OpenUBA: A SIEM-agnostic, Open Source Framework for Modeling User Behavior

May 25, 2020

Jovonni L. Pharr
jpharr2@student.gsu.edu
Atlanta, GA, USA
[v0.0.1]

Abstract

This project describes a system for analytic modeling of user & entity behavior. We make use of the scientific computing community, and the tools used within it. We demonstrate conditions under which modeling invocation takes place, the handling of model results, and a description of model components. Version control is also described for models changing over time. Feedback is described for both rules, and models. Efficient system storage is also demonstrated for anomalies, and events of interest. Various risk calculation approaches are also described. The system takes advantage of turing-completeness for model development, instead of limited development freedom. Lastly, different scenarios in which models, and rules accept feedback are covered. The components described construct a UBA system designed to be extensible, powerful, and open.

Motivation

There exists several security monitoring tools focused on providing User and Entity behavior monitoring capabilities. UBA tools attempt to replace a SIEM solution, but the two tools should work together. Security Operation Centers (SOC) should not be forced to choose between SIEM solution, and a UBA solution. Although, there are SIEM solutions releasing their own UBA features, however, there is a difference between a flexible, and expressive UBA tool, and adding a UBA feature.

1.1 Problems

Many UBA platforms take a "block box" approach to modeling. The very concept of modeling can vary across UBA tools, though more concretely defined in mathematics. Often UBA vendors claim their models are IP, which is an understandable business decision to make. However, ramifications of models being proprietary include vendor lock-in, decreased model developer-friendliness, and hindering community growth & participation.

Also, some SOCs must use multiple tools to gain the benefits of UBA, but this can be expensive depending on the vendor. OpenUBA aims to solve these problems by providing

1. Open/"white box" model standard
2. SIEM-agnostic
3. Turing-complete, expressive modeling
4. Model feedback from cases
5. Model versioning
6. Lightweight architecture
7. Alerting

Modeling

UBA tools use an overarching concept of a model $\boldsymbol{\mathcal{M}}$. A model can be an arbitrary set of commands, of which change the state of the system. Models may execute both stochastic, and deterministic sets of computations. With the depth, and breadth of security use cases, model development in any modern UBA tool should be turing complete, and fully expressive. UBA tools attempt to extend a flexible model development practice, but because it typically results in less than turing completeness, model development is inherently limited, although expressive.

2.1 Mapping Mitre Att&ck

Mapping to the Mitre Att&ck matrix is important for SOCs to monitoring their own performance with respect to understanding their coverage and security posture. Models from the model library are mapped to Mitre Att&ck techniques and tactics. This enables coverage with respect to Mitre Attack to be calculated across the model population. This is becoming common practice for security tools, as Mitre maintains an industry leading position on standardizing attack vectors, and surfaces for the purpose of organizations using a "common language" – this mapping is arbitrary.

2.2 Defining models

We define an abstraction of a model,

$$\boldsymbol{\mathcal{M}} = \{(\boldsymbol{m}, \lambda, s) \mid S_{\texttt{min}} \leq s \leq S_{\texttt{max}}\} \quad (1)$$

Where $\boldsymbol{m}$ is a mitre map, λ is the merkle root hash from the model component tree, s is the contributing risk score for the model, and S is the system's global risk score limits. Each model has a relation to a $m_{\texttt{technique}}$, and $m_{\texttt{tactic}}$.

Models run in model groups, which may contain ≥ 1 model. We define the model execution function,

$$\Xi = \texttt{execute}(\boldsymbol{\mathcal{M}}) \quad (2)$$

which is a function mapping a cartesian product of the set of models, and the set of datasets in the system.

$$\Xi : \boldsymbol{\mathcal{M}} \times D \rightarrow r \tag{3}$$

Where r is a model result, and D is set of data in the system, and $d \in D$, the subset of data used by the `execute` function. Model results can be a set of relations between a user U, and s, given s is between $S_{\mathtt{min}}$, and $S_{\mathtt{max}}$. Such a relation can be "U has new risk score of S", "U has their risk score increased/decreased by S", or another arbitrary risk score calculation.

2.3 Model Library

We make use of a model registry, from which models are installed for use in the system. A model registry can be arbitrary, but must be trusted by the user, although models are verified before installation, and usage.

Components

Models are constructed by ≥ 1 component. Components at a high-level are files. Model library entries contain the file hashes, and encoded-data hashes of each component.

$$\{(c_{type}, f, d) \;:\; c \neq \emptyset, \forall c \in C\} \tag{4}$$

Where f is the file, and d is the encoded data. A model component contains a set of hashes,

$$\boldsymbol{H} := \{\mathtt{H}(f), \mathtt{H}(d) \,|\, f \neq \emptyset, d \neq \emptyset\} \tag{5}$$

where $\mathtt{H}$ is a hash function, and f, and d must be present. Together, these form the model component.

$$c = \{(c_{type}, \boldsymbol{H})\} \tag{6}$$

2.4 Common Modules

For a given model, the model author may be required to rewrite a basic preprocessing step for a model, of which may differ depending on details of how the model is being used. For this reason, we extend common model modules to be used by models. For example, if a model needs to parse incoming data a certain way, the logic can be encapsulated inside of an external module, and invoked by the model, not written within the model itself.

Verification

Model verification takes one of two forms, the system is either verifying the contents from the model library, before installing, or verifying the files the model uses, after being installed, but prior to each execution of the model.

Both verification functions, verifying the encoded data of the model

$$\mathrm{V}_d : C_{data} \rightarrow \{0, 1\} \tag{7}$$

and verifying the files the model use

$$\mathrm{V}_f : C_{file} \rightarrow \{0, 1\} \tag{8}$$

maps an element of the component to a truth value. Overall verification is the logical conjunction of both verification types, and is represented by the boolean expression

$$\mathtt{V} := \mathrm{V}_d \mathrm{V}_f \tag{9}$$

We represent the model invocation function as

$$\Phi = \mathtt{invoke}(\dots) \tag{10}$$

$$\Phi(\Gamma, \boldsymbol{\mathcal{M}}) = \begin{cases} \mathtt{install}(\boldsymbol{\mathcal{M}}), & \Gamma = d \\ \Xi(\boldsymbol{\mathcal{M}}), & \Gamma = f \\ \mathtt{Err}, & \text{otherwise} \end{cases} \tag{11}$$

Using the data and file verifications, we construct the simple verification conditions to be satisfied before model invocation. Invocation occurs if the verification function, V

$$V(\boldsymbol{\mathcal{M}}) = \begin{cases} \Phi(\Gamma, \boldsymbol{\mathcal{M}}), & \text{if } \mathtt{V} \\ \mathtt{Err}, & \text{otherwise} \end{cases} \tag{12}$$

In addition to installation verification, the same process is executed upon each model execution. This ensures the integrity of the models are checked prior to running the model.

2.5 Model Groups

Model groups are sets of models to run with a shared context, and shared datasets. We define executing over model groups by

$$r_j = \Xi(M_{\mathtt{group}_i}, \boldsymbol{\mathcal{M}}_j) \,|\, \forall i, j, \in \boldsymbol{\mathcal{M}}_{\mathtt{enabled}} \tag{13}$$

given that $\boldsymbol{\mathcal{M}}_j$ is enabled, where M is the model configuration, of which defines the group.

Data Loaders

A data loader provides models with the data needed to execute, and any contextual information needed for the model. Context for a model may be the host information if remotely connecting to a data source, or the location of the data source.

If a system has n models enabled for proxy data, data should be loaded at most one time for the model group. The amount of times the same data is loaded, $D_{M_{\text{group}}}$ is not proportional to the number of models, $|\boldsymbol{\mathcal{M}}|$.

$$D_{M_{\text{group}}} \not\propto |\boldsymbol{\mathcal{M}}|, \boldsymbol{\mathcal{M}} \in M_{\mathtt{group}} \tag{14}$$

Data loaders enable model groups to share the data used to model.

2.6 Model Return

Model return types are defined at model configuration time.

$$\rho = \mathtt{result}_{type} \tag{15}$$

and the set of result types is

$$\rho := \{\mathtt{U}_{risk}, \mathtt{U}_{anom}, ...\mathtt{U}_n\} \tag{16}$$

For a return type of U_{risk}

$$\boldsymbol{\mathcal{M}}_{\mathtt{res}} = \begin{cases} \{u_{1_{new}}, ...u_{n_{new}}\}, & \text{if } \rho = \mathtt{U}_{risk} \\ \{u_1, ...u_n\}, & \text{if } \rho = \mathtt{U}_{anom} \\ \mathtt{Err}, & \text{otherwise} \end{cases} \tag{17}$$

where $u_{n_{new}}$ is either the new risk score for user, u, or the risk score to be used for new risk score calculation.

2.7 Sequence Models

$\boldsymbol{\mathcal{M}}$ can be a single-fire model, or a sequence of individual models. Single-fire models perform their logic on the input data, and return a result. Single fire models are markovian, meaning that the state of the system after $\Xi(\boldsymbol{\mathcal{M}})$ is only dependent on the current state, and the state transition initiated by invoking Ξ.

Sequence models depend on outputs from previous models. One approach is to directly feed the output from a model into the input of another model. However, we abstract the connection between models by memoizing model outputs.

The timing of sequence model executions is not always immediately after another. Therefore, it is efficient to memoize prior model output. This approach ensures the length of the time window used during model training/inference can be up to the length of the system historical window. In other words, we can store memoized model outputs, and condition upon them at model execution time.

2.8 Model Versioning

If there exists a model where the model version is the maximum version number in the set of model versions, the system will infer on that model by default. Given a set of all model versions, $\boldsymbol{\mathcal{M}}_{\mathtt{vers}}$

$$\boldsymbol{\mathcal{M}}_{\mathtt{vers}} := \{v_1, v_2, \ldots v_n | \, \boldsymbol{\mathcal{M}}_v \geq 0, \forall v\} \tag{18}$$

$$\exists \boldsymbol{\mathcal{M}} \ni \texttt{max}(\boldsymbol{\mathcal{M}}_{\texttt{vers}}) = \boldsymbol{\mathcal{M}}_v \Rightarrow \Xi(\boldsymbol{\mathcal{M}}_v) \quad (19)$$

This prevents multiple versions of the same model from executing, especially while running on the same data.

2.9 Rules

Traditionally, a rule engine within a SIEM contains a set of conditions. The engines use event data to check against these rules. With OpenUBA, an entire rule engine can be defined within a standalone model. This rule engine executes within a model group, and can output risk scores similar to a traditional model. This approach enables rule engines to be used in the same way models are used.

2.10 Storage

Another issue with current-state UBA tools is the requirement to store a copy of production data in such a way where the tool can access it. This forces UBA users to eventually copy large amounts of data. Although UBA tool tend to delete data beyond a specified time window, copying the data alone can be challenging for users with large amounts of data.

To be lightweight, and "siem-agnostic", we store the anomaly data, and supporting events within a specified time window. This way, the system can still execute on production data, but without the requirement to retain all of the data on disk, or in memory.

If an enterprise proxy log source can result in `1B` record on a given day, which in turn generates $50,000$ proxy anomalies, retaining $50,000$ records on disk is far more efficient than copying the full set of data. This constraint makes the system more lightweight than a traditional bloated SIEM, or UBA system. The number of stored events, $\texttt{e}_{stored}$ is much less than the number of observed events $\texttt{e}_{observed}$

$$\texttt{e}_{stored} \lll \texttt{e}_{observed} \quad (20)$$

If specific models call for sequence of events to occur within an interval of time, $[t^*, t]$, we can efficiently retain the potentially anomalous event for up to $t - \Delta$, where Δ is the amount of time into the past the system can observe an event – typically this is 90 days in traditional SIEM products.

However, this data storage configuration is arbitrary because at any point in time, we can store the anomaly data in an external abstract data store, and a model can simply retrieve it from outside the system, similar to how models retrieve the data used during computation.

Logs

We process logs for both creating users, and entities, and for model execution. Although the process by which the data is processed is the same, the intent is different between the two.

$$\texttt{process}(i, j), \forall i, j \quad (21)$$

3.1 ID Feature

For each log source, a feature to be used as an identifier is configured. The identifier should server as a common key across all log sources, or must be a subset of another key. This identifier makes it less costly to decipher the attributing account for any $\texttt{e}$, not that it cannot be done otherwise.

3.2 Dormancy

Dormancy is determined by the lack of a subset of $\texttt{e}$ in D_u; the dormancy period is arbitrary. If exactly 0 events from preconfigured events, $\texttt{e}_{dorm}$, are found within the logs of u, we say u is dormant.

$$U_{dorm} = \{u \,|\, \forall u \in U, \texttt{e}_{dorm} \notin D_u\} \quad (22)$$

Risk

Risk scores can not only be arbitrary, but can also mislead or confuse stakeholders. "Black box" risk calculations hinder SOCs from fully understanding how case generating anomalies are precisely scored, and their ability to concretely explain the calculations to third parties. There is an inherent tradeoff between interpretable risk scores, and the determinism of risk score calculation. The more a model uses stochastic, and more complex processes to calculate risk scores, the more SOC analysts lose granular explainability of risk results.

The philosophical concept of a risk score is subjective as well, and is designed for human intuition. This goal can be hindered by arbitrary vendor-driven risk score calculations, of which decrease interpretability of model outputs. In data science, there exists an accuracy vs interpretability tradeoff. As more complexed models are invented, of which increase in performance, we lose human intuition on the inner workings of the model. This justifies enabling SOC teams to transparently define their risk score calculation logic as much as possible – hiding concrete risk score logic only harms day-to-day SOC decision making, and requires a great deal of trust from the SOC onto the vendor.

4.1 Risk Calculation

We purposefully abstract the risk calculation, and demonstrate basic risk score calculations for an individual, and a group. If a model results in new user risk score, we can simply overwrite the users risk.

$$u_{\texttt{risk}_{t+1}} = r_u \tag{23}$$

If a model results in s quantity of risk to be used in the final calculation of risk for U, then the abstracted risk score calculation process is invoked using the quantity of risk as an input variable. For example, if the risk calculation for users is a summation between the previous risk score of u, and the model output with respect to u, then

$$u_{\texttt{risk}_{t+1}} = u_{\texttt{risk}_{t-1}} + r_u \tag{24}$$

We define an abstracted risk calculation function, and map over all users

$$u_{\texttt{risk}} = \gamma(u, \texttt{risk}(u,t), \boldsymbol{\mathcal{M}}_r^u), \forall u \in U \tag{25}$$

γ is a cartesian product of the user set, and two real numbers, and yields a real number, representing a new risk score for user, u.

$$\gamma : u \times \Re \times \Re \rightarrow \Re \tag{26}$$

For simple risk score calculations, the function does not need u, however, the system can consider a user's profile as context during risk calculation, opposed to a risk score being independent of u.

Case Feedback

Upon a SOC analyst providing feedback to a model, the models adjust internal logic in some meaningful way. Feedback into a model differs in several ways depending on properties of the model. However, feedback on a rule, must be separated from feedback to a model.

5.1 Model Feedback

Model feedback usually does not include a change in semantic or syntactic forms because model parameters usually change after feedback – although additional feature selection, or preprocessing may also change after feedback.

1. Tuning
   (a) changing parameters in a model
2. Remodeling
   (a) rewriting, or editing the underlying model structure

The feedback process used is mainly for parameter tuning, but more work is to be done on automatically remodeling based on feedback. For a simple statistical model such as

```
if user a triggers rule R
S standard deviations from 'normal'
```

if a result was a false positive, the model would include the false positive metric in its calculation for standard deviation, $\sigma(x)$.

Weighting

Model feedback can also be weighted. Instead of $\sigma(x)$ just having the case-generating event included within its calculation, the function could also apply a weight, $[0, 1]$, of which can damp the feedback. Damping can be done on several variables, including the amount of times the case has been deemed a false positive, or accepted behavior by the SOC.

5.2 Rule Feedback

Rule feedback may look differently from model feedback. Rule feedback may require metaprogramming of the actual rule, or changing a portion. Feedback to a model for a rule such as

```
if eventtype = A
```

may become

```
if eventtype == 0
and if eventtype == 1
```

after feedback. In other words, feedback on rules can change the syntactic form of the rule, or the semantic form, such as variables. Each model defines its own process through which the model interprets feedback given on the case generated.

Future Work

Herein, we have demonstrated an "Open Model" UBA solution for monitoring users and entities. Once systems like OpenUBA are implemented, macro-level SOC activities can be performed using the system's data, and the community data. Several threat feeds exists today, but little to none exists with a focus on analytics, and modeling insights from current threats. Sharing hyperparamaters to fine tune a shared security model on most recent Botnet activity is possible when "Open Model" solutions are used more widely.

6.1 Shared SOC

A SuperSOC consists of a set of small SOCs, all with their own people, processes, and data. SuperSOCs enabled multiple distributed SOCs to function as one organism regarding security threat modeling, and security analytics.

6.2 Distributed Intel

Having community driven intel is not uncommon in several security circles. Isolated, analytics-driven security tools hinder SOCs from building truly knowledge-driven analytic solutions by removing the community participation. More work is to be done in this space.

6.3 Cross-SOC models

SOCs can also begin to collaboratively model security use cases. For example, knowing how anomalous this proxy traffic for a user in your organization, is different when compared to other companies in the same industry. Intersoc collaboration is still an active area of research.