The Mathematics of Strategy Patterns

Thomas John

Abstract

This article sketches the mathematical foundations of strategy patterns. Ongoing threats have persistence which needs to be distinguished from persistent habits of normal users. This requirement is described in terms of separation properties in Mathematical Analysis. Separations are created using strategies for some two-person games. These separators are the strategy patterns that identify persistent threats.

1 Persistence of experiments

Security professionals often talk about persistent threats, or "Advanced Persistent Threat" (APT). The "advanced" part signifies different ways to compromise systems, and the threat corresponds to an active agent.

It is hard to stop malware from entering a system. This has to do with the cleverness of criminals to fool people into doing things that appear to be normal. For example, you could convince someone that an attachment to an email is a legitimate software update or anti-virus feature.

Getting into a system is the first step for hackers. The APT objective is to find and extract valuable information. Thus the APT malware tries not to raise any alerts with excessive cpu, disk or network activity. But the hacker still has to look around to find the valuable information. They may do this quite deliberately, over a long period of time. The process of looking for valuable information however has to involve experimentation; the hacker has to look through multiple directories and computers until they find pieces of useful information. This experimentation is a persistence. Persistence can be mathematically described. Once it is described mathematically, we can formulate algorithms to see if a behavior fits a mathematical description. We do this in many fields, for example we see if a distribution is normal, or if a growth rate is exponential.

Most people are familiar with mathematical descriptions that are like graphs. But persistence is not quite like that. Experimentation involves going through all possibilities, systematically. This means "for every possibility, do an experiment to test that possibility." There is a way to say exactly this, using quantified logic or "predicate calculus." We can say $(\forall x)[P(x) \to T(x)]$

This can be read as "for all x, if x is a possibility then there x is tested in the experiment." For example, if the experimentation is to see if a text file contains

passwords, then $P(x)$ is "x is a text file" and the experimental test T is to see if there are passwords in the text file x.

We can try to see whether the observed data fits this statement. If we see that every text file is tested, say using a pattern matcher such as `grep -i password *.txt`.

But at any time, the experimenter, i.e. hacker, probably has not tested every possible text file. But we can say that we have seen many data instances that suggest that the hacker may be experimenting with all the text files he finds. This is just like fitting mathematical functions to data - we do not get an exact fit, but overall a lot of points are close to the ideal graph.

Thus we can "fit" persistent behaviors to quantified logic formulas. But we cannot do one thing that you can do easily with ordinary graphs. If you had behaviors modeled by ordinary graphs, then you can compare graphs visually to see if one behavior is different from another. For example if one behavior is a linear function $f(t) = At + B$ then it is easy to distinguish this visually from a function $g(t) = At^2 + Bt + C$. With quantified formulas, there is no such easy visual distinction.

Even though the experimentation behavior of a hacker results in a persistence, not all persistences are malicious. For example, many computers run background tasks at regular time intervals. For example, there may be some indexing tasks that help with later searches, such as `updatedb` that run at regular intervals.

2 Separating persistences

Even though it may be difficult to visually distinguish between persistences, there are computational ways to do so. This is related to the mathematics of some topological spaces.

We can describe activities in a computer as sequences of natural (i.e. positive) numbers. We can think of any activity as moving from one situation, or "state", to a new state. There can be a (countably) infinite number of states, and the activity may run for an unbounded amount of time. Thus we can think of the sequence of states as a sequence of numbers.

A hacker or malicious program may go through different chains of activities depending on other conditions. For example, a hacker may try various ways to get into a system. The same technique may not work on every computer in a network, but once they get in the actions in each computer may involve similar experiments.

Thus we can consider all potential activities of a hacker as a set of sequences of numbers. If we think of all the incidents i and all the states in incident i as a sequence of states $S_i = \{s_{i_j}\}$.

If we assume that the states s_{i_j} are obtained as a result of a persistent activity, then we can describe all the possible activities of the hacker with a set

$A = \{S_i = \{s_{i_j}\} | (\forall j) P(s_{i_j})\}$. Such a set is also called a Π_1 set. This also turns out to be a topologically closed set in for the space of all sequences of numbers. Since this set is defined using a universal quantifier $(\forall j)$, that is like an infinite number of logical "and" operations. In Boolean logic, an "and" is considered as a logical product, hence the letter Π stands for the product of all the different assertions $P(s_{i_j})$.

Suppose A and B are Π_1 sets. Suppose A is associated with malicious behavior while B represents some normal, but persistent, activity. There is a way to distinguish between A and B that relies on the topological characterization of these sets as closed sets. In this case, A and B are disjoint sets, i.e. $A \cap B = \emptyset$ since one set is malicious behaviors and the other is made up of normal behaviors. To distinguish between these sets, we can use a concept called "separation". We say that A and B are separated by a set C if $A \subset C$ and $C \cap B = \emptyset$. If A and B are disjoint Π_1 sets then there is a set C that separates A and B such that C and the complement $\overline{C}$ are both Π_1 sets.

Such a set C has some connection to Turing machines. While there are some details involved, essentially one can think of such sets as being defined by converging actions of a Turing machine, thus something related to effectively computable sets. In practical terms, this means that we can try to construct the separating set C, thus giving us a way to distinguish between persistent activities of hackers from persistent activities of normal users.

3 Constructing separators

The method to construct separators defines "safe" positions incrementally.

Recall that we describe actions of a hacker or normal user as sequences of numbers. At any point in time, this sequence is a finite sequence that describes both the history and the current state of the computer system. We can think of all possible such finite sequences as a tree that can potentially keep growing.

Each such sequence can be considered as a list $S = \{s_1, s_2, \ldots, s_n\}$. Thinking of the hacker's actions, suppose the hacker has managed to compromise the system, we can think of this sequence S as a bad situation. When separating a hacker's activity from normal activity, we want to keep the system activity away from such bad situations.

Thus the construction of a separator would include avoiding bad situations as above. In terms of trees, we think of some nodes on the tree that are identified as bad. The separating set should avoid these bad nodes.

Let S_p be a node just before a bad note S. The node S_p can be extended to the bad node S. There may be other bad nodes S_b that also are extended from this node S_p. In such a situation, if there is no way to prevent S_p from being extended to a bad node S_b, we can consider S_p also to be a bad node.

The separator now can be defined in terms of the bad nodes. The separator is the part of the tree that avoids all the bad nodes. These are not just the

nodes like S_b, but also nodes like S_p where you cannot avoid being extended into a bad node.

We can think of a practical example involving a cryptlocker attack. Suppose S_b is a situation where the disk is already locked by a cryptlocker attack. But before this situation, there is a previous situation S_p where the locking malware has been downloaded and is ready to be activated. In most situations, this S_p is already a bad situation where the crypt locking cannot be avoided. The separating set will have to avoid the situation S_q even prior to S_p where malware is about to be downloaded.

The incremental definition of bad nodes can be formalized. This will show that the separators we define are strategy patterns.

4 Separators are strategy patterns

A classical result credited to David Blackwell (sometimes described as a "father of modern statistics") states a result about analytic sets. The result is called separation property. For a casual statement about this theorem see `https://theoryclass.wordpress.com/2010/07/18/david-blackwell/`. Analytic sets arise in Mathematical Analysis related to measure theory.

We can get a similar result for sets of sequences related to hacker and normal behavior. We consider sequences where the hacker "wins" if he is able to force a system into a compromised state. Let us call such situations "bad". In the context of the tree of sequences, we can think of these as "bad nodes." Similarly we can think of a set of "good nodes" where we are safe from compromise. Let A be the set of sequences ending in good nodes and B the set of sequences ending in bad nodes.

Both A and B can be describe using existential quantifiers. $A = \{S|(\exists s \in S)(s = good)\}$. The set B is also defined similarly using bad nodes instead of good nodes. When thinking of these sequences, once we reach a good or bad node, we can think of the computation running on from that point for ever, as is typical in computer systems.

Given a set of n sequences, we can determine whether a sequence reaches a good or bad node using $log(n)$ memory. Therefore we refer to the sets A and B as being Σ_1^{log} sets. For convenience, we will drop the reference to the $log(n)$ memory and simply say that these sets are Σ_1.

We can prove a property called the "reduction property" for Σ_1 sets as considered here (i.e. actually Σ_1^{log} sets. This property states that A and B are Σ_1 sets, then there are two sets A' and B' such that $A \cup B = A' \cup B'$ and $A' \cap B' = \emptyset$ and $A' \subset A$ and $B' \subset B$. This property means that the two sets can be split such that the two split parts have nothing in common. Think of this as two fried eggs, sunny-side-up, but with the white parts joined together, the reduction would split the eggs into two parts where each part will contain one yolk.

This reduction property can be proved by adapting Blackwell's argument using games. We imagine a game, starting from an initial state, where the first player called Player 1 tries to get to a good node sooner than the second player, Player 2 gets to a bad node. Once either one gets to their nodes, they have won the game. But there can be nodes which are neither good nor bad where one of the players has a strategy to get to their type of node, as noted in the last section.

Let σ be a strategy. A strategy is one that extends the current state of play, or in a computer system it is a computational step that extends the current run of the computer. We can use the terminology $\sigma \models \phi$ to say "the strategy satisfies condition ϕ. Let $\gamma(S)$ mean that "S ends in a good node" and similarly let $\beta(S)$ mean "S ends in a bad node." Then we can define our reducing sets A' and B' using strategies. Let $A' = \{S|(\exists\sigma)\sigma \models \gamma(S)\}$. Similarly $B' = \{S|(\exists\sigma)\sigma \models \beta(S)\}$.

As mathematicians tend to say, we can "easily" verify that A' and B' reduce A and B i.e. they satisfy the conditions of the reduction property. Since A and B are computed using logarithmic space, it turns out we can computer A' and B' also using logarithmic space.

Now we can consider two sets C and D that are complements of A and B. That means C is the set of sequences that never reach a good node and D is the set of things that never reach a bad node. These are Π_1 sets. It turns out that Π_1 and Σ_1 sets are the same when dealing with things using logarithmic space. Then the complement of B' is a set that separates us from the strategies that lead to bad nodes.

We can state this another way, again using strategies. Player 2, the bad guy, has strategies to force the system into a bad, i.e. compromised positions. His strategy defines the set B'. If we separate the system from these strategies, then we are safe. To do this, we need to recognize the strategies used to define B'. In this case we can recognize these strategies using logarithmic space.

5 The strategies are actually more complex

So far we have analyzed the problem assuming that the hacker is persistently experimenting with the system. This is not really what happens. In fact hackers try to stay hidden most of the time. Thus their actions are not truly Π_1 but something more complex.

In reality most compromises involve behaviors that are "hyper persistent." What this means is that the hacker is always lurking in the shadows. He is not experimenting all the time, but he never gives up experimenting.

We can state this a different way, again using quantifiers. We can say that for any time t there is a time $t' > t$ such that the hacker performs an experiment at time t'. More formally $(\forall t)(\exists t')\beta(t')$ i.e. some bad node is reached at time t'.

Instead of the persistent threat being Π_1 now it is a more complex criterion Π_2. Accordingly, the strategies associated with these sets is more complex. In fact these strategies are quite a lot more complicated than the strategies for Π_1. These strategy patterns are strategies for games where each player plays strategies, thus these are "meta strategies." Identifying and separating meta strategies is the main task of our system.