Formalising attack trees to support economic analysis

Formalising Attack Trees to Support Economic Analysis

Andrew Simpson[1], Matthias Dellago[2] and Daniel Woods[2]

[1]*Department of Computer Science, University of Oxford*
United Kingdom
[2]*Institute of Computer Science, University of Innsbruck*
Austria
Email: Andrew.Simpson@cs.ox.ac.uk; Matthias.Dellago@student.uibk.ac.at;
Daniel.Woods@uibk.ac.at

Attack trees and attack graphs are both examples of what one might term *attack modelling techniques*. The primary purpose of such techniques is to help establish and enumerate the ways in which a system could be compromised; as such, they play a key role in the (security) risk analysis process. Given their role and the consequent need to ensure that they are correct, there are good reasons for capturing such artefacts in a formal manner. We describe such a formal approach, which has been motivated by a desire to model attacks from the perspectives of attackers, to support economic analysis. As an illustration, we consider exploitation cost.

Keywords: Attack Trees; Formal Modelling; Z; Information Security Economics

1. INTRODUCTION

Attack trees (see, for example, [1]) and attack graphs (see, for example, [2]) are both examples of what one might term *attack modelling techniques*. Schneier [3] describes the former in the following way.

> "Attack trees provide a methodical way of describing threats against, and countermeasures protecting a system. By extension, attack trees provide a methodical way of representing the security of systems." [3]

Such approaches help to establish and enumerate the ways in which a system's security may be compromised. They also identify potential routes to such compromises. Importantly, as both the costs and the probabilities of attacks can be taken into account [3], they can be used in support of cost–benefit analyses and, thus, inform investment decision-making.

Despite their perceived importance to the secure development process (attack trees, for example, are advocated in the SQUARE (Security Quality Requirements Engineering) process model[3] as part of the *Develop Artifacts* step), it is the case that such representations are often still captured manually [4].

Irrespective of the practical reality, a great deal of theoretical work has been undertaken on the formalisation and visualisation of attack modelling techniques. Despite their simplicity, there are good reasons for ensuring that such approaches have formal foundations. For example, Lallie et al. [5] argue that there are "inconsistencies regarding the way cyber-attacks are represented" in attack trees. Furthermore, Mauw and Oostdijk [6] provide the following motivation.

> "What is an attack? Is it just a collection of steps that should be performed or does it have some internal structure? Which conditions should an attribute satisfy before it allows to be synthesized bottom-up? Under what conditions may a projection of a predicate be executed bottom-up? When do two attack trees represent the same set of attacks? How should combined attributes be treated? And which extensions of the formalism (forests, directed acyclic graphs, attack graphs) are possible?" [6]

Ultimately, such formal representations can give greater assurance as to correctness and they can also help underpin automation (see, for example, the work of Vigo et al. [7], Ivanova et al. [8] and Pinchinat et al. [9]).

There has also been recent work on the incorporation of threat-related data (see, for example, the contribution of Cheah et al. [10]). Typically, the focus of such work as been on aiding organisations in giving consideration to costs to defend and to help determine cost measures such as the Return on Security Investment (ROSI) [11], a security-specific metric derived from the

[3] `https://us-cert.cisa.gov/bsi/articles/best-practices/requirements-engineering/square-process`

more familiar Return on Investment (ROI).

While we are also motivated by the need to assist those responsible for determining how and where to invest in security, our focus is slightly different. Specifically, we wish to consider an alternative perspective — that of the attacker (as considered, for example, by van Rensburg et al. [12]) — and consider costs (in terms of effort, as well as financial costs) from their perspective. Examples of such approaches are exemplified by *Cost To Break* (CTB) [13] and *Return on Attack* (ROA) [14]. The aim is to utilise attack modelling techniques to identify which defensive measures most effectively raise costs for the attacker, thereby disincentivising attacks.

In establishing the formal models to underpin such efforts, we leverage the schema language of Z [15] and the support that the ProB tool [16] offers for model-checking and animating Z descriptions in the form of ProZ [17]. (Subsequently, we have also developed models using the B-method [18], but that is not the focus of this paper.) We leverage Z to facilitate a state-based approach, whereby the states of the system under consideration are of relevance. To quote Audinot et al. [4], who also take such an approach: "The advantage of such a state-based approach is that it may benefit from verification and model checking techniques" [4]. We also utilise the Communicating Sequential Processes (CSP) process algebra [19, 20] to help justify the perspective taken (which involves representing trees as sequences / traces).

We start, in Section 2, by providing the background to and the motivation for the contribution described in this paper. Then, in Section 3, we present what might be thought of as the core concepts that underpin what follows. Section 4 builds upon those foundations and we subsequently present our state-based approach in Section 5. Then, in Sections 6 and 7 we demonstrate how the approach gives rise to a degree of modularity. Finally, we summarise the contribution of the paper in Section 8.

2. BACKGROUND AND MOTIVATION

It is well recognised that the question *How secure is this system?* can often be answered with a flippant response of "not 100%". The field of Information Security Economics (as established by the likes of Anderson and Moore [21, 22]) starts from this premise, giving rise to a focus on the development of theoretical solutions to help support the decision-making process via the use of appropriate models. This recognises the fact that cost–benefit analyses are at the heart of such security decision making, and also that (a) the threat landscape is constantly evolving and (b) resources are often limited and contested.

Within the Information Security Economics community, it is well understood that a gap exists between the theoretical models developed and the real-world data that might be used to parameterise them (see, for example, [23, 24, 25]). The *CEMDAT* (Combining Exploit Market Data with Attack Trees) project — as part of which the work described in this paper was developed — attempts to help close that gap by leveraging exploit pricing data from sources such as Zerodium[4]. (As an example, an analysis of longitudinal data of the prices quoted by an exploit broker who claims to sell to governments is discussed in [26] and [27].)

Rather than considering typical metrics such as the aforementioned Return on Security Investment (ROSI) [11] (which, in turn, is based upon the notion of Annual Loss Expectancy (ALE)), the focus of the CEMDAT project is the quantification of attacker costs: we wish to quantify how the cost to the adversary (who we might imagine intends to compromise a software-based system) might vary, based on existing capabilities. For example, exploits that require local device access are shown to be cheaper, whereas zero-click remote exploits are the most expensive exploits in the dataset [26].

Subsequently, these cost estimates can be fed into Return on Attack (ROA) models to support decision making. The conceit is that defenders can then invest in security solutions until the ROA is no longer profitable or feasible for the attacker. (Of course, it is perfectly possible that non-rational attackers will complicate this assumption — but the irrationality of human actors has always been, and always will be, with us [28].)

The project is driven by a desire to move beyond existing approaches to using exploit prices as a proxy for security (in terms of 'cost to compromise') by incorporating systems and economic considerations. The systems perspective recognises that the cost to the attacker depends on existing capabilities; the economic perspective can incorporate market forces such as demand and supply. Importantly, while the formal models discussed in this paper have been developed to underpin reasoning about costs to attackers, there is (as we shall see) nothing to prevent them being used in more traditional ways (i.e. in terms of costs to defend).

As our concern is the use of attack trees to model and reason about potential exploits, we revisit the concept in Section 2.1. We then, in Section 2.2, motivate the incorporation of attacker costs into such representations, before outlining the requirements for our formal approach in Section 2.3.

2.1. Attack trees: a brief introduction

Attack modelling techniques (AMTs) are defined by Lallie et al. [5] as means to "model and visualise the sequence and/or combination of events that enable a successful cyber-attack on a computer or network" [5]. The authors go on to suggest that AMTs can be distributed between three broad categories: use case methods (including, for example, misuse cases),

[4] `https://zerodium.com`

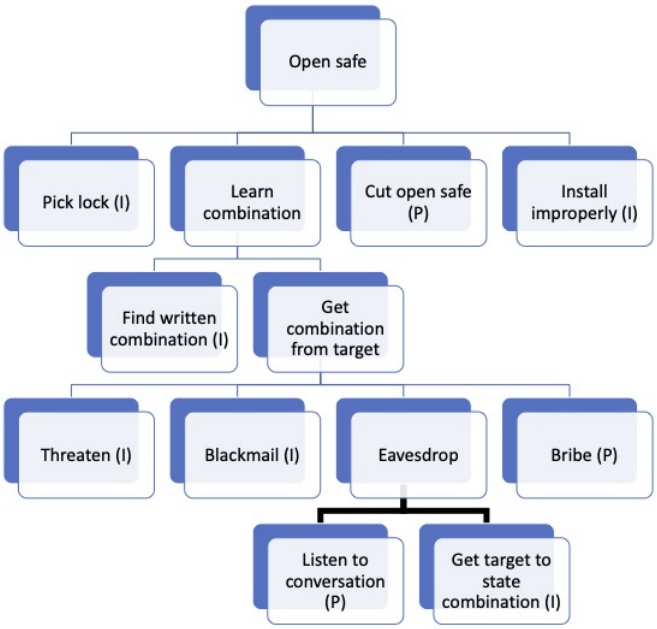

FIGURE 1. An example, drawn from [1].

temporal methods, and graph-based methods. The third of these — our concern in this paper — includes fault trees and attack graphs — with *attack trees* being our particular concern.

Attack trees, which build upon at least Weiss's notion of *threat logic trees* [29] and the contribution of Amoroso [30], are motivated by Schneier [1] thus:

> "If we can understand all the different ways in which a system can be attacked, we can likely design countermeasures to thwart those attacks. And if we can understand who the attackers are — not to mention their abilities, motivations, and goals — maybe we can install the proper countermeasures to deal with the real threats." [1]

It is worth noting that Kordy et al. [31] suggest that the general term *attack tree* subsumes a number of approaches including threat trees [32], threat logic trees [29], cyber threat trees [33], and fault trees for attack modeling [34]. As we move forward, we shall see that these distinctions are beyond the scope of our interests.

No matter which approach one adopts, the simplicity allows one to develop accessible and comprehensible models that can help one to understand and reason about a system and its vulnerabilities in a form that can be easily communicated. That form can subsequently be used to underpin decision making and, if appropriate, reused and / or refined. Attack trees have, to quote Mauw and Oostdijk [6], "proved to be an intuitive aid in threat analysis." The benefits of such approaches are summarised by Kordy et al. [31] thus:

> "The great advantage of graph-based approaches lies in combining user friendly, intuitive, visual features with formal semantics and algorithms that allow for qualitative and quantitative analyses." [31]

Attack trees are relatively simple constructs. Typically, the *nodes* in each such tree represent *attacks*, with the *root node* representing the attacker's *goal*. Each node's *child* represents what might be thought of as a refinement of the goal, with each *leaf* being representative of an attack that can be refined no further. Each refinement can be a conjunct (AND) or a disjunct (OR).

A simple example, based on an example from [1] (and one which is commonly found in the literature — with its familiarity motivating its use here), is illustrated in Figure 1. Here, the goal (and root node) is 'Open safe'. There are four refinements of the goal: 'Pick lock', 'Learn combination', 'Cut open safe' and 'Install improperly'. One of these ('Learn combination') is then refined further. Some of these refinements ('Pick lock', for example) are characterised as impossible — denoted by I; some others ('Cut open safe', for example) are characterised as possible — denoted by P.

The default assumption is that sub-goals on the same level are combined via OR. When sub-goals are combined via AND, it is stated explicitly — as in the case of 'Eavesdrop' being refined into 'Listen to conversation' AND 'Get target to state combination' — with, in this illustration, a bold connector denoting an AND combination.

It is, perhaps, worth noting how attributes such as I and P being associated with nodes hints at how straightforward it would be to associate other attributes

(such as costs and / or probabilities, for example) with nodes, as well as any constraints that might be of interest. As an example, "both I and P cannot both be associated with a single node" is an obvious constraint in this case.

It has been recognised [6, 5] that there are sometimes inconsistencies in both definitions and representation of the various aspects of attack trees and related approaches (which is, of course, one argument for the formalisation of such approaches). We choose to build upon (with some adaptations), the characterisation and terms of Mauw and Oostdijk [6][5]:

> "The purpose of an attack tree is to define and analyze possible attacks on a system in a structured way. This structure is expressed in the node hierarchy, allowing one to decompose an abstract attack or attack goal into a number of more concrete attacks or sub-goals ... An attack tree simply defines a collection of possible attacks ... Each attack consists of the components required to perform this attack. A component may occur more than once in an attack, so an attack is a multi-set of attack components. These attack components are at the lowest level of abstraction that we consider and thus have no internal structure."

At the heart of our approach, then, we will have *components*, *trees* and *forests* (collections of trees). The model's primary deviation from the approach of [6] will be to choose to consider order and assume that components occur no more than once in an attack, meaning that attacks are associated with injective sequences of components, rather than multi-sets (or bags) of components. It is worth noting that, if necessary, adapting the approach from injective sequences to sequences, sets or multi-sets would be a relatively straightforward process.

2.2. CEMDAT: Combining Exploit Market Data with Attack Trees

As previously discussed, the work described in this paper has been undertaken within the Combining Exploit Market Data with Attack Trees (CEMDAT) project[6], which is a collaboration between researchers based at the Universities of Innsbruck and Oxford. The project's key deliverable is an economic framework to reason about the value of existing capabilities and the mediating role of demand and supply. A key part of this framework is a formal model that captures characteristics of attacks. The development of this formal model is the focus of this paper.

A variety of 'costs' can be associated with attack trees. Typically, as we have discussed, these costs are associated with potential investments that can be made by the organisation. However, some researchers have considered things from the perspectives of both attackers and those investing against attacks. As an example, Bistarelli et al. [35] combine defense trees ("an extension of attack trees with countermeasures") and game theory, and consider both ROI and ROA.

This consideration of attacks from both perspectives, i.e. that of the attacker and that of the victim, is an area that has received a degree of consideration in recent years. For example, in [36], Roy and colleagues present what they term an *attack countermeasure tree* (ACT), which "takes into account attacks as well as countermeasures (in the form of detection and mitigation events)". It is worth recognising that Kordy et al. [31] provide an excellent survey of both "attack and defense modeling approaches that are based on directed acyclic graphs (DAGs)" and "propose a taxonomy of the described formalisms". Both have influenced what follows.

Other influences on the following include: recent work on formalising attack and defense trees, with the work of Cheah et al. [10] being an example in this respect; a need to recognise current and emerging commercial needs [37][7]; and a reflection that, as the lines between safety and security are starting to become increasingly blurred in some contexts [38, 39], having an eye to such concerns would be wise in terms of increasing the chances of wider applicability. A relevant example in this respect is the contribution of Kriaa et al. [40], which, in consideration of "the convergence of safety and security concerns," provides "a comprehensive survey of existing approaches to industrial facility design and risk assessment that consider both safety and security."

More broadly, it should be recognised that there is already a wide body of literature related to the amalgamation of attack (and defense) trees with economic concepts. Examples in this respect include the contributions of Niitsoo [41], Dewri and colleagues [42, 43], Buldas and colleagues [44, 45], van Holsteijn [46], Fila and Widel [47], and Patten et al. [48].

2.3. Requirements for a formal approach

The requirements that underpin our formal approach are derived from the concerns stated above and are relatively straightforward. They are, nevertheless, worth stating explicitly.

1. The approach should broadly follow that of [6] and give consideration to *attacks*, collections of attack

[5] It is worth noting that Mauw and Oostdijk argue that, "formally speaking, we study *rooted directed acyclic bundle graphs*, but we will still call them *attack trees*." [6]

[6] `https://informationsecurity.uibk.ac.at/projects/cemdat/`

[7] Gadyatskaya and Trujillo-Rasua [37] suggest that "several promising research directions have emerged recently to address the needs of practitioners": attack tree generation; attack tree visualisation; and empirical studies involving attack trees.

components (in the form of *trees*), and collections of trees (in the form of *forests*).
2. The approach should give rise to machine-readable descriptions to (a) increase confidence in correctness and (b) support automation.
3. The approach should be suitably modular to allow for the incorporation of attributes such as possibility / impossibility, costs and probabilities (and combinations thereof).
4. The approach should allow for suitable economic analysis in terms of both ROA and ROI.

Having provided the motivation for our contribution, we are in a position to start to present our model.

3. FROM TREES TO TRACES

In this section we present what we might think of as the core concepts that underpin what follows. We first justify our approach of representing the information contained in trees in terms of sequences. We leverage the language of Communicating Sequential Processes (CSP) [19, 20] to do so.

It should be noted that our use of CSP is primarily motivational and sets the scene for what follows. Previous contributions that have leveraged CSP in thinking about attack trees and related approaches include those of Nguyen and colleagues [10, 49].

At the heart of our core model is the conceit that each tree can be modelled simply as a collection of sequences.

Consider the following example, which involves four potential attacks and, thus, four attack trees. We use the notation of CSP to illustrate a variety of patterns.

The first such potential attack is straightforward:

$$A \mathrel{\widehat{=}} a_1 \rightarrow a_2 \rightarrow attackA \rightarrow Skip$$

Here, a_1 and a_2 are necessary (in that order) before *attackA* can happen. This process has four possible traces ($\langle\rangle$, $\langle a_1\rangle$, $\langle a_1, a_2\rangle$, and $\langle a_1, a_2, attackA\rangle$), one of which ($\langle a_1, a_2, attackA\rangle$) concludes with *attackA* and is, therefore, the trace of interest to us.

The second involves a characterisation of AND via CSP's partial interleaving operator:

$$\begin{aligned} B \mathrel{\widehat{=}}\ & (b_1 \rightarrow attackB \rightarrow Skip) \\ & \underset{\{attackB\}}{\parallel} \\ & (b_2 \rightarrow attackB \rightarrow Skip) \end{aligned}$$

Here, b_1 and b_2 are both necessary (in either order) before the synchronised event *attackB* can happen. This process has the following possible traces: $\langle\rangle$, $\langle b_1\rangle$, $\langle b_2\rangle$, $\langle b_1, b_2\rangle$, $\langle b_2, b_1\rangle$, $\langle b_1, b_2, attackB\rangle$, and $\langle b_2, b_1, attackB\rangle$. In this case, there are two traces of interest: $\langle b_1, b_2, attackB\rangle$ and $\langle b_2, b_1, attackB\rangle$.

The third involves OR, modelled via CSP's external choice operator, $\Box$:

$$\begin{aligned} C \mathrel{\widehat{=}}\ & c_1 \rightarrow attackC \rightarrow Skip \\ & \Box \\ & c_2 \rightarrow attackC \rightarrow Skip \end{aligned}$$

Here, either c_1 or c_2 needs to happen prior to *attackC*. The possible traces of this process are $\langle\rangle$, $\langle c_1\rangle$, $\langle c_2\rangle$, $\langle c_1, attackC\rangle$, and $\langle c_2, attackC\rangle$. In this case, there are again two traces that end in an attack: $\langle c_1, attackC\rangle$ and $\langle c_2, attackC\rangle$.

The fourth pattern involves two independent paths to exploitation (and which, therefore, would be captured by separate trees), which we model via CSP's interleaving operator, $|||$:

$$D \mathrel{\widehat{=}} D1 \ ||| \ D2$$

Here,

$$D1 \mathrel{\widehat{=}} d_1 \rightarrow d_2 \rightarrow attackD \rightarrow Skip$$

and

$$D2 \mathrel{\widehat{=}} d_3 \rightarrow d_4 \rightarrow attackD \rightarrow Skip$$

Here, we have two trees associated with a single attack, *attackD*. In this case the traces of interest are $\langle d_1, d_2, attackD\rangle$ and $\langle d_3, d_4, attackD\rangle$.

In combining the above, we can think of our *forest* as consisting of these four processes, combined via interleaving:

$$Forest1 \mathrel{\widehat{=}} A \ ||| \ B \ ||| \ C \ ||| \ D$$

The above is sufficient to illustrate the four pattens of interest: a simple sequence, conjunction, disjunction, and multiple independent paths. (Clearly, combinations of the above are possible and can be modelled accordingly.)

It is also worth recognising that we assume that trees (and attacks) are modelled and evolve independently: hence the use of interleaving.

As an illustrative example, we consider again the example of [1], which was presented in Section 2.1.

Initially we have the following.

$$\begin{aligned} & TreeV1 \mathrel{\widehat{=}} \\ & \quad Pick\_lock \rightarrow Open\_safe \rightarrow Skip \\ & \quad \Box \\ & \quad Learn\_comb \rightarrow Open\_safe \rightarrow Skip \\ & \quad \Box \\ & \quad Cut\_open\_safe \rightarrow Open\_safe \rightarrow Skip \\ & \quad \Box \\ & \quad Install\_improperly \rightarrow Open\_safe \rightarrow Skip \end{aligned}$$

Here, there are four refinements of the goal *Open_safe*: *Pick_lock*, *Learn_comb*, *Cut_open_safe* and *Install_improperly*. That is to say, there are four ways by which the safe can be opened (according to the tree).

Recalling that, as per Figure 1, 'Learn combination' is refined[8] into 'Find written combination' (followed by 'Learn combination') and 'Get combination from

[8] Albeit not in the CSP sense of refinement.

target' (again followed by 'Learn combination'), we can consider the next level via the process $TreeV2$:

$$
\begin{array}{l}
TreeV2 \mathrel{\widehat{=}} \\
\quad Pick\_lock \rightarrow Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad (Find\_written\_comb \rightarrow \\
\qquad Learn\_comb \rightarrow \\
\qquad\quad Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad Get\_comb\_from\_target \rightarrow \\
\qquad Learn\_comb \rightarrow \\
\qquad\quad Open\_safe \rightarrow Skip) \\
\quad \Box \\
\quad Cut\_open\_safe \rightarrow Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad Install\_improperly \rightarrow Open\_safe \rightarrow Skip
\end{array}
$$

Here, three of the four original refinements of the goal are unchanged. The other now captures the notion that a combination can be discovered either via *Find_written_comb* or via *Get_comb_from_target*.

Moving forward, 'Get combination from target' is refined into 'Threaten', 'Blackmail', 'Eavesdrop' and 'Bribe'. The process $TreeV3$ captures this: each of the corresponding events can lead to the event *Get_comb_from_target* being observed.

$$
\begin{array}{l}
TreeV3 \mathrel{\widehat{=}} \\
\quad Pick\_lock \rightarrow Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad (Find\_written\_comb \rightarrow \\
\qquad Learn\_comb \rightarrow \\
\qquad\quad Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad\ (Threaten \rightarrow TreeV3a \\
\quad\ \Box \\
\quad\ Blackmail \rightarrow TreeV3a \\
\quad\ \Box \\
\quad\ Eavesdrop \rightarrow TreeV3a \\
\quad\ \Box \\
\quad\ Bribe \rightarrow TreeV3a)) \\
\quad \Box \\
\quad Cut\_open\_safe \rightarrow Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad Install\_improperly \rightarrow Open\_safe \rightarrow Skip
\end{array}
$$

Here, $TreeV3a$ is defined as follows.

$$
\begin{array}{l}
TreeV3a \mathrel{\widehat{=}} \\
\quad Get\_comb\_from\_target \rightarrow \\
\qquad Learn\_comb \rightarrow \\
\qquad\quad Open\_safe \rightarrow Skip
\end{array}
$$

Finally, 'Eavesdrop' is composed (via AND) of 'Listen to conversation' and 'Get target to state combination'. As such, we have the following.

$$
\begin{array}{l}
TreeV4 \mathrel{\widehat{=}} \\
\quad Pick\_lock \rightarrow Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad (Find\_written\_comb \rightarrow \\
\qquad Learn\_comb \rightarrow \\
\qquad\quad Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad\ (Threaten \rightarrow TreeV3a \\
\quad\ \Box \\
\quad\ Blackmail \rightarrow TreeV3a \\
\quad\ \Box \\
\quad\ ((Listen\_to\_conversation \rightarrow \\
\qquad\quad Eavesdrop \rightarrow Skip) \\
\qquad \|_{\{Eavesdrop\}} \\
\quad\ (Get\_target\_to\_state\_comb \rightarrow \\
\qquad\quad Eavesdrop \rightarrow Skip))_{\S} \\
\qquad TreeV3a \\
\quad\ \Box \\
\quad\ Bribe \rightarrow TreeV3a)) \\
\quad \Box \\
\quad Cut\_open\_safe \rightarrow Open\_safe \rightarrow Skip \\
\quad \Box \\
\quad Install\_improperly \rightarrow Open\_safe \rightarrow Skip
\end{array}
$$

The possible traces of the process are given in Table 1.

There are a couple of things to note here. First, consider sequences 5 and 6 of Table 1. In the original attack tree, the sub-goals 'Listen to conversation' and 'Get target to state combination' were combined via conjunction (AND). Allowing the events to appear in either order is consistent with the fact that AND is commutative. However, in this particular case, only one of the two possible orderings ('Listen to conversation', then 'Get target to state combination') would make sense in the real world.

Second, those versed in CSP or related techniques will recognise a potential issue with our representation. Let us imagine, for example, that an attacker tried to pursue two paths in parallel — perhaps bribing one employee while threatening a second and blackmailing a third. In such circumstances, $TreeV4$ could give rise to a deadlock. The reason this is not a concern here is that these processes are intended to motivate how we might use sequences to represent *possible* paths to an attack; the intent is not that the processes represent an 'observation' of an attacker's behaviour in real time.

Returning to $TreeV4$, we can now consider the notions of possibility and impossibility as captured in the original representation. If we 'hide' the impossible events as depicted in Figure 1, i.e. if we consider the process

$$
\begin{array}{l}
TreeV4 \setminus \\
\quad \{\ Pick\_lock, Install\_improperly, \\
\quad\ Find\_written\_comb, \\
\quad\ Threaten, Blackmail, \\
\quad\ Get\_target\_to\_state\_comb\}
\end{array}
$$

1	$\langle Pick\_lock, Open\_safe\rangle$
2	$\langle Find\_written\_comb, Learn\_comb, Open\_safe\rangle$
3	$\langle Threaten, Get\_comb\_from\_target, Learn\_comb, Open\_safe\rangle$
4	$\langle Blackmail, Get\_comb\_from\_target, Learn\_comb, Open\_safe\rangle$
5	$\langle Listen\_to\_conversation, Get\_target\_to\_state\_comb,$ $Eavesdrop, Get\_comb\_from\_target, Learn\_comb, Open\_safe\rangle$
6	$\langle Get\_target\_to\_state\_comb, Listen\_to\_conversation,$ $Eavesdrop, Get\_comb\_from\_target, Learn\_comb, Open\_safe\rangle$
7	$\langle Bribe, Get\_comb\_from\_target, Learn\_comb, Open\_safe\rangle$
8	$\langle Cut\_open\_safe, Open\_safe\rangle$
9	$\langle Install\_improperly, Open\_safe\rangle$

TABLE 1. Possible traces of interest of $TreeV4$.

then we are left with traces 7 and 8 of Table 1, giving rise to the following set:

$$\{\ \langle Bribe, Get\_comb\_from\_target, Learn\_comb, Open\_safe\rangle, \langle Cut\_open\_safe, Open\_safe\rangle\}$$

With regards to this specific example, this tells us that we need to give consideration to two potential sequences (or traces) in defending our system. More generally, the above tells us that we can abstract away from trees and think in terms of traces (or sequences) in constructing our model — while recognising that a starting point based on tree-like structures is beneficial in terms of initial capture of information, and in terms of facilitating communication and understanding.

4. THE FUNDAMENTALS OF THE MODEL

Having motivated our approach, we are now in a position to present the fundamental concepts of our model in terms of the mathematical language of Z [15, 50].

We start by introducing the basic type, *Attack*:

$$[Attack]$$

Borrowing the language of [6], components are the building blocks of attacks. Thus, a set of *attack components* is captured by the basic type *Component*:

$$[Component]$$

As an example, based on an illustrative example of [6], we might have

$$open\_door \in Attack$$

and

$$\{steal\_key, force\_lock, pick\_lock\} \subseteq Component$$

In the case of our example of Section 3, we would have

$$\{attackA, attackB, attackC, attackD\} \subseteq Attack$$

and

$$\{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2, d_3, d_4\} \subseteq Component$$

In the case of our example of Section 2.1, we would have

$$Open\_safe \in Attack$$

and

$$\{Pick\_lock, Cut\_open\_safe, \ldots\} \subseteq Component$$

A *path* can lead to an attack. We define *Path* — a collection of non-empty, injective sequences of components — thus:

$$Path == \mathrm{iseq}_1\ Component$$

Continuing with our illustrative example borrowed from [6], we might have

$$\{\langle steal\_key\rangle, \langle force\_lock\rangle, \langle pick\_lock\rangle\} \subseteq Path$$

As another example, recalling our first example from Section 3, we would have

$$\{\langle a_1, a_2\rangle, \langle b_1, b_2\rangle, \ldots\} \subseteq Path$$

In the case of our second example of Section 3, we would have

$$\{\ \langle Pick\_lock\rangle, \langle Find\_written\_comb, Learn\_comb\rangle, \vdots \}\subseteq Path$$

A *tree* is defined as a pair, in which the first element is an attack and the second element is a (possibly empty) finite set of paths:

$$Tree == Attack \times \mathbb{F}\ Path$$

Continuing with our simple example, we might have

$$(open\_door, \{\langle steal\_key\rangle, \langle force\_lock\rangle, \langle pick\_lock\rangle\}) \in Tree$$

For our other scenarios, we might have

$$(attackB, \{\langle b_1, b_2\rangle, \langle b_2, b_1\rangle\}) \in Tree$$

and (if we limit ourselves — for reasons of space, if nothing else — to possible paths)

$$\begin{array}{l}(Open\_safe,\\ \quad \{\langle Bribe,\\ \quad\quad Get\_comb\_from\_target, Learn\_comb\rangle,\\ \quad \langle Cut\_open\_safe\rangle\})\\ \in Tree\end{array}$$

Finally, we define a *forest* as a (possibly empty) finite set of trees:

$$Forest == \mathbb{F}\ Tree$$

For our simple running example based on [6] we might have

$$\begin{array}{l}\{(open\_door,\\ \quad \{\langle steal\_key\rangle, \langle force\_lock\rangle, \langle pick\_lock\rangle\}),\\ (smash\_down\_door, \emptyset)\}\\ \subseteq Forest\end{array}$$

Recalling our other examples we would have the following for the first scenario

$$\begin{array}{l}\{(AttackA, \{\langle a_1, a_2\rangle\}),\\ (AttackB, \{\langle b_1, b_2\rangle, \langle b_2, b_1\rangle\}),\\ (AttackC, \{\langle c_1\rangle, \langle c_2\rangle\}),\\ (AttackD, \{\langle d_1, d_2\rangle, \langle d_3, d_4\rangle\})\} \subseteq Forest\end{array}$$

and, for the second scenario, we would have

$$\begin{array}{l}\{(Open\_safe,\\ \quad \{\ \langle Bribe,\\ \quad\quad Get\_comb\_from\_target,\\ \quad\quad\quad Learn\_comb\rangle,\\ \quad\ \langle Cut\_open\_safe\rangle\})\} \subseteq Forest\end{array}$$

Having motivated our use of sets and sequences to capture attack trees, we now turn our attention to our state-based model, which we present in terms of the schema language of Z.

5. A STATE-BASED APPROACH

In the previous section we illustrated the core model, consisting of attacks, components, paths, trees and forests. We have developed a state-based model to capture the operations of creating trees, adding them to forests, and so on — and, most importantly, to enforce the checking of constraints to give rise to the assurance of correctness that we seek. We present that state-based model in the following.

5.1. State and initialisation

We start with the state schema, *Core*.

Core

$$components : \mathbb{F}\ Component$$
$$attacks : \mathbb{F}\ Attack$$
$$paths : \mathbb{F}\ Path$$
$$trees : \mathbb{F}\ Tree$$
$$forest : Forest$$

$$\forall p : paths \bullet \mathrm{ran}\ p \subseteq components$$
$$\mathrm{dom}\ trees \subseteq attacks$$
$$\forall a : \mathrm{dom}\ trees \bullet \forall b : \mathrm{dom}\ trees \bullet a = b \Rightarrow trees\ a = trees\ b$$
$$\forall P : \mathrm{ran}\ trees \bullet P \subseteq paths$$
$$\forall t : forest \bullet t \in trees$$

Here, *components*, etc. capture the 'current' values of the key attributes of interest. There are four (finite) sets: *components*, *attacks*, *paths* and *trees*. Finally, *forest* represents the overarching forest of interest.

The first constraint ensures that all elements featuring in *paths* also appear in *components*. The second constraint ensures that all elements featuring in the domain of *trees* should be in *attacks* (recall that a tree is modelled as a pair, consisting of an element of *Attack* and a finite set of elements of type *Path*). The third constraint ensures that no attack can appear in *trees* more than once, i.e. there is at most one collection of paths associated with each attack. The fourth constraint ensures that all elements featuring in the range of *trees* should be in *paths*. Finally, the fifth constraint ensures that only elements of *trees* can appear in *forest*.

The initial state of the model is given thus.

Init

$$Core$$

$$components = \emptyset$$
$$attacks = \emptyset$$
$$paths = \emptyset$$
$$trees = \emptyset$$
$$forest = \emptyset$$

(The format of the initial state — in terms of a before-state, rather than in terms of an after-state — is influenced by the use of the ProZ model-checker [17], whereby the initial state is given in terms of a before-state, rather than in terms of an after-state.)

5.2. Components

Operations to add and remove components are defined as follows.

First, we might wish to consider the addition of a component (such as *Pick_lock*, for example). The operation schema *AddComponent* captures such an action.

AddComponent

$\Delta Core$
$c? : Component$

$c? \notin components$
$components' = components \cup \{c?\}$
$attacks' = attacks$
$paths' = paths$
$trees' = trees$
$forest' = forest$

Here, the input $c? \in Component$ is added to *components* provided that it does not already appear in that set.

The corresponding operation to remove a component is defined as follows.

RemoveComponent

$\Delta Core$
$c? : Component$

$c? \in components$
$\forall p : paths \bullet c? \notin \text{ran } p$
$components' = components \setminus \{c?\}$
$attacks' = attacks$
$paths' = paths$
$trees' = trees$
$forest' = forest$

There are two preconditions that must be satisfied. First, the component to be removed ($c?$) must be an element of *components*. Second, it must not appear in any elements of *paths*. (Recall that any component that appears in *paths* must also appear in *components*.)

5.3. Attacks

Next, we consider the addition of an attack (such as *Open_safe*, for example).

AddAttack

$\Delta Core$
$a? : Attack$

$a? \notin attacks$
$components' = components$
$attacks' = attacks \cup \{a?\}$
$paths' = paths$
$trees' = trees$
$forest' = forest$

The corresponding operation to remove an attack is defined as follows.

RemoveAttack

$\Delta Core$
$a? : Attack$

$a? \in attacks$
$a? \notin \text{dom } trees$
$components' = components$
$attacks' = attacks \setminus \{a?\}$
$paths' = paths$
$trees' = trees$
$forest' = forest$

Again, there are two constraints that must be satisfied. First, the attack to be removed ($a?$) must be an element of *attacks*. Second, it must not appear in *trees*. (Recall that any attack that appears in a tree must also appear in *attacks*.)

5.4. Paths

We now consider the addition of paths.

CreatePath

$\Delta Core$
$p? : Path$

$p? \notin paths$
$\text{ran } p? \subseteq components$
$components' = components$
$attacks' = attacks$
$paths' = paths \cup \{p?\}$
$trees' = trees$
$forest' = forest$

This operation has two constraints. The first is a standard one, as per the previous addition operations: there is a check that a path that doesn't already exist in the set to which it is being added. The second ensures that the elements that appear in the path also appear in *components*. (A sequence in Z is modelled as a function; hence the use of ran. As an example, $\text{ran } \langle a, b, c \rangle = \{a, b, c\}$.)

The corresponding operation to remove paths is defined thus.

RemovePath

$\Delta Core$
$p? : Path$

$p? \in paths$
$\forall P : \text{ran } trees \bullet p? \notin P$
$components' = components$
$attacks' = attacks$
$paths' = paths \setminus \{p?\}$
$trees' = trees$
$forest' = forest$

Again, there are two constraints that must be satisfied. First, the path to be removed ($p?$) must be an

element of *paths*. Second, it must not appear in *trees*. (Recall that any path that appears in a tree must also appear in *paths*.)

5.5. Trees

We now turn our attention to trees. The operation *CreateTree* models the creation of a tree. Recall that a tree is modelled as an ordered pair: in position 1 is an attack and in position 2 is a (finite) set of paths. Any attack, $a \in Attack$, which appears in *attacks* and does not appear in the domain of *trees* can be added to *trees* — and, consequently, associated initially with the empty set of paths.

CreateTree

$\Delta Core$
$a? : Attack$

$a? \in attacks$
$a? \notin \mathrm{dom}\ trees$
$components' = components$
$attacks' = attacks$
$paths' = paths$
$trees' = trees \cup \{a? \mapsto \emptyset\}$
$forest' = forest$

We model the subsequent addition of a path to a tree via the operation *AddPathToTree*. The operation has two inputs: an attack ($a?$) and a path ($p?$). There are three preconditions: $a?$ must appear in (the domain of) *trees*; $p?$ must appear in *paths*; and $p?$ can't already be associated with $a?$. If these preconditions are met, then $p?$ is added to the paths associated with $a?$ (modelled via the override operator, $\oplus$).

AddPathToTree

$\Delta Core$
$a? : Attack$
$p? : Path$

$a? \in \mathrm{dom}\ trees$
$p? \in paths$
$p? \notin trees\ a?$
$components' = components$
$attacks' = attacks$
$paths' = paths$
$trees' = trees \oplus \{a? \mapsto (trees\ a?) \cup \{p?\}\}$
$forest' = forest$

The complementary operation, pertaining to the removal of a path, is defined similarly. Again, there are three conditions: $a?$ must appear in (the domain of) *trees*; $p?$ must be associated with $a?$; and $a?$ can't appear in the forest being modelled. If these preconditions are met, then $p?$ is removed from the paths associated with $a?$ (again modelled via the override operator, $\oplus$).

RemovePathFromTree

$\Delta Core$
$a? : Attack$
$p? : Path$

$a? \in \mathrm{dom}\ trees$
$p? \in trees\ a?$
$a? \notin \mathrm{dom}\ forest$
$components' = components$
$attacks' = attacks$
$paths' = paths$
$trees' = trees \oplus \{a? \mapsto (trees\ a?) \setminus \{p?\}\}$
$forest' = forest$

Finally, the removal of a tree is defined as follows.

RemoveTree

$\Delta Core$
$a? : Attack$

$a? \in \mathrm{dom}\ trees$
$a? \notin \mathrm{dom}\ forest$
$components' = components$
$attacks' = attacks$
$paths' = paths$
$trees' = \{a?\} \lhd trees$
$forest' = forest$

Here, the operation *RemoveTree* has one input: $a? \in Attack$. Assuming that $a?$ is an element of (the domain of) *trees* and does not appear in (the domain of) *forest*, then the tree associated with $a?$ is removed from *trees* (via the domain co-restriction operator, $\lhd$).

5.6. The forest

We are now in a position to give consideration to the *forest* attribute.

A tree, $t?$, can be added to the forest if it appears in *trees* and does not already appear in *forest*:

AddTreeToForest

$\Delta Core$
$t? : Tree$

$t? \in trees$
$t? \notin forest$
$components' = components$
$attacks' = attacks$
$paths' = paths$
$trees' = trees$
$forest' = forest \cup \{t?\}$

A tree that already appears in $t?$ can be removed from the forest.

RemoveTreeFromForest

$\Delta Core$
$t? : Tree$

$t? \in forest$
$components' = components$
$attacks' = attacks$
$paths' = paths$
$trees' = trees$
$forest' = forest \setminus \{t?\}$

5.7. Leveraging the model

We can now consider how we might use the model to learn about the forest of interest. As an example, the *PathsToAttack* operation schema takes an attack as input and returns the paths associated with that attack (if there are any):

PathsToAttack

$\Xi Core$
$a? : Attack$
$P! : \mathbb{F}\ Path$

$a? \in \text{dom}\ forest \Rightarrow P! = forest\ a?$
$a? \notin \text{dom}\ forest \Rightarrow P! = \emptyset$

As an example, if the value of *forest* was given by

$$\{(Open\_safe, \{\ \langle Bribe, Get\_comb\_from\_target, Learn\_comb\rangle, \langle Cut\_open\_safe\rangle, \langle Install\_improperly\rangle\})\}$$

then, if *Open_safe* was passed as input to the *PathsToAttack* operation, it would follow that the output would be

$$\{\langle Bribe, Get\_comb\_from\_target, Learn\_comb\rangle, \langle Cut\_open\_safe\rangle, \langle Install\_improperly\rangle\}$$

6. DEMONSTRATING MODULARITY

In Section 2 we identified four requirements for our approach. The first of these requirements

> *The approach should broadly follow that of [6] and give consideration to* attacks*, collections of attack components (in the form of* trees*), and collections of trees (in the form of* forests*)*

is met in a straightforward fashion. The second of these requirements

> *The approach should give rise to machine-readable descriptions to (a) increase confidence in correctness and (b) support automation*

is met by the fact that the LaTeX representation of the model can be analysed and automated via tools as such as the aforementioned ProZ (as, indeed, it has been).

In this section we give consideration to the third requirement:

> *The approach should be suitably modular to allow for the incorporation of attributes such as possibility / impossibility, costs and probabilities (and combinations thereof).*

Perhaps the most straightforward way of demonstrating this is via the characteristics of possibility and impossibility as considered in Section 2. (We shall consider costs in the next section.)

We start by introducing the free type *PossibilityValue*:

$$PossibilityValue ::= I \mid P$$

We next introduce the schema *Possibilities*:

Possibilities

$components : \mathbb{F}\ Component$
$poss : Component \pfun PossibilityValue$

$\text{dom}\ poss = components$

The two attributes are linked by the single constraint: all elements of *components* must have an associated possibility, captured via the *poss* function (and, indeed, only elements of *components* can have an associated possibility).

We can subsequently combine this aspect with our core model via schema conjunction:

$$CoreWithPossibility \mathrel{\widehat=} Core \land Possibilities$$

This gives rise to a state schema involving the attributes and constraints (combined via conjunction) of both *Core* and *Possibilities*.

Operations to make components possible and impossible are defined thus.

MakeComponentPossible

$\Delta CoreWithPossibility$
$\Xi Core$
$c? : Component$

$c? \in components$
$poss' = poss \oplus \{c? \mapsto P\}$

MakeComponentImpossible

$\Delta CoreWithPossibility$
$\Xi Core$
$c? : Component$

$c? \in components$
$poss' = poss \oplus \{c? \mapsto I\}$

The use of $\Xi Core$ ensures that the attributes of *Core* remain unchanged by these operations.

Most of our operations of Sections 5.2–5.6 are adapted by changing $\Delta Core$ to $\Delta CoreWithPossibility$ and the incorporation of an extra constraint ($poss' = poss$) to ensure that aspect remains unchanged. So, for example, we might have

RemoveTreeFromForest

$$\Delta CoreWithPossibility$$
$$t? : Tree$$

$$t? \in forest$$
$$components' = components$$
$$attacks' = attacks$$
$$paths' = paths$$
$$trees' = trees$$
$$forest' = forest \setminus \{t?\}$$
$$poss' = poss$$

On the other hand, those operations of Sections 5.2–5.6 that update the *components* attribute — such as *RemoveComponent* — now must take into account *poss* due to the fact that the two attributes are linked via the dom $poss = components$ constraint.

As an example, an updated version of *RemoveComponent* is given below.

RemoveComponent

$$\Delta CoreWithPossibility$$
$$c? : Component$$

$$c? \in components$$
$$\forall p : paths \bullet c? \notin \operatorname{ran} p$$
$$components' = components \setminus \{c?\}$$
$$attacks' = attacks$$
$$paths' = paths$$
$$trees' = trees$$
$$forest' = forest$$
$$poss' = \{c?\} \lhd poss$$

The only change is the addition of the constraint $poss' = \{c?\} \lhd poss$. This insists that the removed component $c?$ is also removed (via domain subtraction) from the *poss* function — to ensure the preservation of the invariant dom $poss = components$.

Finally, in addition to *PathsToAttack*, we might introduce *PossiblePathsToAttack*, the output of which rules out impossible paths:

PossiblePathsToAttack

$$\Xi CoreWithPossibility$$
$$a? : Attack$$
$$P! : \mathbb{F}\ Path$$

$$a? \in \operatorname{dom} forest \Rightarrow$$
$$\quad P! = \{p : forest\,(a?) \mid$$
$$\qquad \forall c : \operatorname{ran} p \bullet poss\ c = P)\}$$
$$a? \notin \operatorname{dom} forest \Rightarrow P! = \emptyset$$

Here, the set comprehension insists that only paths in which all components are associated with P (i.e. deemed possible) will feature in the output.

As an example, if the value of *forest* was given by

$$\{(Open\_safe,$$
$$\quad \{\langle Bribe,$$
$$\qquad Get\_comb\_from\_target,$$
$$\qquad\quad Learn\_comb\rangle,$$
$$\quad \langle Cut\_open\_safe\rangle,$$
$$\quad \langle Install\_improperly\rangle\})\}$$

and, as per Figure 1, *Install_improperly* was associated with I, then, if *Open_safe* was passed as input to *PossiblePathsToAttack*, it would follow that the output would be

$$\{\langle Bribe,$$
$$\quad Get\_comb\_from\_target, Learn\_comb\rangle,$$
$$\langle Cut\_open\_safe\rangle\}$$

7. INCORPORATING COSTS

Having considered the first three of the four requirements of Section 2.3, we now turn our attention to the fourth requirement:

> *The approach should allow for suitable economic analysis in terms of both ROA and ROI.*

We first need to introduce a notion of value. For the sakes of convenience and simplicity, we define *Cost* as an abbreviation of the natural numbers:

$$Cost == \mathbb{N}$$

Here, *Cost* could be associated with necessary effort (in terms of time), monetary cost, some other value or some combination thereof. It could, as is typical, pertain to cost to defend; alternatively, it could, as per our motivation, pertain to cost to attack. (A simple extension — involving replacing $\mathbb{N}$ with a tuple to capture a combination of different costs — is, of course, possible.) In the following, we shall consider 'cost to exploit'.

As per the approach taken in the previous section, we start by introducing a new state schema to capture the aspects of interest.

Costs

$$components : \mathbb{F}\ Component$$
$$paths : \mathbb{F}\ Path$$
$$costEC : Component \pfun Cost$$
$$costEP : Path \pfun Cost$$

$$\forall p : paths \bullet \operatorname{ran} p \subseteq components$$
$$\operatorname{dom}(costEC) = components$$
$$\operatorname{dom}(costEP) = paths$$
$$\forall p : \operatorname{dom}(costEP) \bullet$$
$$\quad (\forall c : \operatorname{ran}(p) \bullet costEP\ p \geq costEC\ c)$$

Here, *components* and *paths* are familiar from the *Core* schema. The function *costEC* (for 'cost to exploit component') associates costs with components. (We note that it is not the role of the model to estimate or determine costs of components, but, rather, to capture such information.) The function *costEP* (for 'cost to exploit path') associates costs with paths.

The first constraint is again familiar from the *Core* schema. The other constraints ensure that (a) every element of *components* has an associated cost (and only those components appear in (the domain of) *costEC*); (b) every element of *path* has an associated cost (and only those paths appear in (the domain of) *costEP*); and (c) the cost of each path is at least that of each of its individual component. (It would be inappropriate for our core model to go further than this in constraining the relationships between component costs and path costs. For example, while, in some circumstances, the cost associated of a path might be calculated by simply adding the costs of its components, in the general case the components will not be sufficiently independent to permit such a calculation. Nevertheless, there is, of course, the possibility to add such constraints — such as adding costs and / or multiplying independent probabilities — as appropriate.)

We might layer this aspect on top of our extended model that takes into account possibilities thus.

$$CoreWithPossibilityAndCosts \mathrel{\widehat=} Core \land Possibilities \land Costs$$

Again, some operations remain mostly unaffected by the incorporation of costs, while others need to be adapted. For example, *AddComponent* now takes the following form.

AddComponent

$$\Delta CoreWithPossibilityAndCosts$$
$$comp? : Component$$
$$cost? : Cost$$
$$comp? \notin components$$
$$components' = components \cup \{comp?\}$$
$$attacks' = attacks$$
$$paths' = paths$$
$$trees' = trees$$
$$forest' = forest$$
$$costEC' = costEC \cup \{comp? \mapsto cost?\}$$
$$costEP' = costEP$$
$$poss' = poss \cup \{comp? \mapsto P\}$$

Now it is the case that any component that is added must have an associated cost. In addition, the default / initial assumption is that all components are possible (hence the mapping of *comp*? to P in *poss*).

Further, there are additional operations to reflect the new attributes. The operation *ChangePathExploitationCost*, presented below, is one such example.

ChangePathExploitationCost

$$\Delta CoreWithPossibilityAndCosts$$
$$\Xi Core$$
$$p? : Path$$
$$c? : Cost$$
$$p? \in paths$$
$$\forall c : \mathrm{ran}\,(p?) \bullet c? \geq costEC\ c$$
$$costEP' = costEP \oplus \{p? \mapsto c?\}$$
$$poss' = poss$$

Here, the $\Xi Core$ ensures that the attributes associated with *Core* remain unchanged.

Finally, we might wish to take advantage of the two extensions to the core model to consider the cost of exploitation of possible attacks:

CostToPerformPossibleAttack

$$\Xi CoreWithPossibilityAndCosts$$
$$a? : Attack$$
$$X : \mathbb{F}\ Path$$
$$c! : Cost$$
$$a? \in \mathrm{dom}\ forest$$
$$X = \{p : forest\,(a?) \mid (\forall c : \mathrm{ran}\ p \bullet poss\ c = P)\}$$
$$X \neq \emptyset \Rightarrow c! = (\mu c : Cost \mid (\exists p : X \bullet costEP\ p = c \land (\forall q : X \bullet costEP\ p \leq costEP\ q)))$$

Here, we consider a possible attack, $a? \in Attack$. The set X consists of those paths associated with $a?$ in the current forest in which all components are characterised as possible (as per the *PossiblePathsToAttack* schema of Section 6). Then, the cost to perform $a?$, returned via the output $c!$, is simply the 'cheapest' path found in X. To do so, the μ-operator is used to find the unique cost with the stated property.

8. CONCLUSION

We have described a formal model of attack trees. The model was motivated by the need to underpin analysis of costs from the perspective of attackers — in this case, we considered to 'cost to exploit'. A number of requirements were established for the approach; all have been met.

It is worth noting that there is a broad literature on the formalisation of attack trees (see, for example, [51], [52], [53] and [5]). Widel et al. [53] argue that, classically, two interpretations have been used for attack trees: "the propositional semantics ... where an attack tree is interpreted as a Boolean function

representing the structure of the tree" [53] and "the multiset semantics ... interpreting an attack tree as a set of multisets modeling the attack vectors covered by the tree" [53]. The work of Lenin et al. [54] is offered as an example of the former; the work of Mauw and Oostdijk [6] is offered as an example of the latter. To a degree, we have based our contribution on the latter. It is also worth noting that others have seen value in adopting a state-based approach to representing such constructs (see, for example, the contribution of Fila and Pinchinat [55]).

We intend to build upon this work in a number of ways (and, indeed, have started doing so). First, as mentioned in Section 1, the model described in this paper has already been mapped to the B-method [18] with a view to taking advantage of the tool support offered by ProB [16]. Second, having established the underpinning model, we are now in a position to develop the toolset mentioned in Section 2 — and, in doing so, incorporate probability distributions and actual, estimated costs (as opposed to the more abstract representations provided here to establish the 'proof of concept'). Thus, while the formal model as presented here establishes some healthy foundations and provides a novel perspective on a familiar problem that may be of interest to the research community, there is much work left to do to realise our longer term aims.

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their helpful comments. This research was undertaken as part of the CEMDAT (Combining Exploit Market Data with Attack Trees) project, funded by the US Air Force Research Laboratory. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. No new data were generated or analysed in support of this research.

REFERENCES

[1] Schneier, B. (1999) Attack trees. *Dr. Dobb's Journal*, **24**, 21–29.

[2] Sheyner, O., Haines, J., Jha, S., Lippmann, S., and Wing, J. M. (2002) Automated generation and analysis of attack graphs. *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P 2002)*, pp. 273–284. IEEE Computer Society.

[3] Schneier, B. (2000) *Secrets & Lies: Digital Security in a Networked World*. Wiley, Hoboken, NJ, USA.

[4] Audinot, M., Pinchinat, S., and Kordy, B. (2017) Is my attack tree correct? In Foley, S. N., Gollmann, D., and Snekkenes, E. (eds.), *Proceedings of the 22nd European Symposium on Research in Computer Security (ESORICS 2017) Part I*, Lecture Notes in Computer Science, **10492**, pp. 83–102. Springer.

[5] Lallie, H. S., Debattista, K., and Bal, J. (2020) A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review*, **35**, 100219.

[6] Mauw, S. and Oostdijk, M. (2005) Foundations of attack trees. In Won, D. H. and Kim, S. (eds.), *Proceedings of the 8th International Conference on Information Security and Cryptology (ICISC 2005)*, Lecture Notes in Computer Science, **3935**, pp. 186–198. Springer.

[7] Vigo, R., Neilson, F., and Nielson, H. R. (2014) Automated generation of attack trees. *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF 2014)*, pp. 337–350. IEEE Computer Society.

[8] Ivanova, M. G., Probst, C. W., Hansen, R. R., and Kammüller, F. (2015) Transforming graphical system models to graphical attack models. In Mauw, S., Kordy, B., and Jajodia, S. (eds.), *Proceedings of the First International Workshop on Graphical Models for Security (GraMSec 2015)*, Lecture Notes in Computer Science, **9390**, pp. 82–96. Springer.

[9] Pinchinat, S., Acher, M., and Vojtisek, D. (2015) ATSyRa: an integrated environment for synthesizing attack trees. In Mauw, S., Kordy, B., and Jajodia, S. (eds.), *Proceedings of the First International Workshop on Graphical Models for Security (GraMSec 2015)*, Lecture Notes in Computer Science, **9390**, pp. 97–101. Springer.

[10] Cheah, M., Nga Nguyen, H., Bryans, J., and Shaikh, S. A. (2017) Formalising systematic security evaluations using attack trees for automotive applications. In Hancke, G. P. and Damiani, E. (eds.), *Proceedings of the 11th IFIP WG11.2 International Conference on Information Security Theory and Practice (WISTP 2017)*, Lecture Notes in Computer Science, **10741**, pp. 113–129. Springer.

[11] Sonnenreich, W., Albanese, J., and Stout, B. (2005) Return on security investment (ROSI): A practical quantitative model. *Journal of Research and Practice in Information Technology*, **38**, 239–252.

[12] van Rensburg, A. J., Nurse, J. R. C., and Goldsmith, M. (2016) Attacker-parametrised attack graphs. *Proceedings of the 10th International Conference on Quantum, Nano/Bio, and Micro Technologies (ICQNM 2016)*, pp. 316–319.

[13] Schechter, S. (2002) Quantitatively differentiating system security. *Proceedings of the 1st Workshop on the Economics of Information Security (WEIS 2002)*.

[14] Cremonini, M. and Martini, P. (2005) Evaluating information security investments from attackers' perspective: the return-on-attack (ROA). *Proceedings of the 4th Workshop on the Economics of Information Security (WEIS 2005)*.

[15] Spivey, J. M. (1992) *The Z Notation: A Reference Manual*. Prentice Hall, Englewood Cliffs, NJ, USA.

[16] Leuschel, M. and Butler, M. (2003) ProB: a model checker for B. In Araki, K., Gnesi, S., and Mandrioli, D. (eds.), *Proceedings of the 2003 International Symposium of Formal Methods Europe (FME 2003)*, Lecture Notes in Computer Science, **2805**, pp. 855–874. Springer.

[17] Plagge, D. and Leuschel, M. (2007) Validating Z specifications using the ProB animator and model checker. In Davies, J. W. M. and Gibbons, J. (eds.), *Proceedings of the 6th International Conference*

on Formal Methods (IFM 2007), Lecture Notes in Computer Science, **4591**, pp. 480–500. Springer.

[18] Abrial, J.-R. (1996) *The B-Book: Assigning Meanings to Programs*. Cambridge University Press, Cambridge, UK.

[19] Hoare, C. A. R. (1985) *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, NJ, USA.

[20] Roscoe, A. W. (2005) *The Theory and Practice of Concurrency*. Pearson, Upper Saddle River, NJ, USA.

[21] Anderson, R. and Moore, T. (2007) Information security economics — and beyond. In Menezes, A. (ed.), *Proceedings of the 27th Annual International Cryptology Conference (CRYPTO 2007)*, Lecture Notes in Computer Science, **4622**, pp. 68–91. Springer.

[22] Moore, T. and Anderson, R. (2011) Economics and internet security: A survey of recent analytical, empirical, and behavioral research. Technical Report TR-03-11. Harvard Computer Science Group.

[23] Geer, Jr., D., Soo Hoo, K., and Jaquith, A. (2003) Information security: Why the future belongs to the quants. *IEEE Security & Privacy*, **99**, 24–32.

[24] Anderson, R., Böhme, R., Clayton, R., and Moore, T. (2008). Security economics and the internal market. `https://www.enisa.europa.eu/publications/archive/economics-sec/`.

[25] Sarker, I. H., Kayes, A. S. M., Badsha, S., Alqahtani, H., Watters, P., and Ng, A. (2020) Cybersecurity data science: An overview from machine learning perspective. *Journal of Big Data*, **7**.

[26] Dellago, M., Simpson, A. C., and Woods, D. W. (2022) Exploit brokers and offensive cyber operations. *The Cyber Defense Review*, **7**, 31–48.

[27] Dellago, M., Woods, D. W., and Simpson, A. C. (2022) Characterising 0-day exploit brokers. *Proceedings of the 21st Workshop on the Economics of Information Security (WEIS 2022)*.

[28] Smith, J. E. H. (2019) *Irrationality: A History of the Dark Side of Reason*. Princeton University Press, Princeton, NJ, USA.

[29] Weiss, J. D. (1991) A system security engineering process. *Proceedings of the 14th National Computer Security Conference*, pp. 572–581.

[30] Amoroso, E. G. (1994) *Fundamentals of Computer Security Technology*. Prentice Hall, Englewood Cliffs, NJ, USA.

[31] Kordy, B., Pièrre-Cambacédès, L., and Schweitzer, P. (2014) DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review*, **13–14**, 1–38.

[32] Swiderski, F. and Synder, W. (2004) *Threat Modeling*. Microsoft Press.

[33] Ongsakorn, P., Turney, K., Thornton, M. A., Nair, S., Szygenda, S. A., and Manikas, T. (2010) Cyber threat trees for large system threat cataloging and analysis. *Proceedings of the 4th Annual IEEE Systems Conference*, pp. 610–615. IEEE Computer Society.

[34] Steffan, J. and Schumacher, M. (2002) Collaborative attack modeling. *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pp. 235–259. ACM.

[35] Bistarelli, S., Dall'Aglio, M., and Peretti, P. (2006) Strategic games on defense trees. In Dimitrakos, T., Martinelli, P., Ryan, P. Y. A., and Schneider, S. A. (eds.), *Proceedings of the 4th International Workshop on Formal Aspects in Security and Trust (FAST 2006)*, Lecture Notes in Computer Science, **4691**, pp. 1–15. Springer.

[36] Roy, A., Kim, D. S., and Trivedi, K. S. (2012) Attack countermeasure trees (ACT): Towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, **5**, 929–943.

[37] Gadyatskaya, O. and Trujillo-Rasua, R. (2017) New directions in attack tree research: Catching up with industrial needs. *International Workshop on Graphical Models for Security (GraMSec 2017)*, Lecture Notes in Computer Science, **10744**, pp. 115–126. Springer.

[38] Kavallieratos, G., Katsikas, S., and Gkioulos, V. (2020) Cybersecurity and safety co-engineering of cyberphysical systems — a comprehensive survey. *Future Internet*, **12**, 65.

[39] Ji, Z., Yang, S.-H., Cao, Y., Wang, Y., Zhou, C., Yue, L., and Zhang, Y. (2021) Harmonizing safety and security risk analysis and prevention in cyber-physical systems. *Process Safety and Environmental Protection*, **148**, 1279–1291.

[40] Kriaa, S., Pièrre-Cambacédès, L., Bouissou, M., and Halgand, Y. (2015) A survey of approaches combining safety and security for industrial control systems. *Reliability Engineering & System Safety*, **139**, 156–178.

[41] Niitsoo, M. (2010) Optimal adversary behavior for the serial model of financial attack trees. In Echizen, I., Kunihiro, N., and Sasaki, R. (eds.), *Proceedings of the 5th International Worshop on Security (IWSEC 2010)*, Lecture Notes in Computer Science, **6434**, pp. 354–370. Springer.

[42] Dewri, R., Poolsappasit, N., Ray, I., and Whitley, D. (2007) Optimal security hardening using multi-objective optimization on attack tree models of networks. *Proceedings of the 14th ACM conference on Computer and Communications Security (CCS 2007)*, pp. 204–213. ACM.

[43] Dewri, R., Ray, I., Poolsappasit, N., and Whitley, D. (2012) Optimal security hardening on attack tree models of networks: a cost-benefit analysis. *International Journal of Information Security*, **11**, 167–188.

[44] Buldas, A. and Stepanenko, R. (2012) Upper bounds for adversaries' utility in attack trees. In Grossklags, J. and Walrand, J. (eds.), *Proceedings of the 3rd International Conference on Decision and Game Theory for Security (GameSec2012)*, Lecture Notes in Computer Science, **7638**, pp. 98–117. Springer.

[45] Buldas, A. and Lenin, A. (2013) New efficient utility upper bounds for the fully adaptive model of attack trees. In Das, S. K., Nita-Rotaru, C., and Kantarcioglu, M. (eds.), *Proceedings of the 4th International Conference on Decision and Game Theory for Security (GameSec2013)*, Lecture Notes in Computer Science, **8252**, pp. 192–205. Springer.

[46] Van Holsteijn, F. A. (2015) The motivation of attackers in attack tree analysis. Master's thesis. TU Delft.

[47] Fila, B. and Widel, W. (2020) Exploiting attack-defense trees to find an optimal set of countermeasures. *Proceedings of the 33rd IEEE Computer Security*

Foundations Symposium, (CSF 2020), pp. 395–410. IEEE Computer Society.

[48] Patten, T., Mitchell, D., and Call, C. (2020) Cyber attack grammars for risk/cost analysis. *International Conference on Cyber Warfare and Security*.

[49] Nguyen, H. N., Bryans, J., and Shaikh, S. (2019) Attack defense trees with sequential conjunction. *Proceedings of the 19th IEEE International Symposium on High Assurance Systems Engineering (HASE 2019)*, pp. 247–252. IEEE Computer Society.

[50] Woodcock, J. C. P. and Davies, J. W. M. (1996) *Using Z: Specification, Refinement, and Proof*. Prentice Hall, Englewood Cliffs, NJ, USA.

[51] Tuma, K., Calikli, G., and Scandariato, R. (2018) Threat analysis of software systems: A systematic literature review. *Journal of Systems and Software*, **144**, 275–294.

[52] Mantel, H. and Probst, C. W. (2019) On the meaning and purpose of attack trees. *Proceedings of the 32nd IEEE Computer Security Foundations Symposium (CSF 2019)*, pp. 184–18415. IEEE Computer Society.

[53] Wideł, W., Audinot, M., Fila, B., and Pinchinat, S. (2019) Beyond 2014: Formal methods for attack tree-based security modeling. *ACM Computing Surveys*, **52**, Article Number 75.

[54] Lenin, A., Willemson, J., and Sari, D. P. (2014) Attacker profiling in quantitative security assessment based on attack trees. In Bernsmed, K. and Fischer-Hübner, S. (eds.), *Proceedings of the 19th Nordic Conference on Secure IT Systems (NordSec 2014)*, Lecture Notes in Computer Science, **8788**, pp. 199–212. Springer.

[55] Fila, B. and Pinchinat, S. (2020). State-based attack–defense trees. `https://www.semanticscholar.org/paper/State-based-attackdefense-trees-Fila/3110abb87b4eab87d69487470b837aad18fccbfa#reference`.