting, at random, from a choice of 30 different execution scenarios. On inspection, each norm corresponds to one of the execution scenarios, confirming that the proposed operation $O$ and target $T$ attributes reflect the patterns of behavior that were intended.

This experiment demonstrated that analyzing norms in a log trace can be used to test for existence of behavioral patterns in a simulated system. The search discovers the attributes that characterize the sequences of operations carried out on targets along with the best n-gram profiles for matching the identified norms.

The problem of finding optimal attributes is exponential with respect to number of attributes in the event. For five attributes, an exhaustive search over space of all possible combinations of attribute for target ($2^5$), operation (also $2^5$) and n-gram sizes (3) would require 3072 algorithm iterations. Thanks to some basic optimizations (like not considering operation attributes in targets) in the experiment we reduced that number to 510 which took about 30 minutes on a laptop with 1.6GHz four core CPU.

*1) Simulating anomaly:* In the following side experiment, the system simulating HTTP application was extended to model an access control mechanism. The access control is implemented by assigning each of the scenarios and each of the users one of five roles. The simulation has been modified, so that scenarios are only executed by users with a matching role. For illustration, the scenario related to making an order may be only executed by a users with the role Customer while scenario related to issuing an invoice by users with the role Merchant. Information about user role was included as an event attribute and new events contained 6 attributes (method, action, user, item, role, trans).

The norm search was performed for the modified system. Optimal values identified by the search are role, method, action for the operation and trans for the target. It should be noted that, compared with the system without the access control capability, the operation includes role in addition to method and action. Inclusion of role attribute makes the operation more precisely defined (e.g. ”GET invoice as Customer” compared to just ”GET invoice”) and produces more accurate norm model. The search for similarity level, as presented on Figure 7 (normal) shows the result similar to the original system (without access control) and reflects 30 scenarios for a large similarity range.

In the second part of this experiment, the simulation was modified to model a security flaw. The access control was disabled so any user could run any scenario regardless of their role. The intention of this change was to simulate an accidental misconfiguration that breaks security control and check how such change will reflect on system's norms. The search for similarity level was performed on the trace from modified system. Figure 7 shows number of norms in relation to similarity for both executions (normal and abnormal) of the system with the access control capability.

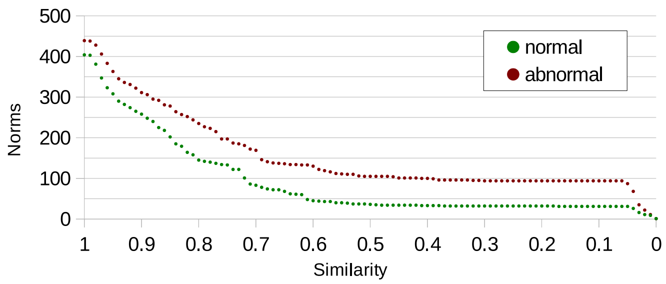

Fig. 7. Number of norms for different n-gram approximations for two configurations of simulated system with access control

This experiment shows that system with broken access control produces much more norms than the one with the access control functioning properly. Although both systems were executing the same same 30 scenarios, they produce traces that are behaviorally different according to identified norm attributes. In the system with enabled access control, different executions of the same scenario produced traces different only because of the control flow. For n-gram similarity between 0.5 and 0.05 they were considered equivalent. In the system with broken access control, executions of the same scenario, but with different roles, produced traces in which *operations* were also different. Even at low similarity levels such traces could not be considered equivalent as they represented completely different behaviors.

This experiment demonstrates that a change in system's behavior may be identified by observing amount of norms that emerge from that system. This is possible even without insight in what individual norms actually represent.

## B. Norms in an Enterprise system

The second experiment was based around an enterprise Java-based social software application. This application provides its users with social communication and content management. The objective of this experiment was to investigate the emergence of norms in an existing large-scale application system.

The application server running the application was configured to include a custom Java Security Manager [12] that recorded every permission check as an event. Six attributes of the events logged by the manager were considered: type of permission (attribute perm, with values FilePermission, SocketPermission, etc.); action (with values, open, read, etc.); name of application server's thread used to perform the operation; application's user on behalf of which the thread executes, and name of the class that invoked the code requiring the permission. Time was recorded using reduced precision.

The application was invoked 1,500 times via its REST API in order to execute 11 different high-level actions (such as file upload, download, adding file to folder, etc.) concurrently for 10 different application users. This was done twice in order to generate the learning and test traces, each containing about 30,000 events. The control trace was created by randomly reordering events from the test trace. N-gram sizes of 3, 5 and 7 were used in the search for norms. Table II provides a selection of operation and target attributes, along with $\mathcal{M}_T^O(l,t)$ (false positive) and $\mathcal{M}_T^O(l,c)$ (false negative) measures that were computed during the search. The results in Table II are for an n-gram size of 3, which, when analyzing this system, produced better results than for other sizes.

| operation $O$ | target $T$ | $\mathcal{M}_T^O(l,t)$ | $\mathcal{M}_T^O(l,c)$ |
|---|---|---|---|
| perm | action, thread, user | 0.38 | 0.49 |
| class, user | action, prtm, time | 0.15 | 0.01 |
| name, perm | thread, trace | 0.51 | 0.37 |
| perm | time, user | 0.99 | 0.11 |
| class, perm | time, user | 0.97 | 0.07 |
| class, perm | thread,time,user | 0.99 | 0.08 |
| action, class | thread,time,user | 0.92 | 0.00 |
| action,class,perm | thread,time,user | 0.92 | 0.00 |

TABLE II. SOME NORMS IN THE ENTERPRISE SYSTEM

The results show that the best candidates for operation attributes are combinations of action, perm and class attributes.

It is interesting to note that the operation with attributes {action,class} has results comparable to {action,class,perm}. Intuitively, the latter seems to be a better choice as it defines actual operation more precisely ([SomeClass, open, file] compared to just [SomeClass, open]). This is not surprising since Java tends to define different permission classes according to different kinds of actions and targets. For example, FilePermission is defined in terms of read/write/execute/delete actions (on file targets), while SocketPermission is defined in terms of accept, connect, listen and resolve actions (on socket targets). Thus, for this system, permission name does not provide any additional information regarding the target of the norm.

Studying the effect that the n-gram approximation model (encoding trace similarity) has on the number of norms identified in $\mathcal{N}_T^O(l)$, for the best $O$ and $T$, provides some interesting insights. In Figure 8, when the degree of trace similarity varies between, 0.34 and 0.17, then 11 norms are identified. This is as expected, as these 11 norms effectively correspond to the underlying behavior patterns resulting from repeatedly invoking the 11 different REST API calls in order to populate the log.

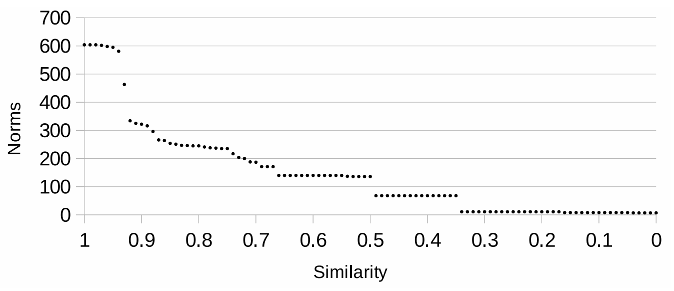

Fig. 8. Number of norms for different n-gram approximations (simulated system)

However, Figure 8 also points to the presence of other potential norms in the system. There are three other regions in the graph that suggest different numbers of norms. For degrees of similarities: between 0.66 and 0.55 there are 140 norms, between 0.49 and 0.45 there are 68 norms, and there are 7 norms between 0.17 and 0.05. We conjecture that these additional norms are a consequence of the application, or its underlying infrastructure, performing different kinds of internal operations for the the same REST API calls. For example, in the same API call we may, at one time, retrieve an object from remote storage while retrieving it from local cache another time; an application may once have established a connection, while at another time uses a previously established connection from a connection pool. Traces generated by different execution paths may be sufficiently different to be considered distinct norms where a high degree of trace similarity is required. This means that it may be possible to generate multiple norm models differing only by degree of trace similarity to have larger number of more precise norms or smaller number of more general ones. While we believe that the 11 norms identified between 0.34 and 0.17 correspond to the events resulting from the REST calls, investigating the cause of the other, emergent, norms is a topic for future research. This experiment was a preliminary attempt to validate that meaningful norms can be identified in a real and complex enterprise system. In building the experiment, many simplifications had to be made, such as ignoring client-side caching and calling only the REST API rather then accessing application resources as the user.

The false positive measure should compare generated norms with norms of abnormal behaviour. Because no traces of abnormal behaviour were available we generated such trace by randomly reordering events in one of existing traces. Investigation on effectiveness of this approach and possible alternatives is a topic for future research.

While the strategy of exhaustive norm search is effective for moderately sized logs, further research is required to develop a scheme to search much larger logs of a scale that is typical of complex enterprise systems.

## V. Discussion and related research

The previous section demonstrates that it is possible to search for norms that characterize underlying patterns of behavior without any prior knowledge about the system.

Norms may provide an efficient way to compare and evaluate system behavior during its lifecycle. If the number of norms increase after system change, such as reconfiguration or patching, it may indicate that some of the security controls (in or outside of the system) are no longer effective. Such behavior was seen in Section IV-A1. Monitoring changes in the number of norms may be useful in detecting potential problems.

The norm model relates to anomaly detection based on trace analysis. One of the problems recognized in this area is finding an anomalous sequence with regards to a sequence database [8], also known as a ‘sense of self’ [11]. It can be considered to be a part of norms model to a limited extent, as it effectively treats the system as a single homogenous norm in which one looses many of the subtleties of interaction. Norms, when used for anomaly detection, are similar to another problem – detecting anomalous sub-sequences within a longer log, called *discords*. They are, however, richer and more accurate then currently used techniques [8]. Rather than using fixed size windows, norm model provides much more adaptive variable length sequences that base on event equivalence. Another advantage of norms is an automatic attribute discovery. Study of anomaly detection recognizes n-grams as one of the techniques for modeling behavior, referred to as window-based [8]. Applicability of other techniques, such as kernel and Hidden Markov Models, to norm model is a subject for future research.

Intrusion detection systems may benefit from using norms at multiple precision levels. As it was demonstrated by experiment in Section IV-B, using different trace similarity level may result in different distinct levels of norms precision. A hybrid IDS employing this technique may match at multiple precision levels and, depending on configuration, adjust alert levels or automatically learn new rules.

Behavioral norms are related to process mining techniques [3], [22] whereby process models, such as Petri-nets, are generated from low level audit logs. These techniques have been used for security conformance [1]. These models tend to use coarse-grained event abstractions, such as operation-name and date. Future research will investigate how the results in this paper might be used as pre-processing step in order to identify additional attributes that that could be used in subsequent process mining.

Research in the area of activity recognition shows that patterns of behavior may be identified by observation [13] and that n-grams may be used to build an underlying model [10]. Investigating the relationship between behavioral norms and activity recognition techniques is a topic for future research.

## VI. Conclusion

Behavioral norms are emergent repetitive patterns that provide a behavior-centric view of a security log. These patterns can emerge at different levels of abstraction. At the highest level of abstraction, a single norm may be identified, corresponding to the conventional [11] 'sense of self'. However, as demonstrated by the experiments described in Section 4, this normal behavior pattern may actually contain further patterns of behavior. These other norms are determined by choosing the right operation and target event abstractions. Some suitable abstractions may be a priori known, such as an event method (operation) carried out on a transaction-id (target), as described in the example in Section IV-A. However, other and/or better abstractions may emerge from analysis of the log; for example, it was demonstrated that the attribute pair (user,item) provides a good alternative to the transaction-id target. In the enterprise system example, analysis points to previously unknown norms in the system. The existence of these emergent norms is not surprising given that the complexity of modern systems is such that a full understanding event attributes/behaviors may not be a priori known.

This paper proposes a model and approach to analyzing system logs and identifying behavioral norms at different levels of abstraction. In identifying these repeating patterns of behavior, norms can be used to help determine the right level of abstraction at which to monitor security violations and incidents in a security log. The contribution in this paper is a behavior-centric approach to security log modeling and a demonstration that emergent norms can be searched for, and exist, at different levels of abstraction. A rudimentary norm-search based on an n-gram model of behavior was used in this paper. The model is not limited to these and future research will investigate alternative behavior models and more efficient search/data mining techniques.

## References

[1] R. Accorsi, "A security-aware simulation method for generating business process event logs," in *Symposium on Data-Driven Process Discovery and Analysis, number to appear in Lecture Notes in Business Information Processing. Springer*, 2012.

[2] R. Accorsi, C. Wonnemann, and T. Stocker, "Towards forensic data flow analysis of business process logs," in *IMF*. IEEE Computer Society, 2011, pp. 3–20.

[3] R. Agrawal, D. Gunopulos, and F. Leymann, *Mining Process Models from Workflow Logs*, ser. EDBT '98. Springer-Verlag, 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=645338.650397

[4] B. Alpern and F. Schneider, "Recognizing safety and liveness," *Distributed Computing*, vol. 2, pp. 117–126, 1987.

[5] M. R. Anderberg, *Cluster analysis for applications*, ser. Probability and mathematical statistics. Academic Press, 1973. [Online]. Available: http://books.google.com/books?id=BZBYAAAAMAAJ

[6] T. Atkins, "Simple log watcher," http://sourceforge.net/projects/swatch, June 2013.

[7] D. A. Basin, F. Klaedtke, and S. Müller, "Policy monitoring in first-order temporal logic," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, 2010, pp. 1–18.

[8] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 823–839, 2012.

[9] N. Dinesh, A. K. Joshi, I. Lee, and O. Sokolsky, "Checking traces for regulatory conformance," in *Runtime Verification, 8th International Workshop*, 2008, pp. 86–103.

[10] K. Farrahi and D. Gatica-Perez, "Extracting mobile behavioral patterns with the distant n-gram topic model," in *Wearable Computers (ISWC), 2012 16th International Symposium on*. IEEE, 2012, pp. 1–8.

[11] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, "A sense of self for unix processes," in *IEEE Symposium on Security and Privacy*, 1996, pp. 120–128.

[12] L. Gong, G. Ellison, and M. Dageforde, *Inside Java 2 Platform Security: Architecture, Api Design, and Implementation*. Addison-Wesley Professional, 2003.

[13] T. Gu, Z. Wu, X. Tao, H. Pung, and J. Lu, "epsicar: An emerging patterns based approach to sequential, interleaved and concurrent activity recognition," in *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*. IEEE, 2009, pp. 1–9.

[14] *ISO/IEC 27001:2005 – Information technology – Security techniques – Information security management systems – Requirements*, International Organization for Standardization, 2005.

[15] K. Kent and M. Souppaya, "SP 800-92. Guide to Computer Security Log Management," Gaithersburg, MD, United States, Tech. Rep., 2006.

[16] A. Luotonen, "The common logfile format," http://www.w3.org/pub/WWW/Daemon/User/Config/Logging.html, 1995.

[17] *Payment Card Industry Data Security Standard (PCI DSS)*, Payment Card Industry Security Standards Council, 2010.

[18] M. Roger and J. Goubault-Larrecq, "Log auditing through model-checking," in *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, ser. CSFW '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 220–. [Online]. Available: http://dl.acm.org/citation.cfm?id=872752.873518

[19] J. P. Rouillard, "Real-time log file analysis using the simple event correlator (SEC)," in *LISA*. USENIX, 2004, pp. 133–150.

[20] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni, "A fast automaton-based method for detecting anomalous program behaviors," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2001, pp. 144–155.

[21] J. M. Spivey, *The Z notation - a reference manual*, ser. Prentice Hall International Series in Computer Science. Prentice Hall, 1989.

[22] W. M. van der Aalst, , T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 9, pp. 1128–1142, 2004.

[23] W. M. van der Aalst, K. van Hee, J. van Werf, and M. Verdonk, "Auditing 2.0: Using process mining to support tomorrow's auditor," *IEEE Computer*, vol. 43, no. 3, pp. 90–93, 2010.