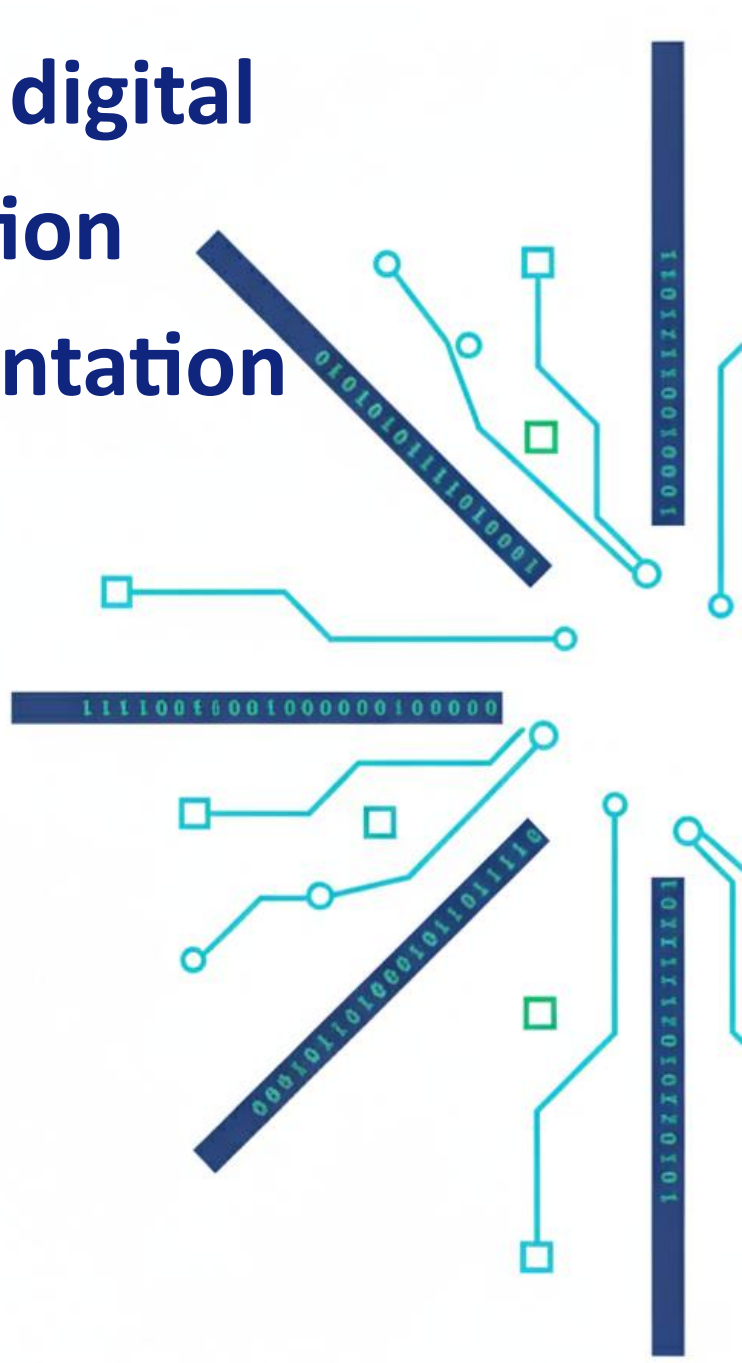
# D16 – Model for digital evidence extraction to graph representation



The project Automatization of digital forensics and incident response (ADFIR) funded by the European Union – Next GenerationEU through the Recovery and Resilience Plan of the Slovak Republic under project No. č. 09-I05-03-V02-00079.

# Outline

# 1 Project description

The project **Automatization of digital forensics and incident response** (hereinafter referred to as **"ADFIR"**) is funded by the **European Union – Next GenerationEU through the Recovery and Resilience Plan of the Slovak Republic** under project No. č. 09-I05-03-V02-00079. This project addresses one of the key challenges in cybersecurity and information security – how to process the massive volume of digital evidence generated during cybersecurity incidents or forensic investigations. Currently, this process is highly demanding in terms of human resources and time. Therefore, automation using machine learning methods can significantly **improve the quality of digital forensic analysis** and reduce the time required to perform it. Overall, this enables security teams to respond more effectively to cyber threats. Main benefits of this project are:

- **Accelerated Resolution of Cybersecurity Incidents**. The project ADFIR introduces automated approaches to collecting, processing, and analyzing digital traces. As a result, security teams can identify the causes of incidents more quickly and adopt effective measures to address them.
- **Reduced Workload for Forensic Analysts**. Routine and time-consuming tasks involved in processing digital traces will be replaced by automated methods. This will allow analysts to focus on more complex cases and strategic decision-making.
- **Higher Quality and Consistency of Outputs**. The use of unified methodologies and tools ensures that the processed digital traces will be more accurate, consistent, and easily verifiable. This significantly reduces the risk of errors caused by human factors.
- **Potential Use in Criminal Proceedings.** The project outputs will be developed in compliance with legal requirements and standards, allowing the digital traces to be accepted as relevant evidence for investigations and court proceedings.

# 2 Introduction

With the growing volume of digital data and the increasing complexity of cyber incidents, the need for effective methods to identify, extract, and interpret digital traces is also growing. Traditional approaches to forensic analysis often deal with many heterogeneous artifacts, such as logs, metadata, file structures, network records, or memory dumps, where their interrelationships may not be immediately apparent. Graph representation offers a suitable method for modeling these entities and their relationships, as it allows for capturing not only the objects of forensic analysis themselves, but also their context, temporal relationships, and interactions among them.

In the field of digital forensic analysis, the relational nature of data is one of the key prerequisites for successfully reconstructing events. Graph models allow digital traces to be represented as a set of nodes and edges, where nodes represent, for example, devices, users, processes, files, or communication entities, and edges express their mutual connections, interactions, or causal relationships. This form of representation supports not only the visualization of evidence but also the application of analytical methods based on graph theory, automated reasoning, and the identification of anomalies in the investigated environment.

A model for extracting digital traces into a graph representation can therefore be understood as a means for the systematic transformation of forensic artifacts from disparate data sources into a unified, structured form. A model created in this way can significantly contribute to a better understanding of the course of an incident, to a more effective search for relevant connections, and to a more convincing presentation of investigation results.

# 3 Sources of Digital Evidence

In digital forensic analysis, we often encounter two approaches: **offline** (post-mortem) and **online** analysis. Offline analysis takes place after the incident, when the system is shut down or isolated, and the expert primarily examines stored data, disk images, logs, and other permanent artifacts. Online forensic analysis, on the other hand, is performed on a running system during the incident to capture volatile data as well, such as RAM contents, active processes, or open network connections.

The main difference between them lies in **what can be captured** and **the risk posed to the integrity of digital evidence**. The post-mortem approach offers the advantage of a stable environment and a lower risk of data alteration, making it well-suited for detailed post-mortem incident investigations. Online analysis, on the other hand, is indispensable in situations where critical evidence may disappear after the device is powered off or the system is rebooted.

## 3.1 Databases of simulated attacks and attacker techniques

One approach to creating datasets involves **the targeted simulation of attacks and attacker techniques** in a controlled and isolated environment. In this case, attacks are carried out deliberately and systematically. Attacks are often performed based on known frameworks, such as MITRE ATT&CK, with the aim of generating forensic artifacts corresponding to specific techniques and phases of the attack.

The main advantage of this approach is **control over the scenario**. The researcher or developer knows exactly which techniques were used, in what order, and for what purpose. This enables precise data labeling, the creation of balanced datasets, and systematic testing of models' ability to detect specific types of behavior. Simulated datasets are also suitable for creating reference data for experimental comparisons and validation studies.

On the other hand, simulated attacks also have their limitations. Even with a high level of expertise, a simulation can be **simplified** and may not capture all the unpredictable aspects of real incidents, such as attacker errors, combined attacks by multiple actors, or long-term, low-intensity campaigns. However, simulated attacks represent an important compromise between realism and data controllability.

## 3.2 Databases from CTF Competitions

Another category consists of **datasets from CTF (Capture the Flag) competitions** (e.g., we used [7–10]**)**, which have been long used in cybersecurity education and training. CTF tasks are typically designed to simulate a specific security incident or part of one, and participants answer predefined questions by analyzing digital traces.

The use of CTF datasets has certain **inherent limitations**. CTF scenarios are often artificially constructed, focused on a specific type of attack, and are typically carried out by a single attacker. Such a model may not fully reflect reality, where multiple attackers may be involved, multi-phase attacks may occur, and the outcome of the investigation is not known in advance. Furthermore, CTF data implicitly assume the existence of a solution, whereas real-world security incidents are characterized by a high degree of uncertainty and ambiguity.

Despite these limitations, CTF competitions have **significant parallels with real-world security incidents**. Even in CTF scenarios, attackers use specialized tools, techniques, and tactics that are often identical or very similar to those used in practice. The process of examining data and digital traces, correlating artifacts, and reconstructing events is comparable to real-world digital forensic analysis.

Although the digital traces in CTF challenges are artificially created, **the** actual **process of data analysis and the extraction of forensic artifacts** is largely the same as in the investigation of real-world incidents. Furthermore, the structured nature of CTF tasks, often based on questions and answers, can foster a systematic analytical approach that is transferable to the real world and can lead to faster identification and resolution of problems.

## 3.3 Semantic Analysis

Semantic analysis is the process of processing input data into an easily interpretable timeline that can be further used to construct a graph representation. It consists primarily of two steps:

1. preprocessing and
2. labeling of input data.

As input for the first step, we use a standard Plaso super timeline [5], from which 6 key attributes are subsequently selected—*datetime*, *sourcetype*, *type*, *MACB*, *user*, *host*—which are used in their raw, "unprocessed" form—and the *desc* attribute, which carries the most important part of the event itself and is truncated to 200 characters. The preprocessed timeline is then divided into windows of 400 rows and proceeds to the second step—data labeling.

Data labeling is performed using inference from the large language model Gemma 4[1] with 27 billion parameters, locally on the end device over the preprocessed timeline. The outputs themselves are then processed to ensure the correct data format (for both graphical display and further analysis). Each output row can potentially be flagged as suspicious; rows flagged in this way are stored and specifically tagged with the following attributes—*attack_type*, *mitre_tactics*, *mitre_techniques*, *killchain_stage*—and made available for export in both JSON and CSV formats.

[1] Gemma 4 model; a description of the model is available on the website: https://deepmind.google/models/gemma/gemma-4/

# 4 Graph as a Tool for Modeling Forensic Data

## 4.1 Graph theory

Graph theory is a branch of discrete mathematics that deals with the study of graphs, their properties, and the relationships between their elements. Graphs serve as a very useful tool for modeling various real-world situations, such as transportation networks, computer networks, social relationships, or various types of connections between objects. In the following text, we present the basic concepts and properties of this field (for more information, see [1]).

A **graph** is usually denoted as *G=(V, E)*, where *V* is the set of **vertices** and *E* is the set of **edges**. Vertices represent individual objects or points, and edges represent connections between them. If the edges are **undirected**, the connection between two vertices has no direction (the edge is bidirectional). In the case of a **directed** graph (digraph), each edge has a specified direction from one vertex to another. In addition, we may also encounter **weighted** graphs, in which edges or vertices are assigned a specific value, such as length, cost, or capacity. A special case is a **multigraph**, where **multiple edges** (between a single pair of vertices) and **loops** (an edge from one vertex to the same vertex) may occur. A generalization of a graph is a **hypergraph**, which allows edges between a set of vertices (not just pairs).

The **degree of a vertex** indicates how many edges are connected to that vertex. In the case of directed graphs, we have both the **in-degree** and **out-degree** of a vertex. Two vertices are **adjacent (**neighboring**)** if they are connected by an edge. The concept of **incidence** expresses the relationship between a vertex and an edge if the vertex is one of its endpoints. **Walk** is an alternating sequence of incident vertices and edges (beginning and ending at a vertex). **Trail** is a walk in which no edge is repeated. **A path** is a walk in which no vertex (and thus no edge) is repeated. **A cycle** in graph theory is a closed trail in which the first and last vertices coincide, and the other vertices do not repeat. **A subgraph** of a graph is a part of the original graph formed by selecting certain vertices and edges.

A graph is **connected** if there is at least one path between any two vertices. If this condition does not hold, the graph is **disconnected** and consists of multiple **connected components**. In directed graphs, **strong connectivity** is also examined, in which a directed path must exist between any two vertices in both directions. An **articulation point** (cut vertex) is a vertex whose removal increases the number of connected components of the graph. A **bridge** (cut edge) is an edge whose removal increases the number of connected components of the graph.

A **complete graph** is a graph in which every vertex is connected to every other vertex. A **regular graph** has all vertices of the same degree. A **bipartite graph** can be divided into two sets of vertices, such that every edge runs only between the sets, not within one of them. A **tree** is a

connected graph that contains no cycles. A graph in which every component is a tree is called a **forest**.

The **shortest path** between two vertices is the path with the minimum length. If the graph is weighted, the length of a path is the sum of the weights of its edges. The **distance between** two **vertices** is the length of the shortest path between them. The **eccentricity of a vertex** is the maximum distance of that vertex from all other vertices in the graph. The **diameter** of a graph is the maximum eccentricity of the vertices in the graph. The **radius** of a graph is the minimum eccentricity of the vertices in the graph. A **spanning tree** of a graph is an acyclic subgraph (not containing a cycle, i.e., a forest) that preserves connected components. A **minimum spanning tree** is a skeleton with the smallest possible sum of edge weights. **Flow** is the amount that flows through a directed network from the initial vertex/-ices (**source**) to the terminal vertex/-ces (**sink**). In flow problems, the capacity of the edges is also considered, i.e., the maximum possible amount that can pass through an edge. The **maximum flow** in a network is the flow that cannot be increased any further. A **topological ordering** in a directed acyclic graph is an ordering of vertices such that every edge goes from a vertex that comes earlier in the ordering to a vertex that comes later in the ordering.

Two graphs are **isomorphic** if they have the same structure, even though they may be drawn differently. This means that there exists a bijection between their vertices that preserves adjacency.

## 4.2 State Machine Diagrams

In UML, a **State Machine Diagram** is one of the fundamental behavioral tools for modeling system dynamics. Its main purpose is to capture **how a specific object transitions between a finite number of states during its lifecycle**, with these transitions triggered by events. It is a form of graphical notation that allows for the precise description of a system's responses to both external and internal stimuli, while also demonstrating that an object's behavior depends on its previous state. This approach is particularly suitable for event-driven systems, where the same event can have a different effect depending on the object's current state.

A state diagram is based on the concept of a finite-state machine (the visualization of the machine is based on a graph representation), which represents an object as an entity capable of being in one of a set of defined states. Each **state represents a specific configuration of conditions** that are true at a given moment—**these** may **include attribute values**, relationships with other objects, or an activity currently in progress. **A transition between states** is a response to an event, which may be external (e.g., user input) or internal (e.g., completion of

an activity). Transitions may be supplemented with **conditions and actions** that are executed upon a state change.

A state machine diagram is a suitable model for analyzing the behavior **of reactive objects**, i.e., those that respond to stimuli while the system is running. In object-oriented design, it is used to describe the behavior of classes, subsystems, or entire systems, allowing for a precise definition of how an object reacts to various events at different stages of its lifecycle. It thus supports software analysis, design, and testing by providing a clear visualization of system dynamics.

The basic elements of a state diagram are:

- **Initial state** – represented by a solid black dot, it describes the beginning of an object's lifecycle.
- **State** – a rounded rectangle representing the situation in which the object finds itself (determined by the values of the object's attributes); it may contain actions and activities.
- **Transition** – an oriented arrow between states, describing an event, optionally a condition, and an action.
- **Terminal state** – a double circle describing the end of an object's life cycle.
- **Branching and connection** – elements enabling the modeling of the start and end of parallel behavior branches.
- **Self-transition** – a transition that does not result in a change of state.

While each lifecycle has exactly one initial state, a lifecycle may have multiple terminal states corresponding to different versions of the system's development.

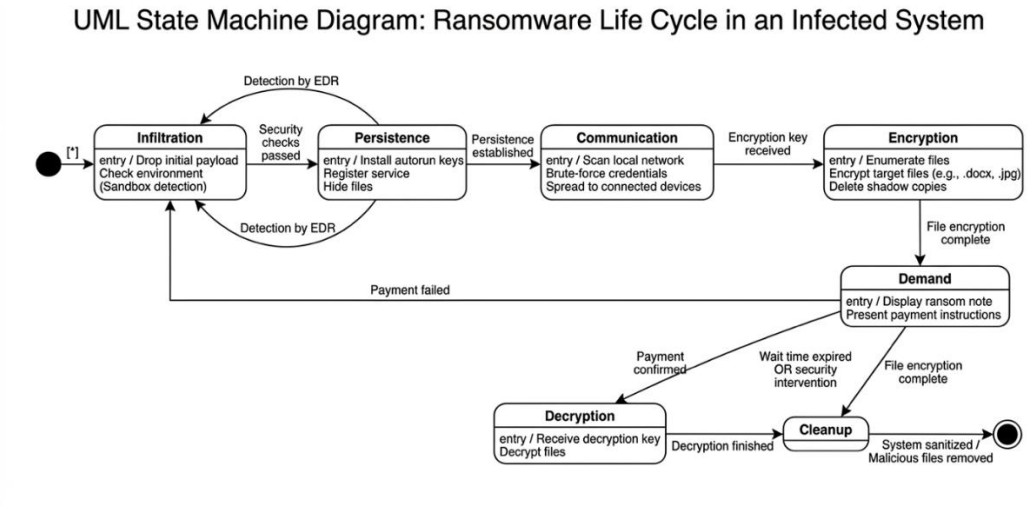



**Figure** 1 **- State diagram for a system infected with ransomware (generated by Gemini 3 Flash on April 30, 2026)**

A state machine diagram is used in situations where it is important to precisely define the system's responses to events and where behavior depends on previous states. From the perspective of digital forensic analysis, suitable objects that can be described using a state diagram include:

- user session,
- process,
- file,
- registry entry.

Based on individual lifecycles (instances), state diagrams can represent classes—general patterns of behavior (see, e.g., [2]). Thanks to the clear structure of states and transitions, it allows one to understand how the system reacts to various events and provides an important tool for system analysis. Since many cyberattacks involve typical behavioral patterns, this approach can aid in prediction or post-mortem analysis of cyberattacks. Its strength lies in its ability to capture the system's dynamics over time and represent them in a way that is understandable to both machine analysis and forensic analysts. An example of such a diagram is shown inFigure 1.

## 4.3 Petri nets

**Petri nets** are a formal tool for **modeling discrete parallel systems**. Conceptually, they are based on a graph structure, and therefore, in addition to algebraic representation, they also provide the ability to visualize processes, which is particularly useful in process simulations. Petri nets allow for the analysis of resource sharing and the synchronous and asynchronous behavior of systems and are a certain type of generalization of finite-state automata.

From a formal perspective, a Petri net is **a directed bipartite graph**, meaning it contains two types of nodes that alternate along the graph's paths:

- **places—**represented by circles, they represent, for example, physical locations, conditions, or states.
- **transitions—**represented by arcs—represent events or actions.

Directed edges in the network can only connect a place to a transition or a transition to a place; they can never connect two nodes of the same type. Edges represent the direction of the evolution of events in the system.

The dynamic aspect of the network is provided by **tokens**, which are placed at individual locations. The assignment of these tokens to individual locations can be represented as a vector and is called **marking**. A marking defines the current state of the system.

The transition rules are simple but allow for complex behavior. We call a node an input node of a transition if a directed edge leads from it to that transition. We define an exit node analogously. In the basic version, edges have a capacity of one; in extended versions, capacities are expressed as positive integers. This number represents the number of tokens required to activate the transition from a given node.

- A transition is **enabled** if each of its input sites contains a sufficient number of tokens.
- When a transition **fires,** tokens are removed from the input places, and new tokens are added to the output places in a quantity corresponding to the capacity of the transitions.

This formalized model allows for the analysis of several process characteristics using graph algorithms:

- **reachability:** The fundamental problem of analysis, which examines whether the system can ever reach a target state (e.g., a final or error state) from its initial state.
- **liveness:** Ensures that the system does not freeze. If the network is alive, in every reachable state it is possible to trigger any transition after a certain sequence of steps. This prevents **deadlock**.
- **boundedness:** Determines whether the number of tokens at any given location does not exceed a specified limit. If the limit is met, we refer to a **safe network**, which is ideal for modeling critical processes.

In practice, the basic model is often supplemented with additional dimensions to better reflect the complexity of real-world situations:

- **Colored Petri nets (CPN):** Tokens are not simple objects but carry a data value (color), which allows for the modeling of complex data flows and processes.
- **Timed Petri Nets (TPN):** A time component (transition delays) is introduced into the model, which is essential for real-time systems.
- **Stochastic Petri nets:** These are used to model performance and reliability, where the firing of a transition depends on a probability distribution based on real statistical observations.

Thanks to their precision, Petri nets are used in the design of distributed systems, communication protocols, manufacturing automation, and even in the analysis of biological

processes. Since a typical feature of cyberattacks is a time-dependent sequence of multiple events that also manipulate multiple types of objects, Petri nets are a good candidate for modeling and formally analyzing both documented and hypothetical attacks. For a deeper understanding of Petri nets, we recommend, for example, publication [3].

To illustrate, let us consider an example of a simple phishing attack model. This scenario is depicted using a Petri net in Figure 2. The individual elements of the Petri net are:

**Places (states):**

- **P1: Start/Threat** (The attacker is ready).
- **P2: Email in the Inbox** (Phishing email delivered to the victim).
- **P3: Local Access** (Malware is running on the HR employee's PC).
- **P4: Admin Rights** (The attacker has taken control of a privileged account).
- **P5: Server access** (Path to the database is open).
- **P6: Encryption complete** (Attack objective achieved).

**Transitions (actions):**

- **T1: Phishing email sent** (Action requires a source/decision in P1).
- **T2: Running a macro** (User interaction that advances the state to P3).
- **T3: Privilege escalation** (Exploitation of a system vulnerability).
- **T4: Lateral movement** (Penetration from the workstation to the server).
- **T5: Activation of ransomware** (Launch of the destructive phase).

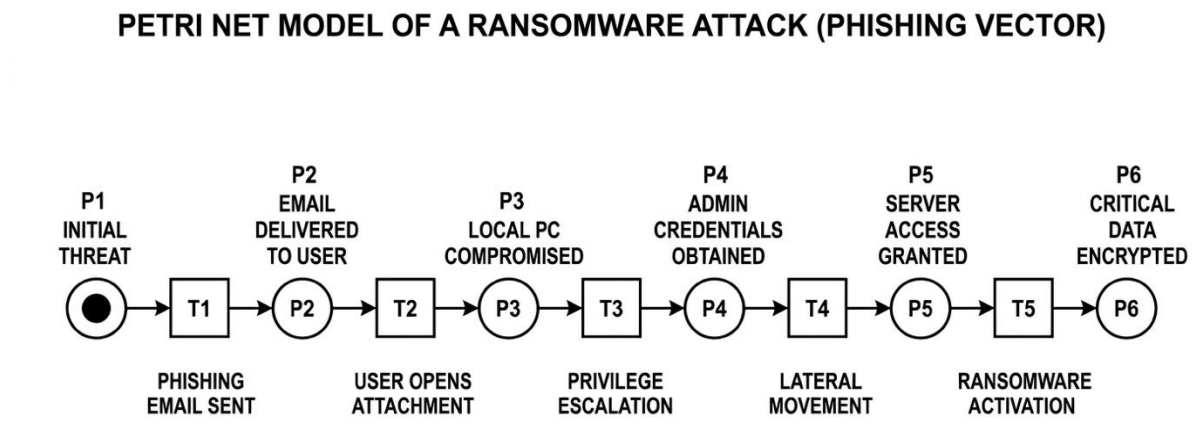



**Figure** 2 **- Petri net for a phishing attack**
**(generated by Gemini 3 Flash on April 28, 2026)**

For more complex attacks, it is of course possible to model parallel processes and more complex conditions affecting individual events.

## 4.4 Process Diagrams

**Business Process Diagrams** are used to model and analyze complex processes of various types. They serve as a visual bridge between the strategic view and implementation on a specific platform. The most widespread and globally accepted standard for these models is **BPMN** (Business Process Model and Notation), Conceptual process modeling is based on **Petri nets**, and in UML, its counterpart is activity modeling. However, BPMN offers far more expressive capabilities and is therefore typically used in combination with UML diagrams.

BPMN is a modeling framework that provides a unified syntax for business analysts, technical developers, and managers. Its main goal is to formally describe and visualize processes within a system. Unlike flowcharts or UML activity diagrams, process diagrams contain specific semantic rules that enable the transformation of a visual model into executable code (e.g., XML, BPEL).

The BPMN notation uses four main categories of graphical elements that define the process structure (for more information, see [3]):

- **Flow Objects:**
  - **Events:** represented by circles, they denote something that "happens" (e.g., receipt of an email, elapsed time); we distinguish between start, intermediate, and end events.
  - **Activities:** rectangles with rounded corners represent actions performed within the process (tasks or subprocesses).
  - **Gateways:** diamonds control branching and the merging of flows based on decision conditions (logical operators AND, OR, XOR, complex, ...).
- **Connecting objects:** define the direction of flow of sequences, messages (communication between participants), and associations to data.
- **Swimlanes:**
  - **Pool:** represents a process participant,
  - **Lane:** a separate area within a pool that assigns responsibility to specific roles or departments.
- **Artifacts:** used to supplement information, such as data objects or text annotations, without directly affecting the flow logic.

**Process diagrams** allow modeling of a single system or the interaction of multiple systems. Related concepts include **orchestration** and **choreography. Orchestration** represents the perspective of a single central entity (process) that controls and coordinates other activities, much like a conductor in an orchestra. This process controls the order of tasks and knows exactly when they are to be performed. **Choreography**, on the other hand, focuses on the interaction between two or more participants without the presence of central control. It models the exchange of messages and the rules of cooperation between partners, much like dancers who know their steps and react to the movements of others without being directly directed by anyone.

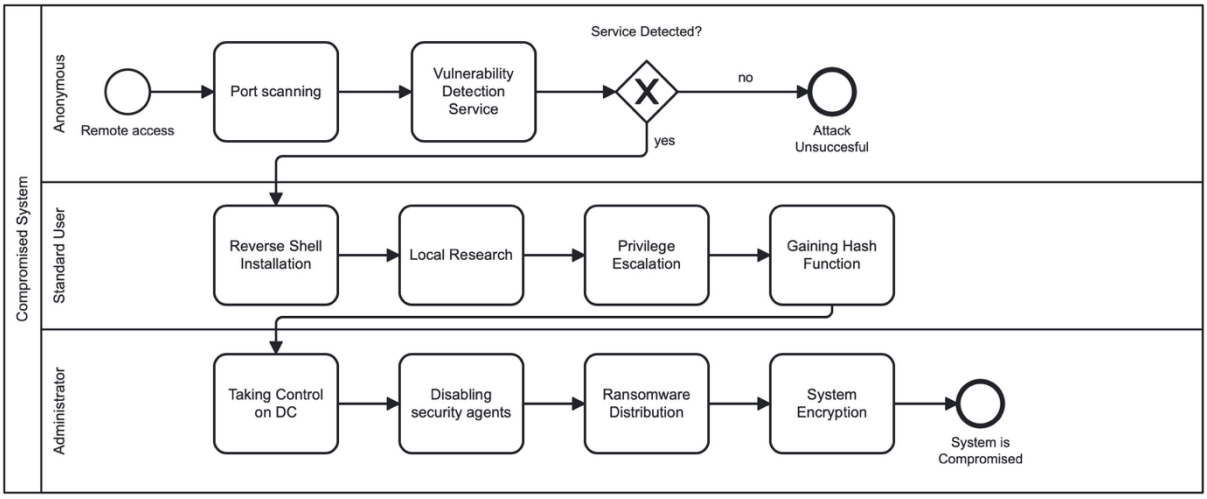



**Figure** 3 **- Process model of a ransomware attack**

Figure 3 shows an example of a process model for a ransomware attack, depicting the sequence of steps from the initial compromise of the system to the final phase of data encryption. The model illustrates how an attacker progressively moves through the various phases of the attack, with individual activities represented as a sequence of interlinked processes. The initial phase begins with scanning the device and available services; the sequence ends with a compromised system containing encrypted files. At the same time, the lines in the process model show how the attacker gradually changed their role—from an anonymous network user, through a standard user, to an administrator.

# 5 Applications of graph models within the ADFIR project

Depending on the nature of the available dataset, the type and structure of objects in the dataset, and the phenomenon under observation, we can choose different graph representations. A suitable representation allows us to visualize selected features of the analyzed dataset and establish the prerequisites for using appropriate algorithms.

In the following sections, we will demonstrate the use of a classic graph, a state diagram, and a process model to analyze the data sources available to us. The examples of classic graphs are based on [4].

## 5.1 Creating Graphs from Digital Evidence

In the case of **post-mortem analysis** of digital traces, the records of those artifacts that are stored on secondary storage (outside of primary operating memory) are usually available. We can analyze these artifacts using various analytical tools (in this material, we focus on the Plaso tools [5] and tools by Eric Zimmerman [6]). Using these tools, we can obtain extracted data (usually in CSV format), from which various graphs can be created (based on individual forensic artifacts).

From the digital traces listed in Windows Event Logs, we created a graph based on two attributes: *user_id* and *execution_process_id*. This results in a bipartite graph, where the vertices represent users (marked in blue in the figure) and processes (marked in red). An edge in the graph indicates that a record containing the given user and the corresponding process was found in the logs. Figure 4 shows a graph from the "NIST Data Leakage Case" dataset [10].

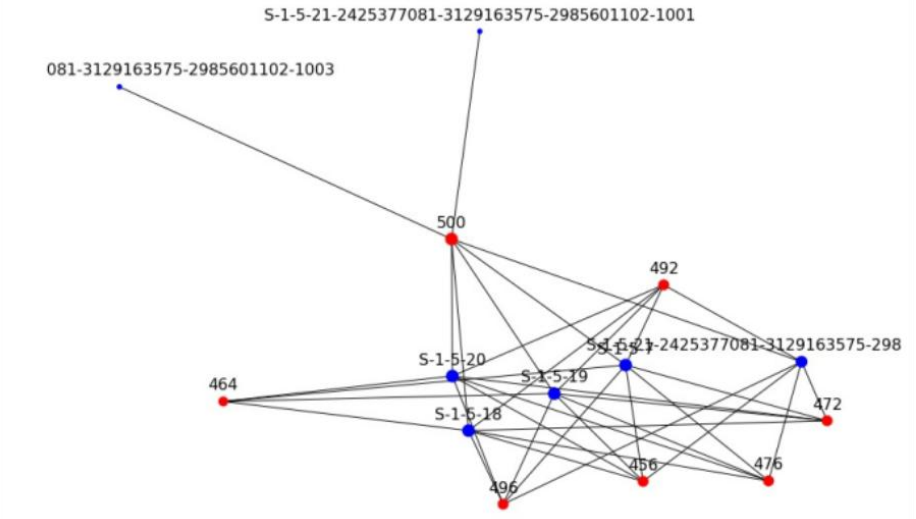



**Figure 4 - Users and processes in the NIST Data Leakage Case graph**

Figure 5 shows a graph created from the attributes *"user*" and "*source,*" also from the "NIST Data Leakage Case" [10]. Red vertices represent user names, while blue vertices represent sources. The source is indicated by the location of artifacts (e.g., registries, logs, ...). An edge represents the existence of a record for the respective user in the source data. For a forensic analyst, this can more clearly indicate which users focus on during forensic analysis.

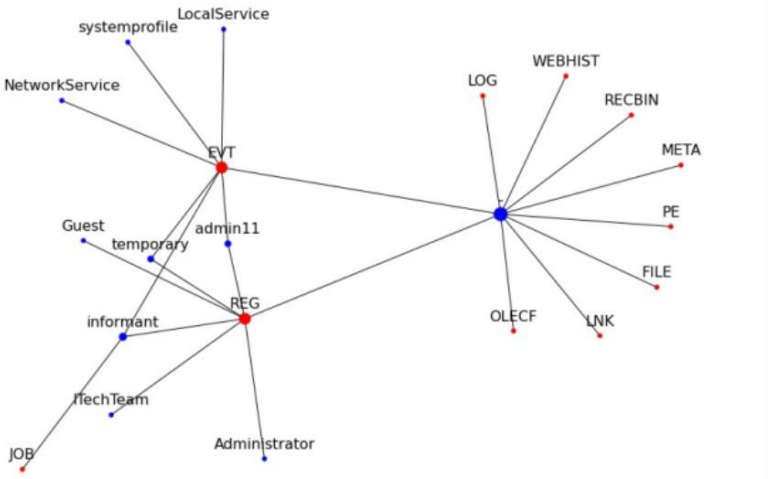



**Figure 5 - Users and sources in the NIST Data Leakage Case graph**

Using data from the MAGNET Capture The Flag competition [8, 9], it is possible to observe the eccentricity of vertices in a graph generated from the *user* and *source* attributes. The center of the graph is formed by a vertex representing logs (WinEVTX) and a vertex representing a single user (in Figure 6).

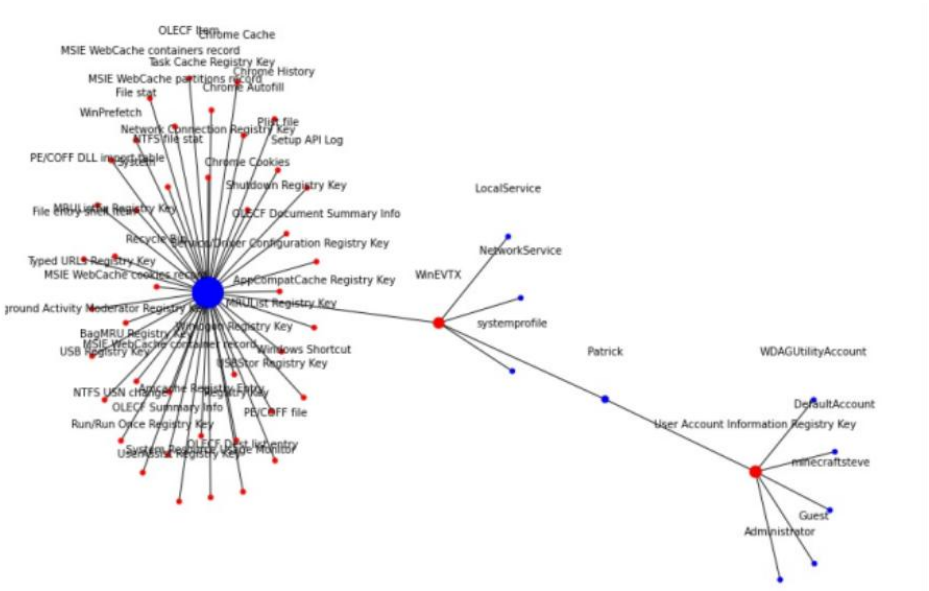



**Figure 6 —Users and Sources in the Magnet CTF 2022 Case Graph**

When creating a graph between users and files (whether by filename or inode number), it is possible to search the graph for vertices with a high degree—this could indicate, for example, ransomware encrypting files, but also a backup process. Figure 7 shows an example of such a graph from the Magnet CTF 2022 case [9]; the data is from the Windows Event Log. Users are marked in red, and file inodes are marked in blue.

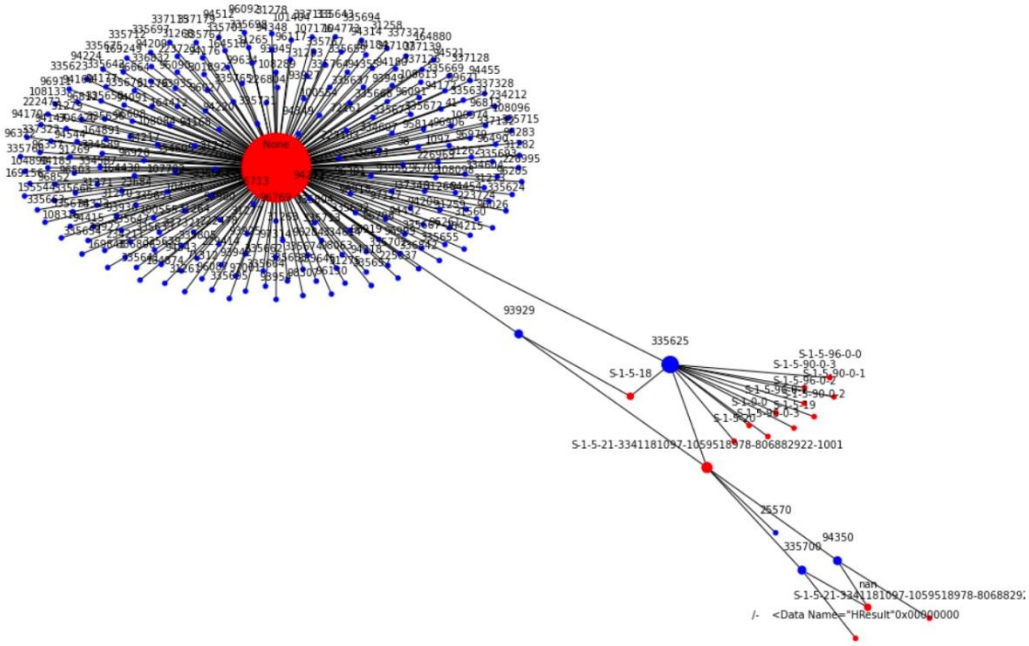



**Figure 7 - Users and file inodes in the Magnet CTF 2022 case graph**

From digital traces related to the file system, it is possible to track individual files and timestamps (*Modification*, *Access*, *Change*, and *Birth*). Figure 8 shows an example of such a graph from the "Stolen Szechuan Sauce" competition [7]. The red vertices represent files, and the blue vertices represent combinations of timestamps.

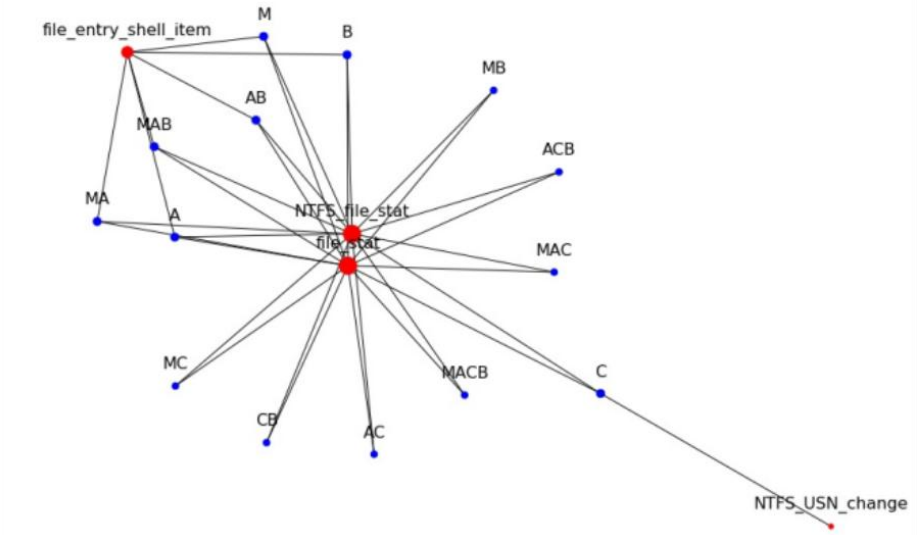



**Figure 8 - Timestamps and file types in the Stolen Szechuan Sauce case graph**

Figure 9 shows a graph from the Stolen Szechuan Sauce case [7] created from the *user* and *inode* attributes. The number of unique values for *the inode* attribute was 81 and the number of unique values *for user* was 14, but the graph shows that not all users worked with the files. The data is from the Windows Event Log. Users are marked in red, and the numerical inodes of the files are marked in blue.

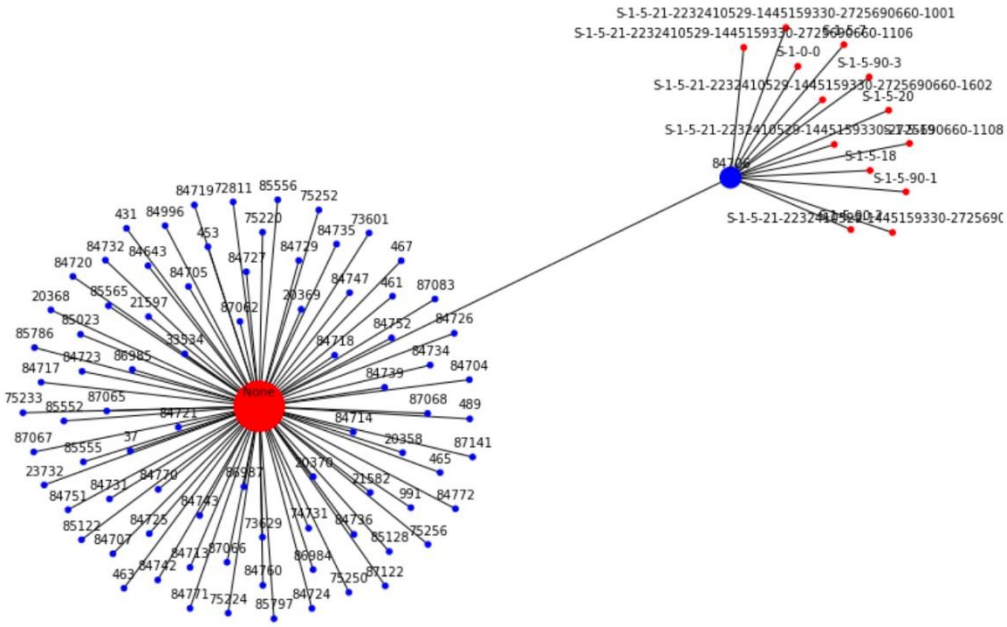



**Figure 9 - Users with inodes in the Szechuan Sauce EVT case**

Figure 10 shows a graph created from the *computer_id* and *event_id* attributes of the "NIST Data Leakage Case" [10]. Here, the search for circles revealed that all circles share three vertices and differ only in one vertex representing *the event_id* (red vertex).

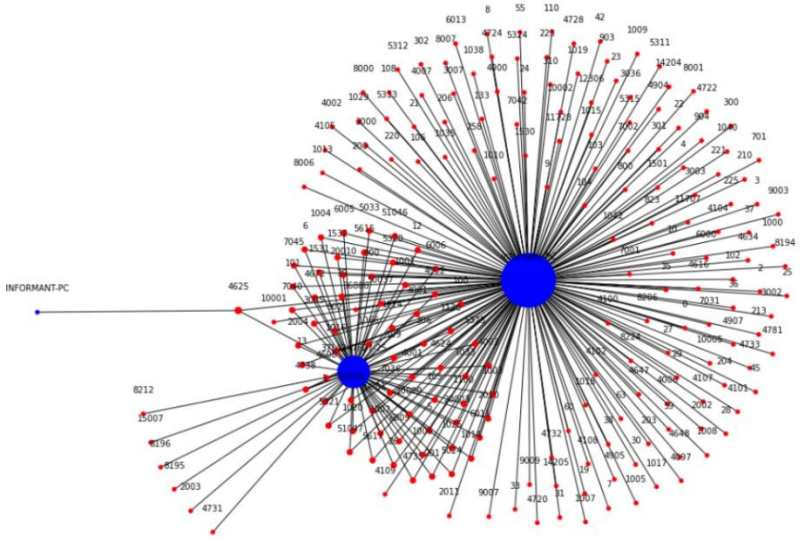



**Figure 10 - Computers and records in the NIST Data Leakage Case graph**

Figure 11 shows a graph from the "Stolen Szechuan Sauce" case [7] created from the *type* and *MACB* attributes, containing separate components. These show how individual types (red vertices) correspond to different file timestamps combinations (blue vertices).


**Figure 11 – Type and set of timestamps in the "Stolen Szechuan Sauce" case graph**

The final example is a graph from the “Stolen Szechuan Sauce” competition [7]. It represents data from the file system—numeric inodes and a combination of timestamps. The more complex Figure 12 shows what to focus on specifically, so we tested various graph properties. We searched for cycles in the graph. We found 43 of them, and seven cycles contained inodes that were manually identified through analysis as relevant to the case.

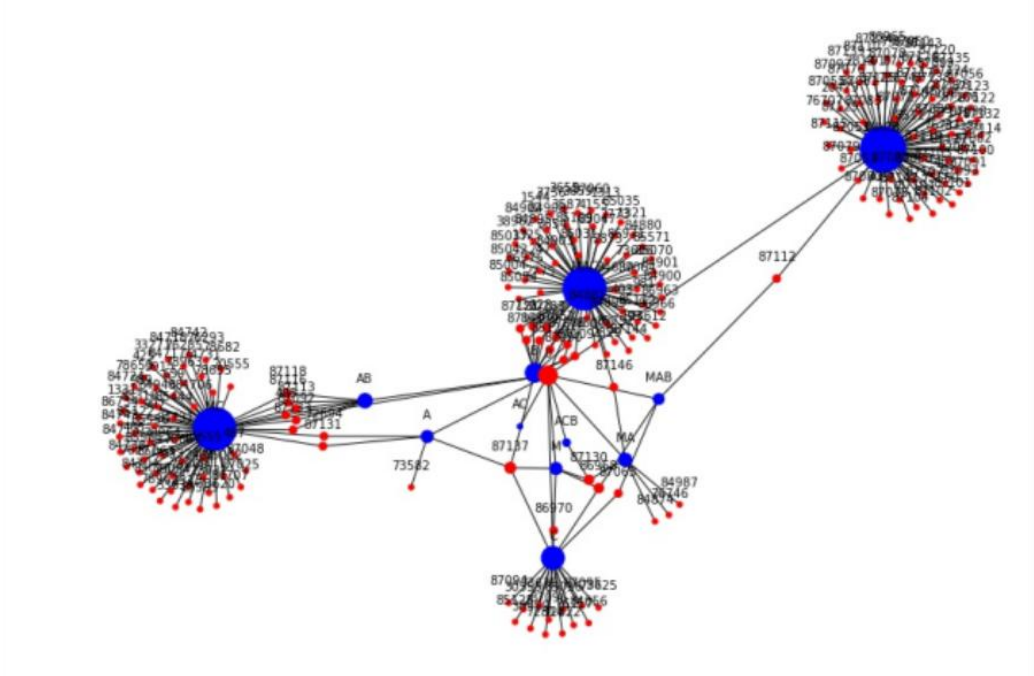



**Figure 12 - Inodes and timestamps in the Stolen Szechuan Sauce case graph**

Only unweighted graphs were displayed; of course, it is possible to add weights to the edges (e.g., the frequency of occurrence of a given event).

In online analysis, it is also possible to analyze state changes—how processes are created or terminated. It is therefore possible to create a directed graph containing multiple types of vertices (user, process, file, registry, network connection, ...). The edge direction is set by default according to the timestamps of individual recorded events.

## 5.2 Creating State Diagrams from Digital Evidence

Table 1 shows selected records that semantic analysis has flagged as suspicious activity. The table represents a snapshot of the event timeline in the operating system, with individual records sorted chronologically and capturing a sequence of operations that together may constitute a comprehensive cybersecurity incident.

| row_id | timestamp | attack_ types | mitre_ techniques | killchain_ stage | source_type |
|---|---|---|---|---|---|
| SSS_DC:10 | September 19, 2020, 3:56:03 AM | Remote Desktop Protocol Connection | T1021.001 | Delivery | Content Modification Time |
| SSS_DC:14 | 2020-09-19 T03:56:03 | Privilege Escalation | | | |
| SSS_DC:15 | 2020-09-19 03:56:03 | Privilege Escalation | T1078 | Exploitation | Content Modification Time |
| SSS_DC:143 | 2020-09-19 T03:56:04 | Service Manipulation | T1543.003 | Installation | Content Modification Time |
| SSS_DC:731 | 2020-09-19 03:56:11 | Privilege Escalation | T1068 | Exploitation | Content Modification Time |
| SSS_DC:787 | 2020-09-19 T03:56:15 | Malware Installation | T1105 | Installation | Metadata Modification Time |
| SSS_DC:788 | 2020-09-19 03:56:15 | Malware Installation | T1105 | Installation | Metadata Modification Time |
| SSS_DC:789 | 2020-09-19 03:56:15 | Malware Installation | T1105 | Installation | Metadata Modification Time |
| SSS_DC:790 | 2020-09-19 03:56:15 | Malware Installation | T1105 | Installation | Metadata Modification Time |
| SSS_DC:792 | 2020-09-19 03:56:15 | Malware Installation | T1105 | Installation | Metadata Modification Time |
| SSS_DC:799 | 2020-09-19 03:56:15 | Malware Installation | T1105 | Installation | Metadata Modification Time |
| SSS_DC:834 | 2020-09-19 T03:56:53 | Privilege Escalation | T1078 | Exploitation | Content Modification Time |
| SSS_DC:837 | 2020-09-19 T03:56:55 | Service Installation | T1543.003 | Installation | Content Modification Time |
| SSS_DC:836 | 2020-09-19 T03:56:55 | Privilege Escalation | T1078 | Exploitation | Content Modification Time |
| SSS_DC:840 | 2020-09-19 T03:56:55 | Privilege Escalation | T1078 | Exploitation | Content Modification Time |
| SSS_DC:844 | 2020-09-19 T03:56:55 | Privilege Escalation | T1078 | Exploitation | Content Modification Time |

**Table 1 Excerpt from the timeline of suspicious activities in System I**

From the complete log for the event timeline in the previous table, it can be seen that the suspicious activities were of the following nature:

- establishing an anonymous connection,
- attempt to obtain administrator privileges,

- modification of access rights,
- initiation of file installation,
- changing access rights,
- installing the suspicious file 87131-3 in multiple steps,
- repeated attempt to change file access rights.

One possible representation of the timeline of the aforementioned suspicious behavior using a state diagram is shown in Figure 13. The state diagram models the individual phases of the activity as discrete system states and the transitions between them as responses to specific events captured in the logs.

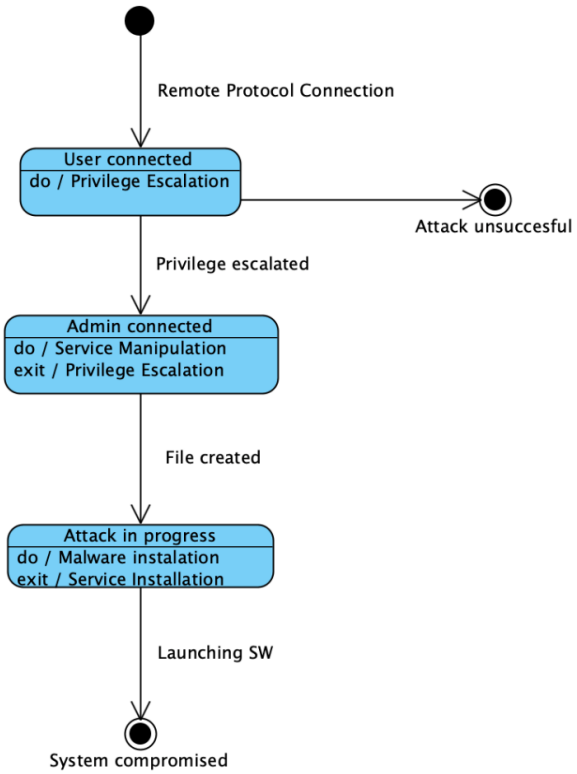



**Figure 12 - State diagram for the timeline of suspicious activities in System I**

Figure 13 illustrates the system's gradual transition from the initial establishment of a remote connection (Remote Protocol Connection) to the "User connected" state, during which an attempt is made to escalate privileges. If unsuccessful, the process ends with the "Attack unsuccessful" state.

In the event of a successful privilege escalation, the system transitions to the "Admin connected" state, which represents the acquisition of administrative privileges and enables

further operations, such as manipulating services. Subsequently, a file is created, and the system transitions to the “Attack in progress” state, during which the installation of malicious software takes place.

The final step is the execution of the malicious program (“Launching SW”), which leads to the final state “System compromised,” representing the successful compromise of the system. The diagram also captures the possibility of attack failure via an alternative branch, thereby providing a more comprehensive view of potential incident development scenarios.

The state diagram created in this way allows one to abstract from individual log entries and focus on the key phases of the attack, thereby facilitating its analysis and understanding.

## 5.3 Creating Process Diagrams from Digital Evidence

In this section, we will demonstrate the use of process modeling using BPMN, which allows us to highlight other types of forensic traces. Table 2—an excerpt from the timeline of suspicious activities in System II—serves as input, capturing the sequence of events relevant for further analysis.

| row_id | timestamp | attack_ types | mitre_ tactics | mitre_ techniques | killchain_ stage |
|---|---|---|---|---|---|
| SSS_DC:7 | 2020-09-19 03:22:07 | Remote Access | Initial Access | T1021.001 | Delivery |
| SSS_DC:77 | 2020-09-19 03:22:08 | Credential Access | | | |
| SSS_DC:155 | September 19, 2020, 3:22:09 AM | Credential Use | Persistence | T1078.002 | Exploitation |
| SSS_DC:161 | 2020-09-19 03:22:09 | Credential Use | Persistence | T1078.002 | Exploitation |
| SSS_DC:538 | 2020-09-19 T03:22:37 | Remote Access | Command and Control | T1021.001 | Command and Control |
| SSS_DC:1123 | 2020-09-19 03:23:01 | Persistence | Persistence | T1547.001 | Installation |
| SSS_DC:1127 | 2020-09-19 03:23:01 | Persistence | Persistence | T1547.001 | Installation |
| SSS_DC:1128 | 2020-09-19 T03:23:01 | Persistence | Persistence | T1547.001 | Installation |
| SSS_DC:1133 | 2020-09-19 T03:23:01 | Persistence | Persistence | T1547.001 | Installation |

| | | | | | |
|---|---|---|---|---|---|
| SSS_DC:1143 | 2020-09-19 T03:23:01 | Persistence | Persistence | T1547.001 | Installation |
| SSS_DC:1566 | 2020-09-19 T03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1567 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1569 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1575 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1577 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1579 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1580 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1581 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1582 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1591 | 2020-09-19 03:23:41 | Web Traffic to Malicious IP | Command and Control | T1071.001 | Command and Control |
| SSS_DC:1647 | 2020-09-19 T03:24:00 | C2 Communication | Command and Control | T1102 | Command and Control |

**Table 1 – Excerpt from the timeline of suspicious activities in System II**

From the complete log corresponding to Table 2 above, the following sequence of suspicious activities in the system can be identified:

- remote access to the system,
- gaining administrator access by spoofing login credentials,
- repeated system scanning,
- installation of a suspicious file,
- writing to the registry,
- attempt to visit the URL 194.61.24.102 as an administrator, downloading the file *favicon.ico,*
- writing to the registry,
- repeated attempt to access the URL as an administrator,
- Repeated access to the URL as an administrator with forged credentials,
- repeated access to the URL as an administrator,
- modification of the registry.

For each such sequence of events, there are multiple interpretations and ways to represent them. One possible model is shown in Figure 14.

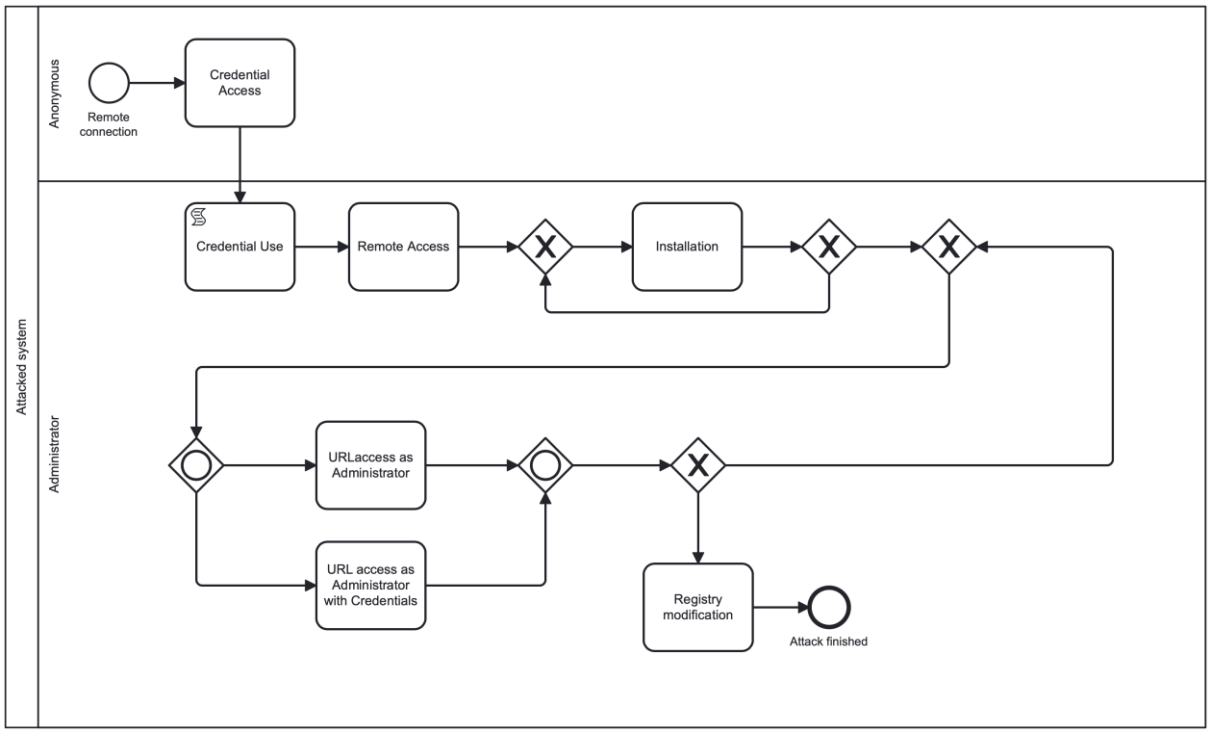



**Figure 14 - Process model for the timeline of suspicious activities in System II**

Figure 14 presents a process model created using BPMN, which captures the progression of suspicious activities in the system from the perspective of individual actors and their interactions. The diagram is divided into several lanes, which represent various entities involved in the incident, such as the attacker (anonymous), the compromised system, and the administrator.

The model begins with the establishment of an anonymous remote connection, followed by the acquisition and use of login credentials (credential access and credential use). These steps lead to gaining remote access to the system, which represents the key point of compromise. Subsequently, a malicious component is installed, and the diagram also captures possible alternative or repeated paths via decision nodes (gateways).

In the next phase, the model depicts activities related to administrative access, specifically access to URL resources either directly or via the obtained credentials. These branches subsequently converge and lead to the modification of the system registry, which may serve as a persistence mechanism or enable further system manipulation.

The entire process concludes with the “attack finished” state, which represents the successful completion of the attack. The diagram also indicates, through decision points and feedback

loops, that the attack does not necessarily proceed in a strictly linear manner, but may involve repeated attempts or alternative scenarios.

The process model created in this way allows for a better understanding of the attack's logic, identifies critical phases, and provides a basis for designing detection or defense mechanisms.

# 6 Conclusion

The proposed model for extracting digital traces into a graph representation provides an effective framework for formally capturing relationships between forensic artifacts. The resulting graph-based process and state diagrams provide a basis for subsequent analysis of the course of a cybersecurity incident, identification of key events, and detection of anomalous structures in the data.

Using graph algorithms (e.g., subgraph search), it is possible to identify patterns representing typical known attack schemes and either predict their development in real time or, as part of post-mortem analysis, gain new insights into the attacker's behavior.

Further analysis should focus on automated relationship detection, comparing multiple cybersecurity incidents, and verifying the extent to which graph representation supports more accurate and faster reconstruction of security events.

# 7 Bibliography


[1] Diestel, R. (2025). *Graph theory* (6th ed.). Springer. https://doi.org/10.1007/978-3-662-70107-2

[2] Olivé, A. (2007). *Conceptual modeling of information systems* (1st ed.). Springer. https://doi.org/10.1007/978-3-540-39390-0

[3] Weske, M. (2024). *Business Process Management: Concepts, Languages, Architectures* (4th ed.). Springer. https://doi.org/10.1007/978-3-662-69518-0

[4] Krišáková, S. P., Sokol, P., & Krivoš-Belluš, R. (2024). Analysis of Forensic Artifacts Using Graph Theory. CEUR Workshop Proceedings. https://ceur-ws.org/Vol-3792/paper26.pdf

[5] Plaso (log2timeline), 2022 [online]. Available at: https://github.com/log2timeline/plaso

[6] Eric Zimmerman's Tools*, 2026* [online]. Available at: https://ericzimmerman.github.io/

[7] Case 001 – the stolen szechuan sauce, 2020 [online]. Available at: https://dfirmadness.com/the-stolen-szechuan-sauce/

[8] Magnet CTF 2019 Windows Desktop, 2019 [online]. Available at: https://digitalcorpora.s3.amazonaws.com/corpora/scenarios/magnet/2019 CTF - Windows-Desktop.zip

[9] Magnet CTF 2022 Windows, 2022 [online]. Available at: https://digitalcorpora.s3.amazonaws.com/corpora/scenarios/magnet/2022 CTF - Windows.zip

[10] Data LeakageCase, 2018 [online]. Available at: https://cfreds-archive.nist.gov/data_leakage_case/data-leakage-case.html