Adversary Tactic Driven Scenario and Terrain Generation with Partial Infrastructure Specification

Ádám Ruman
ruman@fi.muni.cz
Masaryk University
Brno, Czech Republic

Martin Drašar
drasar@ics.muni.cz
Masaryk University
Brno, Czech Republic

Lukáš Sadlek
sadlek@mail.muni.cz
Masaryk University
Brno, Czech Republic

Shanchieh Jay Yang
jay.yang@rit.edu
Rochester Institute of Technology
Rochester, NY, USA

Pavel Čeleda
celeda@fi.muni.cz
Masaryk University
Brno, Czech Republic

ABSTRACT

Diverse, accurate, and up-to-date training environments are essential for training cybersecurity experts and autonomous systems. However, preparation of their content is time-consuming and requires experts to provide detailed specifications. In this paper, we explore the challenges of automated generation of the content (composed of *scenarios* and *terrains*) for these environments.

We propose new models to represent the cybersecurity domain and associated action spaces. These models are used to create sound and complex training content based on partial specifications provided by users. We compare the results with a real-world complex malware campaign to assess the realism of the synthesized content. To further evaluate the correctness and variability of the results, we utilize the kill-chain attack graph generation for the generated training content to asses the internal correspondence of its key components.

Our results demonstrate that the proposed approach can create complex training content similar to advanced attack campaigns, which passes evaluation for soundness and practicality. Our proposed approach and its implementation significantly contribute to the state of the art, enabling novel approaches to cybersecurity training and autonomous system development.

CCS CONCEPTS

• **Security and privacy** → *Systems security*; **Formal security models**; • **Computing methodologies** → *Modeling and simulation.*

KEYWORDS

cybersecurity model, adversary framework, attack scenario generation, cyber terrain generation

ACM Reference Format:
Ádám Ruman, Martin Drašar, Lukáš Sadlek, Shanchieh Jay Yang, and Pavel Čeleda. 2024. Adversary Tactic Driven Scenario and Terrain Generation with Partial Infrastructure Specification. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30-August 2, 2024, Vienna, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3664476.3664523

ARES 2024, July 30-August 2, 2024, Vienna, Austria

1 INTRODUCTION

Training environments are prominent in human experts' education and autonomous systems training. Regardless of the use case, these environments need to be diverse, correct, and up-to-date with the current developments of the targeted domain. Preparing training content for such environments is highly demanding in cybersecurity, requiring detailed knowledge of the latest cybersecurity and system administration practices. The content blueprints also need thorough testing to eliminate unintended but successful decision sequences, which hinder the training's efficacy. Thus, cybersecurity training content can be perceived to be composed of two tightly connected concepts: the ***scenario*** – describing all executable decision sequences, and the ***terrain*** – the technical infrastructure description allowing and limiting the execution of decisions as dictated by the scenario.

To alleviate the difficulties in creating training environment content – stemming from the lack of experts and their time – past cyber exercises and malware campaigns can be used as templates to acquire training content. However, there are two primary challenges in fully utilizing the data they can provide. First, their limited quantity, diversity, and coherence restrict the number and variance of the training. Second, the absence of a standardized data format across the sources entails significant transformational efforts for specific applications.

One avenue involves the artificial generation of adversarial decision sequences to address the limitations of utilizing past exercises and campaigns. It has the potential to deliver a substantial volume of diverse and coherent data shaped by the variations in input specifications in a guaranteed – unified format.

The primary objective of this paper is to design a mechanism for the automated generation of training content – of ***terrains*** and ***scenarios***. The generation should be parameter driven: *requirements* for the training's properties and goals – shaping the scenario, and *hints* limiting or specifying the technical layout of the terrain.

To accomplish its objective, this work presents multiple contributions. First, it introduces a novel action framework ensuring the verifiable logical coherence of the *scenarios*. Second, it devises an

infrastructure model facilitating the generation of *terrains*. Additionally, it proposes a distinctive generative algorithm that diverges from prevailing approaches by concurrently generating *terrains* and *scenarios*. This last contribution encompasses implementation, accompanied by evaluations of its outputs. We first assess the realism of produced results through manual comparison of generated *scenarios* and *terrains* with an example of an advanced persistent threat campaign. Then we utilize kill-chain attack graph generation to automatically create attack graphs for generated *terrains* and compare them with the respective *scenarios* to assess correct correspondence in generated *scenario-terrain* pairs.

The paper is structured as follows. Section 2 describes related work from cybersecurity infrastructure modeling, cyber threat scenario modeling, and their joint modeling. Section 3 introduces a novel domain model and an actor framework. In Section 4, we propose an algorithm for the synthesis of *terrains* and *scenarios* and introduce **PAGAN** – a prototype implementation. In Section 5, we evaluate the generated *terrains* and *scenarios* for mutual correspondence and realism. In Section 6, we conclude the paper.

2 RELATED WORK

To our best knowledge, parallel generation of cybersecurity terrains with non-trivial branching scenarios does not exist as a prior art. Nevertheless, our proposed approach draws upon insights from several research areas: attackers' behavior modeling, topology modeling, and attack plan and threat scenario generation. In the following text, we present notable representatives from each area and discuss them primarily concerning scenario generation.

Some taxonomies and frameworks have been proposed for attackers' behavior modeling, often tailored for specific research problems, such as the Action-Intent Framework [21]. However, only several frameworks can be considered widely recognized and developed. The most prominent are *Lockheed Martin's Killchain* [13], and the *Unified Killchain* [24], which are attack-phase based, and MITRE ATT&CK [6] with its tactics, techniques, and procedures (TTP), which is matrix based.

For the modeling of topology and infrastructure settings, there exist generic models not tied to a specific software tool, such as the *Cyber Terrain* [25] or the *MITRE D3FEND's Digital Artifact Ontology* [5]; the models internal to specific simulation software, such as *NS3* [26] or others analyzed in [8]; and custom models employed by threat scenario generation approaches mentioned later in this section.

Attack plan and threat scenario generation approaches can be divided into two groups: *synthetic*, which produces attack plans without apriori knowledge of the terrain, and *analytic*, which creates the attack plans from the given terrain.

The *synthetic* group generally depends on extending the attackers' behavior models so that some relations between actions can be derived and used to construct plausible attack plans. An example is the tool [32], which enriches the MITRE ATT&CK framework with so-called "promises", and therefore can plan linear attack paths.

The *analytic* group relies on adjustments of the models as well but also requires fixed information about the security posture of targets, such as network topology, and the presence of vulnerabilities, e.g., [3]. Due to its utilization in cybersecurity analysis, this group is more voluminous than the synthetic group.

One of the prevalent methodologies for the analytic group is to employ attack graph generation [16], which attempts to predict possible paths of breaching the infrastructure. Examples of such tools relying on topology descriptions and vulnerability information are *FireMon Security Manager* [10], *MulVAL* [22] or [31]. Others, such as *Cauldron* [14], also accept rules for chaining attack steps together. Therefore, meta-languages have been developed to describe the inputs to these algorithms, such as [15]. The *KCAG* [30] combines the MITRE ATT&CK and the Killchain frameworks and tweaks the output of the *MulVAL* tool to create attack graphs that obey the rules of killchains. An example of a more "exotic" approach is shown in [17], which uses symbolic simulations to construct threat scenarios.

Another option for threat scenario exploration is to base it on data originally intended for reactive defenses. Endpoint Detection and Response (EDR) tools detect ATT&CK techniques and can construct their sequences (for example, Kaspersky's commercial EDR constructs activity trees [1]). Meanwhile, [4] uses intrusion detection alerts to construct attack graphs. The tool presented in [20] uses kernel audit logs to construct provenance graphs, demonstrating that their results are equal to graphs extracted from cyber threat intelligence data. To combine the two previous mentions, [12] improves the results of EDRs by filtering false positive detection alerts using tactical provenance graphs with causal dependencies between alerts.

2.1 Limitations of Existing Approaches

The limitations of the approaches mentioned above are not shortcomings but rather arise from the difference in their objectives compared to our use case. The most notable differences are generally:

Linearity vs. forkings – The majority of existing attack plan generators create linear paths of actions. While this may be the most interesting aspect, some applications can benefit from considering all the different ways to reach a goal (e.g., training autonomous systems).

Success-only approach – Threat scenario generators are frequently goal-driven, occasionally so much that they discard entirely states that do not lead to it. In specific applications, the information about reachable states diverging from intended trajectories may be of equal value.

Attack path optimality – Attackers do not always take the optimal path to reach their objective(s). This might be due to circumstances such as preferring stealthiness, the need to use decoy actions to confuse defenders, or simply due to available skill and resources. We have identified a bias in existing tools to prefer optimal paths.

3 SCENARIO AND TERRAIN MODELING

In the preceding section, we emphasized generating non-trivial branching scenarios and terrains. Such scenarios are relatively uncommon in the cybersecurity landscape, typically represented by attacks at the behest of state actors. However, they are the most damaging and sophisticated, representing a superset of the more

common attacks. Consequently, we utilize these complex scenarios as a template for our models. Our models are constructed to enable the expression of these complex scenarios, and the need for this expressivity is influencing our design.

In this paper, we use the Nitro Advanced Persistent Threat (APT) [23] to illustrate a non-trivial attack to explain our choices. The Nitro APT is a lesser-known campaign endeavoring to exfiltrate industrial secrets and intellectual property. It employs techniques to gain a foothold in the target infrastructure, establish command and control points, escalate privileges, and move laterally. These techniques can be arbitrarily chained depending on the target infrastructure and the response of its users.

An attempt to construct a multitude of Nitro-like *scenarios* in different *terrains* and contexts reveals the main issue, which is discarded by the tools mentioned in the previous section – that the *terrain* and *scenario* have to be generated concurrently, to produce correct and realistic outputs. Therefore, it is impossible to begin by generating the *terrain* and fixing it because this would not guarantee that any *scenario* can be executed within. The other direction also has its issues, namely that there is an infinite number of *terrain* configurations satisfying a *scenario*. This is further elaborated in Section 4.

To address this concurrency issue, we construct our model to enable expressing both the *scenario* and the *terrain* in detail and to enable an easy interplay between the two[1].

3.1 Scenario Models

We classify attack posture, adversarial actions, and their connection to model a scenario. These classifications were distilled by analyzing adversary techniques and focusing on their requirements. While they may appear arbitrary at first glance, this classification supports the terrain model, which will be described in greater detail later in the text.

3.1.1 Attack posture: We created the **Adversary Control State (ACS)** to express the attack posture. This is a collection consisting of three types of "artifacts", as described in Table 1, which the adversary acquires throughout the *scenario* execution.

Table 1: Artifacts of the ACS.

Artifacts	Description
Tokens	Express accesses to sub-networks and devices, the existence of sessions, or infected files/drives/emails.
Credentials	For authenticating into systems and services.
Capabilities	Express fine-grained permissions for using a given system.

While tokens and credentials are self-explanatory, further elaboration on the **capability** artifact type is required. Its values were derived from Microsoft's access control mechanisms in its operating systems and filtered by analyzing prerequisites for MITRE ATT&CK techniques. We present the enumeration of its values in Table 2. In addition to values in the table, **capabilities** belong to one of the four strength classes: *service*, *user*, *administrator*, and *system*.

[1]The particular order of model components in this section does not suggest the chronological order of their generation.

Table 2: Capabilities used in our model.

Family	Capability
System Operations	Config Local User, Config Local Machine, Shutdown, Control Domain, CodeExecution
Security	Debug Processes, Load Drivers, Create Security Tokens (*ST*), Impersonate *ST*
Account Properties	Persist Logout, Impersonate, Domain Login, UAC Elevate, Access Network as Domain
Account Manipulation	Add User, Delete User, Edit User
FileSystem Access	Generic, Elevated, Shared
FileSystem Operations	Read, Write
Utility	Persistence

3.1.2 Adversarial actions: Scenario generation is heavily adversary-driven; thus, the most critical set of available actions is that of the hypothetical attacker. When generating a scenario, mimicking an attacker's decision process is necessary. Therefore, the framework should contain information that encodes when a technique is applicable and what paths it opens.

Existing adversary frameworks such as the Cyber Kill Chain [13] and its unified version [24] aim to map their techniques to a sequence, which is overly strict and deviates from how attackers operate.

MITRE ATT&CK® [6] contains only minimal information about when a technique can be applied (formulated vaguely such as: "you need an administrator account to do this"). On the other hand, it is the most complete framework, so it is sensible to use it as our baseline when enumerating techniques.

None of the existing frameworks fully fit our needs, so the question is: *can we mend or enrich MITRE ATT&CK® to contain granular enough information to base decision-making upon it? If yes, how do we encode this information?*

We have already introduced the **ACS**, which we can use to encode the needed information. The idea is to enrich the techniques so that they express:

- their requirements on the actual **ACS** for their application,
- existence requirements for **enablers** (vulnerabilities, misconfigurations, etc.),
- rules governing the changes in the **ACS** after their application.

The resulting structure of a technique's **usage** is described in Listing 1.

Listing 1: Encoding of technique enriching information.

```
struct UsageScheme {
/*** ACS requirements ***/
  given:  Vec<Token>,
  required_capabilities: dyn Predicate<CapabilityTemplate>
  required_enabler: Option<EnablerTemplate>,
 // special rules for when not to use the action
  except_if: dyn Predicate<Token>,
/*** ACS additions ***/
  grant: Vec<Token>,
```

```
    provided_capabilities: Vec<Capability>,
  // what specifies the inherited privilege level
    privilege_origin: PrivilegeOrigin,
  //What limits the set of inherited capabilities
    capability_specifier: CapabilitySpecifier,
  // same service? other services? any?
    credential_target: CredentialTarget,
  // os creds? app creds?, MFA?
    credential_type: CredentialType
}
impl Predicate for
All, Any, Not, NoneOf, NotJust, Empty {}
```

To create our framework, we augmented ATT&CK®. In this process, some of the original techniques had to be split (e.g., the lateral movement action *Taint Shared Content* was divided into two techniques, one for creating the tainted content, the other for opening it); others, we merged if their enriching usages were equivalent.

The resulting framework, called ***Quiver***, contains 53 actions, each with one or more transformation options. The actions are not constrained by any sequential ordering, and their associations are easily derived from their relation to the **ACS**.

*3.1.3 Connecting **ACS** and **Quiver**:* The **ACS** can be regarded as a state, and the ***Quiver*** framework as a set of transformations of that state. Then, a finite-state automaton as a structure for scenarios naturally offers itself. Figure 1 presents a graphical representation of the scenario structure.

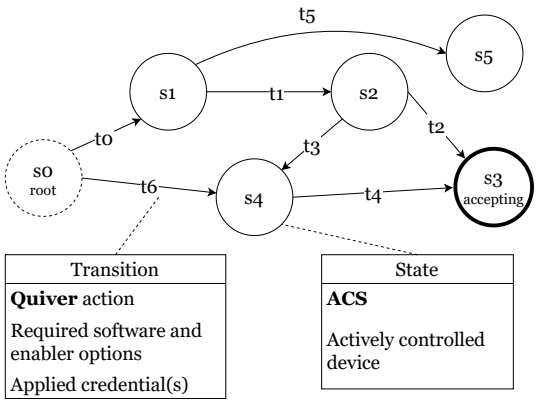


Figure 1: Scenario automaton structure.

3.2 Terrain Models

With the *scenario* modeled as a combination of **ACS** and **Quiver**, we need to model the *terrain* to accommodate and organize the elements of the *scenario* models. The two key questions driving the design of the model are then: *What details do we need to represent with it?* and *How to encapsulate the nuances required by the scenario?*

At the highest level, the *terrain* is represented as a set of interconnected (networked) devices. Its details are derived from the breach mitigation steps provided by the MITRE DEFEND™ [7] and guidelines and descriptions of common security-related processes (such as authentication, access control and authorization, user isolation, etc.) in the Microsoft documentation [19]. Due to the complexity of the model, we describe it as a class diagram in Figure 2 and elaborate on some of the more interesting minutiae (mostly the intertwining with the scenario) in the following text.

The tokens of the **ACS** are exclusive to the attacker and not part of the terrain. On the other hand, capabilities are organized and encapsulated by accounts and, transitively, devices. Credentials are pretty standalone structures of the terrain model, as they represent a piece of knowledge. In the terrain, they provide access to services (in the scenario, this translates to allowing exploits requiring authorization) and accounts (translating to obtaining new capabilities). Other components of the scenario present in the terrain are software and enablers. Software is deployed on devices as an executable or a process under a specific privilege level. Enablers are also deployed on a per-device basis.

3.3 Model Element Instantiation and Relations

Some of the elements of the designed models must mirror their real-world counterparts (software and enablers). Collecting the assets themselves is straightforward (although the trustworthiness and completeness of the data are sometimes questionable), as there are existing databases provided by **NIST**. On the other hand, we rely on **relations** between them, such as which vulnerability facilitates the use of a given adversary action, which software (setup) allows applying a given technique, etc. These relations are more cumbersome to muster. Some connections can be obtained from the same databases as the assets; others have to be derived from unstructured sources (such as analysis reports of system compromises or OSINT). Momentarily, we are working with a small set of manually gathered relations that fairs well in the trial period but will have scalability issues when we move on to large-scale use of our proposed tool. Thus, we started exploring the usage of Large Language Models (LLM) for gathering and enhancing relationships in an automated manner.

4 SCENARIO AND TERRAIN GENERATION ALGORITHM

With the model defined, we introduce the computations, which result in a *scenario* and a corresponding *terrain* that enables the execution of the former. The two processes are tightly tied to each other. In general:

- **Scenario** generation is based on state-exploration. Given a starting state, it discovers all other states of the **ACS** that are reachable under the action space of the attacker, as described by the *Quiver* framework.
- The **terrain** generation is multifaceted, as it imposes restrictions on the reachable states of the scenario. However, it must also obey its results and shape the terrain to map the scenario precisely.

Their dependencies significantly influence the scheduling of *scenario* and *terrain* generation. We have investigated the following options:

- **Scenario generation followed by terrain creation** – This approach proved infeasible due to the arbitrary nature of

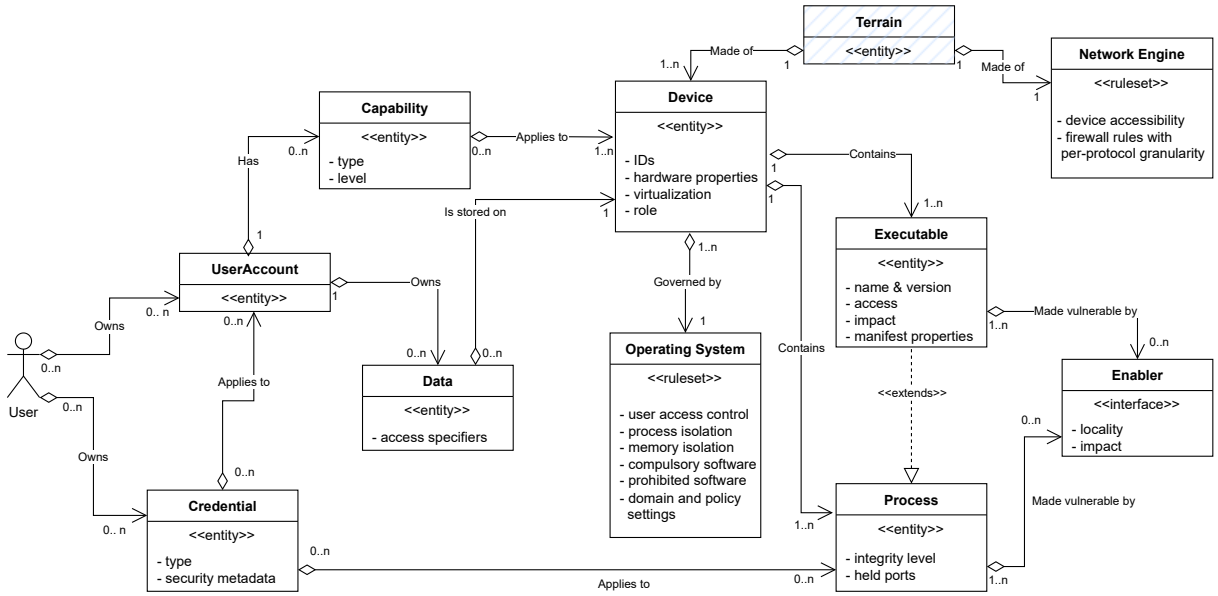


Figure 2: Terrain model as a UML class diagram.

the generated scenarios. We would need to explore all hypothetical states an attacker can reach in any hypothetical infrastructure (the number of technique combinations grows exponentially with the allowed length of the paths). Therefore, we need to cut back generation based on depth or breadth, which is very ineffective. And even then, we lack assurance that the *terrain* mapped onto the *scenario* will make practical sense.

- **Terrain generation followed by scenario creation** – Contrary to the previous approach, we try to generate a meaningful *terrain* that imposes heavy restrictions on the reachable states of the *scenario*. Because we already have tools that can generate attack plans for fixed *terrains*, this approach effectively reduces our problem to that of generating the *terrain*. However, guaranteeing the possibility of a meaningful and logically sound *scenario* is not trivial.
- **Interspersing the two** – From the shortcomings of the previous approaches, we can observe that *scenario* generation needs limitations from the *terrain*, but at the same time, the *terrain* has to permit alterations to accommodate the *scenario*. This can be achieved by dividing the two generation processes into smaller units and scheduling them in an alternating manner.

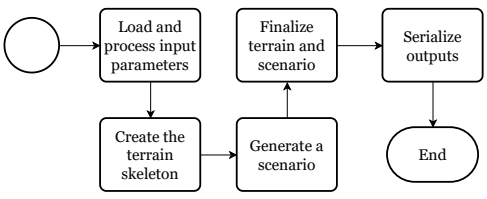


Figure 3: High-level algorithm flowchart.

We have chosen to pursue the last option. Figure 3 depicts the high-level flow of the algorithm. The interweaving of work units at this abstraction level is quite simple, as we divide the terrain generation into two parts and separate them by the scenario generation. In the following subsections, we detail these high-level work units.

4.1 Input Parameters

Although it is desirable to maximize diversity in the generated terrains and scenarios, this must be balanced with realism and meaningfulness. To achieve this, we have opted for a parameter-driven approach to generation. The abstraction level of the parametrization determines the **partial infrastructure specification** required for generation. The following three aspects of parametrization have been chosen:

Users – A set of personas specifying the people using and maintaining the infrastructure. Crucial information about them is their roles within the infrastructure *(e.g., common user, specialized administrator)*, and their IT proficiency, among others. Later, we use these metrics to synthesize user accounts and give them authorizations. They also help with personalizing the scenario (for example, if an adversary action requires some software, and we have multiple choices, we can prioritize based on the person's role in the company).

Adversary – The parametrization includes the objective to be achieved, represented as an action within the *Quiver* framework, along with the type of device to be targeted (e.g., domain controller, database, desktop, etc.). Additionally, it encompasses any additional side effects (e.g., leaving a backdoor on a device), the number of devices to be utilized throughout the attack, and the amount of computing power available for online and offline password cracking.

Infrastructure – The most complex aspect of the three specifies the presence or absence of specialized network segments (domain

control, demilitarized zone, operational technologies, etc.) and devices in them. Also, whether the infrastructure is domain-controlled and some aspects of segment isolation. For the full specifications of parametrization, please refer to the source code [28].

4.2 Terrain Skeleton Synthesis

With the input parameters provided by the user, the system focuses on creating a *terrain* skeleton to limit the *scenario* state space. The terrain skeleton is based on a template (including communication rules and access policies). This is because most real-world networks – even if heavily diverse – conform to some best practices and conventions. We mix this template with the provided input parameters and a little randomness to generate *i)* network segments, *ii)* devices with their roles, operating system, software that must or must not be present, *iii)* reachability between devices (via network, shared filesystem, removable media), *iv)* communication rules (firewall) between devices (with protocol granularity), *v)* user accounts, access control rules, capabilities, and authorizations, and *vi)* credentials to accounts and services, with their strength against cracking.

A challenging part of the *terrain* generation is account and credential synthesis due to their complexity and dependencies with the *scenario* generation. Specific lessons learned we encountered include:

- Without predefined accounts, the options to consider are too numerous. Take, for example, an attacker who has one shot at privilege escalation from a **user** account. Then, we must consider all possible state transformations based on all possible combinations of encapsulated capabilities, further divided by the possible combinations of devices they are applicable on and privilege levels. This would result in an enormous explosion of state space.
- State explosion is also problematic when considering adversary techniques that provide credentials. Let's consider system-level *keylogging*, for example, that can provide virtually any password to any service, device, or account for any person coming into contact with the infected device. Considering that many options would be infeasible, thus pre-generation is necessary.

4.3 Scenario Generation

We begin by generating all the sequences of devices that might create a successful attack path (limited by attack path length from the parameters). Then, we create or expand the scenario automaton for each such sequence. We begin with the **ACS**, where the attacker resides in the untrusted internet (represented by a single device), has access to their target infrastructure either as anyone on the internet, or with special knowledge or access (such as knowing some credentials, already having a backdoor). Then, from this state, we query actions the adversary can take; we consider the minutiae of these actions' applications, such as deviations based on the exploited software, machine settings, the credentials used, etc. If we reach an already explored state (with the same following device sequence), we might terminate the computation at this point. This way, we compute all the possible states the attacker can find themselves in after applying exactly one action. Then, we recursively explore the newly created states in a depth-first-search manner, stopping when the state conforms to the parametrized goal of the attacker (an **accepting state**) or exceeds the allowed number of steps.

Due to the scenario generation being a state-exploration algorithm, we have to limit the state space further. For example, that is why the *Quiver* framework merges MITRE ATT&CK TTPs where possible. Another countermeasure we apply is the usage of *wildcard credentials* in the **ACS**, which can replace multiple existing credential combinations and is then later specialized at the point of usage. This reduces the number of states required to model the adversary's possible states between acquiring and utilizing such credentials.

4.4 Finalization

The automaton generated, as described in the previous section, is still too extensive and contains too many attack sequences to be considered realistic. Thus, we randomly choose some accepting state(s) and some of the paths leading to them. Then, we tweak the terrain to make these paths executable (we deploy the software and settings from the transitions using heuristics when multiple options are available), then freeze it. Afterward, we do a full graph search and cut the transitions that only require artifacts not present in the terrain.

4.5 Prototype Implementation

We have created a prototype implementation of the designed algorithm, called **PAGAN** [28, 29], that is mostly true to the design. The most notable deviations are:

- To ease the computation, PAGAN randomly chooses one device sequence before scenario generation and considers only that one, not all the possibilities.
- When faced with multiple options for account takeover (either when choosing a credential to apply or the account that should be running the exploited software), PAGAN eagerly prioritizes the option with the highest immediate rewards. For example, suppose it has to choose from two user-level accounts, where one has slightly more power on the targeted device. In that case, it will be prioritized over the other, even if it is more potent on a different device that is compromised in the future.
- PAGAN can only work with goal specifications on the final device if they are defined by tokens (with capability-defined specifications, this is possible). Also, we can only create specifications that can be described solely with tokens or capabilities, not simultaneously.
- PAGAN only uses privilege escalation techniques to get higher-level capabilities and not to change accounts on the same level.

5 EVALUATION

PAGAN is designed to guarantee the logical soundness of the generated scenarios and their executability in the terrains. We conduct two experiments to evaluate PAGAN in this paper: one based on real-world analyst reports and the other using a third-party attack graph generation tool.

5.1 Evaluation against Analyst's Report

We use analyst's incident report to re-create attack scenarios on a real-world network infrastructure. We expect PAGAN to generate scenarios that include what was reported.

5.1.1 Methodology. First, we manually transform the incident report content into a *ground-truth attack path*, a *terrain skeleton*, a set of *software* and *enablers* to be considered, and rules describing the applicable relations between the triplets of *software*, *enablers* and *techniques*. The above allows us to run the scenario generation process (recall the third block in Figure 3) to obtain the plausible scenario. Then, we manually compare and verify whether the reported attack path exists in the scenario.

We consider a specific historical incident report on the Nitro APT attack. Our choice fell on this particular case as it is sufficiently – but not overly – complex (the manual transformation takes ∼8 hours of mostly mundane work). The terrain for this case consists of two devices only: one (*host A*) reachable from the internet via email, the other (*host B*) intranet-only. There are four user accounts, one with administrator privileges on *host A*, another with administrator privileges on both, and one user account for each device. We limit the available software to those present in the report to minimize the size of the scenario automatons. The attacker's goal is to provide the action **Data Exfiltration** on *host B*. For specialization, we have multiple options: we can require a DATA token to enforce **Data Collection/Processing** on the path; PERSISTENCE capability to enforce **Planting a Backdoor**; or specify the filesystem-related capabilities to be at least of administrator level. We go with the first option. Figure 5a shows the manually derived ground truth attack scenario based on the report.

5.1.2 Results. For this experiment, we tested the scenario with and without the finalization process described in Section 4.4. The scenario without the finalization contains 121 nodes and 761 edges, with 24 accepting states and at least 96 hypothetical attack paths[2]. The finalization process reduced the scenario to 77 nodes and 457 edges with a minimum of 48 successful attack paths (see the electronic attachments for the full result). We found that the ground-truth attack scenario **is present** among the hypothetical attack paths, thus confirming the ability of PAGAN to recreate complex attack scenarios. The exact attack path is shown in Figure 5b.

One may wonder how to verify the validity of the remaining alternative attack paths. The following section presents a thorough experiment to compare PAGAN against a 3rd-party tool that generates attack paths.

5.2 Evaluation against 3rd-Party Tools

Despite its importance for evaluation, the previous method has two shortcomings. First, because of the manual work included, it does not scale well with the number of reports or the complexity of the infrastructures in them. Second, it does not say anything about the alternative paths in the scenario.

Therefore, another evaluation approach is required to eliminate the need for manual work and consider all the attack paths encoded in one scenario.

[2]Due to the mechanism of early return from already explored states, counting the exact amount is complicated.

5.2.1 Methodology. For the second phase of evaluation, we generate a *terrain* and *scenario* with **PAGAN**, feed the *terrain* into the third-party tool – **KCAG**, a state-of-the-art attack graph generator – and compare its output with the PAGAN-*scenario* (see Figure 4). As the outputs of both tools can be considered discrete graphs, graph matching algorithms [18] – that consider both structure and labels – can be used for the comparison.

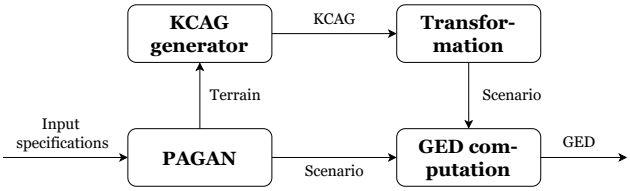


Figure 4: Workflow of graph edit distance computation between PAGAN and KCAG.

Due to the differing output and foundations of KCAG compared to PAGAN, the outputs must undergo lossless transformations before entering comparison. Following the transformations, the two outputs have equal structure but not equal labels. Consequently, we cannot use the traditional strict equality over these labels. Instead, we verify that the vertices and edges do not contradict each other in the two graphs. This means that the global **ACS** encoded in the scenario automaton's vertices of PAGAN is a superset of the more sparse "**ACS**" of the KCAG. Regarding edges, we verify that the adversary actions from the different ontologies used map to each other, including their enablers.

Having output graphs with a very similar structure and a well-defined comparison operation for both vertex and edge labels, we can apply graph edit distance (GED) [2] to express how much the outputs differ. For this purpose, we utilize six edit operations listed in Table 3. Any other change in the graph must be defined as a sequence of these. The GED computation's resulting value is the global minimum of all the possible edit operation sequences that transform one graph into the other, with the operation costs presented in Table 3.

Table 3: Operation costs for graph edit distance, with substitutions differentiated with sublabel granularity.

Operation	Label Change			Cost
	Device	Action	Software	
Vertex Insertion & Deletion	–	–	–	1
Edge Insertion & Deletion	–	–	–	2
Vertex Substitution	✓	–	–	1.5
Edge Substitution	–	✓	✗ / ✓	3
	–	✗	✓	1.5

Legend – ✓: changed, ✗: not changed, –: not considered by operation.

The costs are defined according to the relationships between the operations. In general, operations over edges should be more expensive than operations with vertices because attack techniques are information that matters the most. At the same time, aspects of application vary slightly in different threat models. Moreover,

substituting vertices and edges should be cheaper than combining deletion and insertion (substitutions affect the labels of vertices and edges only, while deletion plus insertion also changes structural likeness). To illustrate, consider a path of three vertices where the middle one is mislabeled. In this case, applying a simple vertex substitution and paying 1.5 is the straightforward solution. Now, if we were to do that with deletion and insertion only, the recipe is: remove the edge on the right, remove the edge on the left, remove the vertex and add a new vertex, add the edge to the right, add the edge to the left; for a total cost of 10.

KCAG attack graphs are typically shorter because they enumerate only attack steps necessary to achieve attack goals, while the *scenario* has paths representing other possibilities. Therefore, we use the evaluation algorithm in Listing 2.

Listing 2: Evaluation algorithm.

```
fn eval(
  KCAG: DirectedGraph,
  PAGAN: DirectedGraph,
) -> (f64, f64) {
  kcag_paths = edge_disjoint_paths(KCAG);
  geds = Vec::new();
  for accepting_state in PAGAN {
    component = subgraph(
      PAGAN,
      rule = "accepting_state_reachable_from_node",
    );
    component_geds = Vec::new();
    for path_kcag in kcag_paths {
      ged_min = GED(
        path_kcag,
        component,
        rule = "all_labels_match",
      );
      ged_max = GED(
        path_kcag,
        component,
        rule = "no_labels_match",
      );
      ged_kcag = GED(path_kcag, component);
      ged_norm = (ged_kcag - ged_min) / (ged_max - ged_min);
      component_geds.push(ged_norm);
    }
    geds.push(min(component_geds));
  }
  return (min(geds), max(geds));
}
```

The *PAGAN component* is a subgraph of nodes from which the particular accepting state is reachable, together with the edges between them. The `ged_min` represents the edit costs for the hypothetical best-case scenario, that is, when we need only to do structural changes (because we consider a path from KCAG but a whole component from PAGAN, there are many of these), and no label substitution at all. On the contrary, `ged_max` is for the hypothetical worst-case scenario, where every label must be altered on top of all the structural changes. Lastly, `ged_kcag` is the actual GED between the two outputs. These three values are then used to compute a normalized value for each *PAGAN component* and KCAG-path combination. The minimum of these normalized values is taken for each *PAGAN component* and then compiled into a range.

Since even unrelated pairs of *PAGAN components* and KCAG paths can be created, the primary indicator of success is the lower value within the range. Conversely, the higher value within the range indicates the meaningfulness of the most diverging path of the PAGAN scenario with respect to the KCAG.

To illustrate the comparison with an example, we again turn to the Nitro APT scenario. When fed with the terrain, **KCAG** generates the attack path shown in Figure 5d. The computation then follows.

First, asset properties and levels of privileges are merged into a single vertex for each attack step. For example, vertices 9, 10, and 11 are considered one vertex, with their merged data as a label. Nodes representing countermeasures are ignored. Therefore, we obtain a simple attack path where each second vertex is an attack technique. The second step converts the attack techniques from vertices to edges to resemble an attack path from a *scenario*. The final step is to apply the computation from Listing 2, which determines the combination of operations that provides the minimum distance. For all accepting states from the scenario, at least one KCAG path with normalized GED from 12% to 31% exists. The component containing the path that most faithfully represents the manual attack sequence in Figure 5a has 12% GED compared to one of the KCAG paths. Figure 5c showcases the most diverging path from the component.

5.2.2 Setup. Because **KCAG** works with a different set of adversary techniques, we had to limit **PAGAN** to employ only the intersecting techniques from *Quiver*. The intersection (with the help of the MITRE ATT&CK framework, as it is mappable to both tools) is: *Active Scanning (T1595)*; *Exploit Application (T1190)*; *Phishing (T1566)*; *Valid Accounts (T1078)*; *Remote Services (T1021)*; *External Remote Service (T1133)*; *Create or Modify System Process (T1543)*; *Abuse Elevation Control Mechanism (T1548)*; *Exploitation for Privilege Escalation (T1068)*; *Brute Force (T1110)*; *Exploitation for Credential Access (T1212)*; *Internal Spearphishing (T1534)*; *Data Manipulation (T1565)*; *Network Denial of Service (T1498)*; *Service Stop (T1489)*.

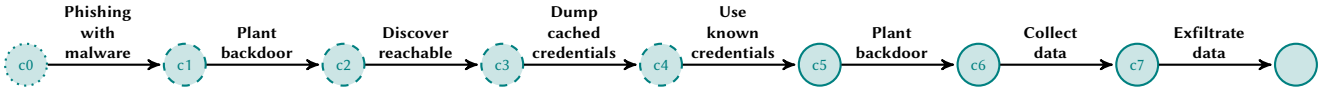
These cover enough attack life-cycle phases to describe complex scenarios. Also, we work with an artificial set of software artifacts and enablers.

For the evaluation, we have generated ten scenarios. Table 4 provides a brief description of their attributes. Scenarios 1-5 are generated with one set of parameters and 6-10 with another. The first scenario forcefully uses the shortest possible sequence of devices, the others are choosing at random. At first sight, the device and account numbers don't change much, as these numbers are heavily based on the parameters. More differences can be seen in the network connections and firewall rules, but there is no way to represent those quantitatively.

Table 4: Attributes of the generated scenarios and terrains.

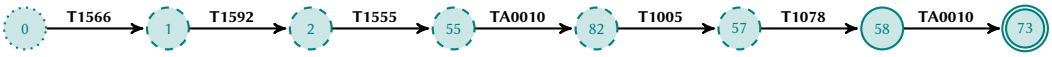
No.	Nodes	Edges	Devices	Accounts	Min. Paths
1	102	630	13	47	N/A
2	1241	9558	13	48	N/A
3	1016	10373	13	48	396
4	7884	70401	13	48	2784
5	15890	157890	13	48	6378
6	103533	758588	11	49	59204
7	40736	346494	11	49	32627
8	16591	108521	11	49	14600
9	28462	319140	11	49	21527
10	36661	352718	11	49	16196

Note: Scenarios 1 & 2 did not store the list of paths.

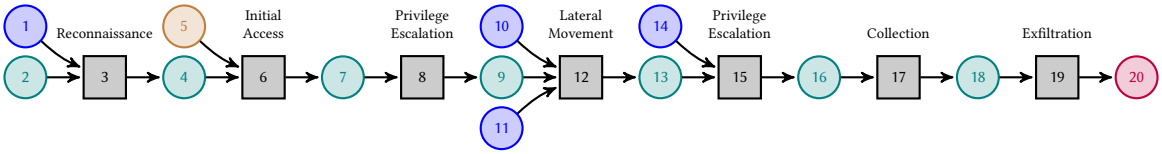


(a) Manually created attack path under *Quiver*. The states contain the newly acquired Tokens (see the table).

(b) Accepting path from the scenario equal to the manual analysis, with the IDs of the states, and with actions mapped to ATT&CK.



(c) Path from the PAGAN-component with the worst GED_{NORM}.



(d) Incomplete attack path excerpt from KCAG containing IDs of vertices and ATT&CK tactics. Green vertices represent levels of privileges, blue vertices asset properties, brown vertex a countermeasure, black squares attack techniques, and purple color an attack goal.

ID	Description	ID	Description	ID	Description	ID	Description
c0	External phishing mail	––	host B	2	Mail address on website	12	T1021.004 – Remote Service (SSH)
c1	Session, All capabilities	T1566	Phishing with malware	3	T1594 – Search websites	13	Can login to account
c2	Persistence Capability	T1547	Plant backdoor for persistence	4	Attacker can send emails	14	Windows OS installed
c3	Access via network	T1592	Discover devices and services	5	No sender reputation analysis	15	T1547 – Boot / Logon Autostart
c4	Credential	T1555	Credential Access from LSASS	6	T1566.001 – Spearphishing	16	Breached authorization (hostB)
c5	Session, All capabilities	T1078	Valid Accounts	7	Send a phishing email	17	T1005 – Data from local system
c6	Persistence Capability	T1547	Plant backdoor for persistence	8	T1547 – Boot / Logon Autostart	18	File-copy authorization breach
c7	Data	T1005	Data collection	9	Breached authorization (hostA)	19	TA0010 – Exfiltration
...	Untrusted Internet	TA0010	Data exfiltration	10	Network service on port 22	20	Violated confidentiality of files
---	host A	1	External actor from internet	11	Devices in the same network		

Figure 5: Artifacts for the Nitro APT attack used for the evaluation and its example-based explanation. The table contains elaborations where appropriate.

5.2.3 Results. The results for the introduced scenarios are listed in Table 5. Since graph edit distance computation is an NP-hard problem [33], we were limited by its time complexity. Therefore, we focus on comparing smaller parts of the scenario (components) with edge-disjoint paths from KCAG only and not with all existing paths in KCAG (see Listing 2). Due to this adjustment, only scenarios 1 – 3 are covered entirely by our evaluation, while the other scenarios are only partially due to a higher count of accepting states (see Table 5).

GED_{norm} values for the first three scenarios – where exhaustive comparison could be accomplished – are less than 20%. The remaining scenarios indicate a higher GED_{norm} range. Averages of GED_{norm} are close to or below 25% for these scenarios, except for scenarios 6 and 10. For these two scenarios, the suboptimal values are caused by limitations of the selection algorithm described above, which sometimes discards optimal paths. By manual inspection, we can identify a *PAGAN component* and KCAG-path pair for scenario no. 6 with a GED_{norm} of ~25%. Therefore, we postulate that scenarios 6 and 10 contain paths with GED_{norm} similar to the other scenarios.

While GED_{norm} around 20% may indicate significant differences between PAGAN and KCAG, it shows near-total correspondence between the two. This is because KCAG explicitly utilizes reconnaissance at the beginning of each attack path, while PAGAN implicitly begins with the knowledge gained by this activity. This difference equals an edge and a vertex insertion for each path, adding ~15% to

Table 5: Normalized graph edit distances, coverage of accepting states, and lengths of the shortest paths (PAGAN/KCAG).

No.	GED_{norm}	Average GED_{norm}	Accepting State Coverage	Shortest Path Length
1	15 – 20%	16.2%	12 / 12	6 / 5 vertices
2	15 – 20%	15.2%	48 / 48	6 / 5 vertices
3	15 – 20%	17.6%	36 / 36	6 / 5 vertices
4	16 – 40%	28.1%	102 / 540	7 / 5 vertices
5	17 – 36%	23.6%	102 / 813	7 / 5 vertices
6	41 – 60%	48.4%	158 / 23842	8 / 5 vertices
7	20 – 30%	24.3%	111 / 8944	9 / 5 vertices
8	15 – 30%	22.3%	401 / 3865	6 / 5 vertices
9	17 – 35%	23.6%	103 / 5921	7 / 5 vertices
10	40 – 60%	46.1%	287 / 5589	8 / 5 vertices

Note: Due to the time complexity of *GED*, evaluation results are computed only for a fraction of the accepting states.

each GED_{norm} for a given shortest path length. The remaining percentages result from PAGAN allowing more technique permutation on the paths and multiple impact tactics, thus adding edge/vertex renaming to *GED* computation while achieving the same output.

The evaluation results thus confirm that the *scenario* generated by PAGAN can materialize in the respective cyber *terrain*. Scripts used for evaluation, detailed results, and other evaluation artifacts are listed in the supplementary materials [27].

5.3 Constraints and Room for Improvements

The design proposed in this paper prioritizes completeness and detailedness over efficiency. This manifests in the complexity of the models presented in Section 3, the looseness of the limitations by the terrain, and the high abstraction of input parameters. While beneficial for scenario diversity and correctness, these choices adversely affect efficiency and cost.

5.3.1 Time Complexity. To express quantitatively, the generation process for scenarios 7 and 8 takes around 101, respectively 146 minutes for PAGAN. Some of this time complexity stems from the tool's implementation, which uses a rather primitive and sequential state exploration. Thus, in the near future, we plan to assess and experiment with parallel algorithms [9] to see how much of a speed-up is achievable.

5.3.2 Memory Complexity. The more concerning limitation is that of computational memory [11]. We have observed a few occasions where we had to stop the computation early due to it already using up ≈860 GB of RAM memory (of our 1 TB limit). We are currently working on identifying whether this problem stems from an implementation bug or purely the nature of the algorithm. We also plan to experiment with reducing the memory footprint of the state exploration.

5.4 Discussion

The evaluation demonstrates that we can create complex and non-linear *scenarios*, including non-successful paths. It also shows that the generation process can correctly produce the *terrain* where the *scenario* can be realized. We want to argue that these achievements are a product of the models we introduced, which are suited for the parallel generation of *terrains* and *scenarios*. The other approaches, which focus on either *scenario* or *terrain* generation, cannot produce such results because of the inevitable state-space explosion. We would also like to argue that while other approaches need not use **ACS**, **Quiver**, or be based on MITRE frameworks, we expect the interplay of *scenario* and *terrain* generation to be a necessary mode of operation.

6 CONCLUSION

We have established a mechanism for the automated concurrent generation of non-trivial attack *scenarios* together with instantiable cyber *terrain* descriptions necessary for the reliable execution of these scenarios. This generation requires only a partial specification of key elements required from the user while filling in the blanks in a logically sound manner.

The contributions of this work bring novelty from multiple perspectives. First, by introducing an innovative infrastructure model enabling the generation of cyber terrains, second, by devising a novel action framework to ensure the verifiable logical coherence of the generation process. The presented approach diverges from prevailing methods by concurrently creating terrains and scenarios, thus overcoming pitfalls stemming from generating the *scenario* and the *terrain* sequentially. Moreover, by introducing PAGAN – an implementation of the framework – and conducting an extensive evaluation of its outputs[3].

Two sets of experiments were conducted to evaluate PAGAN and our models. The outputs of PAGAN were compared to the *terrain* and *scenario* described in a writeup of the Nitro APT campaign. The similarity of these two confirmed the realism and relevance of the *scenarios* and *terrains* generated by PAGAN. Subsequently, a kill-chain attack graph generation process was employed to automatically derive attack graphs for the generated *terrains*, which were then compared with the corresponding *scenarios*. This assessment demonstrated the correctness of the generated *scenario-terrain* pairs.

We have identified vital distinctions that emphasize the suitability and necessity of our methodology for applications such as cybersecurity exercise generation or training of autonomous cybersecurity agents. Our approach accommodates non-linear paths of action, recognizes the value of information beyond immediate success, and acknowledges the variability in attacker behavior, including the preference for suboptimal paths in specific contexts.

We discovered some technical limitations in the implementation of the proposed models related to computational complexity. Therefore, we will focus on optimizations as the next avenue of research.

In conclusion, this research represents a significant advancement in the field, providing a robust framework for the automated generation of adversary activities and instantiable infrastructure descriptions. Our methodology addresses critical limitations of existing approaches and offers a valuable tool for a range of applications in cybersecurity training, autonomous system development, and threat analysis.

ACKNOWLEDGMENTS

This research was supported by the Strategic Support for the Development of Security Research in the Czech Republic 2019–2025 (IMPAKT 1) program granted by the Ministry of the Interior of the Czech Republic under No. VJ02010020 – AI-Dojo: Multi-agent Testbed for Research and Testing of AI-driven Cybersecurity Technologies.

[3]The prototype implementation can be found at [28], the work-in-progress implementation for production is at [29]. Evaluation artifacts are published in [27].

REFERENCES

[1] AO Kaspersky Lab. 2023. Mapping EDR to ATT&CKs. https://www.kaspersky.com/enterprise-security/mitre/edr-mapping Accessed on 23 Oct 2023.

[2] David B. Blumenthal and Johann Gamper. 2020. On the exact computation of the graph edit distance. *Pattern Recognition Letters* 134 (2020), 46–57. https://doi.org/10.1016/j.patrec.2018.05.002 Applications of Graph-based Techniques to Pattern Recognition.

[3] Ghanshyam S. Bopche and Babu M. Mehtre. 2014. Attack Graph Generation, Visualization and Analysis: Issues and Challenges. In *Security in Computing and Communications*, Jaime Lloret Mauri, Sabu M. Thampi, Danda B. Rawat, and Di Jin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 379–390. https://doi.org/10.1007/978-3-662-44966-0_37

[4] Yu-Chin Cheng, Chien-Hung Chen, Chung-Chih Chiang, Jun-Wei Wang, and Chi-Sung Laih. 2007. Generating Attack Scenarios with Causal Relationship. In *2007 IEEE International Conference on Granular Computing (GRC 2007)*. IEEE, New York, NY, USA, 368–368. https://doi.org/10.1109/GrC.2007.117

[5] The MITRE Corporation. 2022. Digital Artifact Ontology. The MITRE Corporation. https://d3fend.mitre.org/dao/ Accessed on 7 Nov 2023.

[6] The MITRE Corporation. 2023. MITRE ATT&CK®. https://attack.mitre.org

[7] The MITRE Corporation. 2023. MITRE D3FEND™. https://d3fend.mitre.org

[8] Martin Drašar, Ádám Ruman, Pavel Čeleda, and Shanchieh Jay Yang. 2024. The Road Towards Autonomous Cybersecurity Agents: Remedies for Simulation Environments. In *Computer Security. ESORICS 2023 International Workshops*, Katsikas et al. (Ed.). Springer Nature Switzerland, Cham, 738–749. https://doi.org/10.1007/978-3-031-54129-2_43

[9] Jonathan Ezekiel and Gerald Lüttgen. 2008. Measuring and Evaluating Parallel State-Space Exploration Algorithms. *Electronic Notes in Theoretical Computer Science* 198, 1 (2008), 47–61. https://doi.org/10.1016/j.entcs.2007.10.020 Proceedings of the 6th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC 2007).

[10] Firemon, LLC. 2023. Security Manager. https://www.firemon.com/products/security-manager/ Accessed on 23 Feb 2024.

[11] Patrice Godefroid, Gerard J. Holzmann, and Didier Pirottin. 1995. State-space caching revisited. *Formal Methods in System Design* 7, 3 (01 Nov 1995), 227–241. https://doi.org/10.1007/BF01384077

[12] Wajih Ul Hassan, Adam Bates, and Daniel Marino. 2020. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 1172–1189. https://doi.org/10.1109/SP40000.2020.00096

[13] Eric Hutchins, Michael Cloppert, and Rohan Amin. 2011. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. *Leading Issues in Information Warfare & Security Research* 1 (01 2011). https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf

[14] Sushil Jajodia, Steven Noel, Pramod Kalapa, Massimiliano Albanese, and John Williams. 2011. Cauldron mission-centric cyber situational awareness with defense in depth. In *2011 - MILCOM 2011 Military Communications Conference*. IEEE, Baltimore, MD, USA, 1339–1344. https://doi.org/10.1109/MILCOM.2011.6127490

[15] Pontus Johnson, Robert Lagerström, and Mathias Ekstedt. 2018. A Meta Language for Threat Modeling and Attack Simulations. In *Proceedings of the 13th International Conference on Availability, Reliability and Security* (Hamburg, Germany) *(ARES '18)*. Association for Computing Machinery, New York, NY, USA, Article 38, 8 pages. https://doi.org/10.1145/3230833.3232799

[16] Kerem Kaynar. 2016. A taxonomy for attack graph generation and usage in network security. *Journal of Information Security and Applications* 29 (2016), 27–56. https://doi.org/10.1016/j.jisa.2016.02.001

[17] Jong-Keun Lee, Min-Woo Lee, Jang-Se Lee, Sung-Do Chi, and Syng-Yup Ohn. 2005. Automated Cyber-attack Scenario Generation Using the Symbolic Simulation. In *Artificial Intelligence and Simulation*, Tag Gon Kim (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 380–389. https://doi.org/10.1007/978-3-540-30583-5_41

[18] Lorenzo Livi and Antonello Rizzi. 2013. The graph matching problem. *Pattern Analysis and Applications* 16 (08 2013). https://doi.org/10.1007/s10044-012-0284-8

[19] Microsoft. 2023. Technical Documentation. https://learn.microsoft.com/en-us/docs/

[20] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V.N. Venkatakrishnan. 2019. POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1795–1812. https://doi.org/10.1145/3319535.3363217

[21] Stephen Moskal and Shanchieh Jay Yang. 2020. Cyberattack Action-Intent-Framework for Mapping Intrusion Observables. https://doi.org/10.48550/arXiv.2002.07838 arXiv:2002.07838 [cs.CR]

[22] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. 2005. MulVAL: A Logic-based Network Security Analyzer. In *USENIX security symposium*, Vol. 8. Baltimore, MD, USENIX Association, Baltimore, MD, 113–128. https://www.usenix.org/legacy/event/sec05/tech/full_papers/ou/ou_html/

[23] Gavin O'Gorman and Eric Chien. 2011. The Nitro Attacks Stealing Secrets from the Chemical Industry. https://paper.seebug.org/papers/APT/APT_CyberCriminal_Campagin/2011/the_nitro_attacks.pdf

[24] Paul Pols. 2017. THE UNIFIED KILL CHAIN. https://www.unifiedkillchain.com Accessed on 1 Nov 2023.

[25] David Raymond, Tom Cross, Gregory Conti, and Michael Nowatkowski. 2014. Key terrain in cyberspace: Seeking the high ground. In *2014 6th International Conference On Cyber Conflict (CyCon 2014)*. IEEE, Tallin, Estonia, 287–300. https://doi.org/10.1109/CYCON.2014.6916409

[26] George F. Riley and Thomas R. Henderson. 2010. *The ns-3 Network Simulator*. Springer Berlin Heidelberg, Berlin, Heidelberg, 15–34. https://doi.org/10.1007/978-3-642-12331-3_2

[27] Ádám Ruman, Martin Drašar, Lukáš Sadlek, Shanchieh Jay Yang, and Pavel Čeleda. 2024. Supplementary Materials: Adversary Tactic Driven Scenario and Terrain Generation with Partial Infrastructure Specification. Zenodo. https://doi.org/10.5281/zenodo.11183639

[28] Ádám Ruman et al. 2024. PAGAN prototype implementation. https://gitlab.ics.muni.cz/cyst-public/pagan

[29] Ádám Ruman et al. 2024. PAGAN-RS. https://gitlab.ics.muni.cz/ai-dojo/pagan-rs

[30] Lukáš Sadlek, Pavel Čeleda, and Daniel Tovarňák. 2022. Identification of Attack Paths Using Kill Chain and Attack Graphs. In *NOMS 2022 - 2022 IEEE/IFIP Network Operations and Management Symposium* (Budapest, Hungary). IEEE Xplore Digital Library, Budapest, Hungary, 1–6. https://doi.org/10.1109/NOMS54207.2022.9789803

[31] Oleg Sheyner and Jeannette Wing. 2004. Tools for Generating and Analyzing Attack Graphs. In *Formal Methods for Components and Objects*. Springer Berlin Heidelberg, Berlin, Heidelberg, 344–371. https://doi.org/10.1007/978-3-540-30101-1_17

[32] Geir Skjotskift, Fredrik Borg, Martin Eian, and Siri Bromander. 2021. Adversary Emulation Planner. Mnemonic. https://github.com/mnemonic-no/aep Accessed on 4 Dec 2023.

[33] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. 2009. Comparing stars: on approximating graph edit distance. *Proceedings of the VLDB Endowment* 2, 1 (aug 2009), 25–36. https://doi.org/10.14778/1687627.1687631