Deception in Network Defences using Unpredictability

JASSIM HAPPA, Information Security Group, Royal Holloway, University of London, UK
THOMAS BASHFORD-ROGERS, Department of Computer Science and Creative Technologies, University of the West of England, Bristol, UK
ALASTAIR JANSE VAN RENSBURG, Department of Computer Science, University of Oxford, UK
MICHAEL GOLDSMITH, Department of Computer Science, University of Oxford, UK
SADIE CREESE, Department of Computer Science, University of Oxford, UK

In this paper, we propose a novel method that aims to improve upon existing moving-target defences by making them unpredictably reactive using probabilistic decision-making. We postulate that unpredictability can improve network defences in two key capacities: (1), by re-configuring the network in direct response to detected threats, tailored to the current threat and a security posture, and (2), by deceiving adversaries using pseudo-random decision-making (selected from a set of acceptable set of responses), potentially leading to adversary delay and failure. Decisions are performed automatically, based on reported events (e.g. IDS alerts), security posture, mission processes, and states of assets. Using this codified form of situational awareness, our system can respond differently to threats each time attacker activity is observed, acting as a barrier to further attacker activities. We demonstrate feasibility with both anomaly- and misuse-based detection alerts, for a historical dataset (playback), and a real-time network simulation where asset-to-mission mappings are known. Our findings suggest that unpredictability yields promise as a new approach to deception in laboratory settings. Further research will be necessary to explore unpredictability in production environments.

CCS Concepts: • **Networks → Network simulations**; **Network experimentation**; • **Security and privacy → Firewalls**; **Information flow control**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**;

Additional Key Words and Phrases: Network Defences, Decision Trees, Situational Awareness, Simulation

ACM Reference Format:
Jassim Happa, Thomas Bashford-Rogers, Alastair Janse van Rensburg, Michael Goldsmith, and Sadie Creese. 2021. Deception in Network Defences using Unpredictability. *Digit. Threat. Res. Pract.* 0, 0, Article 0 (2021), 26 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Present day network defences are typically deterministic: for any given observed attacker action, there is an expected reaction. Deft adversaries can use any such insights to their advantage. A significant body of work propose methods related to deception, including the use of honeypots and moving-target defences to mislead adversaries [1–4]. However, deception depends on the adversary believing reactions to be authentic. Moving-target defences involve pseudo-randomised

Authors' addresses: Jassim Happa, Information Security Group, Royal Holloway, University of London, UK; Thomas Bashford-Rogers, Department of Computer Science and Creative Technologies, University of the West of England, Bristol, UK; Alastair Janse van Rensburg, Department of Computer Science, University of Oxford, UK; Michael Goldsmith, Department of Computer Science, University of Oxford, UK; Sadie Creese, Department of Computer Science, University of Oxford, UK, firstname.surname@cs.ox.ac.uk.

2576-5337/2021/0-ART0 $15.00
https://doi.org/0000001.0000001

configuration (or dynamic reconfiguration) of systems, but do not make such decisions in direct response to the state of the system or known events. Little progress has been made on investigating the benefits of reactive unpredictability as a defence mechanism [5, 6]. In this paper, **we investigate whether reactive unpredictability can be of benefit to network defences**. We describe a decision-tree grammar, which defines how the system's defences can respond according to adversary behaviour, contextual information and a security posture. Each decision is a *Course of Action* (COA) (instruction to reconfigure the network). Performing reactive reconfigurations enable actions to be adjusted according to the perceived threat level, context and mission-criticality of assets. Our decision-tree grammar makes use of:

- **Security alerts**. Information contained in IDSs alerts can be used to inform the decision-making process, so that any decision is tailored to the threat represented by the alert.
- **Security posture** can provide global priority flags, specifying the system's current acceptance of risk and criticality. These flags could relate to willingness to accept *confidentiality*, *integrity*, and *availability* impacts, and the decision system can steer attackers towards impacts that defenders prefer.
- **Mission processes** can be mapped to assets and used to inform the decision system of any dependencies and to restrict access to critical assets under threat.
- **Asset status** can be used to estimate the impact of attackers and, thus, the optimal reconfiguration.

The contributions in our paper are as follows:

(1) **A novel deception technique** for network defences that uses probabilistic decision trees and security postures to achieve reactive unpredictability. We introduce **a novel decision-tree grammar** which enables analysts to define and determine possible responses based on alerts, security postures, mission processes, and asset states.
(2) **An implementation, demonstration and assessment** of our approach, using both a historical dataset and using a live (real-time) simulation of an organisation and its activities.
(3) **A novel simulation approach** to model complex network activities driven by underlying business processes, derived from Business Process Modeling and Notation (BPMN)[7] descriptions.
(4) **A discussion on unpredictability in network defences**, including the feasibility of deception using probabilistic decision-making in production environments, as well as suggestions for future research.

2 RELATED WORK

2.1 Deception and Unpredictability in Cyber Defences

Deception seeks to fool the adversary as a defence (and offensive) mechanism and has a long history of effective use in military contexts. The core idea is that defenders may improve the effectiveness of defences because deception will likely yield in attacker uncertainty [8–10]. Honeypots [11–15] and moving target defences [4, 16, 17] are popular approaches to deception. Moving target defences for instance, alter the external attack surface of the network in order to invalidate prior reconnaissance activities. Common network-deception tactics [18] include:

- **Concealment**. Hide valuable assets as seemingly innocuous ones and divert attackers to honeypots. These efforts aim to waste the attacker's resources and time.
- **Camouflage**. Obscure assets as moving targets by creating and removing resources at will. Software-Defined Networks (SDN) for instance can make use of virtualization for this deception process.
- **Disinformation**. Diverting or confusing attackers with incorrect information. While similar to concealment, it is broader in scope and differs in purpose: such as providing error messages or claims that files cannot be downloaded or opened when they can. Disinformation should not be easily disproved.
- **Displays/Ruses**. Adding counterfeit resources, such as fake credentials and activities on decoys, ruses offer low false positives and little bandwidth as legitimate users have no reason to interact with decoys.
- **Feints**. Pretending to succumb to one form of attack in order to conceal a second, less-obvious defence.
- **Insights**. Developing comprehensive situational awareness to shift from defending against attacks to anticipating them and acting accordingly.

Deception using unpredictable behaviour in defence systems is a relatively new approach to cyber defences. Gutzwiller et al. [19] discuss a human factors-centric approach to deception, particularly how "oppositional human factors" can aid cyber defences. They discuss how disrupting human attention through interference, manipulation of attention allocation, task interruptions, interruption anticipation, memory load and attention load on an attacker can be very effective methods to improve cyber defences. Sun et al. [5, 6, 20, 21] argue that software diversity and unpredictable behaviour in applications can improve security of operating systems, but do not explore the topic in a network defence decision-making perspective.

We postulate that introducing unpredictability in network defences using probabilistic decision making has potential to benefit network defences, akin to ideas explored by Sun et al. [5], and Gutzwiller et al. [19]. Our incident response system has a business-process centric view of the network (where one is available). Analysts can specify their own decision trees, security posture and probabilities based on what they know about the mission and assets. This aspect is crucial to understand our method, as we deem it vital to have an in-depth understanding about the core mission tasks as well as assets to be able to make good use of unpredictability in cyber defences.

2.2 Network Simulations and Red Team Exercises

Network attack data is important in many areas of cybersecurity research. Simulation of attacks is one approach to generate data to studying attacker and defender behaviour in safe environments [22–31]. Ring et al. [32] recently published an in-depth survey of simulation tools and dataset. Attacker behaviour can be studied using simulations, from historical data as 'playbacks', or during live red-team exercises. Amit et al. [33] examine challenges of realistic data set creation such as topics related to: lateral movement by attackers, lack of attacker-defender games, lack of labelled samples and certainty of ground truths, imbalanced data sets, issues with metrics, access to data sets, inability to create data of varying types of families. There are many important cyber-security data sets like Microsoft's malware data set [34], Knowledge Discovery and Data Mining (KDD 99) dataset [35], DARPA's datasets from 1998/1999/2000 [36], and Los Alamos's traffic data set [37].

Real captures are often not shared externally, may not represent a wide range of network attacks, are quickly outdated, or are anonymised (with the payload removed) which reduces their usefulness to researchers. For thorough assessment of intrusion-detection tools, it is ideal to use multiple datasets representing the range of attacks that might be faced. Captures of real network traffic often cannot be manipulated to suit user requirements, they are often unlabelled, and it can be difficult to know when an attack is actually occurring amidst a deluge of traffic, such that usability for ID evaluation purposes is low. To our knowledge, no network simulation tool makes use mission processes (e.g. using BPMN or UML). The activities that take place in our simulation are scripted according to BPMN tasks (a flow-chart model of the tasks that needs doing on a daily basis in an enterprise), and react to network attacks. If an attack has severely compromised an asset, a mission task is unable to complete. We therefore saw a need to develop such a system, in order to demonstrate the capabilities of our unpredictable defence method.

Ferguson-Walter et al. [38] makes use of behavioural science to study technologies and techniques for defensive deception in four studies using deception. The conditions were the following: (1) determine effectiveness of decoys when the red team was not made aware of the presence of decoys; (2) examine whether decoys are still effective even when the red-team knows about the presence of decoys. (3) examine whether the belief of the presence of decoys (when there are none) would be of some benefit to the red-team; (4) examine whether decoys would still be effective if the red-team had technical details about how the decoys function. Initial data indicate that during (1) and (2), subjects first questioned their tools and techniques before questioning the validity of the network. In (2) the confidence levels were noted as extremely high when assessing a real asset as fake.

Finally, Ferguson-Walter et al. [39] describe a study that empirically measures effectiveness of cyber deception on an attacker's ability to perform reconnaissance and exploitation. The paper describes their experimental design, methodology, cyber range, participant population, and planned future analyses.

3 REQUIREMENTS

Deception requires situational awareness in order to make suitable decisions in response to attacker activities. It is common convention to develop systems that behave predictably, including moving-target defences (even if the attacker might initially perceive them as unpredictable). Our approach aims to enable analysts to **codify incident responses for known situations, allowing analysts to focus their efforts on unknown situations**. Unlike previous work, we make our moving-target defences react directly to the attacker in a probabilistic manner that is weighted by a security posture. This is characterised by the presence of a decision system, which automatically updates the network's security configuration in response to live alerts. The decision system takes in contextual information, together with an alert, and outputs a COA (see Figure 1), based on a decision-tree configuration, which defines the system's security posture, mission priorities and underlying business processed modelled. This COA is passed to an actuation system which enacts the chosen COA on the system.

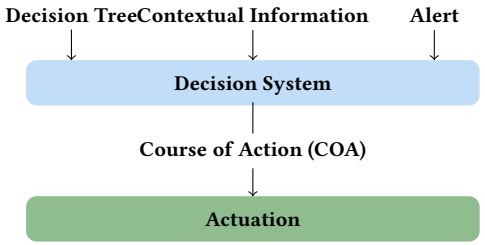

Fig. 1. I/O of our decision system. Contextual information is a list of assets and mission processes, how mission processes depends on assets (dependency trees), security posture data (flags that specify the systems' acceptance of risk and priority) and asset status (e.g. is a machine on/off, what software is installed, etc.).

3.1 Course of Action

Once contextual and alert information has been processed by the decision system it will produce a COA. **A COA contains a set of actions to be taken by the network in order to reconfigure it**. By performing reconfiguration in light of detected intrusions, the decision system can provide COAs that are tailored to the known properties of the attack. Uses include:

- **Preventing further intrusion.** By using the alert as a basis for an understanding of where the attacker is inside the network, the system can perform targeted reconfiguration to prevent further intrusion by the attacker. Such targeted reconfiguration could include disabling known (or suspected) vulnerable services that interact with compromised components, or temporarily imposing stricter controls on affected subnets. Performing this in a targeted fashion provides a stronger justification for actions to be taken that may impact service usability, because of the increased risk to the network. Additionally, controls that may have an associated cost or business impact can be enacted temporarily as a direct response to serious intrusion.
- **Mitigation.** The decision system can also suggest COAs that prevent the attacker from utilizing privileges they have already gained. By performing actions in direct response to a known attack,

these actions can be taken despite impacts to the network, in order to mitigate clear risks. This can take the form of direct reactions to attacker privileges, such as revoking stolen credentials, or through indirect actions to lessen the impact of the intrusion, such as redirecting traffic to non-compromised alternatives. For example, the decision system may decide to take a database of personal information offline when an intrusion on it is clearly detected. While this may cause impact to services provided by the database, it may be decided that the risk of data being stolen is significant enough that the impact is worthwhile. Contextual information can be used to make this decision; in some cases it may indicate that the database is too critical to be taken offline. In this case, alternative measures such as disabling remote access could be taken.

- **Reconfiguring attack surfaces.** Aside from considering the impact of the detected attack, a decision system can be used to reconfigure the attack surface of the network, in order to prevent future intrusions through the same means (or, equally, to prevent the attacker re-gaining entry once their privileges have been mitigated). This can be performed when the alert and contextual information is enough to infer likely entry-points onto the network. With these determined, COAs can be suggested that perform randomised dynamic reconfiguration, similar to typical moving-target defence techniques, that alter the external attack surface of the network and invalidate reconnaissance performed by the attacker.
- **Diverting resources.** Finally, the decision system can divert resources to the affected area, either human resources, by reassigning security analysts, or computational resources, by increasing logging and forensics in the affected parts of the network. This can be done automatically, and in response to less-confident alerts. While this information may be too expensive to gather constantly across all systems, performing dynamic reconfiguration of logging detail can provide more information in important systems without considerable overhead.

3.2 Automation

Automating decision-making in network defences has the advantage of potentially reacting to attacks while they are still in progress, making each of the four aims described above more feasible. Circumstances may arise in which confidence in the alert is not sufficient to justify a response. In this case, obtaining feedback and giving control to a human analyst is desirable. This can be performed at any stage of the unpredictable network defences model. In the simplest case, a suggested COA could contain some (or only) actions that are marked as requiring manual approval. These decisions can be sent to the actuators only once approval has been obtained. Alternatively, alert-input to the decision system does not need to come directly from IDSs, but instead could be the result of manual input. An alert to the decision system could take the form of a human-produced notification about a suspected breach. This enables the reaction to a suspected alert to be automated; when an analyst determines a particular situation has occurred, they can supply this information to the decision system which can determine an appropriate COA, taking into account contextual information. In circumstances with particularly critical systems, or where situational awareness may be poor, these two approaches can be combined, so that the decision system acts as a support system for analysts. Given a suspicion about a situation on the network, an analyst can submit a suitable alert to the decision system and receive a suggested COA in return. They can then consider each of the suggested actions and act upon those they deem necessary.

3.3 Contextual Information

Along with the details of the alert, the decision system is supplied with contextual information, which provides context for the decision. **Contextual information is independent of the particular incident that the alert relates to, but could contain information about the relevant**

hosts or other entities. For example, the contextual information could include the current workload of a host, together with its criticality in the business process. This information can then be used to enrich the decision-making process and avoid taking actions with unexpected consequences.

Aside from host-related contextual information, information about the whole state of the system can be included, such as: time of day or overall workload of the system, to enable some COAs to be enacted only when the workload is low. Security analysts can also specify contextual information manually, to give overall directives to the decision-making system. These directives, together, form a codified security posture for the system.

In the simplest case, a security posture could provide a general security level, so that analysts can direct the decision system towards greater or lower risk depending on current appetite. More specifically, the posture can direct risk to selected areas and harden defences based on known incoming threats. For example, a company might anticipate an increased threat from insiders, and can reconfigure their security posture so that it provides more controls focused on countering this threat. If similar companies have faced a string of attacks, security analysts could configure the security posture to combat the expected techniques of the other attacks.

Defining the security posture through contextual information separates the implementation of the posture from its specification, enabling analysts to implement COAs relevant to each potential situation, and then rapidly decide on a security posture in response to changing circumstances.

4 SPECIFICATIONS

Our implementation is intended as a practical demonstrator, in which analysts have full control and specify precisely the decision-making process and COAs that result. Other implementations may reduce this control in favour of allowing machine learning techniques to aid the decision system, increasing the possible complexity of decisions and the number of factors that can be considered.

In our implementation, analysts define the system's responses to intrusion by specifying one or more decision trees associated with a business process task. Each tree takes as input the alert and contextual information; both of which can contain any relevant information that may be wanted by the decision system. The distinction between the alert and contextual information is that the alert is generated in direct response to a specific incident (or by an analyst) and the contextual information relates to the state of the system and is independent from a particular incident.

Decision trees are described through a custom grammar, designed to be simple and lightweight so that trees can be readily understood. Each (non-leaf) node in the tree is an expression, and each child is labelled with a condition on its parent's expression. When processing an alert, the tree begins at the root node and proceeds recursively by evaluating the expression and then testing each child condition, in order. Upon reaching a satisfied condition, the process repeats using the expression of that child. Further child conditions are not evaluated once a condition has been satisfied, so that each decision tree reaches a single decision. Leaves of the tree are action nodes, which define the COA to employ when that node is reached. Here, the COA is a single action, but this could be replaced with more complex combinations of actions.

Figure 2 shows a simple example tree. The tree is labelled as `SimplestTree`, and contains four nodes, three of which are leaves. The non-leaf node contains the expression `alert.priority`, which evaluates to the `priority` property of the alert that triggered the tree. The leaves are labelled with the values 1, 2, and 3, determining the COA that should be taken for each priority. To process an alert on this tree, the system will first evaluate the expression of the root node `alert.priority`, and then compare this value to each of the children in turn. When reaching the child corresponding to the evaluated value, this process will stop and the system will repeat this process using the child node. Here, the child node is an action, so the system will stop and return this action. If no arguments are provided the actions use a default set.

```
~SimplestTree~
alert.priority? (
          1: STOP
          2: DENY
          3: REPORT
)
```

Fig. 2. A simple tree, color-coded with name, expressions, conditions, and actions. Actions use OpenC2 vocabulary [40].

4.1 Basic Tree Specification

The tree parser accepts trees inputted in a plain text format. The ~ (tilde) symbol signals the beginning and end of a tree name. A tree file can define as many trees as required.

4.1.1 Expressions. Expressions act as questions asked by the decision tree, and evaluate to a value which is then tested against each of the conditions in its child nodes. Expressions take one of the following forms:

`value`	Evaluates to the value of the named input value, such as properties of the alert or contextual information.
`%`	Evaluates to a random number between 0 and 1, which is used to make trees probabilistic (see Section 4.2).
`TreeName<`	Evaluates to the decision of the named tree. The tree is executed using the same inputs as the current tree, and its value is returned. This enables trees with common parts to be reused to save time and reduce code duplication.

4.1.2 Conditions. Each condition is tested against the value of its parent expression until a matching condition is found. In general, conditions consist of a comparison operator and a value, which is compared against the value of the expression using the operator. Conditions are specified using the following comparison operators:

`=` `!`	Test for equality and inequality respectively. If no operator is specified, the tree assumes equality is being tested.
`<` `<=`	Test for the corresponding inequalities. Equivalents for greater than are also available.
`@`	Tests for the existence of the value as a substring of the expression.
`*`	Always matches. This can be used as the final answer to catch all otherwise-unmatched values.
`&` `|`	combine the results of two other operators in "and" and "or" operations respectively.

Each operator is paired with a value which is compared by the operator against the expression (with the exception of `*`, which does not need a value, and `&` and `|`, which combined two other operators).

4.1.3 Actions. The leaf nodes of the trees are referred to as actions, and typically represent an action (or set of actions) for the decision system to suggest. Actions can have additional arguments specified in brackets after their name, so that additional detail can be specified. Arguments prepended with `$` are treated as expressions, which are evaluated before being returned.

In general, any string can be an action, and when a subtree is called, the expression evaluates to the string representing the action that results from it. Through this, subtrees can return strings that do not correspond to actions, so that the parent tree can use the result string in its decision-making process. This is illustrated in the example in Figure 3, where a tree is called that returns one of `manual`, `graceful`, or `immediate` which is then used to determine the action of the parent tree. This example tree contains two trees, `SyntaxTree` and `STOPMethod`, and illustrates and example of our decision tree grammar. `SyntaxTree` uses the subtree to determine the desired method of shutdown in a reusable fashion. The condition `2|3` is equivalent to `=2|=3`, which tests if the expression is equal to `2` or `3`, and the condition `*` catches all unmatched values, so that if `alert.priority` is not

1, 2 or 3 it will result in NOACTION[1]. The STOP action is present on two leaf nodes, with a different argument (specified in square brackets []) each time.

```
~SyntaxTree~
entity.is_on? (
     true: alert.priority? (
          1: STOP[\$ STOPMethod<]
          2|3: STOP
          *: NOACTION
         )
     *: REPORT
)

~STOPMethod~
entity.is_critical? (
     true: manual
     false: entity.active_users? (
          <=10: immediate
          *: graceful
         )
)
```

Fig. 3. Two simple trees demonstrating the syntax of the tree grammar. The STOPMethod tree is called as a subtree of SyntaxTree. This shows how decision trees can be used to determine parameters of an action about to be taken. Our naming convention is to use the suffix Tree to as a generic tree for action decisions, while the suffix Method to highlight to the tree developer that this tree relates to action parameters.

4.2 Probabilistic Trees

Our strategy to potentially confuse and delay the attacker is to insert probabilities in the decision tree. Probabilities can be added at each level if the analysts deem it to be appropriate to place there. It is worth highlighting that **assigning probabilities and action parameters is always optional**. Indeed, some decisions may (and perhaps should always) be non-probabilistic (e.g. some safety critical systems as determined by risk-owners). It should be up to the analyst during the tree creation stage to decide whether to insert probabilities or not. Figure 4 shows a simple example of a probabilistic tree, with examples of action parameters being explicit. **Where no parameter is specified in the tree, default parameters are used.** Action can be made on the source (attacker), target (victim) or a separate network defence asset (e.g. Firewall).

The choices in which how to build an effective probabilistic tree should be dictated by the requirements and risk appetite of the individual analyst and organisation. We assume that all probabilistically available choices should exist if they are a) valid, and b) weighted correctly[2]. Judging whether the weighting is done correctly is a challenge that can be resolved heuristically by the analyst, or by making some basic assumptions. SimpleTree02 in Figure 4 shows that: `1: %? (<0.5: STOP[device,shutdown,method=immediate] *: DENY[port,80,method=blackhole])` can be read as: if the ML alert is of severity 1, generate a random number between 0 and 1. If the

[1]Note: NOACTION and SUPPRESS are non-OpenC2 actions we use. These are discussed further in Section 7.7.
[2]We deem the means to accurately determine probability values out of scope for this paper as this has the potential to become a research paper in its own rights. For the purposes of this paper, we assume that it is possible to determine them using heuristics and network security testbed experiments.

```
~SimpleTree02~
  entity.type? (
   Machine: entity.is_on? (
       false: REPORT
       true: entity.type? (
            MLAlert: alert.priority? (
              1: %? (<0.5: STOP[device,shutdown,method=immediate]
                        *: DENY[port,80,method=blackhole]
                     )
              2: %? (<0.5: DENY[port,80,method=blackhole]
                        *: REPORT)
              3: REPORT
            )
            IDSAlert: alert.priority? (
              1: %? (<0.5: DENY
                      <0.7: STOP
                         *: CONTAIN
                     )
              2: %? (<0.5: REPORT
                     *: SUPPRESS)
              3: SUPPRESS
            )
        )
    )
    Person: SUPPRESS
 )
```

Fig. 4. A probabilistic tree example. %? generates a number between 0 and 1.

value is above 0.5, shutdown the device immediately. If the pseudo-random number generated is anything above 0.5, blackhole any traffic to port 80 on the device. If another value between 0.5 and * existed, e.g. `1: %? (<0.5: STOP[ device,shutdown, method=immediate] <0.8: DENY[port,80,method=blackhole] *: REPORT)` should read as: if the ML alert is of severity 1, generate a random number between 0 and 1. If this value is above 0.5, shutdown the device immediately. If the number is anything greater than 0.5 but below 0.8, then blackhole any traffic to port 80 on the device. For anything above 0.8, then REPORT.

4.3 Security Posture Tree

We define a Security Posture is at its core to be **a set of flags that enriches the options available for traversing a decision tree** (including the probabilistic component). This enables analysts to be able to encode security priorities. Figure 5 shows a simple example of a Security Posture flag used in a decision tree. We can query the priorities set by an analyst, where we check for a context flag. If c is true, we can STOP, if false, we can DENY. Here, we do not even check the alert, we simply check whether the context flag c has been set, enabling for fast reconfigurations prior to alerts coming in (if the network defence needs to quickly switch its security priorities).

```
~SimpleTree03~
  context.flag-c? (
    true: STOP
    false: DENY
  )
```

Fig. 5. The simplest security posture tree example, identifying whether the confidentiality flag has been set.

Probabilistic trees and security postures can be combined. This increases complexity of the decision system using another layer of decision-influencing factors. This second layer (Security Posture) sets flags to favour some routes over others, see Figure 6.

```
~SimpleTree04~
  entity.type? (
   Machine: entity.is_on? (
      false: REPORT
      true: alert.event_type? (
        MLAlert: alert.priority? (
          1|2: CIA<? (a: REPORT
                      c:STOP
                      i:DENY
                      *: %? (<0.8: DENY *: STOP))
          *: CIA<? (a: REPORT
                     *: %? (<0.8: CONTAIN *: DENY))
        )
        IDSAlert: alert.priority? (
          1: STOP
          *: CIA<? (a: REPORT
                    *: %? (<0.8: DENY *: STOP))
        )
      )
    )
    Person: SUPPRESS
  )
```

Fig. 6. Example tree identifying which flags are set in the program. The CIA< expression then queries the CIA tree in Figure 7 to identify any security posture to take into account.

```
~CIA~
  context.flag-c? (
    true: context.flag-i? (
      true: context.flag-a? (
        true: cia
        false: ci
      )
      false: context.flag-a? (
        true: ca
        false: c
        )
      )
      false: context.flag-i? (
      true: context.flag-a? (
          true: ia
          false: i
        )
        false: context.flag-a? (
          true: a
          false: n
        )
     )
  )
```

Fig. 7. Security posture tree example. Here we can query any CIA priority flags have been set.

In SimpleTree04 we show that by nesting the CIA query out as a separate tree (CIA, Figure 7), we can input the results of that tree into another. Note that the security posture is a flag that globally

applies to all BP tasks, but could also be set to individual BP tasks. CIA are simple boolean flags. This is done in the interest of allowing analysts more flexibility when designing trees – however, we see it also possible to define flags to be visible to some BP tasks only. We use CIA as examples of what one might use flags for. The benefit of using flags is that there is nothing preventing an analyst from using this flag mechanism in other ways, such as: only to make a certain decision in the tree during working hours or only if a specific person is available. Currently we set the flags to be true or false during startup, but these could be manipulated during run-time as well.

4.4 Architecture: Data-Processing Pipeline

Our full pipeline is designed around three core modules as shown in Figure 8, all of which implemented in Python:

A Machine Learning (ML) Module. The primary purpose of the ML module is to demonstrate that our decision system works with anomaly detection systems as well as misuse detection alerts. The ML module loads a list of known entities (digital assets and people) that belongs to the organisation, builds a training profile on them based on ten features, generates anomaly alerts using a user-define standard deviation threshold from observed raw logs such as network traffic activities (e.g. packets or Netflow). These anomaly alerts as well as other misuse detection alerts (in our DataSim assessment, we used Snort and our own MLModule) are then forwarded to the decision system.

A Decision System (DS) Module. The decision system consumes alerts and maintains the current state entities (assets belonging to or external to an organisation). The alerts the decision system consumes come from both the ML module, and from other sources of alerts (such as AV, IDS or any other alert that can be parsed), and produces COAs on how to act on these alerts. The decision system creates COAs by consuming alerts and traversing a decision tree, checking whether the alert pertains to a mission task, and context to make a decision. Context is derived from knowing the current state of an entity, understanding how an alert and entity relates to a BP tasks and the BPs priority, and is checked against a security posture.

A Visual Analytics (VA) Module. This module presents decisions, entities and alerts to analysts (e.g. in a SOC). The purpose of this module is to provide analysts with insight about the tool's behaviour. We implemented this module as a dashboard using bokeh (https://bokeh.pydata.org/).

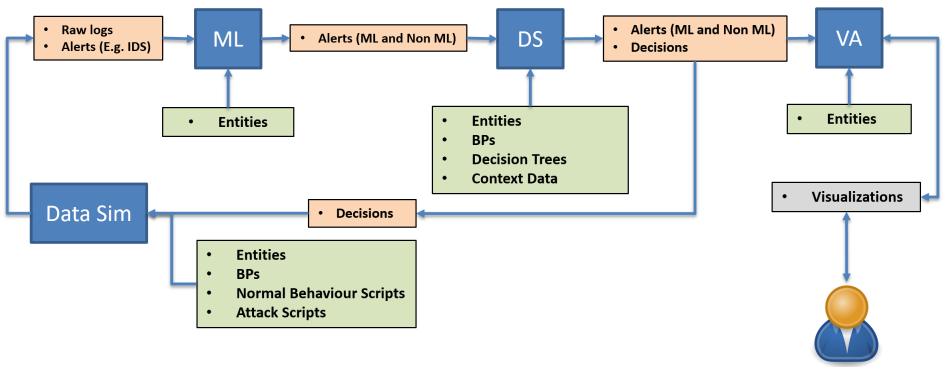

Fig. 8. High-level view of data flow for the tool. Blue boxes represent our implementation: Machine Learning (ML), Decision System (DS) and Visual Analytics (VA) respectively. DataSim is expanded in Figure 9. Green boxes indicate configurations being loaded in, while orange boxes indicate data flowing through the system.

There are two types of I/O: data read into the system to be analysed (orange boxes), and data loaded into the system to be used to aid analysis (green boxes). The orange boxes are the various alerts and decisions that get passed on between modules, whereas the green boxes indicate configurations that are loaded into the system during startup.

Entities are directly loaded from a configuration (JSON files), and this configuration is loaded into all three modules, allowing for the same entities to be used for different purposes: one for ML processing, one for making decisions and the final for the processing of statistics to present to the viewer. This is done to keep the modules separate, following a *loosely coupled, highly cohesive* design. BPs and decision trees are also loaded into the decision system, the only module (apart from the Data Simulator) requiring more information than simply the *internal entities* (those belonging to the enterprise). To aid our entity creation efforts, we built a standalone tool to pseudo-randomly create entities that meet the requirements of any particular DataSim simulation.

In the full pipeline, the decision system begins by loading in BP tasks, which entities those relate to and other contextual data (e.g. Security Posture flags) as well as the decision trees themselves. After having loaded a configuration, it awaits new alerts (either from the ML module or a third-party application). Once new alerts have arrived, the decision tree is traversed, if a decision can be made based on the new alert, a decision is sent out. After a COA has been sent out the state returns to idle, awaiting new alerts to arrive.

5 EXPERIMENTS

To demonstrate the value of our approach, we implemented a prototype system and performed two tests using different scenarios: a historical dataset (captured), and a live/reactive simulation. These were chosen to validate our system in two settings:

(1) **Historical Dataset**. The historical dataset is a capture of events in a real-world system, but is not *reactive*, meaning that decisions made can be reported, but not acted upon. This also is representative of the common scenario where BPs are either not clearly specified or unknown. We tested our system on the CRATE dataset [41], which has unknown BPs and entities from the perspective of our system. This consists of logs of network traffic and IDS events and demonstrates real-world network traffic applied to our system, and the scalability of our approach, see Section 5.1.
(2) **Simulated Environment**. The second scenario was to use a simulated environment. This allows for a controlled assessment of our system under a wide range of conditions and attacks, which can be fully specified. This also allows complete knowledge of the network, entities and BPs. This system is *reactive* allowing for the simulation to respond to attacks in a configurable and realistic manner. Crucially, the simulator-based approach removes the requirement of validating the proposed defence mechanism on a real network, while at the same time understanding the performance of various probabilistic trees without running any risks to existing systems. As no existing simulator is able to simulate machines, people, networks and BPs at a fine-grained accuracy, while providing scripting capabilities to emulate a real network environment, we build a custom simulator, as outlined in Section 5.2.

5.1 Historical Dataset (CRATE)

We used the Swedish Defence Research Agency's CRATE dataset [41] as a stand-in as a relevant dataset for scalability-testing. Our emphasis has been to build a robust, scalable system that works for relevant data, i.e. data that would work in real environments. As this data is pre-recorded, the decisions cannot be actively applied to the entities in the system; however it allowed us to test the performance and scalability of the tool on real data. In this case, we do not use information about

the structure of the network on which this data was gathered, and we have no information about any BPs that were running. We therefore used a default tree in this experiment, which could only make decisions based on the selected security posture and alert severities.

Using the tool in this non-reactionary circumstance additionally allows at least two uses:

- **As a playback-investigation tool**. Playback of past events can help analysts study how they could have performed better in the past, as well examine weaknesses of the existing configurations.
- **As preparation for live missions**. Playback can also provide insight that can help improve present day network defences prior to actual missions.

The CRATE dataset consists of 8.3GB NetFlow logs and IDS sensor outputs. Each NetFlow log stores details about traffic flowing through the sensor machine, and the IDS is implemented using Snort. We considered a subset of the data where timestamps of the NetFlow logs and IDS data overlapped, comprising of 2.2GB log files with approx. 24 million entries. We sorted the data and streamed the log files minute by minute. This allowed us to evaluate the data throughput of our tool in a realistic context in terms of the volume of data. The system was trained on the first hour of network traffic, and was run from the start of the second hour.

5.2 Simulated Environment (DataSim: Design and Implementation)

Our decision system is designed to be used in a live environment where decisions affect the state of the network in consideration. For example, if a `STOP` decision is applied to a machine and acted on, then the machine would be powered off, resulting in no network traffic from that machine. In a pre-recorded dataset, this would not be the case. Therefore, in the absence of a real network on which to test this tool, a simulator was developed that mimics the behaviour of a real network, and is reactionary to the decisions made by our decision system. This also provides a controlled environment where the decisions and their impacts on business processes an be demonstrated. It is worth noting that DataSim is not specific to our decision system, and can easily be applied to other investigations of network performance or network defence.

5.2.1 DataSim System Model. The architecture of the DataSim tool supports a scripting language that describes activities between entities as specified by the classification of activities scheme and BPs, and we use this system to test and optimise performance of our tool. DataSim is built on using business processes as a way to describe day-to-day activities and assign these tasks to the people and network nodes in the simulation. The system is implemented as a graph with scripting capabilities. It is reactionary in the sense that each node in the simulation can be affected by external forces (such as attacks). Internally, the simulation keeps track of where in the business process it is, and generates activities on entities on a schedule. We simulate network traffic, people active on those machines, power generators, IDS sensors and firewalls.

At the outset we intended to create more types of data sources, but have continually had to make pragmatic decisions in the interest of keeping the datasets used realistic. This means that in principle, and without much effort, it would have been possible to write sensors that keep track of a number of non-networked devices. However, as a key requirement of this work has been realism, we have cut these down to realistic data fields, meaning that in principle, with not much effort we believe we could deploy the decision system in an organisation – simply because our focus has been on realism, at the expense of exploring richer, more heterogeneous datasets.

5.2.2 DataSim Architecture. DataSim is composed of multiple layers, each responsible for simulating the different aspects that can impact our tool. Figure 9 shows our architecture of the DataSim. The top level of the simulator (the grey box in Figure 9) consists of simulation of Business Processes. The role of the Business Processes is to define the function performed by the organisation, in

the case of the scenario, a medication support mission from a Main Operating Base and Forward Operating Base. These are described by a directed graph linking individual operations performed by the organisation. Associated with each node in the graph is an encoding, via a scripting language, of how entities in the next layer are supposed to perform these tasks.

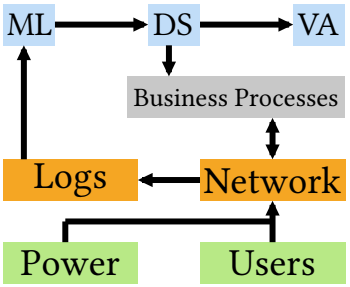

Fig. 9. DataSim overview showing the interaction between the tool and different layers of the simulator.

The digital entity layer (orange boxes in Figure 9) comprises the simulation of digital entities. These are the computers, servers, routers, IDS, firewalls, etc. that make up the physical network. Each of the entities in this layer produces and consumes network traffic, which if passed through a simulated NetFlow or IDS sensor, will produce the log files required for the tool. As the tool requires statistics about general flow of network traffic rather than precise information about each packet, the simulation of network traffic was conducted via a group of packets, conveniently akin to each entry in a NetFlow log. This network traffic is produced from small, programmable scripts which we call '*Simulated Machines*' (SM) which are designed to produce a wide variety of network traffic. In addition, these scripts can specify attack behaviour, such as a DDoS, data exfiltration, or a worm attacking a SCADA system. The next layer consists of two types of simulations. The first is a power layer (green box in Figure 9) which is connected to the entities, and is a requirement for the entities to produce network traffic. This allows the effects of an attack on a generator to be simulated, and the resulting consequences of the network and Business Processes to be examined. The second is Users (yellow box in Figure 9), which are connected to entities. This allows an increase in network traffic during working hours to be simulated.

5.2.3 Network Simulation. The output of the simulator is two types of log file; NetFlow and IDS (based on Snort). These log files are formed as the result of network traffic in the simulation passing through a sensor mimicking NetFlow or an IDS. Instead of performing a packet level simulation which would require the extensive engineering task of replicating not only packet structure, but large portions of the underlying software which generate packets, we chose to run the simulation at a higher level, that is statistics about groups of packets. These quantities are exactly those required by NetFlow, and we label the groups of packets in the simulation as NFPackets. These contain the following information: Source IP Address; Source Port; Destination IP Address; Destination Port; Number of packets summarized in the NFPacket; Number of bytes summarized in the NFPacket; How long the NFPacket was in transit; Protocol used (e.g. TCP, UDP etc); An attack ID index for a table of IDS responses. This can be used when an attack is simulated; and a business process query to transport flags associated with BPs around the network.

5.2.4 Entity Simulation. As stated before, the network is comprised of entities. Each of these has common shared attributes, such as a list of scripts or SMs, a software configuration, a list of vulnerabilities, a set of associated users, a list of open ports, and an auxiliary quantity. The auxiliary

quantity allows the entity to be associated with, and control an external process, such as a generator. Entities in the simulation consist of the following types: Clients, Servers, Routers, IDSs, Firewalls, Generators, External Nets (a catch-all entity which encompasses the external communications infrastructure).Entities are simulated through the use of programmable Simulated Machines (SM). These generate network traffic in a manner similar to real programs running on physical hardware. Each SM consists of interpreted custom assembly code comprising the script, a 32 element register file of 32 bit registers, a program counter (pc) for flow control, and optional arguments allowing program reuse. The scripting language for the assessment can be found in Table 1 in the Appendix.

This scripting language allows entities to send NFPackets to other entities. In order to send a packet, the *send_packet* instruction is used with arguments specifying the target IP, and a flag detailing how to construct the packet. Details such as protocol, size (from which bytes and duration are derived), the flag associated with a BP, and an optional Attack ID to be used for the IDS are encoded into a 32 bit unsigned integer, and used to build the packet. For more information, please see Appendix 1. The auxiliary quantity q is represented by a value $\mathbb{R} \in [0..1)$ which describes the physical quantity being simulated. In addition, a time derivative is supplied stating how the quantity changes over time $\frac{dq}{dt}$, and to add randomness, the quantity is allowed to very according to a distribution at each time step $q^{'} \sim q + \mathcal{N}(0, \sigma^2)$ where $\mathcal{N}(0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$; i.e. a zero-mean Normal Distribution, with a variance σ^2 associated with the quantity. Samples are generated using the Box-Muller method [42] and are added at each time step. An example use of the auxiliary quantity is to model the state of supplies which may need ordering, information which maybe be required for successful completion of BPs. Simulation of this auxiliary quantity allows this to happen in a predictable and controllable manner. An example program that loads a target into register 0, and a flag into register 1, then sends a NFPacket to the target IP is shown below:

```
arg_to_register   0   0
arg_to_register   1   1
send_packet       0   1
exit
```

Entities also have associated vulnerabilities which can be exploited by SMs with the aim to reproduce the behaviour of malware, DDoS, SCADA attacks and worms. To facilitate the implementation of the various attacks, SMs can be propagated around the network, or be configured to mimic the types of network traffic associated with these attacks, see Section 6.2.

5.2.5 Business Processes. The simulation of BPs is performed by moving from node to node in a directed graph of BPs. Each BP has an impact on network traffic, for example if a task is to send a report from a machine to a server, then the network traffic required to accomplish this task must be generated. Likewise, if the network has been compromised, and traffic cannot move between machines, the state of the BP needs to reflect this. The operations on the network assigned from BPs are specified via a scripting language. This allows a wide range of operations to be supported, such as querying state of the entities in the network, their auxiliary quantities, and can launch SMs on the network to produce traffic. BPs can additionally assign a flag specific to the BP which is also transported around the network, and can be used to signal the delivery of information (e.g. a report) from one machine to another. This scripting language is outlined in Table 2 in the Appendix. An example of using the BP script is shown below. This finds an available machine in a range of machines, and runs the code from Section 5.2.4 on the found machine, then moves to the next BP:

```
find_first_free_machine  192.168.10.[100 - 107]
build_send_flag          10  FTP  1
run_program_on_machine   Communicate  192.168.20.209
```

5.2.6 Implementation. DataSim was implemented in C++ for speed with an average of 140, 000 NetFlow entries per second, each corresponding to a simulated flow of data through the system, from which, corresponding IDS alerts are generated (Snort). The number of NFPackets can be throttled to more realistic amounts. Routing between entities is implemented by a routing table generated for each entity responsible for flow of network traffic, typically routers. This happens at startup, and takes 50ms to construct the network used.

5.2.7 DataSim Scenario. Our system is based on a fictional military infrastructure consisting of a Main Operating Base (MOB) and Forward Operating Base (FOB). The simulation had 166 known Entities monitored on 81 processes. These Entities included: routers, generator/SCADA, clients, servers, firewalls, IDSs, and people. The purpose of the monitored organisation was to conduct medical-support missions, which include the use of cyber-physical power systems, people, machines. The MOB (among other things) generates medical support missions and send them to the FOB. Both have power supplies and both use clients, routers, servers, firewalls and IDSs to conduct their mission and to protect their infrastructure. The FOB conducts medical support missions in two areas: at the base, but they also send out patrols on a daily basis. People entities work from early morning to late afternoon. Machines have three types of hardware: A server hardware, a client hardware and a router/firewall hardware (Cisco ASA). The software includes, MSOffice/Libre Office, Firefoxv48, AcrobatReader, WinSCP/gFTP, Edge/Chromium, CiscoAnyConnectv3, and Outlook. Each of these have vulnerabilities associated with them. This static list that is unknown to the people entities in the simulation. The IDSs use Ubuntu, Windows and Cisco ASA firmware (firewall).

5.2.8 Adversary Model. In DataSim the adversary's goal is to severely compromise the confidentiality, integrity and availability of core systems in the military organisation. The adversary can freely choose who to attack on the network. This can be achieved pseudo-randomly or via a GUI to give analysts the option to customise their order of a small library of attacks. The attacks include: reconnaissance, data exfiltration (confidentiality), disabling the power generators (availability), DoS-ing the network (availability), and finally using a worm to propagate the network and modify system files (integrity). We assume the adversary to be perfect and fully potent [43]. If the attacker can be successful, then they are (i.e. perfect), and the maximum harm is always achieved (fully potent). This approach assumes worst-case scenarios for attackers, and enables us to consider how our decision system would behave in worst-cases.

6 RESULTS

Next, we present results for our implementation using the CRATE dataset and DataSim simulation.

6.1 CRATE

As discussed in Section 5, we used the CRATE dataset as an example of a worst-case configuration for the analyst. CRATE is a use case in which the analyst knows very little about what assets they are defending (beyond IP ranges). We also use this dataset to demonstrate our ability to make decisions on historical activities. While running the tool, we piped the dataset through tcpreplay to replay the events. There are no BP or mission descriptions associated with the dataset.

Figure 10 shows the main dashboard of our system, indicating the number of alerts, decisions and top entities with both alerts and decisions. This shows that the proposed tree, even with no context about the underlying network and BPs, is able to make decisions. The efficacy of this decision making process is less relevant for historical data, as the decisions cannot be acted upon in retrospect. This figure also shows the volume and origin of the alerts and decisions, and a summary display of values over time. The purple line shows ML alerts, the blue line shows alerts from the IDS and the black line shows the volume of decisions made by our system. Our Python system processed

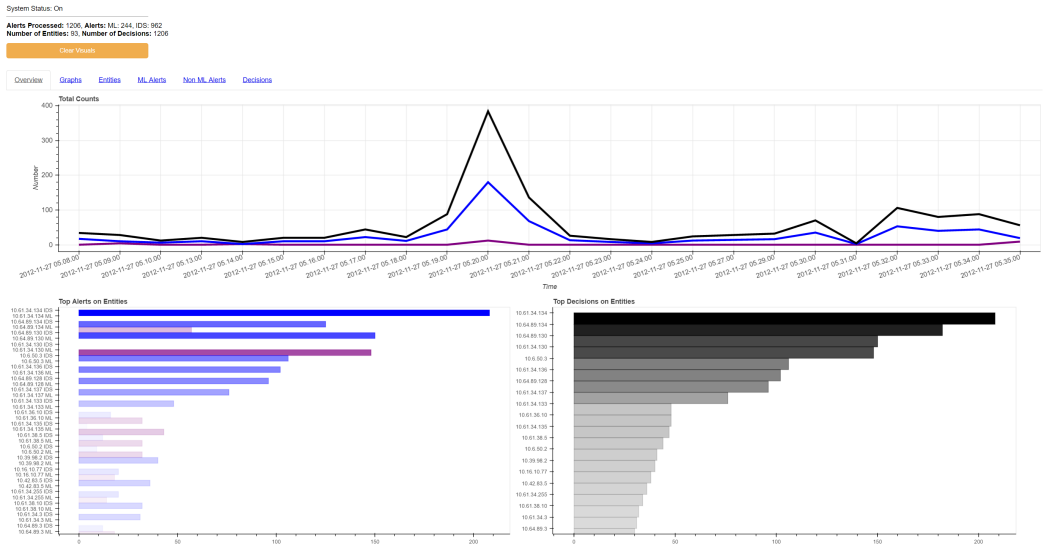

Fig. 10. Visuals while running the CRATE dataset. The tool identifies new entities, and the visualization tool shows which assets have had actions taken on them, alongside summary information over a time window (top).

2012-11-27 05:01:14.863	act	10.61.36.6	DENY	port	80	method=blackhole
2012-11-27 05:01:14.863	act	10.61.36.10	DENY	port	80	method=blackhole
2012-11-27 05:01:14.863	act	10.61.36.10	DENY	port	80	method=blackhole
2012-11-27 05:01:14.863	act	10.61.36.6	DENY	port	80	method=blackhole
2012-11-27 05:01:15.404	act	10.61.36.10	STOP	device	shutdown	method=immediate
2012-11-27 05:01:15.404	act	10.61.33.105	STOP	device	shutdown	method=immediate
2012-11-27 05:01:15.404	act	10.61.36.10	NOTIFY	email	SOC_analyst	method=immediate
2012-11-27 05:01:15.404	act	10.61.33.105	NOTIFY	email	SOC_analyst	method=immediate
2012-11-27 05:01:16.561	act	10.61.36.10	NOTIFY	email	SOC_analyst	method=immediate
2012-11-27 05:01:16.561	act	10.61.33.105	NOTIFY	email	SOC_analyst	method=immediate
2012-11-27 05:01:16.561	act	10.61.36.10	NOTIFY	email	SOC_analyst	method=immediate
2012-11-27 05:01:16.561	act	10.61.33.105	NOTIFY	email	SOC_analyst	method=immediate
2012-11-27 05:01:29.102	act	10.61.34.137	STOP	device	shutdown	method=graceful

Fig. 11. A selection of early decisions made on the CRATE dataset showing timestamps, entities, actions, and information related to the selected actions.

events in the CRATE dataset faster than real-time, taking 0.13s per 500, 000 events. Figure 11 shows a selection of the decisions taken on the CRATE dataset, and illustrates a rational approach to preventing an attack on a machine. In this case blocking a port, then shutting the machine down, then notifying a SOC analyst about what has happened. As stated before, this is an illustrative example of our system, and if deployed could be configured to respond in different ways.

6.2 DataSim Attacks and Results

A series of attacks were created using the SM scripts to simulate various attack behaviour. These simulated attacks are launched manually by an analyst using a interface which allows available attacks to be selected an run on, or targeted at, specific machines. Our decision system handled the attacks by relying on sensors on the network to report them as alerts to the decision system. Based on the results of the decision system, the actions are applied back into the simulator environment, and the state of the network then reflects the results of the decision system in a reactive manner.

Note that these attacks can be applied concurrently, although we show results for individual attacks for clarity. The following sections describe the attacks, and how the decision system reacted to these attacks. Below follows details about each attack:

6.2.1 Port Scan. The simplest attack tested was a port scan. This originated at the EXTERNAL NET and targeted a server. A NFPacket was constructed with an attack ID corresponding to a port scan. Figure 12 shows the resulting decision from the tool, and the following is the SM code used to generate this attack:

```
arg_to_registers  0  0
init_to_register  1  2147749888
send_packet       0  1
exit
```


Fig. 12. Example resulting decision from a Port Scan

In this case the tree made the decision to query the software running on the device, allowing an analyst to decide whether the machine is vulnerable. This decision may change depending on the profiles selected, or if a different tree is used.

6.2.2 Data Exfiltration. The second attack consisted of data exfiltration. This assumed a CLIENT had been compromised and was sending large amounts of data to the EXTERNAL NET. Figure 13 shows a visualization of this attack in the visuals, and Figure 14 shows decisions made to attempt to combat this attack. The following is the script used to create this attack:

```
arg_to_register   0  0
init_register     1  205520896
init_register     2  0
arg_to_register   1  3
compare_register  2  3  8  5
send_packet       0  1
add               2  1
loop              4
exit
```

The decisions resulting from this attack are to investigate and contain the activity on the machine. As can be seen in the dip in number of alerts (the purple line) in Figure 13, our system successfully stops the data-exfiltration attack.

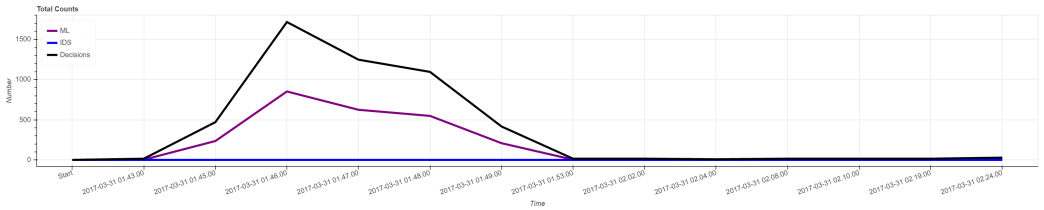

Fig. 13. Visualization of NetFlow traffic during data exfiltration, and number of decisions

2017-03-31 01:48:48.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:48.000	act	MOB-S-S07	SNAPSHOT	device	memory	method=immediate
2017-03-31 01:48:48.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:48.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:48.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:49.000	act	MOB-S-S07	SNAPSHOT	device	memory	method=immediate
2017-03-31 01:48:49.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:49.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:49.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:49.000	act	MOB-S-S07	CONTAIN	device	whole	method=immediate
2017-03-31 01:48:49.000	act	MOB-S-S07	INVESTIGATE	device	memory	method=immediate

Fig. 14. Example decisions made on the affected entity during data exfiltration

6.2.3 SCADA Attack. We simulated an attack on a SCADA device, in this case a portable power generator. This resulted in entities associated with this device also losing power. The resulting decisions from this attack are shown in Figure 15, and the SM script follows:

```
init_register  0  0
init_register  1  4026798080
send_packet    0  1
off
```

The decisions made on this critical asset correspond to restarting the generator machine and updating the software.

2017-03-31 02:05:22.000	act	FOB-MT-POWER1	UPDATE	device	software	method=immediate
2017-03-31 02:05:22.000	act	FOB-MT-POWER1	RESTART	device	power	method=immediate

Fig. 15. Decisions made on a critical SCADA Entity, in this case a generator

6.2.4 DDoS. A DDoS attack was simulated as origination from the EXTERNAL NET to a server in the organisation. The significant increase in network traffic is shown in Figure 16, resulting decisions in Figure 17, and the script used to generate this attack is shown below:

```
arg_to_register  0  0
init_register    1  205520896
send_packet      0  1
loop
```

In the case of a DDoS attack, the decisions made by the tool were to deny access to a port, reboot the machine, and check for vulnerabilities. While these actions cannot stop the DDoS attack, they can attempt to ensure the integrity of the affected entity.

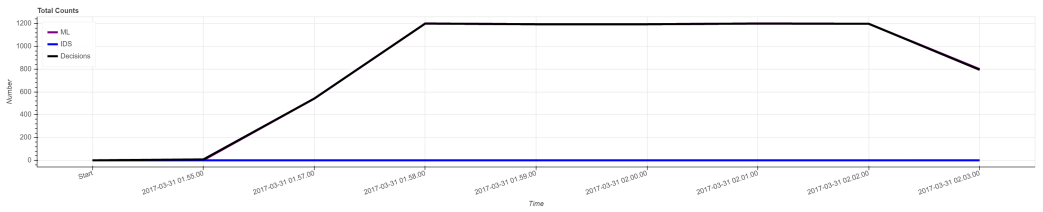

Fig. 16. Visualization of NetFlow traffic during a DDoS attack

2017-03-31 01:58:17.000	act	MOB-S-S02	DENY	port	value	method=immediate
2017-03-31 01:58:17.000	act	MOB-S-S02	RESTART	device	power	method=immediate
2017-03-31 01:58:17.000	act	MOB-S-S02	SCAN	device	known_vulnerabilities	method=Nessus

Fig. 17. Example decisions made during a DDoS attack

6.2.5 Worm. Finally, we simulated a worm propagating through the network. This moves from vulnerable machine to vulnerable machine until it finds its target and then starts data exfiltration. This also communicates with the EXTERNAL NET, and raises both ML alerts in combination with IDS alerts. The resulting decisions are shown in Figure 18, and the following script is used to produce this behaviour:

```
exit_if_running
arg_to_register   0  0
is_ip             0  1
compare_value     1  1  16  4
find_vulnerable   2  0
send_program      2
init_register     0  0
init_register     1  3221491712
send_packet       0  1
rand_to_register  1  10
add               1  10
init_register     0  0
compare_register  0  1  15  13
add               0  1
loop              12
exit
init_register     0  0
init_register     1  205520896
init_register     2  0
compare_value     2  10  23  20
send_packet       0  1
add               2  1
loop              19
exit
```

The resulting decisions attempt to contain the data exfiltration and investigate the cause.

2017-03-31 02:23:39.000	act	MOB-SO-WS01	CONTAIN	device	whole	method=immediate
2017-03-31 02:23:43.000	act	MOB-SO-WS01	SNAPSHOT	device	memory	method=immediate
2017-03-31 02:23:43.000	act	MOB-SO-WS01	INVESTIGATE	device	memory	method=immediate

Fig. 18. Resulting decisions from a worm performing data exfiltration on a target machine

6.2.6 Validation. As no similar approaches exist, we focused our efforts on reliably demonstrating feasibility, rather than compare against a deception approach which is different in scope (making for an unfair comparison), and mainly discuss performance in laboratory conditions. With unit, integration and functional testing we consistently demonstrated that actions based on our decision trees reconfigured the DataSim network, according to specification. We also ran a red-team demonstration exercise with an external party using our DataSim scenario. A human attacker would run any of our simulated attacks at will using a GUI to execute each attack with basic parameters (what entity to attack and what attack). If an attacker executed any of the aforementioned attacks, our system would respond reliably, according to specification.

7 DISCUSSION AND FUTURE WORK

7.1 Results

For historical data, the system generated decisions that matched our expectations. It is worth pointing out that more decisions are generated than necessary. This is due to the lack of information about the structure of the network or the associated BPs means the system will output decisions for any IP of assets that we wish to protect. If this information is available, we see fewer, and more targeted decisions. When applied to the reactive simulated environment, our system successfully and consistently mitigated multiple common types of attacks. Our demonstration of unpredictability in network defences show promise, but expect production environments to have additional practical challenges that we have yet to identify.

7.2 Business process-centric security postures

Our grammar provides a business-process centric language to define priorities in network defences. Instead of assuming that all tasks and assets are created equal, we recognise that some entities and tasks require different defence priorities. Security postures provide a straightforward way to re-configure a collection of decision trees and mission priorities. Instead of tweaking probabilities at each decision-tree node, we can apply a preferred profile across all trees immediately. We envisage this being particularly useful in volatile missions. Nested trees is particularly encouraged, and re-use of trees can become a collaborative effort between analysts, in which each analyst incorporates their understanding of the mission and how the defence should respond, and equally important: have their understanding peer-reviewed.

7.3 Manual labour, learning curve and automation

We assume that a competent network defence suitably reflects the priorities of the organisation. This often requires manual labour in advance of deployment to understand the human-level needs of the organisation. In practice, many assets are treated with equal importance. Future work will investigate to what degree automation can facilitate the mapping of assets and missions dependencies. For now, analysts would need to learn our grammar, we are currently exploring ideas for GUIs to make tree creation and optimisation more straightforward.

7.4 Effectiveness and efficiency

The ordering of the tree levels does not matter for its effectiveness, it can impact its efficiency. For instance, whereas our `SimpleTree00` tree checks the entity type first, `SimplestTree` only considers the severity (priority) of the alert. As a guide, entities that need protecting at all costs (e.g. safety-critical assets) should have small trees to ensure traversal is kept to a minimum.

7.5 Order of decision-tree execution

Our approach does not check whether the order of COAs matters w.r.t. BPMN tasks. For instance, if two BP tasks relate to the same alert, both trees get executed in order of the BPMN task. This imposes a form of ordering which may not reflect analyst preferences. We consider this future work and identify four corner cases:

- *An entity is responsible for multiple BP tasks.* This makes it unclear what order to execute the BP trees.
- *Alerts from different sensors can refer to the same incident and report different severity.* This may result in conflicting decisions.
- *Repeated decisions can affect entities.* Should duration of actions be remembered, appended or overwritten?
- *Timeouts may affect decisions in production environments* – should we acknowledge decisions?

7.6 Integration with external systems

Our approach is easy to parse and interface. It is also strict and thus straightforward to validate. Indeed, before a decision tree is used, its syntax is validated in our implementation. The implementation is distributed, parallelisable and easy to deploy for load balancing and specialist decision making (e.g. decisions that should not be made available for certain members of staff). This yields security benefits akin to containers such as Docker, and our solution is easy to containerise.

7.7 On OpenC2 integration

OpenC2 is a suite of specifications to command and execute cyber defence functions [40]. OpenC2 commands require *actions* and *targets*, while *arguments* and *actuators* are optional. Key benefits in using OpenC2, as perceived by us, include: it is abstract, concise, straightforward to integrate and unambiguous. However, as OpenC2 was not designed with either unpredictability in defences or cyber analytics decision systems in mind, we faced limitations as some of our use cases fell outside the scope of OpenC2.

OpenC2 is designed to logically command cyber assets directly, not provide commands to reason about them (e.g. performing analytical actions on the network defence itself, i.e. not directly tied to protection of an asset, but instead tied to the improvement of network defence decision-making). Currently, make use of two non-OpenC2 actions: `NOACTION` and `SUPPRESS`. `NOACTION` means making the informed decision to "*not do anything*", while `SUPPRESS` means that we do not have enough information to make a decision, and thus we "*ignore making a decision*". `SUPPRESS` acts as a placeholder for a future decision to be inserted. We envisage other actions such as `GATHER` (i.e. "*gather more Cyber Threat Intelligence before making a decision*, e.g. from STIX [44], MISP [45], PROTECTIVE [46] or other CTI systems") and `LEARN` (i.e. "*feed this decision back into our machine learning module*") along with others actions could be useful for network defence decision systems.

Our understanding is that OpenC2 assumes that all entities have to be digital assets. In our system, we assume people can be cyber assets that need protecting (and defending against) and should also be able to receive actions via digital assets. OpenC2 appears to make no distinction between *reporting to* an entity and requiring a person (or a person in a role, e.g. IT support) to do a *physical-space action* (e.g. turn a valve manually). Such a "`HUMANACT`" action may be necessary in air-gapped environments, cyber-physical systems, or for responding to insider threats activities. Our modules also support people alerts (for insider threat detection) based on anomalous behaviour of a person.

7.8 Future Work

We wish to continue **investigating characteristics and applications of reactive unpredictability**. We plan to investigate whether OpenC2 provides a comprehensive view of decision-making capabilities for an unpredictable network defence. We also plan to investigate the theoretical value of reactive unpredictability in network defences, by examining how reconfiguration of the defences through actions available (see OpenC2 [40] or Agrafiotis et al. [47]), and in mission tasks and dependency mapping themselves can affect the security of an organisation.

Other aspects we would like to study include **the effectiveness of unpredictability in red-team exercises** and deploying our system in production environments. We envisage multiple aspects of unpredictability needing formalisation and assessment, including its uses against real attackers as well as the **use cases and usability of our decision tree grammar**. This will enable us to measure how capable a decision tree is in terms of performance in delaying attackers. We are also interested in simulating the effectiveness of the probabilistic approach on a simulated attacker in DataSim.

We believe that our probabilistic decision system can be used to **learn optimal security postures and decision tree order execution**. Manually generated, or automatically created trees could also be optimised similar to Norouzi et al. [48] to further improve performance, especially when trees become large.

Adding new assets in our system is a trivial task, however it is unclear **how the decision system should treat deletion of assets**. Presently, deleted assets are flagged as '*no longer visible*', but never removed entirely, akin to a publish/subscribe model. This is because we cannot rule out that an IDS system will not observe activities on those assets again.

We are also interested in **deception that makes use of cyber threat intelligence standards** such as STIX/TAXII [44], IDEA/WARDEN [49], MISP [45] etc. we may be able to preempt attacks and use this in our decision system.

The current implementation does not **learn from attacks**. We suspect third-party tools could be combined with our system to improve this capability. Learning strategies may target:

- *"the attacker"*, spending more time on a decision about an attacker, may allow for more intelligent deception.
- *"the effectiveness of the organisation's own defence"*, enabling analysts to benchmark current network defences.
- *"the vulnerabilities of assets"*, keeping in mind which components of the system need patching.

Finally, we are interested in **automating asset-to-mission mappings**. Deriving business processes and their asset dependencies from raw logs over time may remove some of the configuration burden for analysts. It may also reveal dependencies that analysts were previously unaware of.

8 CONCLUSION

This paper proposed a novel approach to decision making in network defences. We deceive adversaries by computing decisions using probabilistic decision trees. These trees are created from a grammar that defines how a system's defences should respond to threats. We have implemented our approach and evaluated it using a historical dataset (playback), and a real-time network simulation.

For historical data, the system generated decisions which matched analysts expectations. However, this generated more decisions than strictly required, this is because of the lack of information about the structure of the network or the associated BPs. If this information was available, we expect to see fewer, and more targeted decisions. When applied to the reactive simulated environment, our system successfully repelled common types of detected attacks. As part of this assessment, we deemed it necessary to design a real-time simulation system that is capable of running business processes alongside network assets and cyber-physical systems that can be affected by network attacks. The design and implementation of this system was also presented.

We envisage that our approach can be used was a way to examine effectiveness of existing defences using historical playback and identifying optimal decisions – and can therefore be used as an analyst learning tool. Our findings suggest that deception through probabilistic decision making yields promise and should be pursued further in future research.

ACKNOWLEDGMENT

This document is the results of a research project on automated network defences funded by the UK Defence Science and Technology Laboratory (DSTL).

REFERENCES

[1] L. Spitzner, "Honeypots: catching the insider threat," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, 2003, pp. 170–179.

[2] E. Vasilomanolakis, S. Karuppayah, P. Kikiras, and M. Mühlhäuser, "A honeypot-driven cyber incident monitor: lessons learned and steps ahead," in *Proceedings of the 8th International Conference on Security of Information and Networks*. ACM, 2015, pp. 158–164.

[3] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011, vol. 54.

[4] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the First ACM Workshop on Moving Target Defense*. ACM, 2014, pp. 31–40.

[5] R. Sun, M. Bishop, N. C. Ebner, D. Oliveira, and D. E. Porter, "The case for unpredictability and deception as os features," *USENIX; login*, 2015.

[6] R. Sun, D. E. Porter, D. Oliveira, and M. Bishop, "The case for less predictable operating system behavior," in *15th Workshop on Hot Topics in Operating Systems (HotOS {XV})*, 2015.

[7] S. A. White, "Introduction to bpmn," *Ibm Cooperation*, vol. 2, no. 0, p. 0, 2004.

[8] T. E. Carroll and D. Grosu, "A game theoretic investigation of deception in network security," *Security and Communication Networks*, vol. 4, no. 10, pp. 1162–1172, 2011.

[9] M. H. Almeshekah and E. H. Spafford, "Planning and integrating deception into computer security defenses," in *Proceedings of the 2014 New Security Paradigms Workshop*, 2014, pp. 127–138.

[10] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv preprint arXiv:1802.04528*, 2018.

[11] N. Provos *et al.*, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, no. 2004, 2004, pp. 1–14.

[12] P. Wang, L. Wu, R. Cunningham, and C. C. Zou, "Honeypot detection in advanced botnet attacks," *International Journal of Information and Computer Security*, vol. 4, no. 1, pp. 30–51, 2010.

[13] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in network security: a survey," in *Proceedings of the 2011 international conference on communication, computing & security*, 2011, pp. 600–605.

[14] R. M. Campbell, K. Padayachee, and T. Masombuka, "A survey of honeypot research: Trends and opportunities," in *2015 10th international conference for internet technology and secured transactions (ICITST)*. IEEE, 2015, pp. 208–212.

[15] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," *arXiv preprint arXiv:1608.06249*, 2016.

[16] G.-l. Cai, B.-s. Wang, W. Hu, and T.-z. Wang, "Moving target defense: state of the art and characteristics," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 11, pp. 1122–1153, 2016.

[17] J. Zheng and A. S. Namin, "A survey on the moving target defense strategies: An architectural perspective," *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 207–233, 2019.

[18] J. Ferreira, A. Grahn, J. Nelson, D. O'Leary, and D. Poarch, "6 ways to deceive cyber attackers," https://edge.siriuscom.com/security/6-ways-to-deceive-cyber-attackers, 2018.

[19] R. Gutzwiller, K. Ferguson-Walter, S. Fugate, and A. Rogers, ""oh, look, a butterfly!" a framework for distracting attackers to improve cyber defense," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 62, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2018, pp. 272–276.

[20] R. Sun, A. Lee, A. Chen, D. E. Porter, M. Bishop, and D. Oliveira, "Bear: A framework for understanding application sensitivity to os (mis) behavior," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 388–399.

[21] R. Sun, X. Yuan, A. Lee, M. Bishop, D. E. Porter, X. Li, A. Gregio, and D. Oliveira, "The dose makes the poison—leveraging uncertainty for effective malware detection," in *2017 IEEE Conference on Dependable and Secure Computing*. IEEE, 2017, pp. 123–130.

[22] S. Keshav, *REAL: A network simulator*. University of California Berkeley, Calif, USA, 1988.

[23] M. Lacage and T. R. Henderson, "Yet another network simulator," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, 2006, pp. 12–es.

[24] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[25] T. Issariyakul and E. Hossain, "Introduction to network simulator 2 (ns2)," in *Introduction to network simulator NS2*. Springer, 2009, pp. 1–18.

[26] M. Piorkowski, M. Raya, A. L. Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux, "Trans: realistic joint traffic and network simulator for vanets," *ACM SIGMOBILE mobile computing and communications review*, vol. 12, no. 1, pp. 31–33, 2008.

[27] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras, "Atemu: a fine-grained sensor network simulator," in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004*. IEEE, 2004, pp. 145–152.

[28] G. Chen, J. Branch, M. Pflug, L. Zhu, and B. Szymanski, "Sense: a wireless sensor network simulator," in *Advances in pervasive computing and networking*. Springer, 2005, pp. 249–267.

[29] S. Sundresh, W. Kim, and G. Agha, "Sens: A sensor, environment and network simulator," in *37th Annual Simulation Symposium, 2004. Proceedings*. IEEE, 2004, pp. 221–228.

[30] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "Estinet openflow network simulator and emulator," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110–117, 2013.
[31] C. Garcia Cordero, E. Vasilomanolakis, A. Wainakh, M. Mühlhäuser, and S. Nadjm-Tehrani, "On generating network traffic datasets with synthetic attacks for intrusion detection," *arXiv preprint arXiv:1905.00304*, 2019.
[32] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, 2019.
[33] I. Amit, J. Matherly, W. Hewlett, Z. Xu, Y. Meshi, and Y. Weinberger, "Machine learning in cyber-security-problems, challenges and data sets," *arXiv preprint arXiv:1812.07858*, 2018.
[34] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.
[35] S. Hettich and S. D. Bay, "The UCI KDD Archive," University of California, Department of Information and Computer Science. http://kdd.ics.uci.edu, 1999.
[36] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2. IEEE, 2000, pp. 12–26.
[37] M. J. Turcotte, A. D. Kent, and C. Hash, "Unified host and network data set," *ArXiv e-prints*, vol. 1708, 2017.
[38] K. J. Ferguson-Walter, D. S. LaFon, and T. Shade, "Friend or faux: deception for cyber defense," *Journal of Information Warfare*, vol. 16, no. 2, pp. 28–42, 2017.
[39] K. Ferguson-Walter, T. Shade, A. Rogers, M. C. S. Trumbo, K. S. Nauer, K. M. Divis, A. Jones, A. Combs, and R. G. Abbott, "The tularosa study: An experimental design and implementation to quantify the effectiveness of cyber deception." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 5 2018.
[40] O. Command and C. O. L. D. Document, "OpenC2," http://openc2.org/, 2016.
[41] Swedish Defence Research Agency, "CRATE – Cyber Range And Training Environment," https://www.foi.se/en/our-knowledge/information-security-and-communication/information-security/labs-and-resources/crate---cyber-range-and-training-environment.html, 2010.
[42] G. Box and M. E. Muller, "A note on the generation of random normal deviates," *The Annals of Mathematical Statistics*, vol. 2, pp. 610–611, 1958.
[43] B. Roscoe, "The perfect spy for model- checking crypto- protocols," Technical Report, 1997.
[44] S. Barnum, "Standardizing cyber threat intelligence information with the structured threat information expression (stix)," *MITRE Corporation*, vol. 11, pp. 1–22, 2012.
[45] C. Wagner, A. Dulaunoy, G. Wagener, and A. Iklody, "MISP: The design and implementation of a collaborative threat intelligence sharing platform," in *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. ACM, 2016, pp. 49–56.
[46] J. Happa, "PROTECTIVE: A European-Wide NREN Cyber Threat Intelligence Sharing Platform – Lessons Learnt To Date," OASIS & FIRST Borderless Cyber Conference and Technical Symposium, 2017.
[47] I. Agrafiotis, A. Erola, M. Goldsmith, and S. Creese, "Formalising policies for insider-threat detection: A tripwire grammar." *JoWUA*, vol. 8, no. 1, pp. 26–43, 2017.
[48] M. Norouzi, M. Collins, M. A. Johnson, D. J. Fleet, and P. Kohli, "Efficient non-greedy optimization of decision trees," in *Advances in neural information processing systems*, 2015, pp. 1729–1737.
[49] P. Kácha, "Idea: Designing the data model for security event exchange," in *17th International Conference on Computers: Recent Advances in Computer Science*, 2013.

Table 1. Scripting Language Designed for the Simulation

Name	Command	Description
init_register	r1 v	Put the value v in register r1
arg_to_register	id r1	Places argument id in register r1
send_packet	r1 r2	Sends a packet from current machine to destination ip stored in register r1, with flag stored in r2
send_packet_exp	v r2	Sends a packet from current machine to destination ip v, with flag stored in r2
send_program	r	Sends a copy of the current code to the destination ip stored in register r
loop	l	Move the program counter to line l
probe	r1 r2	Move entity flag specified in r2 value to register r1
compare_value	r1 v ls lf	Compare value in r1 to value v. If equal move to ls, else lf
compare_register	r1 r2 ls lf	Compare value in r1 to value in r2. If equal move to ls, else lf
flag_bp	r1	Tell the entity to use the value in r1 as BP flag
random_machine	r1	Put the IP address of a random machine on the network into r1
next_connected	r1 r2	Put the IP of the next connected machine after the value in r1 into r2. If none left, r2 equals 0
is_ip	r1 r2	If the ip in register r1 is the entity, r2 equals 1 otherwise 0
find_vulnerable	r1 v	Put the IP of a vulnerable machine with vulnerability ID v into r1
add	r1 v	Add the value of v to the value stored in r1
alter_aux	r1 r2	Update the auxiliary quantity r2 to the value stored in r1 (this should be [0..512], and will be scaled [-1..1]. r2 takes the value [0..6] for the quantity to change
rand_to_register	r1 v	Put a random number between 0 and v into r1
send_bad_packet	r1 r2	Send a packet that will trigger the IDS to the IP in r1. The attack ID is specified in r2
off	-	Stop all programs and power off
exit_if_running	-	Stop the program if an instance is running
exit	-	Stop the program, and remove from entity

Table 2. Scripting Language for Business Processes

Name	Command	Description
find_first_free_machine	ip1 ip2 ...	Finds first free machine in list of ips
find_random_free_machine	ip1 ip2 ...	Finds a random free machine in list of ips
find_any_free_machine	-	Finds any free machine on the network
run_program_on_machine	name ip	Runs program with name on found machine. The destination ip is arg1, and the flag must have been constructed previously
wait_for_completion	flag s l	Waits till the machine has signalled completion of BP for s seconds. If exceeded move to line l
loop	l	Move pc to line l
finish	-	Finish the BP Node
run_program_on_machine_exp	name ip protocol scale	Runs program with name on found machine. The destination ip is arg1, and flags is arg2. Protocol and scale are used to create the flag
wait	s	Waits for s seconds
wait_rand	min max	Waits for a random time between min seconds and max seconds
next_bp	ID	Move to BP with ID
start_bp	ID	Start a new BP with ID
exit	-	Use on an end BP Element
query_entity	ip threshold l_g l_l	Query the auxiliary quantity on entity with IP ip and compare to threshold. If greater than threshold, move to line l_g, otherwise move to line l_l
rand_bp	prob1:ID1 prob2:ID2	Probabilistically move to BP with ID1 with probability prob1, and so on
check_power	ip s f	Check if machine at ip has power. Move to line s if has power, otherwise f
find_machine	ip	Find the entity associated with ip
build_send_flag	scale protocol bpid	Build a flag to use to deposit bpid in target machine
build_fetch_flag	scale protocol bpid	Build a flag to use to try to fetch bpid from the target machine

APPENDIX: DATASIM FEATURES

The scripting language run on the simulated machines developed for DataSim is described in more detail in Table 1. The example scripts in Section 6.2 use this scripting language. To allow BPs to control behaviours on simulated machines, we also developed a scripting language to express this behaviour programmatically, and this is specified in Table 2.

Received July 2020; revised XXXX 2020; accepted XXXX 2020