From TTP to IoC: Advanced Persistent Graphs for Threat Hunting

Aimad Berady, Mathieu Jaume, Valérie Viet Triem Tong, Gilles Guette

From TTP to IoC: Advanced Persistent Graphs for Threat Hunting

Aimad Berady, Mathieu Jaume, Valérie Viet Triem Tong, and Gilles Guette

***Abstract*—Defenders fighting against Advanced Persistent Threats need to discover the propagation area of an adversary as quickly as possible. This discovery takes place through a phase of an incident response operation called Threat Hunting, where defenders track down attackers within the compromised network. In this article, we propose a formal model that dissects and abstracts elements of an attack, from both attacker and defender perspectives. This model leads to the construction of two persistent graphs on a common set of objects and components allowing for (1) an omniscient actor to compare, for both defender and attacker, the gap in knowledge and perceptions; (2) the attacker to become aware of the traces left on the targeted network; (3) the defender to improve the quality of Threat Hunting by identifying false-positives and adapting logging policy to be oriented for investigations. In this article, we challenge this model using an attack campaign mimicking APT29, a real-world threat, in a scenario designed by the MITRE Corporation. We measure the quality of the defensive architecture experimentally and then determine the most effective strategy to exploit data collected by the defender in order to extract actionable Cyber Threat Intelligence, and finally unveil the attacker.**

***Index Terms*—Advanced Persistent Threat, Tactics Techniques Procedures, Threat Hunting, IOC, SIEM**

I. Introduction

THE rise of collective awareness of Advanced Persistent Threats (APT) has required companies to reconsider their approach to cybersecurity. The resilience level of a company's information system can be challenged today through a *Red versus Blue* exercise that simulates a realistic attack campaign, aimed at reproducing the behavior of the adversary. In these exercises, the Blue Team seeks the Red Team in a Threat Hunting operation [1], while the Red Team tries to carry out their attack as stealthily as possible. These exercises allow for testing the security controls, sensors, Security Information and Event Management (SIEM), and incident response processes. In such a game an attacker, the Red Team, intends to break into the infrastructure defended by the Blue Team. Such exercises of attacking and defending are inspired by similar military maneuvers whose objectives are to test the soldier readiness and attack effectiveness through simulations. In cybersecurity, these exercises help organizations keep their assets safe. The Red Team is composed of highly trained individuals playing the role of potential attackers motivated by a strategic objective (e.g., stealing sensitive information, using organizations' capabilities for malicious purposes, defeating the availability of victim's services). The Blue Team defends the company, and has to ensure that its assets are not compromised, in the event of the Red Team finding a vulnerability and exploiting it. The Blue Team thus needs to rapidly remediate the incident to control the Red Team's network propagation and contain the threat. To estimate the effectiveness of their respective games, we can naively measure the time it took the Red Team to dominate the target and the time it took the Blue Team to detect and respond to the attack. We believe that this measure would be greatly improved with knowledge of the compromised components, by the Red Team, from the victim's network (i.e., its propagation area) and how aware the Blue Team was of this.

In this article, we formalize both defensive processes and the attacker's offensive approaches, allowing for confronting their respective perceptions during the same attack campaign. The attacker's perception of the campaign is built from (1) the execution of his procedures chosen from among his *Tactics, Techniques and Procedures (TTP)* [2]; (2) his exposed resources during these executions; and (3) the victim's components he compromised. The defender's perception of the attack is built from (1) the collected traces on the targeted information system; and (2) the exploitation of these traces through his defensive procedures. The benefits of the proposed model are twofold. First of all, it provides a high-level representation of the attack campaign, allowing to quickly assess the attacker and defender progressions. This representation can also be used by an omniscient third party to measure the success of a *Red versus Blue* exercise. Second, the model highlights how the defender can improve the efficiency of his detection process by tweaking the input (configurations and rules) of some of his defensive procedures. Here, we conduct an experiment with our model, using an attack campaign issued from the public project *Mordor* [3]. This attack campaign mimics the real-world threat APT29 in a scenario designed by MITRE for the purposes of ATT&CK Evaluations [4]. The confrontation of these two perceptions allows us to define a metric to estimate the deployed detection chain efficiency.

This article is structured as follows. Section II provides an overview of the concepts manipulated in the model. Sections III and IV present respectively the attacker and the defender's perspectives. Section V details the experiment architecture and specifies requirements for integration in existing infrastructures. Section VI discusses how to evaluate the efficiency of the detection chain and how to enhance it.

A. Berady and V. Viet Triem Tong are with CentraleSupélec, Inria, Univ Rennes, CNRS, IRISA, F-35042 Rennes, France (e-mails: {aimad.berady}{valerie.viet_triem_tong}@irisa.fr)

M. Jaume is with Sorbonne Université, CNRS, LIP6, F-75005 Paris, France (e-mail: mathieu.jaume@lip6.fr)

G. Guette is with Univ Rennes, Inria, CentraleSupélec, CNRS, IRISA, F-35042 Rennes, France (e-mail: gilles.guette@univ-rennes1.fr)

II. Overview

This article formalizes the Threat Hunting process conducted by an incident response team, towards ultimately evaluating the efficiency of a detection chain. We start our process with the attacker's point of view; the attacker has initiative and sets the tempo.

We begin by specifying the scope of this study and explaining the terms we will use in the rest of this article. Thus, this section successively details the infrastructure under consideration, the attacker's scope, and then the defender's scope before outlining how we will be able to compare their two points of view with the help of two graphs.

a) The infrastructure: The targeted network is the information system hosting the attacker's final objective. In this network, we distinguish the components from the objects. The components are assets (i.e., machines) with logging capabilities over which the defender has full control, such as computers, servers, or appliances. The objects are measurable events or stateful properties relative to malware characterization, intrusion detection, incident response or digital forensics. These objects correspond to the observable objects defined in the STIX standard [5]. Each object plays a precise role in the context of an event. This role specifies the function that the object holds in the event. The set of possible roles is here denoted by $\mathbf{R}$. Each role $\mathbf{r}$ is associated with a unique type that specifies the nature of any object playing this role. By denoting $\mathbf{T}$ the set of possible types, the type associated with a role is defined through the function $\tau : \mathbf{R} \rightarrow \mathbf{T}$.

For our implementation, we used the MITRE Cyber Analytics Repository (CAR) [6] data model to name objects' roles. An object can play different roles associated with different types. For example, the object `maliciousfile.com` can hold the role *destination hostname* with the type domain, the role *file name*, or even the role *executable* with the type file. The three columns on the left of the Table I detail some examples of objects and their relative types and roles. *IP addresses*, *domain names*, or *files* are examples of frequently observed types.

Although the attacker also has his own machines, which could be components, in this model, we only consider the victim's components since those of the attacker cannot be reached by the defender, in a strictly defensive posture. Nevertheless, the defender may have an insight into some of the attacker's objects (e.g., a domain name, a hash value, an IP) because the attacker would have exposed them. The attacker will, however, be able to discover and gain access to part of the victim's components through objects. For these reasons, this article highlights only the set of components of the targeted network, refering to it as $\mathbf{C}$. We denote by $\mathbf{O}_D$ the set of objects relative to the victim (some of them related to components of $\mathbf{C}$), by $\mathbf{O}_A$ the set of objects relative to the attacker and by $\mathbf{O}$ the set $\mathbf{O}_D \cup \mathbf{O}_A$.

b) The attacker scope: The attacker can be an individual, a group, or a Red Team, but for the sake of clarity, we simply refer here to *the attacker*. In the same way, the presence of multiple attackers in the victim's network is not an obstacle because the ambition of this model is to provide a more exhaustive view of the compromised components. The attacker is at the initiative of the attack campaign and has his own components at his disposal, which is part of his infrastructure. He also owns a collection of attack procedures (denoted $\mathbf{TTP}_A$) often related to techniques described in the MITRE ATT&CK matrix. These procedures may be parameterized by objects from $\mathbf{O}_D$ as well as objects from $\mathbf{O}_A$. These procedures are executed on components in the targeted network (in $\mathbf{C}$) only if the attacker has already discovered these components. The attacker scope is completely formalized in section III.

c) The defender scope: The defender can be the victim itself, the security team of a company, an external security team, or a Blue Team. For the same reasons, we simply refer here to *the defender*. The defender has defensive procedures to cope with an attack campaign. The defender only observes the events occurring on components in $\mathbf{C}$ that he has chosen to monitor. The relevant events to be monitored by sensors are specified in their configuration $\mathcal{S}$. The defensive procedure p_{logs} allows him to generate traces from events that occur on these components. The defensive procedure p_{hunting} exploits these traces in an attempt to identify in $\mathbf{C}$ the components compromised by the attacker. The defender scope is formalized in section IV.

d) Confronting perspectives: We propose here to represent the attacker network propagation during an attack campaign by a persistent graph $\mathbf{G}_A$ between objects and components. This graph will be computed from the sequence of attack procedures and the involved objects. These objects represent characteristic information that the attacker cannot conceal and is aware of exposing.

Similarly, we represent the defender perception of the attacker's propagation by a second persistent graph $\mathbf{G}_D$. This graph computation relies on the defensive procedures executed by the defender. Figure 1 gives a global overview of this process, including the attacker and the defender scopes.

TABLE I
Examples of objects, their relative types, possible roles, and existence in both directories.

$\mathbf{o} \in \mathbf{O}_D$ Object	$\mathbf{r} \in \mathbf{R}$ Role	$\tau(\mathbf{r}) = \mathbf{t} \in \mathbf{T}$ Type	$\mathbf{D}_A(\mathbf{t}, \mathbf{o})$ Attacker directory (extract)	$\mathbf{D}_D(\mathbf{t}, \mathbf{o})$ Defender directory (extract)
`10.0.1.4`	src_ip dest_ip	ip address	(ip address, scranton.dmevals.com)	(ip address, scranton.dmevals.com)
`m.exe`	file_name exe	file	None	None
`dmevals.com`	dest_hostname file_name	domain file	(domain, `newyork.dmevals.com`) None	None None
`warehouse`	dest_hostname	domain	None	(domain, `warehouse.dmevals.com`)

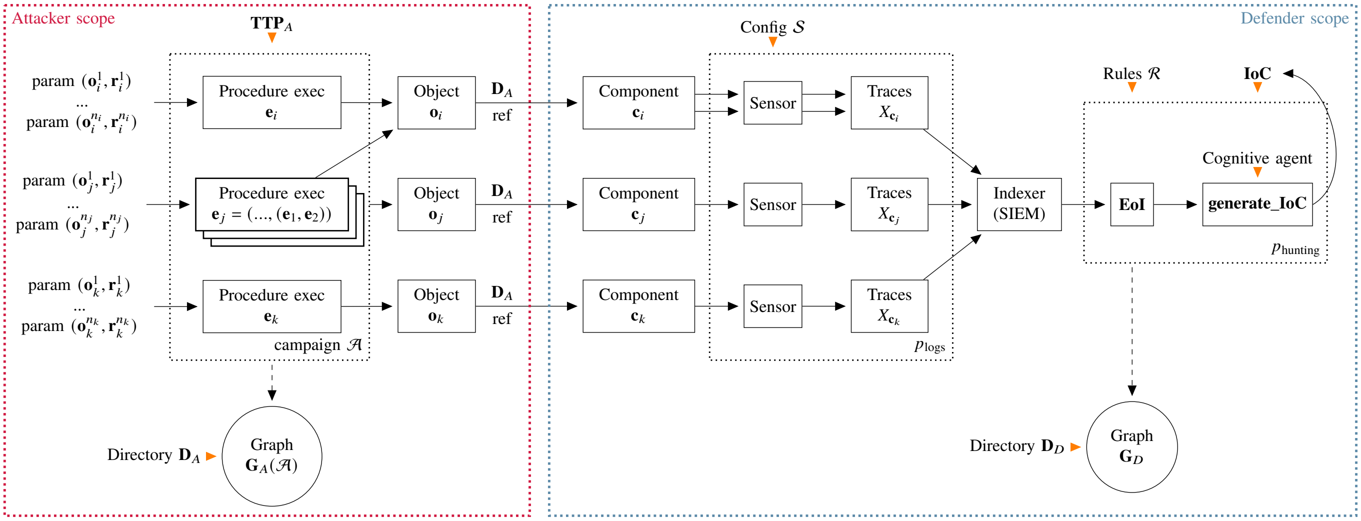
Attacker scope
$\mathbf{TTP}_A$
param $(\mathbf{o}_i^1, \mathbf{r}_i^1)$
...
param $(\mathbf{o}_i^{n_i}, \mathbf{r}_i^{n_i})$
Procedure exec $\mathbf{e}_i$
Object $\mathbf{o}_i$
$\mathbf{D}_A$ ref
param $(\mathbf{o}_j^1, \mathbf{r}_j^1)$
...
param $(\mathbf{o}_j^{n_j}, \mathbf{r}_j^{n_j})$
Procedure exec $\mathbf{e}_j = (\ldots, (\mathbf{e}_1, \mathbf{e}_2))$
Object $\mathbf{o}_j$
$\mathbf{D}_A$ ref
param $(\mathbf{o}_k^1, \mathbf{r}_k^1)$
...
param $(\mathbf{o}_k^{n_k}, \mathbf{r}_k^{n_k})$
Procedure exec $\mathbf{e}_k$
Object $\mathbf{o}_k$
$\mathbf{D}_A$ ref
campaign $\mathcal{A}$
Directory $\mathbf{D}_A$
Graph $\mathbf{G}_A(\mathcal{A})$
Defender scope
Config $\mathcal{S}$
Component $\mathbf{c}_i$
Sensor
Traces $X_{\mathbf{c}_i}$
Component $\mathbf{c}_j$
Sensor
Traces $X_{\mathbf{c}_j}$
Component $\mathbf{c}_k$
Sensor
Traces $X_{\mathbf{c}_k}$
p_{logs}
Indexer (SIEM)
Rules $\mathcal{R}$
IoC
Cognitive agent
EoI
generate_IoC
p_{hunting}
Directory $\mathbf{D}_D$
Graph $\mathbf{G}_D$

Fig. 1. Model overview.

The knowledge of the attacker and the defender of the targeted network are enriched by their own directory. For both the attacker and the defender, their directory determines how an object with a given type is relative to a component from his own perspective. In the following, $\mathbf{D}_A$ will be the attacker directory and $\mathbf{D}_D$ the defender directory. Both $\mathbf{D}_A$ and $\mathbf{D}_D$ are possibly imperfect or incomplete; they are here represented by two functions from $\mathbf{T} \times \mathbf{O}_D$ to $\mathbf{C} \cup \{\mathsf{None}\}$. When the attacker (resp. the defender) has no information concerning an object $\mathbf{o}$ with a type $\mathbf{t}$ or if the object $\mathbf{o}$ in a type $\mathbf{t}$ does not correspond to a component, then $\mathbf{D}_A(\mathbf{t}, \mathbf{o})$ is equal to None (resp. $\mathbf{D}_D(\mathbf{t}, \mathbf{o}) = \mathsf{None}$). The representation of a component $\mathbf{c}$ from the attacker's point of view may be different from the representation of the same component $\mathbf{c}$ from the defender's point of view. For example, the defender may know that a machine has a given IP address, is a web server, owns files, and has a name, while the attacker only knows the IP address of this machine. In the Table I example, the two right columns show if an object with a specific type is related to a component using the attacker and the defender directory.

In the following, Section V explains how the comparison of $\mathbf{G}_A$ and $\mathbf{G}_D$ allows confronting the perceptions of the attack campaign from both points of view: that of the attacker and that of the defender. We also show how the Threat Hunting operation can be greatly improved by increasing the quality of Indicators of Compromise (IoC) and Events of Interest (**EoI**) used by the defender.

e) Experiment: APT29 simulated campaign: In the context of cybersecurity product evaluations, MITRE creates attack scenarios inspired by real-world threats. Two of them concern the so-called APT29, which is a state-sponsored attackers group that has been active since 2008. During these attacks, the attacker used different procedures to collect and exfiltrate sensitive files from the targeted network after exploring it. These two scenarios detail the steps of an APT campaign using the threat actor's TTPs as they were observed by the infosec community. Subsequently, Roberto Rodriguez [3] was at the initiative of a dataset of logs recorded on an infrastructure allowing the procedure execution from scenarios. We decide to merge these two scenarios because of their similarities in targeting the same infrastructure and the fact that they mimic the same APT actor. Among the traces in the dataset, we focused on those coming from the *Sysmon* sensor, which is the one recommended according to state of the art [7]. Experimentation infrastructure is detailed in section V, and the results of this experiment are discussed in section VI.

III. Attacker's perspective

The attacker is modeled through a graph of objects and components. The components belong to the targeted network, with which the attacker interacts while executing an offensive procedure. The objects represent the traces that the attacker is aware of leaving on the target infrastructure while executing these procedures. This section details the construction of this graph by taking into account both actions, which define the whole attack campaign, and progression of the attacker's knowledge about the targeted network.

A. Actions of the attacker

An attack campaign $\mathcal{A}$ is composed of a sequence $\mathbf{e}_1, \cdots, \mathbf{e}_N$ of executions of N attack procedures $p_1, \cdots, p_N$ on components in $\mathbf{C}$ of the targeted network. We assume here that the attacker knows at least one initial component of the targeted network. This component could have been discovered by the attacker through an external reconnaissance of the exposed services provided by the targeted network or even by a social engineering attack. When the attacker has compromised at least one component of the targeted network, he is able to apply attack procedures inside it. Here begins the *Network Propagation* phase [8].

The execution $\mathbf{e}$ of an attack procedure $p \in \mathbf{TTP}_A$ requires the knowledge of:

- *a machine $\mathcal{M}(\mathbf{e})$:* a component $\mathbf{c} \in \mathbf{C}$ named through a relative object $\mathbf{o}$ and a type $\mathbf{t}$ occurring in its directory $\mathbf{D}_A$ where the procedure will be executed (thus $\mathbf{c} = \mathbf{D}_A(\mathbf{t}, \mathbf{o})$).

This machine is typically the host where the procedure will be executed, satisfying a technical intention (also known as Tactic) such as `Privilege Escalation`, `Discovery` or even `Persistence`;

- *some parameters* $((\mathbf{o}_1, \mathbf{r}_1), \cdots, (\mathbf{o}_n, \mathbf{r}_n))$*:* objects $\mathbf{o}_i$ in $\mathbf{O}$ with their roles $\mathbf{r}_i$ that configure the procedure;
- *some sub-executions* $(\mathbf{e}_1, \cdots, \mathbf{e}_m)$: corresponding to the invocation of sub-procedures orchestrated by p and such that for each $\mathbf{e}_i$, the host component $\mathcal{M}(\mathbf{e}_i)$ is accessible through a relative object appearing in $((\mathbf{o}_1, \mathbf{r}_1), \cdots, (\mathbf{o}_n, \mathbf{r}_n))$ the parameters of p.

An execution $\mathbf{e}$ (designated by a unique identifier $\mathsf{id}_\mathbf{e}$) of an attack procedure $p \in \mathbf{TTP}_A$ on a component $\mathbf{c} = \mathbf{D}_A(\mathbf{t}, \mathbf{o}) \neq \mathsf{None}$ can thus be formalized by:

$$\mathbf{e} = (\mathsf{id}_\mathbf{e}, p, (\mathbf{o}, \mathbf{t}), ((\mathbf{o}_1, \mathbf{r}_1), \cdots, (\mathbf{o}_n, \mathbf{r}_n)), (\mathbf{e}_1, \cdots, \mathbf{e}_m))$$

Listing 1 gives the example of an attack procedure commonly used by the attacker. This procedure performs a lateral movement using `psexec`. It allows the attacker to execute a command-line process on a remote machine and redirect console applications output to its local system. In this example, the host component for the main procedure is `SCRANTON.dmevals.local`, the component for the sub-procedure is `NASHUA.dmevals.local`. The attacker launches the malware file named `python.exe` remotely, with the user's privileges of `pbeesly` on both sides.

```
{ "name": "8.C",
  "cmp": "SCRANTON.dmevals.local",
  "description": "Execute payload on secondary computer",
  "obj": [
    {"role": "dest_ip", "value": "192.168.0.5"},
    {"role": "user", "value": "pbeesly"},
    {"role": "command_line", "value": ".\\PsExec64.exe −accepteula \\\\
           NASHUA −u \"dmevals\\pbeesly\" −p \"Fl0nk3rt0n!T0by\" −i
           {WILDCARD} \"C:\\Windows\\Temp\\python.exe\""},
    {"role": "exe", "value": "PsExec64.exe"},
    {"role": "dest_hostname", "value": "NASHUA"},
    {"role": "image_path", "value": "C:\\Program Files\\SysinternalsSuite
           \\PsExec64.exe"}],
  "invol": [
    { "name": "8.C.1",
      "cmp": "NASHUA.dmevals.local",
      "description": "Execute payload",
      "obj": [
        {"role": "dest_ip", "value": "192.168.0.4"},
        {"role": "dest_ip", "value": "10.0.1.6"},
        {"role": "dest_fqdn", "value": "SCRANTON.dmevals.local"},
        {"role": "user", "value": "pbeesly"},
        {"role": "exe", "value": "python.exe"},
        {"role": "image_path", "value": "C:\\Windows\\Temp\\python.exe"
           }
      ]}]}
```

Listing 1
ATTACK PROCEDURE: LATERAL MOVEMENT USING `psexec`.

B. Attacker propagation in the targeted network from the attacker point of view

We represent here the attacker propagation by a graph whose nodes are the objects involved in the attack campaign and their potential relative components from the attacker perspective. Each execution of an attack procedure requires the attacker to use objects and components either from his own resources or objects and components he has already discovered in the targeted network. In this model, we consider that the attacker's knowledge about the targeted network is frozen and already fully described through his directory $\mathbf{D}_A$. If we wanted to make this directory dynamic, it is necessary to formalize a `Discovery` procedure that allows the attacker to improve his knowledge of the targeted network infrastructure.

a) Small step propagation: From each execution $\mathbf{e}$ of an attack procedure p, we build an oriented graph $\mathbf{G}(\mathbf{e})$, whose nodes are the objects and their relative components involved in this execution referenced by the attacker's directory. This graph represents one step of his attack campaign. Formally, $\mathbf{G}(\mathbf{e})$ is defined for $\mathbf{e} = (\mathsf{id}_\mathbf{e}, p, (\mathbf{o}, \mathbf{t}), ((\mathbf{o}_1, \mathbf{r}_1), \cdots, (\mathbf{o}_n, \mathbf{r}_n)), (\mathbf{e}_1, \cdots, \mathbf{e}_m))$ by:

$$\mathbf{G}(\mathbf{e}) = (V_\mathbf{e}, \rightarrow_\mathbf{e}) \oplus \bigoplus_{i=1}^{m} \mathbf{G}(\mathbf{e}_i)$$

and computed from:

- the nodes $V_\mathbf{e}$, which are the host component, all the objects used in the procedure execution, and their potential relative components known by the attacker:
$$\{\mathcal{M}(\mathbf{e}), \mathbf{o}_1, \cdots, \mathbf{o}_n\} \cup \{\mathbf{c}' = \mathbf{D}_A(\tau(\mathbf{r}_i), \mathbf{o}_i) \neq \mathsf{None}\}$$
- the edges $\rightarrow_\mathbf{e}$ connecting the host component with parameters (objects) and the components linked to these objects according to the attacker's directory:
$$\bigcup_{i=1}^{n} \left\{ \mathcal{M}(\mathbf{e}) \xrightarrow{\mathsf{id}_\mathbf{e}, \mathbf{r}_i} \mathbf{o}_i \right\} \cup \left\{ \begin{array}{l} \mathbf{o}_i \xrightarrow{\text{ref}} \mathbf{c}' \mid \\ \mathbf{c}' = \mathbf{D}_A(\tau(\mathbf{r}_i), \mathbf{o}_i) \neq \mathsf{None} \end{array} \right\}$$
- and the union of graphs $\bigoplus_{i=1}^{m} \mathbf{G}(\mathbf{e}_i)$ issued from sub-procedures called during $\mathbf{e}$. The operator $\oplus$ denotes here the classical union between two graphs.

Figure 2 presents graph $\mathbf{G}$(`psexec`) computed from the specific `psexec` attack procedure execution as previously described in Listing 1. In this graph, dark blue nodes are components and light blue nodes are objects. The labels on thin black edges define the object role in the procedure. The thick green edges define objects that are related to another component according to the attacker directory. Given that a `Lateral Movement` procedure, by definition, involves two components, both are represented on the graph, connected by "ref" edges.

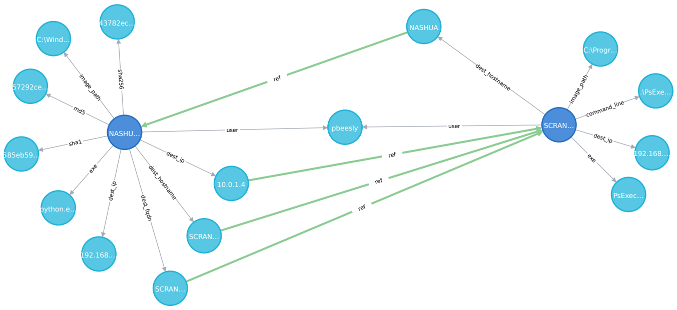

Fig. 2. $\mathbf{G}$(`psexec`)

b) Big step propagation: Finally, an attack campaign $\mathcal{A}$ composed of executions $\mathbf{e}_1, \cdots, \mathbf{e}_N$ can be modeled by the graph $\mathbf{G}_A(\mathcal{A})$ resulting from the union of all the graphs associated with each execution: $\mathbf{G}_A(\mathcal{A}) = \bigoplus_{i=1}^{N} \mathbf{G}(\mathbf{e}_i)$. The computation of this graph allows the attacker to represent all the objects he exposes during his attack campaign. This graph can be seen as an attack footprint exposed to a defender. Moreover, in a *Red versus Blue* exercise, this graph allows an omniscient team (i.e., the White Team) to measure the distance between what the Blue Team has actually found and the attacker's actual footprint.

Figure 3 presents the graph $\mathbf{G}_A$ computed by the attacker during the APT29 simulated campaign in 49 steps corresponding to execution procedures, 4 compromised components `SCRANTON`, `NASHUA`, `NEWYORK` and `UTICA`, which are the four on which the attacker actually executed procedures during the attack campaign. Those 4 components are also defined in his directory.

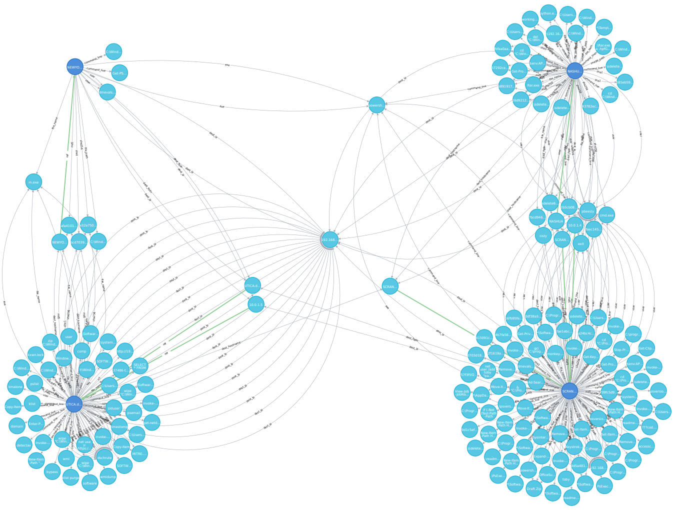

Fig. 3. $\mathbf{G}_A$ computed by the attacker during APT29 simulated campaign.

IV. Defender's perspective

We consider the same attack campaign but now from the defender's perspective. The defender is never considered to have initiative. The aim of the defender is to compute a representation of the attacker propagation in the targeted network from his (the defender's) point of view. In other words, his goal is to compute a graph of objects and components that are as similar as possible to the one produced by the attacker on his side. To this end, the defender uses his own defensive procedures. The semantic framework proposed in this article allows for generalizing offensive or defensive procedures. We specify that we have already identified several other tactics that would be generalizable. These include, for example, $p_{\text{discovery}}$, which consists for the attacker to browse a namespace to discover new components.

In the following, we describe two main defensive procedures: p_{logs} and p_{hunting}, which must be implemented by the defender to conduct incident response operations. Attack campaigns against the targeted network generate events on its components. We represent an event ev by a tuple $(\epsilon, \mathbf{c}, O)$ where $\epsilon \in \mathcal{E}$ denotes the type of ev (with $\mathcal{E}$ the set of event types that can be observed), $\mathbf{c} \in \mathbf{C}$ is the component over which the event is observed, and $O = \{(\mathbf{o}_1, \mathbf{r}_1), \cdots, (\mathbf{o}_m, \mathbf{r}_m)\} \subseteq \mathbf{O} \times \mathbf{R}$ contains all the objects involved in this event associated with their role from the defender's point of view.

Depending on the configuration of sensors, events can generate traces. A trace $\mathbf{x}$ is a tuple $(\mathsf{id}_\mathsf{x}, t, \epsilon, \mathbf{c}, O)$ where id_x is the trace identifier, t is a timestamp, $\epsilon \in \mathcal{E}$ is the type of events causing this trace, $\mathbf{c} \in \mathbf{C}$ is the component where the event causing this trace has been observed and O are all the objects together with their roles involved in this trace. A single object can assume several roles in a single event and therefore appears several times in the O set. In contrast, each role is unique within the same trace.

In a Threat Hunting process, the defender can influence the quality of his results on three aspects: sensor configuration, detection rules, and his **IoC** database.

First, the defender decides through his defensive procedure p_{logs} which components on the targeted network are monitored by sensors and which events produce traces that will be forwarded to the Security Information Event Management (SIEM). This procedure, detailed in subsection IV-A, will generate a huge number of traces allowing the defender to identify the traces relative to the attacker's activity.

The second defensive procedure called p_{hunting}, detailed in subsection IV-B, exploits those traces.

A. Targeted network monitoring

Technically, the defender is able to monitor almost any event occurring on the targeted network's components. Nevertheless, many of these events tend to be irrelevant from a security point of view and may generate too many false-positives in a SIEM alerting system.

As the traces raised by the sensors are the only way for the defender to perceive a part of the attacker's activity, the defender has to pay very close attention to the configuration $\mathcal{S}$ of sensors. The more meaningful a trace is, the more valuable it is to the defender.

We assume here that all components can be observed, and their monitoring is configured by a sensor configuration $\mathcal{S}$. In this configuration, the defender defines the relevant event types for each component $\mathbf{c} \in \mathbf{C}$, which need to be traced and filtered on objects according to their roles. The configuration $\mathcal{S}(\mathbf{c})$ makes it possible to monitor the component $\mathbf{c}$. The configuration is defined by a set of tuple $(\epsilon, \phi, \mathsf{R}_\epsilon)$ where $\epsilon \in \mathcal{E}$ is the event type raised by the sensor when the condition ϕ holds true. ϕ expresses properties on the objects involved in the observed event $(\epsilon, \mathbf{c}, O)$ and their corresponding roles. We write $O \models \phi$ when objects in O satisfy condition ϕ. The syntax of ϕ and satisfaction relation $\models$ are defined in Appendix A.

$\mathsf{R}_\epsilon \subseteq \mathbf{R}$ contains the roles considered to be relevant to this type of event. Objects with one of these roles could be exploited in a Threat Hunting approach because of their searchability. For example, the string `10.0.1.4` (the object) with the role *dest_ip* and the observable type `IP address` can be searched in the logs.

```
<Sysmon schemaversion="4.22">
    <EventFiltering>
        <ProcessCreate onmatch="exclude">
            <CommandLine condition="is">
                \SystemRoot\System32\smss.exe
            </CommandLine>
        </ProcessCreate>
        <NetworkConnect onmatch="include">
            <Image condition="begin with">
                C:\Windows\Temp
            </Image>
        </NetworkConnect>
    </EventFiltering>
</Sysmon>
```

Listing 2. Sysmon sensor configuration example, part of a p_{logs} procedure.

Listing 2 presents the part of a Sysmon configuration. Sysmon is Windows service that monitors and logs system activity. In this example, the monitored component is the machine running on Windows; the condition ϕ expresses that Sysmon will generate `ProcessCreate` traces for all the Process creation events, except if the command line is exactly `\SystemRoot\System32\smss.exe` (false-positives are frequently generated, possibly due to legitimate command line). Moreover, this Sysmon instance will generate `NetworkConnect` traces for all images (i.e., PE binary files) stored in `C:\Windows\Temp` filesystem folder. Table II gives examples of relevant roles R_ϵ for these two event types. As a model of system-level events, we use the MITRE Cyber Analytics Repository (CAR) [6].

TABLE II
RELEVANT ROLES IN A SYSMON TRACE AND THEIR CAR NAME.

Event type $\epsilon \in \mathcal{E}$	Relevant roles $\mathsf{R}_\epsilon \subseteq \mathbf{R}$
ProcessCreate (CAR: `process create`)	Image (image_path) CommandLine (command_line) ParentImage (parent_image_path) ParentCommandLine (parent_command_line) User (user) Hashes (md5, sha1, sha256)
NetworkConnect (CAR: `flow start`)	Image (image_path) User (user) SourceIp (src_ip) SourceHostname (src_hostname) DestinationIp (dest_ip) DestinationHostname (dest_hostname)

Finding an adapted sensor configuration is very tricky because the defender has to find an optimal position between logging every single observed event and defining highly restrictive conditions (including specific or excluding generic objects). The first option will generate too many traces and risks producing many false-positives. The second option will rarely be positively satisfied and thus will produce a very few numbers of traces, which will lead the defender to misjudge the threat.

Finally, the defender's defensive procedure determines the monitored events and the reported traces in the procedure p_{logs}. This procedure is parameterized by $\mathcal{S}$ that specifies the configurations associated with components and by E a set of events that occurred on the components.

p_{logs} generates a set of traces $X_\mathbf{c}$ for each component $\mathbf{c}$:

```
procedure $p_{\text{logs}}(\mathcal{S}, E)$ :
    for all $ev = (\epsilon, \mathbf{c}, \mathcal{O}) \in E$:
        if $(\epsilon, \phi, \mathsf{R}_\epsilon) \in \mathcal{S}(\mathbf{c})$ and $\mathcal{O} \models \phi$:
            $X_\mathbf{c} = X_\mathbf{c} \cup \{(\mathsf{id}_X, t, \epsilon, \mathbf{c}, \{(\mathbf{o}, \mathbf{r}) \in \mathcal{O} \mid \mathbf{r} \in \mathsf{R}_\epsilon\})\}$
    return $\{X_\mathbf{c} \mid \mathbf{c} \in \mathbf{C}\}$
```

Listing 3 is an example of a trace raised by Sysmon sensor. It happens because the configuration in Listing 2 observed a `Network Connection` event for which the object with the role `Image` (i.e., executable file name) matched with an expected location in the filesystem (i.e., the Windows `Temp` folder). The sensor thus logged it. The produced trace indicates that this event has occurred on the component `NASHUA.dmevals.local` and has some objects to give more information to this event, such as `DestinationIp`, `User`, or the full path of the `Image`.

```
{ "Hostname": "NASHUA.dmevals.local",
  "UtcTime": "2020-05-02 03:16:19.454",
  "RecordNumber": 353256,
  "SourceName": "Microsoft-Windows-Sysmon", "EventID":3,
  "SourceIp": "10.0.1.6",
  "SourcePort": "60215",
  "DestinationIp": "192.168.0.4",
  "DestinationPort": "8443",
  "Image": "C:\\Windows\\Temp\\python.exe",
  "ProcessId": "2172",
  "User": "DMEVALS\\pbeesly", ... }
```

Listing 3. Extract of a "Sysmon/Network connection" trace.

B. Attacker propagation from the defender's point of view

We assume here that all traces produced by the sensors are reported to a SIEM. The second defensive procedure used by the defender is the procedure p_{hunting}. This procedure helps the defender to construct a graph $\mathbf{G}_D$ from the observed traces. The graph $\mathbf{G}_D$ is built on the same model as the graph $\mathbf{G}_A$ constructed by the attacker: nodes are objects or components, edges between two nodes indicate that these objects or components are relative in this attack campaign from the defender point of view. The graph $\mathbf{G}_D$ is not built directly from the raw set of traces reported to the SIEM because these traces cover a lot of objects and components irrelevant to the hunting process. It is for this reason that the defender has to filter the traces reported to the SIEM to focus only on traces dealing with an Event of Interest (**EoI**).

a) Highlighting Events of Interest: The defender relies on a database of Indicators of Compromise **IoC** and a set of detection rules $\mathcal{R}$ to define the traces that have to be considered as Events of Interest. An Indicator of Compromise (**IoC**) is an object **o** with a type **t** that indicates, with high confidence, malicious activity on a network. An IoC is similar to an *artifact* generated along with a malicious activity. Table III gives an extract of the **IoC** database used by the defender during the APT29 Threat Hunting process. In this example, objects like `toby` or `m.exe` with the observable types of *user* and *file* respectively, are searchable in a local scope. Those are *local* IoC. The defender decides this classification because the user has been created in the target information system, so it makes little sense to look for it globally, in other victims' networks, due to the high risk of false-positives caused by its detection. However, `cod.3aka3.scr` and `9d1c5ef38e6073661c74660b3a71a76e`, with the observable types of *file* and *hash* respectively, are searchable in a larger scope and may make sense in other networks: those are *global* IoC.

TABLE III
EXTRACT OF THE **IoC** DATABASE USED BY THE DEFENDER DURING APT29 THREAT HUNTING PROCESS.

Object $\mathbf{o} \in \mathbf{O}_D$	Type $\mathbf{t} \in \mathbf{T}$	Scope
`toby`	user	local
`m.exe`	file	local
`cod.3aka3.scr`	file	global
`9d1c5ef38e6073661c74660b3a71a76e`	hash	global

A database $\mathbf{IoC} = \{(\mathbf{o}_1, \mathbf{t}_1), \cdots, (\mathbf{o}_n, \mathbf{t}_n)\}$ is maintained by the defender who can update it through sharing his knowledge with other security teams (global IoCs) or through his own investigations on his network (local IoCs). A detection rule $r \in \mathcal{R}$ expresses a defender-specific condition specifying that a trace has to be considered as an Event of Interest. We write $\mathbf{x} \models r$ when the condition specified by r is satisfied by the trace $\mathbf{x}$. The syntax of rules and satisfaction relation $\models$ are inductively defined in Appendix A.

Finally, the set of detection rules $\mathcal{R}$ and the **IoC** database allow the defender to specify the function $\mathbf{EoI}_{\mathcal{R},\mathbf{IoC}}$, which filters the traces in order to highlight the traces that are Events of Interest exclusively. More precisely, $\mathbf{EoI}_{\mathcal{R},\mathbf{IoC}}(\mathbf{x})$ provides two sets: the subset $R_{\mathbf{x}}$ of $\mathcal{R}$ containing rules satisfied by $\mathbf{x}$, and the subset $O_{\mathbf{x}}$ of **IoC** containing objects occurring in $\mathbf{x}$. Hence, a given trace $\mathbf{x}$ is an Event of Interest if at least one of these subsets is not empty. Formally, given a trace $\mathbf{x} = (\mathsf{id}_{\mathbf{x}}, t, \epsilon, \mathbf{c}, O)$ this function is defined by $\mathbf{EoI}_{\mathcal{R},\mathbf{IoC}}(\mathbf{x}) = (R_{\mathbf{x}}, O_{\mathbf{x}})$ where:

$$R_{\mathbf{x}} = \{r \in \mathcal{R} \mid \mathbf{x} \models r\}$$
$$\text{and} \quad O_{\mathbf{x}} = \left\{ \begin{array}{l} (\mathbf{o}, \mathbf{t}) \mid (\mathbf{o}, \mathbf{t}) \in \mathbf{IoC} \text{ and} \\ (\mathbf{o}, \mathbf{r}) \in O \text{ and } \tau(\mathbf{r}) = \mathbf{t} \end{array} \right\}$$

Listing 4 details a rule commonly used by a defender to detect execution of `psexec` by analyzing `process_creation` event types observed by sensors such as Sysmon with an object matching `*\PsExec64.exe`. The object has the role `Image` (i.e., executable file name). Following this detection, the **EoI** function returns, in particular, the trace (Cf. Listing 3), which resulted in the verification of the condition set out in the rule.

b) Small step: Each observation of a trace $\mathbf{x} = (\mathsf{id}_{\mathbf{x}}, t, \epsilon, \mathbf{c}, O)$ considered as an Event of Interest leads to the creation of a graph $\mathbf{G}(\mathbf{x}, R_{\mathbf{x}}, O_{\mathbf{x}})$ (where $\mathbf{EoI}_{\mathcal{R},\mathbf{IoC}}(\mathbf{x}) = (R_{\mathbf{x}}, O_{\mathbf{x}})$) representing this trace. The nodes of this graph are:

- the component $\mathbf{c}$ where the trace has been collected;
- the objects $\mathbf{o}$ occurring in O;
- the components referred by an object from the trace in the defender's directory $\{\mathbf{c}' = \mathbf{D}_D(\tau(\mathbf{r}), \mathbf{o}) \neq \mathsf{None} \mid (\mathbf{o}, \mathbf{r}) \in O\}$.

```
action: global
title: PsExec Tool Execution
id: 42c575ea−e41e−41f1−b248−8093c3e82a28
description: Detects PsExec service installation and execution events
tags:
    − attack.execution
    − attack.t1035
    − attack.s0029
detection:
    condition: 1 of them
fields:
    − EventID
    − CommandLine
    − ParentCommandLine
    − ServiceName
    − ServiceFileName
...
logsource:
    category: process_creation
    product: windows
detection:
    sysmon_processcreation:
        Image: '*\PSEXESVC.exe'
        User: 'NT AUTHORITY\SYSTEM'
```

Listing 4. Extract of Sigma detection rule[1] for `psexec`.

The edges in $\mathbf{G}(\mathbf{x}, R_{\mathbf{x}}, O_{\mathbf{x}})$ permit to connect:

- all the objects $\mathbf{o} \in O$ to the component $\mathbf{c}$:
$$\bigcup_{(\mathbf{o},\mathbf{r}) \in O} \left\{ \mathbf{c} \xrightarrow{\mathbf{r}, R_{(\mathbf{o},\mathbf{r})}, \text{is\_ioc}} \mathbf{o} \right\}$$
Each edge is labeled by the role of $\mathbf{o}$, the subset $R_{(\mathbf{o},\mathbf{r})}$ of $R_{\mathbf{x}}$ defined by:
$$R_{(\mathbf{o},\mathbf{r})} = \{r \in R_{\mathbf{x}} \mid (\mathsf{id}_{\mathbf{x}}, t, \epsilon, \mathbf{c}, O \setminus \{(\mathbf{o}, \mathbf{r})\}) \not\models r\}$$
and by a boolean is_ioc, which is true iff there exists $(\mathbf{o}, \mathbf{t}) \in \mathbf{IoC}$ such that $\tau(\mathbf{r}) = \mathbf{t}$. Hence, to remove the edge $\mathbf{c} \xrightarrow{\mathbf{r}, R_{(\mathbf{o},\mathbf{r})}, \text{is\_ioc}} \mathbf{o}$ from $\mathbf{G}(\mathbf{x}, R_{\mathbf{x}}, O_{\mathbf{x}})$ it suffices to disable rules in $R_{(\mathbf{o},\mathbf{r})}$ and to remove $(\mathbf{o}, \mathbf{t})$ from **IoC** when is_ioc is true.
- and all the components $\mathbf{c}'$ referenced in the defender directory $\mathbf{c}' = \mathbf{D}_D(\tau(\mathbf{r}), \mathbf{o}) \neq \mathsf{None}$ (such edges are just labeled by ref).

Figure 4 gives an example of graph computed by the defender from the trace described in Listing 3. This graph is thus the perception of the execution of the `psexec` attack procedure on the network. At the center of the graph in Figure 4 is the component on which the trace has been observed. Around this component are the relevant objects involved in the trace. This graph can contain objects similar to the small step attacker's graph, presented in Figure 2.

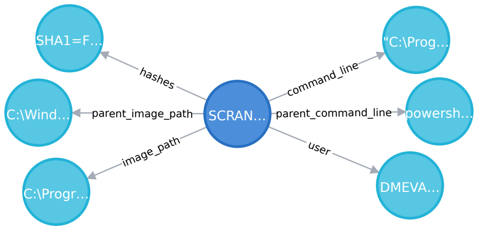

Fig. 4. Computed graph from a single trace which is considered by the defender as an **EoI**.

[1] https://github.com/Neo23x0/sigma/blob/master/rules/windows/other/win_tool_psexec.yml

c) Big step: Finally, an attack campaign $\mathcal{A}$ observed by a defender through a collection of traces X can be modeled by the graph $\mathbf{G}(X) = \bigoplus_{\mathbf{x}\in X} \mathbf{G}(\mathbf{x}, R_{\mathbf{x}}, O_{\mathbf{x}})$ resulting from the union of all the graphs associated with each Event of Interest computed from a set of traces X.

Figure 5 is the graph computed by the defender during the APT29 simulated campaign with rules from the public project Sigma. Section V gives all the details on its computation and discusses how to deal with the objects present.

The graph on the Figure 5 has to be compared with the graph representing this same attack campaign from the attacker's perspective on the Figure 3.

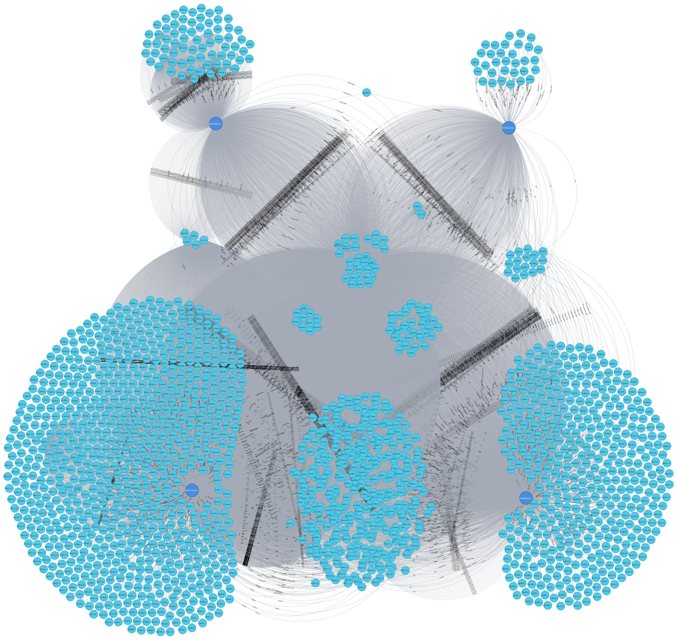

Fig. 5. $\mathbf{G}_D$ computed by the defender during APT29 simulated campaign.

d) Quality of the defender perspective: The quality of the graph $\mathbf{G}(X)$ constructed by the defender is influenced by three parameters: the configuration of the sensors $\mathcal{S}$, the set of detection rules $\mathcal{R}$ database, and the **IoC** database. In a Threat Hunting process, updating the sensor configuration or the set of detection rules is too long and too impactful to be done straightaway. On the other hand, the IoC database can and must be updated each time a trace is considered to be of interest by the function $\mathbf{generate\_IoC}((R_{\mathbf{x}}, O_{\mathbf{x}}), \mathbf{x}, \mathbf{IoC})$. All objects that appear in a trace considered as an Event of Interest are candidates to become IoC and, in particular, objects with roles corresponding to types of observables such as `IP address`, `hashes` or `domain`. Although Kurogome et al [9] have already proposed to automate this function **generate_IoC**, the intervention of an expert can be considered. The defender therefore has two main defensive procedures p_{logs}, which he uses to designate the components to monitor and information to report. The p_{hunting} procedure completes this first defensive procedure by computing the defender's graph and updating the **IoC** database for each trace considered as an Event of Interest. p_{hunting} can be formalized as follows. Starting from scratch, $\mathbf{G}_D = (\emptyset, \emptyset)$.

```
procedure $p_{\text{hunting}}(X, \mathbf{G}_D, \mathbf{IoC})$ :
    for all $\mathbf{x} \in X$:
        $(R_{\mathbf{x}}, O_{\mathbf{x}}) = \mathbf{EoI}_{\mathcal{R},\mathbf{IoC}}(\mathbf{x})$
        if $R_{\mathbf{x}} \neq \emptyset$ or $O_{\mathbf{x}} \neq \emptyset$:
            $\mathbf{G}_D = \mathbf{G}_D \oplus \mathbf{G}(\mathbf{x})$
            $\mathbf{IoC} = \mathbf{IoC} \cup \mathbf{generate\_IoC}((R_{\mathbf{x}}, O_{\mathbf{x}}), \mathbf{x}, \mathbf{IoC})$
    return $(\mathbf{G}_D, \mathbf{IoC})$
```

During a Threat Hunting operation, the defender goes to build from $\mathbf{G}_D$, a restricted graph that highlights the objects shared between several components. This means that these objects with relevant roles and involved in Events of Interest have been observed on at least two components. This graph is useful for orientation during the hunt.

V. Model Experimentation

Integrating our approach in a production environment would allow both the attacker and the defender on either side to become aware of the traces left on the victim's network. The attacker could use this information to improve the stealthiness of his procedures, and the defender could use it to improve his Threat Hunting process. In practice, its deployment in existing architectures requires:

- on the attacker's side: a logging system for executed procedures formatted according to Listing 1; a directory that corresponds to the attacker's current knowledge of the victim's network;
- on the defender's side: sensors installed on defended components and configured as presented in Listing 2 and whose relevant roles are specified as presented in Table II; a directory that corresponds to the defender's current knowledge of the victim's network; an indexer enriched with a set of detection rules as in Listing 4; an IoC database, such as in Table III, an analyst for tasks that are not automatable to date, such as IoC generation.

In this article, we take the point of view of an omniscient actor, which allows us to answer the following questions.

How to measure the quality of defensive architecture?
The comparison of the graphs $\mathbf{G}_A$ and $\mathbf{G}_D$ allows calculating the coverage rate of objects coming from the attacker into the defender's graph. This allows estimating the relevance of the detection chain.

How to reduce the defender's graph to unveil the attacker?
Too many objects in the defender's graph $\mathbf{G}_D$ can make it unusable. We are therefore looking for ways to reduce the number of objects while maintaining a sufficient coverage rate to provide potential IoC to a Threat Hunting team.

For this, we exploited traces of a realistic attack scenario on defensive infrastructure representative of that which we find in modern companies.

A. Attack scenario

We choose to experiment our model with an independent and representative attack scenario. We rely on the cybersecurity project Mordor [3] maintained by Roberto Rodriguez.

The Mordor project provides pre-recorded security events generated after simulating adversarial techniques. It was updated in 2020 with a new dataset called APT29. The dataset provides the logs built by replaying both parts of an attack scenario designed by MITRE in the context of their ATT&CK Evaluations [4]. The attack scenario emulates a 2-part attack led by the threat group APT29. The attack aims to collect and exfiltrate sensitive data. The first part is a rapid "smash and grab" collection and exfiltration of specific file types after an initial infection due to a widespread phishing campaign. Then the attacker drops a toolkit used to further explore and compromise the network. The second part is a targeted and methodical breach. It is a low and slow takeover of the target.

A part of the attack procedures is described in the listing 5. The complete list of attack procedures is described on the MITRE-Engenuity website[2].

```
1.A The scenario begins with an initial breach, where a legitimate user
        clicks (T1204) an executable payload (screensaver executable)
        masquerading as a benign word document (T1036). Once executed,
        the payload creates a C2 connection over port 1234 (T1065) using
        the RC4 cryptographic cipher.
...
2.A The attacker runs a one−liner command to search for filesystem for
        document and media files (T1083, T1119), collecting (T1005) and
        compressing (T1002) content into a single file (T1074).
...
6.C The attacker then harvests password hashes (T1003).
...
8.C This new payload is executed on the secondary victim via the PSExec
        utility (T1077, T1035) using the previously stolen credentials (T1078)
```

Listing 5. Part of all procedures from the Mordor APT29 attack campaign.

Two videos were also recorded by the author which gives an informal understanding of the attacker's perspective during this attack and of the objects that he was aware of exposing to the defender. We wrote all the attack procedures of this campaign according to the format presented in Listing 1. This allows us to build the attacker graph $\mathbf{G}_A$. Thus for each procedure, a central node that corresponds to the component is created. It is then connected by edges to each of the objects, presumed to be the attacker's, to be exposed during the execution of the procedure. Each of these edges is tagged with the role of the object in the procedure and also with an identifier. Where an involved procedure exists, an object designating the third component is created. The object is linked to the appropriate node with an edge annotated "Ref". Figure 3 presents this graph, characterized by 4 components, 180 unique objects, and 359 edges.

B. Targeted infrastructure and defensive architecture

The targeted infrastructure, on which the scenario that produced the Mordor APT29 dataset took place, has been reproduced according to the environment described by MITRE as part of their ATT&CK Evaluations as shown in Figure 6.

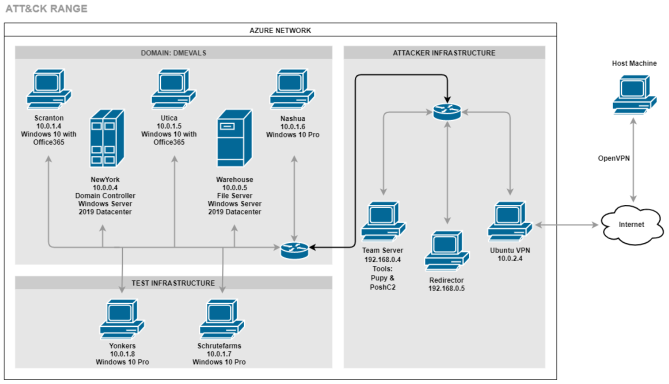

Fig. 6. APT29 Evaluation Environment (source: MITRE[3]).

The victim's network consists of three workstations, one file server, and one domain controller. All ran Microsoft Windows operating systems. The targeted infrastructure's systems are monitored by Microsoft Windows Sysmon, which provides detailed information about processes, network connections, and file manipulations. Microsoft Windows Sysmon produces the traces used to compute the defender view. These traces form a dataset of 783367 log entries corresponding to two days of observation (attack duration).

We have injected all the traces from the Mordor dataset into a Splunk[4] indexer. In this raw dataset, we now have to reveal the Events of Interest (**EoI**). In this experiment, the set of rules $\mathcal{R}$ is formed by the 565 commonly used rules from the public detection project Sigma[5]. Sigma is an open community project which aims to capitalize on detection rules sharing the same formalism and which are thus convertible to a large number of SIEM or directly integrated into malware analysis platforms such as VirusTotal[6]. At the beginning of the experiment, our Indicators of Compromise **IoC** database is empty. The first pass made it possible to detect 6100 **EoI**, thanks to the matching of 22 detection rules among the 565 enabled rules. We can now build the defender's graph with these EoI[7]. The computed graph has 4 components and 1758 unique objects. Table IV presents this graph's specifications.

Through this experimentation, we evaluated the relevance of two rule-disabling strategies in order to determine the approach, which is the most efficient to reduce the number of false-positives and make the defender graph $\mathbf{G}_D$ exploitable.

VI. Results

As we previously discussed, the attacker graph misses certain objects that the attacker is not aware of exposing. In our experiment, the components present in the attacker's graph are all present in the defender's graph, which is not surprising because they have proper monitoring, and all rules are enabled.

However, the number of objects in the attacker's graph (180) is significantly inferior to the number of objects (1758) in the defender's graph. This suggests that the defender's graph contains many false-positives. In this context, false-positives

[2] https://attackevals.mitre-engenuity.org/APT29/operational-flow.html

[3] ©2018 - 2020 The MITRE Corporation. This work is reproduced and distributed with the permission of The MITRE Corporation.

[4] https://www.splunk.com

[5] https://github.com/Neo23x0/sigma

[6] https://developers.virustotal.com/v3.0/reference#sigma-analyses

[7] Datasets, code, and full graphs are publicly available at https://gitlab.inria.fr/cidre-public/from-ttp-to-ioc-dataset

TABLE IV
DEFENDER POINT OF VIEW: BIG GRAPH SPECIFICATIONS.

item	value
Components	4
Objects	1758
Relations (edge between objects and components)	15788
Events of Interest injected	6100
Objects connected to 4 components	18
Objects connected to 3 components	48
Objects connected to 2 components	398
Objects connected to 1 component	1294
Attacker perspective's objects coverage	27.78 %
Unique rules producing **EoI**	22

are generally legitimate or native objects of the system and whose normal behavior triggers inappropriate or lax detection rules $\mathcal{R}$, or even verbose sensor configuration $\mathcal{S}$. In other words, a false-positive is an object that can never become an IoC since it would not make sense to look for it in a wider scope or because it is not directly related to the malicious event that occurs on the victim's system.

In order to reduce these false-positives, the intervention of a cognitive agent is often necessary. However, it would be possible to automatically mark objects which correspond to legitimate behaviors of the system while paying attention to malicious actions falling within the *Living-off-the-Land* [10] paradigm. These actions have the particularity of staying under the radar since they rely on native objects of the system in order to satisfy the attacker's technical intentions. The next step is to find a method to remove these false-positives. It has to be done because their preponderance on the graph may reduce the importance of interesting objects. The defender will, therefore, have to find a more efficient way to identify them instead of manually qualifying each of them.

A. How to measure the quality of defensive architecture

From an omniscient perspective, the model allows for comparing data from both sides, attacker and defender. The number of objects from $\mathbf{G}_A$, existing also in $\mathbf{G}_D$, allows for estimating the efficiency of the entire defender detection chain. We compute that 27.78% of all attacker objects are effectively considered by the defender as Events of Interest (**EoI**). If this percentage is high enough to allow the defender to initiate a Threat Hunting operation, the large number of objects present in the defender's graph may disturb him. Consequently, the defender may not pay attention to particular objects in $\mathbf{G}_D$ that could become new IoCs. Some of them are objects that have very discriminating characteristics, like those presented in Table III, and it would be productive to search for them in a wider scope. Those could be qualified by a cognitive agent as new IoC and be added in the **IoC** database.

B. How to reduce the defender's graph to unveil the attacker

Threat Hunting is a cyclical discipline where the defender identifies new IoCs, modifies detection rules to search for them in a wider scope, analyzes the collected objects, and extracts new IoCs. In this process, not all objects can become IoCs.

For example, if an attacker uses the attack procedure `psexec` to perform lateral movements, and if the victim's administration team's performs remote tasks with this tool, then considering `psexec.exe` or one of its hashes as an IoC will cause a large number of false-positives. It is then necessary to write a detection rule which will allow us to specify the legitimate context of these administrative actions (e.g., by specifying the source IP addresses and user accounts involved). The disabling of too verbose rules can be done in post-processing to clean up $\mathbf{G}_D$ and to make it more exploitable. We have experimented with two rule-disabling strategies called *Top-objects* and *Top-events*. The *Top-objects* strategy consists of a sequence of rounds of disabling the rule that created the largest number of new unique objects in the defender's graph. This stops upon reaching the highest coverage rate, without having too many objects in the graph and having the fewest rules disabled. The *Top-events* strategy is similar but first disables the rules, which are at the origin of the largest number of **EoI**.

Given a defender's graph $\mathbf{G}_D = (V_D, \rightarrow_D)$, disabling a set of rules R_D leads to a new defender's graph, written $\text{Update}(\mathbf{G}_D, R_D)$, defined by removing from $\mathbf{G}_D$ all the edges:

$$\mathbf{c} \xrightarrow{\mathbf{r}, R_{(\mathbf{o},\mathbf{r})}, \text{is\_ioc}} \mathbf{o}$$

such that $R_{(\mathbf{o},\mathbf{r})} \setminus R_D = \emptyset$ and is_ioc $= false$ and to remove (if it exists) the edge labeled by ref starting from **o** together with the component ending such edge. Starting from:

- the attacker's graph $\mathbf{G}_A = (V_A, \rightarrow_A)$ containing objects in $O_A = V_A \cap \mathbf{O}$;
- the defender's graph $\mathbf{G}_D = (V_D, \rightarrow_D)$ containing objects in $O_D = V_D \cap \mathbf{O}$;
- an empty set R_D of disabled rules;
- and an initial value coverage $= v > 0$ for coverage;

```
while coverage $> 0$:
        $r$ = most_used($\rightarrow_D$)
        $R_D = R_D \cup \{r\}$
        Update($\mathbf{G}_D, R_D$)
        coverage = $\frac{Card(O_A \cap O_D)}{Card(O_A)} \times 100$
return $R_D$
```

Listing 6. Algorithm to compute coverage rates.

Listing 6 defines an algorithm to compute R_D according to the *Top-events* strategy, where:

- most_used($\rightarrow_D$) is a function providing the rule appearing the most times in $\mathbf{G}_D$;
- Update($\mathbf{G}_D, R_D$) is a function that updates the graph $\mathbf{G}_D$ view by excluding the events resulting from the rules R_D.

Implementing R_D as a list allows us to keep the order in which the rules were disabled and therefore revert the graph $\mathbf{G}_D$ to an earlier state.

Figure 7 shows two 3D curves which correspond respectively to the deactivation of *Top-objects* and *Top-events* strategies. We can observe that the *Top-objects* strategy seems to be the most effective because it allows maintaining high coverage while considerably reducing the number of objects. So when there are only 63 objects left in the graph and only 4 rules have been deactivated, the coverage rate is 24.4%.

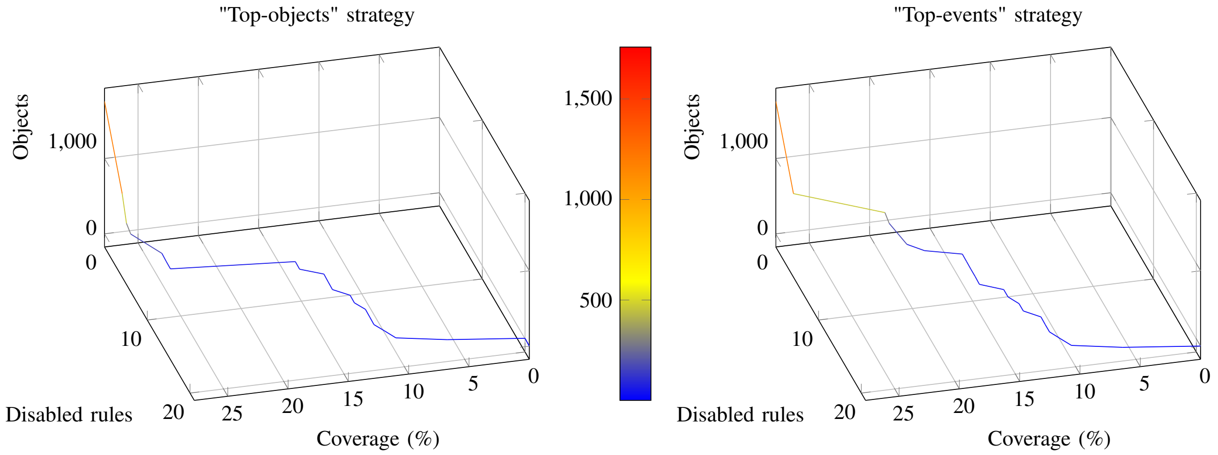

Fig. 7. Evolutions of coverage rates and count of objects present in the graph $\mathbf{G}_D$ according to the number of disabled rules.

C. Discussion

By applying a rule disabling strategy, the defender will thus considerably reduce the number of objects to be investigated while maintaining a sufficiently high coverage rate. However, it should be emphasized that, unlike a detection system which aims to automatically contain a technical threat (e.g., anti-malware, Endpoint Detection and Response), the defensive infrastructure, as described in this article, aims to collect as much Cyber Threat Intelligence as possible related to the adversary in order to be able to better hunt it in the victim's network. Thus, an exhaustive detection is not necessary since the objective is not to block unknown threats but simply to ensure the sufficient number of **IoC** that will allow it to be tracked in the victim's network. This approach of measuring the coverage rate is possible only in the context of *Red versus Blue* exercises. However, the disabling of too verbose rules can be done in post-processing without observing the evolution of the coverage rate, but simply in order to clean up the graph generated by the defender and to make it more exploitable.

VII. Related Works

Threat Hunting is an agile and iterative process of searching, characterizing, and later identifying attackers who may have compromised the victim's network. In 2021, it is still a widespread focus of cyber defense research. We believe that this area still requires further formalization efforts in order to allow a good understanding of the different layers and their interactions. We observe two main lines of research that should converge: those that formalize the relationships between real-life components; and those focusing on the improvement of actors' strategies, for instance, based on Game Theory.

a) Need for unified views: Gianvecchio *et al.* [11] point out the semantic gap between the defender and the attacker. Indeed, the attacker operates at the level of strategy and tactics; he focuses on target discovery, and can deploy various kill chains tactics [8, 12]. However, the defender spends significant time processing low-level, rule-generated alerts and single-log analysis can hardly reveal the complete attack story for complex, multi-stage attacks. Gianvecchio *et al.* reduce this gap by proposing an explicit model of the attacker strategy using machine-readable data structures and clustering of security events around TTP annotations from a well-known behavioral taxonomy. This model enables the defender to operate similarly to the attacker at the strategic level without sacrificing their ability to drill into evidential details. For their implementation, they used CALDERA, previously introduced by Applebaum *et al.* [13] in order to automate Red Teaming while retaining the concept of TTP.

b) From Events of Interest to objects: In [14], Najafi *et al.* are one of the first to introduce the intuition behind global features for threat detection. They define a SIEM-based knowledge graph allowing highlighting the most important entities (what we call *objects*) and relationships observed in Event Logs of Interest extracted from DNS and proxy logs. These relations are enriched with information gathered from publicly available sources of threat intelligence. Over this knowledge graph, Najafi *et al.* design MalRank, a graph-based inference algorithm designed to infer a node maliciousness score based on its associations to other entities presented in the graph. MalRank has successfully been helpful in identifying previously unknown malicious entities such as malicious domain names and IP addresses.

c) From traces to Indicator of Compromise: In [9], Kurogome *et al* propose to enhance the Threat Hunting process by automatically generating accurate and interpretable IoCs from malware traces. They design EIGER that takes a dataset of traces computed from malware as input. EIGER then computes IoCs of different abstraction levels using an *enumerate-then-optimize* algorithm. Kurogome *et al* demonstrate that their generated IoCs bear comparison with manually generated ones, which indicates that EIGER is an appealing complement to endpoint malware detection in real-world security operations. This is an example of a concrete implementation for the function **generate_IoC** formalized here.

d) From logs to defender's perspective: Pei *et al* describe in [15] HERCULE a log-based intrusion analysis system. It models the relationship between multiple logs in the system and automatically generates a multidimensional weighted graph with potentially valuable information for the defender embedded within. Their proposed graph provides *a panoramic view* of the logs generated by different system components and help the defender to understand the whole attack trace. Recently, Burr *et al* published a study [16] that focuses on community detection in graphs constructed from Intrusion Detection System (IDS) alerts.

This article is in line with these works and proposes a richer model that offers a dual view of the Threat Hunting process by also taking the perception into account, but also knowledge and actions of the campaign from the attacker perspective. We believe that in the near future, this work will allow us to join the research conducted in Game Theory and in Security Games [17, 18] where the Threat Hunting process requires more accurate models [19]. In these games, the defender, who is monitoring some collection of resources, has to decide how to deploy a certain number of sensors or honeypots [20] at some predetermined cost. His goal is to properly protect this network at a minimal cost. The attacker's goal is to cheat the defender in order to reach his objectives.

VIII. Conclusion

Threat Hunting is a fundamental step of an incident response operation that allows for spotting the components of an information system compromised by an attacker. In this process, the defender's ambition is to shed light on the attacker's propagation area in order to best prepare the operation for his eradication.

In this article, we have proposed a model to analyze both the attacker propagation and the defender knowledge of that propagation. All the steps involved in a Threat Hunting approach have been carefully formalized here. Our approach allows enhancing the knowledge base of the defender with new Indicators of Compromise, which can subsequently enable proactive threat detection. Furthermore, our model and its experimentation highlight the existence of false-positives in particular because of lax detection rules. This feature is valuable for the defender since it allows him to gain efficiency and to improve his detection tools. In particular, because the graph analysis subserves the attack correlations by identifying event patterns. During this study, we better understood the origin of objects which then become IoC. We also emphasize that some objects, which have a meaning only in the context of the information system where they were found, can be very interesting to exploit for Threat Hunting. Finally, our model explains the mutual inference necessary for attacker and defender to understand each other. The most valuable intelligence is the understanding of attacker's procedures.

In future work, we plan to focus on graph similarity computation in order to design metrics that could testify to the quality of attacker and defender points of view. Such experiments require implementing different comparison algorithms and benchmarking them. However, beyond a metric confrontation between two graphs, we hope the semantics introduced in this paper allows for interpreting the differences between attacker and defender perceptions at a deeper level, towards designing other defensive procedures.

In the long run, our goal is to adjust defender levers defined in this article dynamically: sensor configuration, detection rules, and IoC database to better reveal the presence of an attacker. To achieve this goal, we might need to define a defensive strategy that takes deployment costs into account. Thus, a *Red versus Blue* exercise would have concrete and immediate technical repercussions on the company's cybersecurity to fight Advanced Persistent Threats.

References

[1] E. C. Thompson, "Threat Hunting," in *Designing a HIPAA-Compliant Security Operations Center*. Apress, Berkeley, CA, 2020.

[2] F. Maymi, R. Bixler, R. Jones, and S. Lathrop, "Towards a definition of cyberspace tactics, techniques and procedures," in *2017 IEEE International Conference on Big Data*. IEEE, 2017.

[3] R. Rodriguez. (2020) APT29 activity from the ATT&CK evaluations. [Online]. Available: https://github.com/hunters-forge/mordor/tree/master/datasets/large/apt29

[4] MITRE Corporation. (2019) ATT&CK evaluation. [Online]. Available: https://attackevals.mitre-engenuity.org/

[5] OASIS Cyber Threat Intelligence. (2017) STIX a structured language for cyber threat intelligence. [Online]. Available: https://oasis-open.github.io/cti-documentation/stix/intro

[6] MITRE Corporation. (2018) The MITRE Cyber Analytics Repository (CAR). [Online]. Available: https://car.mitre.org/

[7] V. Mavroeidis and A. Jøsang, "Data-Driven Threat Hunting Using Sysmon," in *Proc. of the 2nd Int. Conf. on Cryptography, Security and Privacy*. ACM Press, 2018.

[8] A. Berady, V. Viet Triem Tong, G. Guette, C. Bidan, and G. Carat, "Modeling the Operational Phases of APT Campaigns," in *6th Annual Conf. on Computational Science and Computational Intelligence*. IEEE, 2019.

[9] Y. Kurogome, Y. Otsuki, Y. Kawakoya, M. Iwamura, S. Hayashi, T. Mori, and K. Sen, "EIGER: automated IOC generation for accurate and interpretable endpoint malware detection," in *Proc. of the 35th Annual Computer Security Applications Conf.* ACM, 2019.

[10] Sudhakar and S. Kumar, "An emerging threat fileless malware: a survey and research challenges," *Cybersecurity*, 2020.

[11] S. Gianvecchio, C. Burkhalter, H. Lan, A. Sillers, and K. Smith, "Closing the Gap with APTs Through Semantic Clusters and Automated Cybergames," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Int. Publishing, 2019.

[12] P. N. Bahrami, A. Dehghantanha, T. Dargahi, R. M. Parizi, K.-K. R. Choo, and H. H. S. Javadi, "Cyber Kill Chain-Based Taxonomy of Advanced Persistent Threat Actors: Analogy of Tactics, Techniques, and Procedures," *JIPS*, 2019.

[13] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent, automated red team emulation," in

Proc. of the 32nd Annual Conf. on Computer Security Applications. ACM, 2016.

[14] P. Najafi, A. Mühle, W. Pünter, F. Cheng, and C. Meinel, "MalRank: a measure of maliciousness in SIEM-based knowledge graphs," in *Proc. of the 35th Annual Computer Security Applications Conf.* ACM, 2019.

[15] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph," in *Proc. of the 32nd Annual Conf. on Computer Security Applications*. ACM, 2016.

[16] B. Burr, S. Wang, G. Salmon, and H. Soliman, "On the detection of persistent attacks using alert graphs and event feature embeddings," in *NOMS 2020*. IEEE, 2020.

[17] A. H. Anwar and C. Kamhoua, "Game theory on attack graph for cyber deception," in *Decision and Game Theory for Security*. Springer Int. Publishing, 2020.

[18] T. E. Carroll and D. Grosu, "A Game Theoretic Investigation of Deception in Network Security," in *2009 Proceedings of 18th International Conference on Computer Communications and Networks*. IEEE, 2009.

[19] M. Bilinski, J. diVita, K. Ferguson-Walter, S. Fugate, R. Gabrys, J. Mauger, and B. Souza, *Lie Another Day: Demonstrating Bias in a Multi-round Cyber Deception Game of Questionable Veracity*. Springer, 2020.

[20] R. Píbil, V. Lisý, C. Kiekintveld, B. Bošanský, and M. Pěchouček, "Game Theoretic Model of Strategic Honeypot Selection in Computer Networks," in *Decision and Game Theory for Security*. Springer, 2012.

APPENDIX

a) Sensor configuration: Expressions ϕ of sensor configurations are built from logical constants true, false and operators and, or, not applied over pairs $(\mathbf{r}, c)$ where $\mathbf{r}$ is a role and c is a property over the object playing this role (we write $c(\mathbf{o})$ to express that an object $\mathbf{o}$ satisfies the property c):

$$\phi ::= \mathsf{true} \mid \mathsf{false} \mid (\mathbf{r}, c) \mid \mathsf{not}\ \phi \mid \phi\ \mathsf{and}\ \phi \mid \phi\ \mathsf{or}\ \phi$$

Then, given a set $O = \{(\mathbf{o}_1, \mathbf{r}_1), \cdots, (\mathbf{o}_n, \mathbf{r}_n)\} \subseteq \mathbf{O} \times \mathbf{R}$, the satisfaction relation $\models$ of a condition is defined by:

$$\begin{array}{lll} O \models \mathsf{true} & & \\ O \models (\mathbf{r}_i, c) & \text{iff} & c(\mathbf{o}_i) \\ O \models \mathsf{not}\ \phi & \text{iff} & O \not\models \phi \\ O \models \phi_1\ \mathsf{and}\ \phi_2 & \text{iff} & O \models \phi_1 \text{ and } O \models \phi_2 \\ O \models \phi_1\ \mathsf{or}\ \phi_2 & \text{iff} & O \models \phi_1 \text{ or } O \models \phi_2 \end{array}$$

We assume here that for each role in ϕ there exists an object in O playing it and that a role occurs at most one time in O.

b) Detection rule: Detection rules r are built from logical constants true, false and operators and, or, not applied over pairs (α, c) where α is a timestamp t, or an event type $\epsilon \in \mathcal{E}$, or a component $\mathbf{c} \in \mathbf{C}$, or a role $\mathbf{r} \in \mathbf{R}$, and c is a property over α (we write $c(\alpha)$ to express that α satisfies the property c):

$$r ::= \mathsf{true} \mid \mathsf{false} \mid (\alpha, c) \mid \mathsf{not}\ r \mid r\ \mathsf{and}\ r \mid r\ \mathsf{or}\ r$$

Then, given a trace $\mathbf{x} = (\mathsf{id}_\mathsf{x}, t, \epsilon, \mathbf{c}, O)$, the satisfaction relation $\models$ of a detection rule r is inductively defined by:

$$\begin{array}{lll} \mathbf{x} \models \mathsf{true} & & \\ \mathbf{x} \models (t', c) & \text{iff} & t' = t \text{ and } c(t) \\ \mathbf{x} \models (\epsilon', c) & \text{iff} & \epsilon' = \epsilon \text{ and } c(\epsilon) \\ \mathbf{x} \models (\mathbf{c}', c) & \text{iff} & \mathbf{c}' = \mathbf{c} \text{ and } c(\mathbf{c}) \\ \mathbf{x} \models (\mathbf{r}_i, c) & \text{iff} & (\mathbf{o}, \mathbf{r}_i) \in O \text{ and } c(\mathbf{o}_i) \\ \mathbf{x} \models \mathsf{not}\ r & \text{iff} & \mathbf{x} \not\models r \\ \mathbf{x} \models r_1\ \mathsf{and}\ r_2 & \text{iff} & \mathbf{x} \models r_1 \text{ and } \mathbf{x} \models r_2 \\ \mathbf{x} \models r_1\ \mathsf{or}\ r_2 & \text{iff} & \mathbf{x} \models r_1 \text{ or } \mathbf{x} \models r_2 \end{array}$$

c) Main notations:

– $\mathbf{C}$ is a set of components of the targeted network; $\mathbf{c}$ is a component in $\mathbf{C}$

– $\mathbf{O} = \mathbf{O}_D \cup \mathbf{O}_A$ where $\mathbf{O}_D$ (resp. $\mathbf{O}_A$) is the set of objects relative to the victim (resp. to the attacker); $\mathbf{o} \in \mathbf{O}$

– $\mathbf{R}$ is a set of roles associated with objects

– $\mathbf{T}$ is set of types associated with roles; $\mathbf{t}$ is a type in $\mathbf{T}$

– $\tau : \mathbf{R} \rightarrow \mathbf{T}$ is the function from roles to types

– $\mathbf{D}_A$ (resp. $\mathbf{D}_D$) : $(\mathbf{T} \times \mathbf{O}_D) \rightarrow (\mathbf{C} \cup \{\mathsf{None}\})$ is the attacker (resp. defender) directory

– $\mathcal{E}$ is a set of types of observable events (by the defender) on components; ϵ is an event in $\mathcal{E}$

– $ev = (\epsilon, \mathbf{c}, O)$ is an observable event on a component

– $O \subseteq \mathbf{O} \times \mathbf{R}$ is a (sub)set of objects with their role

– $\mathbf{x} = (\mathsf{id}_\mathsf{x}, t, \epsilon, \mathbf{c}, O)$ is a trace observed on $\mathbf{c}$ at date t

– $\mathcal{S}$ is a set of sensor configurations

– $\mathcal{S}(\mathbf{c})$ is a sensor configuration on the component $\mathbf{c}$

– $\mathbf{TTP}$ is a set of procedures; p is a procedure in $\mathbf{TTP}$

– $\mathcal{A} = \mathbf{e}_1, \cdots, \mathbf{e}_N$ is an attack campaign

– $\mathbf{e}$ is a procedure execution

– $\mathcal{M}(\mathbf{e})$ is a machine (component) on which $\mathbf{e}$ is executed

– ϕ is a condition over some objects and their roles

– $\mathsf{R}_\epsilon \subseteq \mathbf{R}$ are relevant roles for an event type ϵ

– $(\epsilon, \phi, \mathsf{R}_\epsilon)$ is an element of a sensor configuration

– $\mathbf{IoC} \subseteq \mathbf{O} \times \mathbf{T}$ is an *Indicators of Compromise* database

– $\mathcal{R}$ is a set of detection rules; r is a rule in $\mathcal{R}$

– $\mathbf{EoI}_{\mathcal{R},\mathbf{IoC}}(\mathbf{x}) = (R_\mathbf{x}, O_\mathbf{x})$ is the function providing relevant rules ($R_\mathbf{x} \subseteq \mathcal{R}$) and objects ($O_\mathbf{x} \subseteq \mathbf{O}$) from a trace generated by an *Event of Interest*

– $\mathbf{G}(\mathbf{e}) = (V_\mathbf{e}, \rightarrow_\mathbf{e})$ is an attacker's graph relating to an execution $\mathbf{e}$

– $\mathbf{G}_A(\mathcal{A}) = (V_A, \rightarrow_A)$ is the attacker's network propagation graph resulting from the $\mathcal{A}$ attack campaign

– $\mathbf{G}(\mathbf{x}, R_\mathbf{x}, O_\mathbf{x})$ is a defender's graph relating to a trace $\mathbf{x}$

– $\mathbf{G}_D(X) = (V_D, \rightarrow_D)$ is the defender's perception graph of the attacker's propagation associated with each *Event of Interest* computed from a set of traces X