

# Containment Strategy Formalism in a Probabilistic Threat Modelling Framework

Per Fahlander<sup>1</sup><sup>a</sup>, Mathias Ekstedt<sup>1</sup><sup>b</sup>, Preetam Mukherjee<sup>1,2</sup><sup>c</sup> and Ashish Kumar Dwivedi<sup>1</sup><sup>d</sup>

<sup>1</sup>*Department of Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden*

<sup>2</sup>*School of Computer Science and Engineering, Digital University Kerala, Kerala, India*

**Keywords:** Threat Analysis, MAL, Containment Strategies, Simulated Annealing.

**Abstract:** Foreseeing, mitigating and preventing cyber-attacks is more important than ever before. Advances in the field of probabilistic threat modelling can help organisations understand their own resilience profile against cyber-attacks. Previous research has proposed MAL, a meta language for capturing the attack logic of a considered domain and running attack simulations in a model that depicts the defended IT-infrastructure. While this modality is already somewhat established for proposing general threat mitigation actions, less is known about how to model containment strategies in the event that penetration already has occurred. The problem is a fundamental gap between predominant threat models in cyber-security research and containment in the incident response lifecycle. This paper presents a solution to the problem by summarizing a methodology for reasoning about containment strategies in MAL-based threat models.

## 1 INTRODUCTION


Cyber-attacks of sufficient magnitude can have adverse consequences not only for organisations - but for individuals - whether it is denied functionality, lock-out or information leakage. The burden of foreseeing, mitigating and preventing cyber-attacks may also be more important than ever before, but in spite of high stakes, resilience can be very challenging against an aggressive adversary. It may also be particularly difficult to reason about how to protect specific assets and infrastructure without advanced and domain-specific expertise.


Various methods to computationally detect vulnerabilities and reason about conceivable threats have previously been proposed by multiple different researchers (Stan et al., 2019; Soikkeli et al., 2019; Poolsappasit et al., 2012; Li et al., 2020; Fila and Wideł, 2020). All these methodologies may mitigate potential security issues with the use of threat models akin to the IT-infrastructure in question, albeit by assuming that the attack has not yet already transpired. Techniques that concern counter-measures


subsequent to the attack appear unprecedented in the literature on threat modelling, which constitutes a gap in initial assumption such that existing methodologies may not be directly applicable in this context.


Earlier research has for example highlighted utility in probabilistic threat modelling techniques and attack simulations. The Meta Attack Language (MAL) is one framework that has repeatedly been mentioned in the context of cyber-security ((Johnson et al., 2018), (Katsikeas et al., 2020), (Katsikeas et al., 2019)). The idea is to first model the assets and infrastructure of the organisation, simulate an adversary intruding in the system, and then demonstrate viable routes to penetration attributed with respective probability estimations. By elucidating the vulnerabilities of a system it can be learned how to mitigate these risks and prevent penetration from occurring in the first place.

While threat modelling techniques are recognized as a tool for prevention of penetration in cyber-security networks, less is known about how threat models could be useful in the event that penetration already has occurred. That is, how would threat models be used to reason about containment of threats, specifically? This paper summarizes a methodology that consolidates the MAL-based threat modelling paradigm with a novel interpretation on containment from the incident response framework. The ideas

<sup>a</sup>  <https://orcid.org/0000-0002-1639-2673>

<sup>b</sup>  <https://orcid.org/0000-0003-3922-9606>

<sup>c</sup>  <https://orcid.org/0000-0003-2549-6578>

<sup>d</sup>  <https://orcid.org/0000-0002-4641-9240>

proposed here hence work at the intersection of two otherwise independent domains, enabling discriminative reasoning about containment strategies consistent with the particular requirements of the threat model.

Due to a lack of previous research on the distinct matter, this paper presents early ideas for the use of the threat model in cyber-security containment decisions. In this paper, we have summarized the key findings from the independent thesis work by (Fahlander, 2021) who have made the following contributions: (1) outlined the landscape that is containment in the literature, (2) presented a modality where concepts from “incident response” methodology are formalized and expressed in relationship to MAL-based threat models, (3) framed containment strategy selection as an optimization problem and (4) evaluated a particular implementation for solving this optimization algorithmically.

The rest of the paper is organized as follows. Section 2 gives the background necessary to achieve a firm understanding of the rest of the work in detail. Section 3 outlines the landscape in terms of other notable methodologies of interest and approaches to containment that are commonplace in the literature. Section 4 formally defines a set of concepts used in the methodology presented. Section 5 details the representation of these concepts by sharing implementation-specific features and technical assumptions. Section 6 describes the methodology used to find optimal containment strategies by utilizing the aforementioned concepts. Section 7 describes two distinct procedures used to evaluate the work critically and reveals the outcome yielded by doing so. Section 8 concludes the paper with the main takeaways. In section 9 we finally end with a discussion about the role of future work on the matter.

## 2 BACKGROUND

### 2.1 Meta Attack Language

The *Meta Attack Language (MAL)* (Johnson et al., 2018) provides the building blocks in terms of formalized syntax and semantics for encoding the attack logic of a considered domain. For example, a domain could refer to cloud systems, embedded systems or general IT infrastructure. MAL can be considered meta in the sense that something written in the language, in itself, is a language. Such a language can be used to manually or semi-automatically create a model of organizational infrastructure where attack simulations can be run. As each domain has its own set of theoretical constructs, these can be codified

with the use of MAL-based syntax. Examples of previously developed domain-specific MAL languages include *coreLang*, *powerLang* and *vehicleLang* ((Katsikeas et al., 2020), (Hacks et al., 2020), (Katsikeas et al., 2019)).

A central idea to MAL is the meta modelling hierarchy that discerns three different layers: a meta language, a domain-specific language and a system-specific model. The meta language provides formal syntax for expressing many different types of attack logic. The domain-specific language delimits the scope to a considered domain by codifying the attack logic that characterizes its generic features, a man-in-the-middle attack within IT-infrastructure, for example. A system-specific model can then be devised, through the use of this language, to capture idiosyncratic characteristics of the system defended, which could be a network or other IT-infrastructure belonging to an organization, for example.

The system-specific model breaks down into two types of main elements: assets and associations (Katsikeas et al., 2020). *Assets* model the elements found in the considered domain, for example, “System”, “Application” and “Vulnerability” are asset types used to depict IT-infrastructure. Relationships between assets can also be described with *associations*, such as “network access” between two hosts. An asset may be considered an attack surface that consists of several *attack steps* (here denoted  $\alpha$ ) that represent the threats that the asset is vulnerable to. Attack steps can be *performed* which is equivalent to saying that its time to compromise value is 0, as explained in a moment. These steps may furthermore be interconnected with parent and children steps to form *attack paths* that are explored when a simulation is run within the system-specific model, as visualized in Figure 2. There are also attack steps of type “AND”, that require that all parental steps are performed beforehand, and attack steps of type “OR” that requires at least one. Lastly, a *defense* is a form of attack step that may be implemented on an asset, thereby modelling a threat mitigation action that the defending organization may attempt. Refer to Figure 1 for an illustration of assets and associations in a system-specific model and Figure 2 for an understanding how attack steps may be interrelated in general.

Finally, the time to compromise (*ttc*) is a concept of paramount importance to the aforementioned attack simulations. It is a metric repeatedly mentioned in the context of estimating risk and resilience from cyber-attacks ((McQueen et al., 2006), (Ekstedt et al., 2015), (Johnson et al., 2018), (Katsikeas et al., 2020)). Although difficult to estimate, the number indicates the amount of effort required by an attacker.

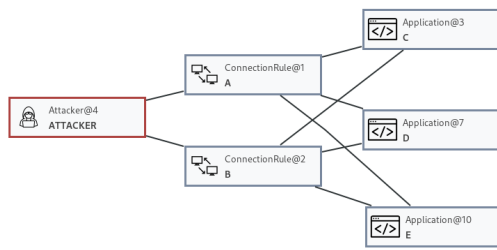


Figure 1: A coreLang-based system-specific model visualized in securiCAD Professional Edition, an application developed by foreseeti.

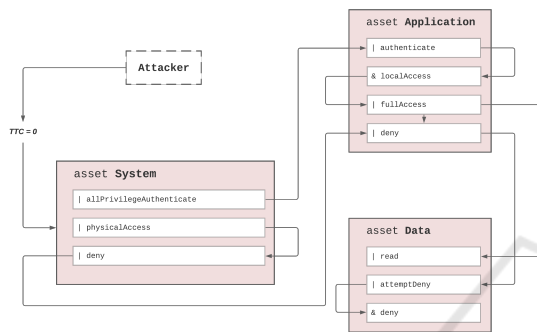


Figure 2: An illustration showing how attack paths may interconnect attack steps between multiple assets in a coreLang-based model.

(McQueen et al., 2006) formally defined the metric as “the time needed for an attacker to gain some level of privilege  $p$  on some system component  $i$ . This time depends on the nature of the vulnerabilities and the adversary’s skill level. In the context of MAL, this value is estimated locally for each attack step in the system-specific model, and subsequently also globally to account for the difficulty in performing the required parental attack steps. This means that each attack step has an adjusted “global” time to compromise value that depends on both the probability of the adversary performing the current attack step as well as all preceding attack steps that lead up to that point in the attack path. Note also that in this paper we will be working with the full time to compromise distribution from 0 to infinity.

## 2.2 CoreLang

In this work we have used a domain-specific language called coreLang, but acknowledge that the methodology presented in this paper is agnostic to the MAL-based language used. coreLang should hence merely be viewed as an example for demonstrative purposes and is in fact a work in progress in itself. This particular language happens to model IT-infrastructure, a context in which it could be especially relevant to im-

plement containment. The language also has several assets categories so that assets fall under either “System”, “Vulnerability”, “User”, “Identity”, “Data resources” and “Networking”. The available assets can be associated to depict IT-networks or other types of relevant structures and include for example “Application”, “Data” and “High Complexity Vulnerability”. (Katsikeas et al., 2020)

## 3 RELATED WORK

### 3.1 Incident Response

NIST Institute has put forward an incident response framework that incorporates four distinct phases. Containment is a central activity of the third phase “Containment, eradication & recovery” in which efforts are made to mitigate the impact of incidents. By incident, we refer to a *computer security incident*, which is formally defined by NIST as: “A violation or imminent threat of violation of computer security policies, acceptable use policies, or standard security practices”. Incidents are the provocations that incentivize containment as a remediating response. It is of primary importance to contain incidents in a system before the damage taken escalates or its resources are overwhelmed. Complicating the matter, however, is the fact that a majority of incidents require containment and different incidents require disparate containment measures. Such measures are here referred to as *containment strategies*, but the term *containment action* is also used in some contexts. The decisions incorporated into such a strategy or action could for example be: “shutting down a system”, “disconnect it from a network” or “disable certain functions”. (Cichonski et al., 2012)

### 3.2 Descriptive Instances of Containment in Literature

As various literature on “incident response” methodology illustrates containment, practical instances of containment are sometimes given as examples. Refer to Table 1 for a limited sample of descriptive instances. The containment actions listed in the literature are explicitly said to address either general containment or one of either: phishing attacks, denial of service attacks, malware outbreaks or instances with hostile employees, specifically. This should bring some insight into what containment might look like without preconceived notions. Note, that references older than four years were intentionally omitted in fa-

vor of the relevance of these cross-referenced descriptions.

## 4 A FORMALISM FOR CONTAINMENT

### 4.1 Incident

To express the notion of computer security incidents within a MAL-based framework their characteristics need to be modelled. In this paper we model an incident as a combination of two separate constructs: the tentative instance and indicative markers. An incident can also be either *indicated* or *latent*, reflecting whether the incident is occurring in the model. If so, the system is in a vulnerable state as a result of malicious use of the assets mapped by the tentative instance.

#### 4.1.1 Tentative Instance

The *tentative instance* is a particular configuration of assets that may be utilised by an adversary in a hostile manner to start an incident. Such an instance is tentative in the sense that it is only a hypothetical threat and can remain dormant without causing problems. It is formally defined as a surjective mapping function where the domain is the set of *roles* tied to the incident, and the co-domain, all assets in the system-specific model:  $T : \{role \mapsto asset\}$ . A role in this context is a natural language term used by incident responders to address a certain asset involved in an incident. For example, some asset representing an account endangered by a phishing attack or an application that cannot be used as a result of a denial of service attack. It thus has to be mapped to an asset in a system-specific model in order to be of practical utility. Finally, whether an incident is indicated is determined by a set of attributed indicative markers.

#### 4.1.2 Indicative Marker

An *indicative marker* is defined as a function  $I$  where  $I : \{\alpha \rightarrow ttc\} \rightarrow b$  where  $ttc$  is a value in the domain  $[0, \infty]$  and  $b$  is a binary number. Here, the input is a mapping of attack steps  $\alpha$  to their respective time to compromise values  $ttc$ . The probability that these attack steps are performed such that  $b = 1$  is an estimate of the risk of the incident occurring. The function returns 1 if the incident is indicated for this input, otherwise 0, thereby modelling the detection of some reality-corresponding phenomena that realistically could justify containment.

In the model, a sufficiently low time to compromise value for some attack step is a cue that incentivizes containment. The incident can thus be considered an offset in some time to compromise values that swings the pendulum such that containment now is incentivized. For example, an indicative marker could state that if the time to compromise values of attack steps “guessCredentials” and “use” (both belonging to asset “Credentials”) are lower than a threshold, then a brute-force attack has succeeded, which would constitute an incident to contain.

However, for only entertaining the hypothetical idea of an incident being indicated, the indicative markers may instead be used to put the system in a vulnerable state. This would be done by changing the time to compromise value of some attack step(s)  $\{\alpha\}$  such that  $b = 1$ . Note, that the existence of an edge from the attacker to some attack step in the defended model implies a vulnerable state. To express this vulnerability, the attacker may be illustrated to be attached to the system-specific model.

### 4.2 Attack Step Protection

An attack step  $\alpha$  is said to be *protected* by a containment action  $\beta$ , if  $\alpha_{ttc} < \alpha'_{ttc}$ , where  $\alpha_{ttc}$  and  $\alpha'_{ttc}$  refers to the time to compromise value of  $\alpha$  before and after  $\beta$  is applied, respectively. Protection occurs anytime the time to compromise value is increased by any amount over the pre-containment baseline. If the time to compromise value changes to  $\infty$ , then  $\alpha$  is said to be *completely protected* by  $\beta$ .

### 4.3 Containment Action

A *containment action* is here defined as an alteration of the system-specific model that satisfies the following properties:

1. It is expressed as an associated set of *Containment Structural Attribute Flags (CSAF)* that indicate hypothetical mutations to the system-specific model, e.g. what if you suspend a host, or prohibit network traffic in a particular direction? To test out the consequences of such hypothetical changes, CSAF-flags may for example show that an association should be replaced or that an asset should be removed. The consequences of these structural changes are altered time to compromise values and we will say that a containment action is *deployed* when these altered values are used. Refer to Table 2 for a few examples of CSAF-flags that were used in this study.
2. It must be capable of theoretically limiting the spread of an incident through this mechanism;

Table 1: Some suggestions for containment according to literature on “incident response”. The pure descriptive citations are grouped by the general idea for containment that they convey (inferred containment action).

Containment action	Descriptive citations from the literature	References
Disable a network connection ( <i>BlockSpecificConnection</i> )	1. “Unplugging the network cable”, 2. “[Unplug] the network cable, [disable] wireless access, or [disable] the connection through the operating system.”, 3. “Disabling the network switch port to which a particular system is connected”, 4. “[Physically remove] the network cable and disabling wireless networking on the device”	(Thompson, 2018), (Johansen, 2020), (Roberts and Brown, 2017), (Sheward, 2018)
Put a machine in sleep mode ( <i>SuspendHost</i> )	1. “Putting the machine in sleep mode (Powering it off causes volatile memory loss and the loss of forensic evidence.)”	(Thompson, 2018)
Isolate a machine from the network ( <i>IsolateHost</i> )	1. “Take the identified end points off the network; do not power off”, 2. “[T]he affected servers or computers can be taken offline by disconnecting them from the organization’s network”	(Thompson, 2018), (Roberts and Brown, 2017)
Isolate a network segment ( <i>IsolateNetworkSegment</i> )	1. “[Contain] network traffic at the perimeter and [work the] way to the specific subnets containing the impacted systems”, 2. “Other isolation techniques include remediation virtual local area networks (VLANs), which are special network segments that use access control lists to prevent hosts ... from talking to the internet”	(Johansen, 2020), (Sheward, 2018)
Filter network traffic ( <i>DropInboundTraffic</i> , <i>DropOutboundTraffic</i> )	1. “Block traffic with perimeter devices.”, 2. “... a more common technique for denial of service containment is traffic filtering or scrubbing.”, 3. “Blocking access to malicious network resources such as IPs (at the firewall) and domains or specific URLs (via a network proxy)”	(Thompson, 2018), (Sheward, 2018), (Roberts and Brown, 2017)
Lock user account ( <i>LockUserAccount</i> )	1. “Temporarily locking a user account under the control of an intruder”	(Roberts and Brown, 2017)
Disable app / service ( <i>StopService</i> )	1. “Disabling system services or software an adversary is exploiting”, 2. “Temporarily disable applications and services affected by the attack.”	(Roberts and Brown, 2017), (Thompson, 2018)
Reroute or rebalance load	1. “In many cases involving the cloud, containment will involve the shutdown of the affected instances or apps, with traffic being rerouted to alternate sites.”, 2. “Add servers and load balancers, as needed.”, 3. “In the case of an attack against an IP address, an easy containment method is to move the service to a new IP address and, where applicable, update DNS records to point to the new address.”	(Roberts and Brown, 2017), (Thompson, 2018), (Sheward, 2018)

that is, protect one or more attack steps in a system-specific model through CSAF-flags that reflect structural changes.

3. The action must in reality be able to do this protection even after an incident has occurred. This implies that the protection of attack steps has to be justified from a modelling point of view. For example, imagine a scenario where containment involves the act of locking a user account. It may be ineffective to so if the adversary has had enough privileges to create a new account for some amount of time. If the containment action protects time to compromise values that reflect access through any user account, then it must be able

to eject the adversary out of the system completely (i.e. from any exploited or created user account) and not merely prevent this access in the future. A seemingly minor protection may hence be a very ambitious goal in reality.

4. It has a specified deployment point *DP* that consists of one or more assets. It is represented as a surjective mapping of assets by roles  $DP : role \mapsto asset$ . A role is natural language term that describes an asset involved in the action, and has to be mapped to a specific asset in the system-specific model. This mapping of assets by roles indicates where the containment action takes effect.

Table 2: Examples of Containment Structural Attribute Flags (CSAF) that were used in this work.

CSAF-flag	Semantic interpretation
ASSET_DETACH	Indicates whether to detach an asset from the system-specific model. Equivalent to detaching all associations (incl. attacker associations) from an asset.
ASSOC_DETACH	Indicates whether to detach an association from the system-specific model.
ASSOC_SWAP	Replaces an old association in the system-specific model with a new one if given a value (not empty string).

A containment action may be denoted by a name followed by a specified deployment point in parentheses. For example, *SuspendHost(host=A)* for an action that suspends a host named *A*. A containment action can also be expressed more formally with the terminology used in Equation 1. Here, a set of attack steps  $\{\alpha\}$  each map to a new corresponding time to compromise value  $\alpha'_{ttc}$  such that  $\alpha'_{ttc} > \alpha_{ttc}$  holds. The two variables refer to the time to compromise before and after the action is applied, respectively.  $f(DP)$  refers to a function specific to the containment action in question that maps the deployment point to a set of attack steps affected by the action.

$$\beta : \{\alpha \mapsto \alpha'_{ttc} > \alpha_{ttc} \mid \alpha \in f(DP), \alpha_{ttc} : [0, \infty), DP : role \mapsto asset\} \quad (1)$$

#### 4.4 Valuation

A *valuation* is defined as a tuple of two sets of externally estimated values that reflect what is at stakes in the IT-infrastructure. This construct is denoted by  $(\{\alpha_{conseq.}\}, \{\beta_{conseq.}\})$  and includes two different types of numeric values in the domain of  $[0, 100]$ :

- $\alpha_{conseq.}$ : an estimate of the potentially preventable consequence entailed if the attack step  $\alpha$  is performed (e.g., the relative downtime cost per hour of a service being denied, or the number of gigabytes of data stolen). Refer to Table 3 for an example of an assignment for this estimate.
- $\beta_{conseq.}$ : an estimate of the consequence entailed that is inherent when deploying the containment action (e.g., downtime cost per hour of system suspension, or a quantified measure of the social repercussions of locking a user account). Refer to

Table 3: An example of the  $\alpha_{conseq.}$  part of a valuation.

Attack step ( $\alpha$ )	Valuation ( $\alpha_{conseq.}$ )
C.deny	100
D.deny	100
E.deny	100
...	0

Table 4: An example of the  $\beta_{conseq.}$  part of a valuation. Assets A-E refer to assets in the system-specific model shown in Figure 3.

Containment action ( $\beta$ )	Valuation ( $\beta_{conseq.}$ )
BlockSpecificConnection(conn.=A)	20
BlockSpecificConnection(conn.=B)	20
DropInboundTraffic(receiver=C)	20
DropInboundTraffic(receiver=D)	20
DropInboundTraffic(receiver=E)	20
DropOutboundTraffic(sender=C)	20
DropOutboundTraffic(sender=D)	20
DropOutboundTraffic(sender=E)	20
StopService(service=C)	20
StopService(service=D)	20
StopService(service=E)	20

Table 4 for an example of an assignment for this estimate.

### 4.5 Containment Strategy

A *containment strategy* is defined as an unordered set of containment actions. A strategy  $S$  can hence be expressed as:  $S = \{\beta_1, \beta_2, \dots, \beta_n\}$ , where all  $\beta$  are containment actions in  $S$  and  $n$  is the total number of actions in the strategy considered (here limited to 3). Picking the optimal set of containment actions - with the most preferential loading on  $\alpha_{conseq.}$  and  $\beta_{conseq.}$  values - is a central objective in this study. Containment strategies are in this view differentiated by the attack step protection they provide and how problematic the containment actions they include are. The effectiveness of every containment strategy is thus dictated both by its effects in the model as well as contextual parameters.

## 5 REPRESENTATION

### 5.1 System-specific Model

In order to encode the system-specific model we used the graph computing framework Apache TinkerPop™ (TinkerPop, 2021). The types of graphs considered in

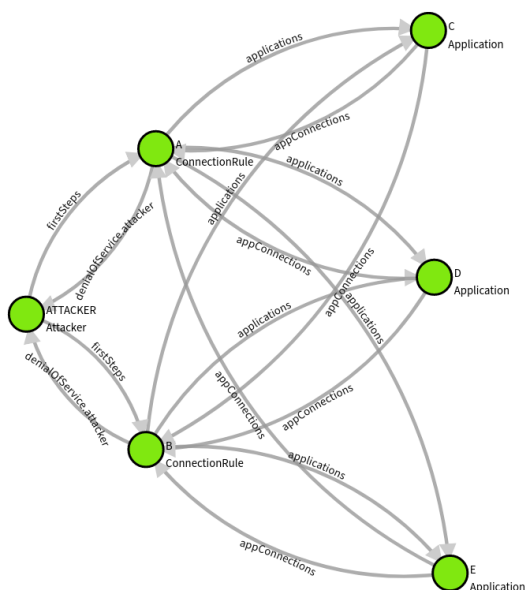


Figure 3: The system-specific model from Figure 1 interpreted as an Apache TinkerPop™ graph.

this framework are structures comprised of vertices, that embody discrete objects, and edges between vertices, that reflect relationships between objects. Both constructs may furthermore contain their own set of key-value pairs that hold non-relational information referred to as properties in this framework. In this modality, the system-specific model is represented using vertices and edges in an Apache TinkerPop™ graph as demonstrated by Figure 3. Vertices denote “assets”, and edges between them represent “associations”. A property on each vertex captures the type of an asset (e.g., *Application* or *Identity*). For each association, two directed edges in the opposite direction are used and labelled by the role.

## 5.2 Incident

### 5.2.1 Tentative Instance

A tentative instance has a set of assets and associations that could be involved in an incident. Since these constructs manifest in terms of vertices and edges in the Apache TinkerPop™ graph manifestation, these instances can be identified using graph traversal queries. Every query would be specific to the incident in question and each pattern it yields would constitute a point of entry for the adversary. The relevant attack steps are thus affiliated with the asset representations retrieved from this query. Typically, such queries look for some definite features in the pattern arrangement, such as what types of assets are associated with which. It may therefore be the case that only

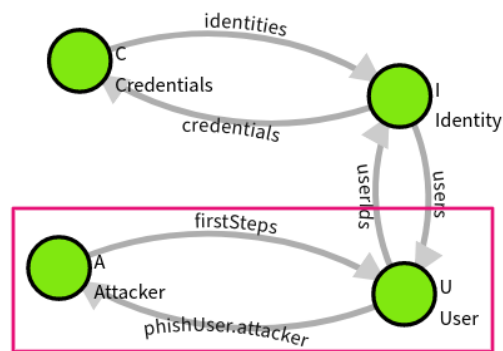


Figure 4: An example of a tentative instance for a phishing-attack where the marked *phishUser.attacker* - *firstSteps* edge symbolizes the point of entry. This edge denotes a time to compromise value of 0 for the attack step *phishUser* on asset *User*.

certain types of assets are considered for a given incident. For example, an incident that involves phishing may always have a pattern arrangement that includes an asset for the user, as illustrated in Figure 4.

### 5.2.2 Indicative Markers

Indicative markers are represented in terms of one or more edges between the attacker and some set of assets in the system-specific model. One of the end vertices (i.e. head or tail) of these edges are always assets from a tentative instance. When such an edge is added between the attacker and such an asset this reflects that the attacker is able to perform some attack step on the asset already from the beginning (i.e.  $ttc = 0$ ) and the incident is indicated. By adding and removing an edge that corresponds to an indicative marker, the system is respectively switched between a vulnerable and a safe state. This is helpful for controlling what hypothetical incidents should be entertained in the analysis.

## 5.3 Containment Action

As CSAF-flags pertain to specific assets and associations in the system-specific model they may highlight points of infrastructural mutability useful for the sake of containment. Such alterations to the model will be implied by the deployment of the containment action. To represent containment actions and CSAF-flags we used graph traversal queries in the Apache TinkerPop™ framework. These queries allow us to set properties on both assets and associations, thus enabling us to flag for whether any given mutation should be realized. For example, consider that the containment action *BlockSpecificConnection* mutates the model to remove an asset that represents a connection, in which case, the query would need to flag for a corresponding

property value update. The following implementation was used to express this behaviour:

```
g.V(connection)
  .has("metaConcept", "ConnectionRule")
  .property(CSAF.ASSET_DETACH, deployed)
  .hasNext();
```

Here, *metaConcept* refers to the asset type and *deployed* to a boolean value reflecting whether the containment action should be deployed.

## 6 FINDING AN OPTIMAL STRATEGY

### 6.1 Optimization Problem

The framing of containment strategy selection as an optimization problem begs the question of what factors might distinguish a good strategy from a poor one. It appears that no strategy is appropriate in all circumstances or environments; an activity that benefits one system could be detrimental to the state of another, for example. Structural variance in IT-infrastructure suggests that we include the system-specific model as an input parameter to the optimization, and this has indeed been a presumption of the study from the beginning. However, even if the model is given, what dictates the preference for keeping system A online over system B? This subjectivity hints at an additional utility in analyzing what is at stakes in the infrastructure, explaining the use of the valuation as an auxiliary input parameter to the optimization. Finally, proper containment has to be well-timed which implies that a measure of the progress of the adversary should be included. This measurement also accounts for variability in the types of incidents contained since an incident may be thought of as initial progress made by the adversary that could escalate if not contained.

A few assumptions are required in order to express these inputs in a MAL-based paradigm. Firstly, adversarial propagation is estimated as the set of time to compromise values for the attack steps in the model. Secondly, the incident that we are responding to might be thought of as an initial offset in some time to compromise values for some specific attack steps. Thirdly, a combination of the time to compromise metric and the external valuation should govern the manner in which we contain the threat. A question left unaddressed is where these time to compromise values come from. While the methodology presented here is agnostic to any specifics that relate to the generation and assignment of time to compromise values

to attack steps, this study used test code for MAL-based attack simulations compiled from the coreLang repository (mal lang, 2021). We will address such an assignment of time to compromise values to respective attack steps by the term *benchmark* here.

An additional consideration is that the externally estimated consequence of attack steps has to be put into the perspective: is it really likely that the attack step will be performed soon? For this reason, attack step valuations (i.e.  $\alpha_{conseq.}$ ) have to be adjusted by the time to compromise assigned to the attack step. This adjustment can be considered a discount to the attack step valuation and is proportional to the time to compromise value. However, since the former type of value lies in the domain  $[0, 100]$  and the latter in the domain  $[0, \infty]$ , this proportionality must clearly not be linear. Besides, protection is a lot more meaningful if the initial time to compromise value is somewhat low even if the estimation changed by the same amount. That is to say that disproportional urgency also discredits a linear model. For the adjustment intended, the function should yield the output 1 for the input 0, and the output 0 for the input  $\infty$ , reflecting the fact that the time to compromise metric lacks an upper bound. Conversely, only when the time to compromise value approaches 0 would the attack step be of relevance to any containment decision. A standard function  $\phi(ttc)$  that matches this description was identified, refer to Equation 2. It maps a time to compromise value  $\alpha_{ttc}$  to a coefficient that can be used to track the significance of  $\alpha_{conseq.}$  according to adversary propagation. This concept is illustrated in Figure 5.

Finally, this leaves us with the actual optimization function  $\theta$  shown in Equation 3. The output of this function represents the optimal containment strategy in the context of the aforementioned input parameters. Here,  $S$  represents an arbitrarily evaluated containment strategy candidate.  $A$  is the set of all attacks steps  $\{\alpha\}$  affected by the strategy.  $\alpha_{ttc}$  and  $\alpha'_{ttc}$  are the time to compromise values of  $\alpha$  before and after all actions  $\beta$  in the strategy  $S$  are deployed, respectively.  $\alpha_{conseq.}$  is the externally estimated consequence if  $\alpha$  should be performed by an adversary, reflecting what is at stakes in the attack.  $\beta_{conseq.}$  is a similar estimation with regards to the consequence inherent to  $\beta$ , capturing the negative consequences associated with containment itself. The numerator denotes the estimated benefit of protecting all attack steps in  $A$  by weighing in  $\alpha_{conseq.}$ , while the denominator factors in the inherent downside to this containment as per  $\beta_{conseq.}$ .

$$\phi(ttc) = 1 - \frac{ttc}{k \cdot ttc + 1}, k = 1 \quad (2)$$



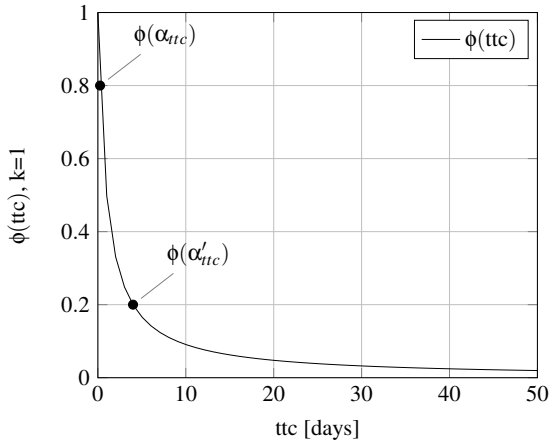


Figure 5: The standard function  $\phi(ttc)$  plotted from  $ttc = 0$  to  $ttc = 50$  ( $ttc < \infty$  is the real upper bound, however) for  $k = 1$ .  $\alpha_{ttc}$  and  $\alpha'_{ttc}$  are arbitrary time to compromise values translated by this function to illustrate the resulting coefficient value before ( $\phi(\alpha_{ttc})$ ) and after ( $\phi(\alpha'_{ttc})$ ) containment strategy deployment, respectively.

$$\theta = \underset{S}{\operatorname{argmax}} \frac{\sum_{\alpha \in A} [\alpha_{conseq.} \cdot (\phi(\alpha_{ttc}) - \phi(\alpha'_{ttc}))]}{\sum_{\beta \in S} \beta_{conseq.}} \quad (3)$$

## 6.2 Solving the Optimization

### 6.2.1 Simulated Annealing

After framing the issue of containment strategy selection as an optimization problem, the question remains how to solve this optimization algorithmically. Here we used a technique called simulated annealing to approximate the optimal solution (i.e. containment strategy). The idea for simulated annealing (Leemon@cs. cmu. edu, 2021) is inspired by the process of annealing in thermodynamics where metals cool and anneal. The algorithm is initially very willing to accept any solution randomly, but as the simulated temperature decreases, it gradually decreases this acceptance to act more like a hill climbing algorithm. This would be analogous to the metal becoming less malleable and more rigid when cooling. Hence, the algorithm always accepts better solutions, but is also somewhat open to worse solutions, albeit only to the extent that the temperature allows for it. Simulated annealing is effective for finding the global optima despite prevalence of a large set of local optima. The algorithm is typically applied for problems with large but discrete configuration spaces wherein the optimal solution is to be found.

We must also address how the containment strategies are generated and altered such that the simulated

annealing algorithm can explore similar and neighbouring configurations in the configuration space (i.e. a *neighbor*). First of all, the initial strategy is without any containment actions. Then, for every subsequent iteration, the strategy is altered by either adding (always done for the initial strategy), removing or replacing a containment action in the current strategy. A list of containment actions to pick for a strategy are automatically generated from the system-specific model by looking at all possible combinations of assets and containment action deployment points. Two distinct sets of containment actions are considered: one for the actions used in current strategy and one for unused containment actions that may be considered in another iteration. The two lists always exchange exactly one action with each other (i.e. replacement) or one list gives an action to the other (i.e. adding or removal) in an iteration of the simulated annealing algorithm. To guarantee that the algorithm does not get stuck in a neighbourhood of incommensurate solutions where no further move can be performed, an additional policy is introduced. The algorithm selects a previously safe solution (here: empty strategy or best strategy seen yet with equal probability) with a probability of  $\frac{1}{n+1}$  where  $n$  is the maximum number of actions in a strategy considered (here: 3). The algorithm may thus on average reach a strategy where all actions have been replaced before returning to the last safe solution.

We finally need to elucidate the implementation used to computationally attain the results described next. Algorithm 1 shows the implementation of the objective function used for this particular application of simulated annealing. It evaluates a containment strategy  $S$  according to the metric described by the optimization function in Equation 3. The same variables are therefore reused here, alongside  $\sigma_{\alpha}$  and  $\sigma_{\beta}$  that refer to numerator and denominator in Equation 3, respectively.

A version of the simulated annealing algorithm is shown in Algorithm 2. In this pseudocode,  $t$  refers to the simulated temperature value used to influence the probability of accepting a subpar solution. *coolingRate* is the rate at which the temperature is decreased in each iteration and is expressed as a coefficient.  $S_{accepted}$  is the currently accepted strategy that we have arrived at,  $S_{neighbor}$  is a related strategy with a modification and  $S_{best}$  is the best strategy yet seen.  $\theta_{accepted}$ ,  $\theta_{neighbor}$  and  $\theta_{best}$  refer to the respective values of the objective function for each of these strategies.

---

Algorithm 1: Objective function  $\Theta$  used as a metric for the optimization problem in Equation 3.

---

**Input:** containment strategy  $S$  for evaluation

**Result:** calculated strategy score

$A \leftarrow \text{benchmark}(S)$

$\sigma_\alpha \leftarrow 0$

**foreach**  $\alpha \in A$  **do**

$\sigma_\alpha \leftarrow \sigma_\alpha + \alpha_{conseq.} \cdot [\phi(\alpha_{ttc}) - \phi(\alpha'_{ttc})]$

**end**

$\sigma_\beta \leftarrow 0$

**foreach**  $\beta \in S$  **do**

$\sigma_\beta \leftarrow \sigma_\beta + \beta_{consequence}$

**end**

**return**  $\frac{\sigma_\alpha}{\sigma_\beta}$

---



---

Algorithm 2: Simulated annealing for containment strategy selection.

---

**Result:** best containment strategy found

$t \leftarrow 10000000$

$\text{coolingRate} \leftarrow 0.98$

$S_{accepted} \leftarrow \emptyset$

$S_{best} \leftarrow S_{accepted}$

$\Theta_{best} \leftarrow 0$

**while**  $t > 1$  **do**

$S_{neighbor} \leftarrow \text{neighbor}()$

$\Theta_{accepted} \leftarrow \Theta(S_{accepted})$

$\Theta_{neighbor} \leftarrow \Theta(S_{neighbor})$

**if**  $\Theta_{neighbor} > \Theta_{accepted}$  **then**

$\Theta_{best} \leftarrow \Theta_{neighbor}$

$S_{accepted} \leftarrow S_{neighbor}$

$S_{best} \leftarrow S_{accepted}$

**else if**

$\exp((\Theta_{neighbor} - \Theta_{accepted})/t) \geq \text{rand}()$

**then**

$S_{accepted} \leftarrow S_{neighbor}$

**else**

        ;

$t = t \cdot \text{coolingRate}$

**end**

**return**  $S_{best}$ ;

---

## 7 EVALUATION

For the purpose of evaluating the methodology presented, two separate procedures were used in this study. The first procedure involves a process of verification to confirm that the containment actions work as intended. The second procedure is validation of the containment strategy selection algorithm, which uses the constructs verified by the former procedure.

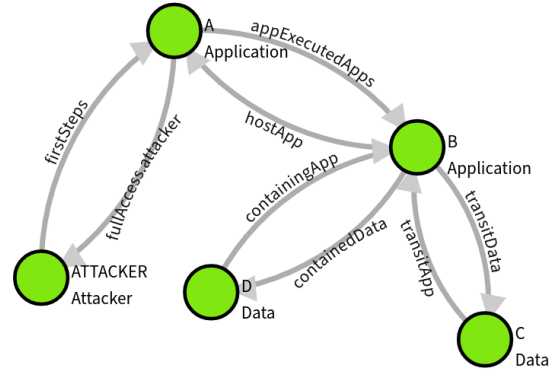


Figure 6: A system-specific model interpreted as an Apache TinkerPop™ graph used for the verification procedure.

Note, that only by doing the first procedure is the latter justified, since selecting among a set of ill defined containment actions would undermine the credibility of such testing.

### 7.1 Verification

The verification procedure strengthens the intuition that any containment action implementation really models the action it claims to model in a satisfying manner. In this qualitative approach, the set of protected attack steps is critically perused. This section presents a case where this procedure is applied to the containment action  $\text{StopService}(\text{service}=A)$  where ‘A’ refers to an asset in the system-specific model depicted in Figure 6. This containment action models any action that stops a running service or application. For example, usage of SIGKILL on Linux or termination via a user interface. The service in question is asset ‘A’ in this model. Other instances of verification can be found in (Fahlander, 2021, §5.2.1).

#### 7.1.1 Procedure

The approach is more specifically divided into four chronological steps:

1. Create an imaginary scenario to test the containment action.
2. Infer what subset of attack steps should be protected by the containment action given its purpose.
3. Test the containment action implementation in the same scenario to retrieve the subset of attack steps protected.
4. Compare the two subsets and verify that they are equal.

### 7.1.2 Results

It was found that no attack steps were performed after the protection achieved with deployment of containment action  $StopService(service=A)$ , as seen in Table 5. It was found that all attack steps were either completely protected or never performed in the first place.

Table 5: A summary of the post-containment (\*) outcome after deploying  $StopService(service=A)$  in the system-specific model specified in Figure 6.

Asset	Performed attack steps (*)	Protected attack steps
A	0	13
B	0	16
C	0	17
D	0	17

### 7.1.3 Discussion

This containment action constitutes one out of a total of seven containment actions verified in (Fahlander, 2021, §5.2.1). While a limited set of initial tests are available, these should not be misconstrued as exhaustive tests of the containment actions. These results merely demonstrate proof-of-concept for a limited set of scenarios, and while infinitely many scenario configurations are possible, more complicated scenarios could be used for more thorough verification. Here, the containment action  $StopService(service=A)$  was verified in the context of the system-specific model shown in Figure 6. The expected outcome is that the containment action should protect all attack steps on the service ‘A’, as well as the sub-process ‘B’ and associated data ‘C’ and ‘D’. The adversary should effectively be unable to perform any attack step on any asset ‘A’, ‘B’, ‘C’ or ‘D’ in the model post-containment, which is in alignment with the results attained.

## 7.2 Validation

The validation procedure evaluates whether the output from the algorithm is sound and accurate. To do so, we can reason about the probability of seeing certain containment strategies be endorsed by the algorithm. We postulate that validity can be asserted if the algorithm endorses the correct solution with statistical significance despite compelling odds to the contrary. The argument thus requires that there must exist one single, correct and objectively superior solution in the given problem context. For the sake of prudence, relatively uncomplicated models are chosen where the

correct solution can be inferred confidently. This section presents a case where this procedure is applied to the system-specific model illustrated in Figure 3 and a valuation described by Tables 3 and 4.

### 7.2.1 Procedure

The procedure can be divided into four consecutive steps:

1. Imagine a scenario with a system-specific model and a valuation where all containment actions have the same  $\beta_{conseq.}$ .
2. Reason about what is the correct (i.e. optimal) containment strategy for this scenario.
3. Run the algorithm for this scenario 5 times in a row to see what containment strategies are endorsed in each iteration.
4. Reject  $H_0$  if the correct containment strategy is endorsed in 5 consecutive iterations and the probability of this event is  $< 0.05$ .

Through means of statistical hypothesis testing, the procedure is an attempt to reject the null hypothesis  $H_0$  by calculating the corresponding p-value.  $H_0$  is defined as “the amount of times the correct solution is endorsed by the algorithm can be explained strictly by coincidence”. The threshold for rejecting the null hypothesis  $H_0$  is set to 0.05. The algorithm is run a total of five times in order to calculate the p-value, which is the probability of selecting the correct containment strategy of at most  $m = 3$  actions among  $n = 11$  total possible containment actions raised to five, as seen in Equation 4.

$$p\text{-value} = \left(\frac{1}{\binom{n}{m}}\right)^5 \tag{4}$$

### 7.2.2 Results

It was found that the same containment strategy was endorsed throughout the five runs, as seen in Table 6. Note, that the order of containment actions in a strategy is irrelevant due to its definition. The p-value was calculated to:  $\left(\frac{1}{\binom{n}{m}}\right)^5 = \left(\frac{1}{\binom{11}{3}}\right)^5 = \left(\frac{1}{165}\right)^5 = 8.1767417 \cdot 10^{-12}$ , and the null hypothesis hence rejected as p-value  $< 0.05$ .

### 7.2.3 Discussion

In this model, it is critical that assets ‘C’, ‘D’ and ‘E’ are protected from a denial of service attack (as reflected by the valuation:  $\alpha_{conseq.} = 100, \forall \alpha \in \{C.deny, D.deny, E.deny\}$ ). Other attack steps are not

Table 6: Raw output from the containment strategy selection algorithm (right column) in each of the five iterations achieved with the system-specific model shown in Figure 3 and valuation described in Tables 3 and 4. All iterations yielded an endorsement of the same containment strategy (irrespective of containment action order per definition).

It.	Endorsed containment strategy
1.	(BlockSpecificConnection(connection=B), BlockSpecificConnection(connection=A))
2.	(BlockSpecificConnection(connection=B), BlockSpecificConnection(connection=A))
3.	(BlockSpecificConnection(connection=B), BlockSpecificConnection(connection=A))
4.	(BlockSpecificConnection(connection=B), BlockSpecificConnection(connection=A))
5.	(BlockSpecificConnection(connection=A), BlockSpecificConnection(connection=B))

addressed in the valuation so they can safely be ignored. The total prevented consequence thus amounts to:  $\sum_{\alpha}^{\{C.deny,D.deny,E.deny\}} \alpha_{conseq.} = 300$ .

By deploying an action of type *DropInboundTraffic*, *DropOutboundTraffic* or *StopService*, it should be expected that three actions are required to address the denial of services ‘C’, ‘D’ and ‘E’ since each action only addresses one service at a time. By using *BlockSpecificConnection*, two actions are sufficient as the simultaneous removal of connections ‘A’ and ‘B’ stops the attack by making ‘C’, ‘D’ and ‘E’ unreachable. This solution (with an consequence of  $\beta_{conseq.} \cdot 2 = 40$ ) is clearly superior to any solution addressing three applications by three different actions ( $\beta_{conseq.} \cdot 3 = 60$ ) given that they have the same  $\beta$ -consequence and a total  $\alpha$ -consequence reduction of 300 regardless. Note, that the order of containment actions in the strategy does not matter since the problem at hand can be solved without such differentiation (the time to compromise for reaching ‘A’ and ‘B’ are the same). The only correct solution for this scenario is hence the following, which incorporates the containment actions in no particular order:

$$S_{optimal} = \{BlockSpecificConnection(connection=A), \\ BlockSpecificConnection(connection=B)\}$$

Given that all containment actions are weighted equally ( $\beta_{conseq.} = 20$ ), the probability of guessing this solution once is:  $\frac{1}{\binom{11}{3}} = \frac{1}{165}$ . Doing so five times in a row would be very unlikely:  $(\frac{1}{165})^5 \approx 8.1767417 \cdot 10^{-12}$ . The null hypothesis is rejected since  $8.1767417 \cdot 10^{-12} < 0.05$ .

## 8 CONCLUSION

The paper presents early ideas for the use of the threat model in cyber-security containment decisions by summarizing key findings from the independent thesis work by (Fahlander, 2021). A set of literature-derived incidents and containment actions are modelled with MAL-based semantics in order to facilitate discriminative reasoning about effective containment decisions in computer models. An algorithm for containment strategy selection is described and its implementation outlined to elucidate the methodology proposed. The assumptions made during modelling as well as the proposed metric are also evaluated critically with a verification and validation procedure, respectively. The formalism is presented and implemented using practical examples that are then used to evaluate the feasibility of the suggested approach. Finally, the work is aimed at creating a systematic and infrastructure-tailored platform for decoding the complexity surrounding containment. It may perhaps one day improve overall cyber resilience. Note, that a prototype implementation is available at <https://github.com/mal-lang/containment-course-of-action>.

## 9 FUTURE WORK

There are a several interesting directions that future work may explore. Firstly, the current work could be extended with a larger set of incidents, containment actions and supported CSAF-flags. Secondly, research on effective containment practices could answer the question of what level of abstraction the underlying MAL-based language should work at. Thirdly, you could address performance concerns and limit the search space by for example disqualifying containment strategies that operate too far from the incident, or those that include any two containment actions redundant when combined. Finally, the methods proposed here could be used to reason about temporary load balancing between hosts or other types of resource-related interim practices.

## ACKNOWLEDGEMENTS

This research was partially supported by grants allocated by the European Union’s H2020 research and innovation programme under the Grant Agreement no. 833481 (Project SOCCRATES).

## REFERENCES

- Cichonski, P., Millar, T., Grance, T., and Scarfone, K. (2012). NIST Special Publication 800-61 : Computer Security Incident Handling Guide. <https://doi.org/10.6028/NIST.SP.800-61r2>.
- Ekstedt, M., Johnson, P., Lagerström, R., Gorton, D., Nydrén, J., and Shahzad, K. (2015). Securi cad by foresee: A cad tool for enterprise cyber security management. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 152–155.
- Fahlander, P. (2021). Containment strategy formalism in a probabilistic threat modelling framework. Master's thesis.
- Fila, B. and Wideł, W. (2020). Exploiting attack-defense trees to find an optimal set of countermeasures. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, pages 395–410. IEEE.
- Hacks, S., Katsikeas, S., Ling, E., Lagerström, R., and Ekstedt, M. (2020). powerlang: a probabilistic attack simulation language for the power domain. *Energy Informatics*, 3(1).
- Johansen, G. (2020). *Digital forensics and incident response : incident response techniques and procedures to respond to modern cyber threats*, pages 46–49. Second edition.. edition.
- Johnson, P., Lagerström, R., and Ekstedt, M. (2018). A meta language for threat modeling and attack simulations. In Doerr, S., Fischer, M., Schrittwieser, S., and Herrmann, D., editors, *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018*, pages 38:1–38:8. ACM.
- Katsikeas, S., Hacks, S., Johnson, P., Ekstedt, M., Lagerström, R., Jacobsson, J., Wällstedt, M., and Eliasson, P. (2020). An Attack Simulation Language for the IT Domain. In Eades III, H. and Gadyatskaya, O., editors, *Graphical Models for Security*, pages 67–86. Cham. Springer International Publishing.
- Katsikeas, S., Johnson, P., Hacks, S., and Lagerström, R. (2019). Probabilistic modeling and simulation of vehicular cyber attacks: An application of the meta attack language. In *Proceedings of the 5th international conference on information systems security and privacy (ICISSP)*, page 175.
- Leemon@cs. cmu. edu, L. B. (2021). What is Simulated Annealing? <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/photoz/.g/web/glossary/anneal.html>. [Online; accessed 28. Aug. 2021].
- Li, F., Li, Y., Leng, S., Guo, Y., Geng, K., Wang, Z., and Fang, L. (2020). Dynamic countermeasures selection for multi-path attacks. *Computers & Security*, 97:101927.
- mal lang (2021). coreLang. <https://github.com/mal-lang/coreLang>. [Online; accessed 22. Aug. 2021].
- McQueen, M. A., Boyer, W. F., Flynn, M. A., and Beitel, G. A. (2006). Time-to-compromise model for cyber risk reduction estimation. In Gollmann, D., Massacci, F., and Yautsiukhin, A., editors, *Quality of Protection*, pages 49–64, Boston, MA. Springer US.
- Poolsappasit, N., Dewri, R., and Ray, I. (2012). Dynamic security risk management using Bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1):61–74.
- Roberts, S. J. and Brown, R. (2017). *Intelligence-Driven Incident Response: Outwitting the Adversary*, pages 30–31. O'Reilly Media, Incorporated, Sebastopol.
- Sheward, M. (2018). *Hands-on incident response and digital forensics*, pages 134–137. 1st edition. edition.
- Soikkeli, J., Muñoz-González, L., and Lupu, E. (2019). Efficient attack countermeasure selection accounting for recovery and action costs. In *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*, pages 3:1–3:10. ACM.
- Stan, O., Bitton, R., Ezrets, M., Dadon, M., Inokuchi, M., Ohta, Y., Yagyu, T., Elovici, Y., and Shabtai, A. (2019). Heuristic approach towards countermeasure selection using attack graphs. *CoRR*, abs/1906.10943.
- Thompson, E. C. (2018). *Cybersecurity Incident Response: How to Contain, Eradicate, and Recover from Incidents*, pages 99–116. Apress, Berkeley, CA.
- TinkerPop, A. (2021). Apache TinkerPop. <https://tinkerpop.apache.org>. [Online; accessed 9. Apr. 2021].