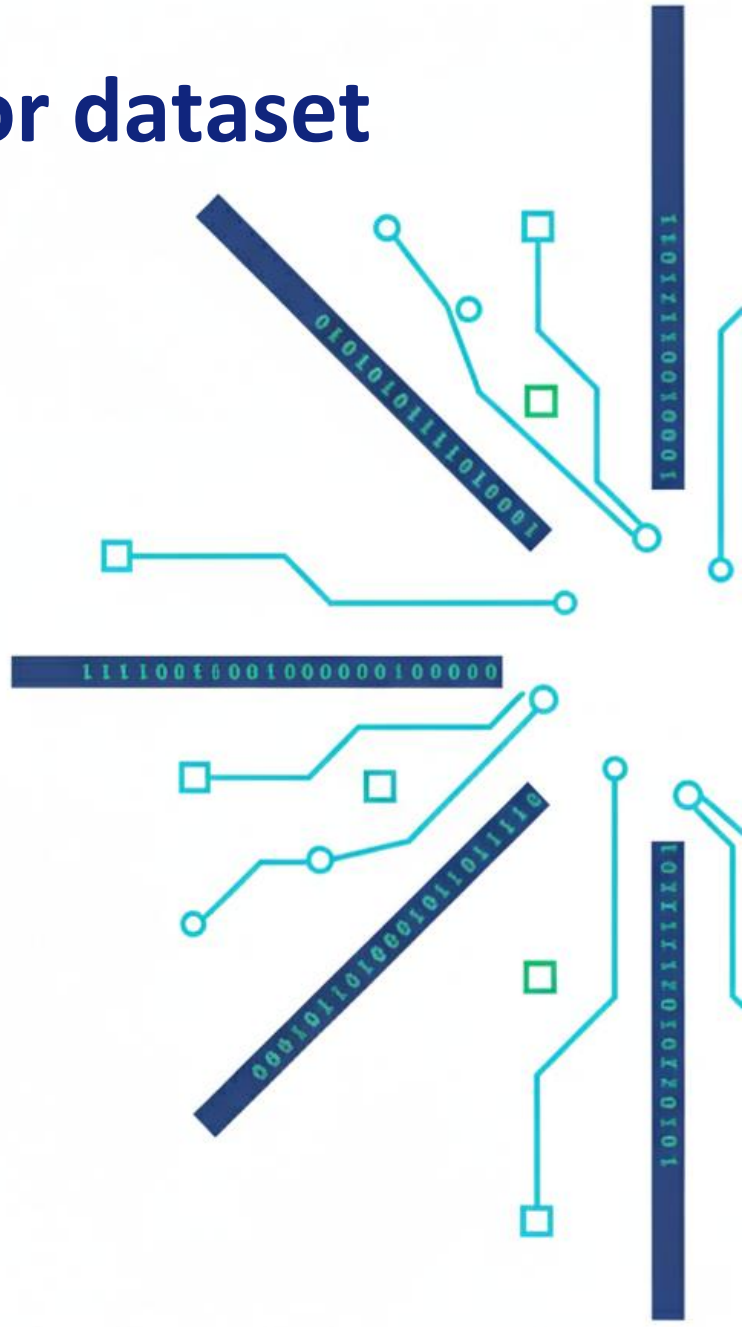
D13 – Method for dataset creation



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

The development of machine learning methods and advanced data analysis also has a significant impact on the field of digital forensic analysis. However, automated processing of large amounts of forensic data, identification of behavior patterns, correlation of events, and detection of anomalies require high-quality, well-structured, and representative data sets. The way in which these datasets are created has a fundamental impact on the usability and reliability of the resulting models.

The development of machine learning methods and advanced data analysis is also having a significant impact on the field of digital forensic analysis. However, automated processing of large amounts of forensic data, identification of behavior patterns, correlation of events, and anomaly detection require high-quality, well-structured, and representative data sets. The way these data sets are created has a fundamental impact on the usability and reliability of the resulting models.

One of the main challenges in the field of digital forensic analysis is the **lack of high-quality datasets** that meet several key requirements, such as the availability of comprehensive case studies and a sufficient number of relevant digital artifacts [1,2]. This problem is highlighted in several scientific publications, which emphasize the need for well-structured datasets suitable for forensic purposes and at the same time point out the **overall lack of such resources** in this area [1,2,3].

From the perspective of model learning and the application of machine learning in digital forensic analysis, three basic types of datasets can be identified:

1. **Real datasets** obtained directly from the resolution of actual security incidents,
2. **Created datasets from simulated attacks and attacker techniques**, carried out in a controlled environment,
3. **Datasets originating from CTF (Capture The Flag) competitions**, in which attacks are simulated and analytical tasks and questions are subsequently prepared.

Each of these approaches has its advantages, limitations, and specifics that must be taken into account when designing the research methodology and interpreting the results.

2.1 Real datasets from actual security incidents

Real data from actual security incidents are the **most valuable source** for digital forensic analysis in terms of authenticity and realism. They capture the actual behavior of attackers, real system configuration errors, unforeseen combinations of events, and the complex dynamics of incidents in a production environment. For this reason, they provide an ideal basis for the development and testing of forensic analysis methods.

However, the use of such data presents **significant problems and limitations**. Real-world datasets often cover only a **limited range of attacker techniques, tactics, and procedures**

(TTPs) that occurred in a given security incident. They are therefore not a representative cross-section of the entire spectrum of possible attacks, but rather a specific and context-bound case. In addition, real-world security incidents often vary in the quality and completeness of available artifacts, as data may be corrupted, deleted, or incomplete.

However, the biggest obstacle is the **practical impossibility of sharing this data**. This data is sensitive in nature and may contain personal data and other sensitive information (e.g., intellectual property, trade secrets), and its use is restricted by legislative frameworks, personal data protection, trade secrets, and internal security policies of organizations. For this reason, its use for academic research and development, replicability of experiments, and comparability of results is significantly limited. This is also confirmed in article [4], where the authors point out in their findings that when creating a dataset, it is important to ensure that the datasets are not only created correctly, but also reflect scenarios that may occur in real-life situations. They also emphasize that creating and sharing datasets can be challenging due to legal restrictions or data management and protection. The same is stated by Breitinger and Jotterand [3], who concluded that creating and sharing datasets is essential for progress and for enabling the comparison of results. They also emphasized the need to be cautious with regard to specific laws, such as copyright or licensing.

2.2 Datasets from simulated attacks and attacker techniques

The second approach to dataset creation is **targeted simulation of attacks and attacker techniques** in a controlled and isolated environment. In this case, attacks are carried out consciously and systematically. Attacks are often carried out 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 allows for accurate 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, the simulation may 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. Nevertheless, simulated attacks represent an important compromise between realism and data controllability.

2.3 Datasets from CTF competitions

The third category consists of **datasets from CTF (Capture the Flag) competitions**, which have long been used in cybersecurity education and training. CTF tasks are typically designed to

simulate a specific security incident or part of it, and participants answer predefined questions by analyzing digital traces.

The use of CTF datasets has certain **inherent limitations**. CTF scenarios are often artificially created, focused on a specific type of attack, and usually carried out by a single attacker. Such a model may not fully reflect reality, where multiple attackers may be involved, multi-stage attacks may take place, and the outcome of the investigation is not known in advance. Furthermore, CTF data implicitly assumes the existence of a solution, while 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 tasks are artificially created, the **process of data analysis and forensic artifact extraction** itself is largely the same as in the investigation of real incidents. Furthermore, the structured nature of CTF tasks, often based on questions and answers, can promote a systematic analytical approach that is transferable to the real world and can lead to faster problem identification and resolution.

2.4 Motivation for the selection of data sources

In view of the above, we decided to focus this study primarily on **datasets created from simulated attacker techniques and datasets originating from CTF competitions**.
The main reason is the limited availability of data from real security incidents and the practical impossibility of sharing it in an academic and research environment.

The focus on **Windows operating system disk images** is based on the scope and complexity of digital forensic analysis as a discipline. Individual subdomains, such as network forensic analysis, forensic analysis of Windows, Linux, or mobile devices, differ significantly in data structure, tools used, and analytical procedures. Each of these areas requires specific methods of data preprocessing and analysis, which makes a universal approach impossible.

By using simulated attacks and CTF datasets in a Windows environment, it is possible to create **reproducible, shareable, and methodologically controllable datasets** that provide a suitable basis for research and development of machine learning methods in digital forensic analysis.

3 Method for forensic dataset creation based on attackers' techniques

For the purpose of training and validating machine learning (ML) models designed to automate forensic analysis, a dataset was created that faithfully reflects the behaviour of modern attackers in the post-exploitation phase. Unlike synthetic datasets generated randomly, in the ADFIR project we used a combination of automated simulation and manual execution of advanced offensive TTPs (Tactics, Techniques, and Procedures - TTPs).

The data collection was conducted on a virtualized workstation designated as RD05W-W11-1, operating on Windows 11 (version 23H2) and hosting the Cymulate agent for testing purposes. The underlying infrastructure utilizes the VMware ESXi hypervisor. The testing environment simulates a standard corporate domain structure, where the endpoint is managed by two redundant domain controllers running Windows Server 2022 Standard.

To generate synthetic yet realistic attack simulation, the Cymulate platform [5] was utilized within the experimental environment. Cymulate is a Breach and Attack Simulation (BAS) tool designed to continuously validate cyber resilience through automated attack simulations.

Unlike traditional penetration testing, Cymulate allows for the execution of comprehensive attack scenarios in a controlled environment without disrupting production operations. Crucially for this research, the platform generates high-fidelity digital footprints at both the operating system and network levels.

For the purposes of the research, the following features of the platform were essential:

- MITRE ATT&CK® Alignment: Cymulate simulates the Tactics, Techniques, and Procedures (TTPs) of real-world adversaries mapped to the MITRE ATT&CK framework. This ensures that the generated data reflects current trends in cyber threats (e.g., APT groups, ransomware, and advanced trojans).
- Generation of Forensic Artifacts: Although the attacks are simulated, the interaction with the operating system is authentic. The execution of these simulations leaves specific traces necessary for training the forensic automation tool, including:
    - Entries in system logs (Windows Event Logs).
    - Modifications to the Windows Registry and file system.
    - Residual data in volatile memory (RAM).
    - Network communication patterns (C2 traffic).

3.1 Dataset creation

To construct a dataset suitable for training machine learning models, it is best to establish a controlled generation pipeline focused on producing realistic forensic artifacts. The input to this phase is a clean, instrumented virtual environment, while the output consists of raw

forensic disk images and memory dumps containing both benign and malicious activity patterns.

The data generation process was designed to simulate the complete lifecycle of a cyberattack and was executed through the following methodological steps:

- orchestration of multi-stage attack scenarios,
- injection of advanced evasion techniques,
- temporal annotation (ground truth generation), and
- forensic data acquisition.

Orchestration of Multi-Stage Attack Scenarios: The core of the dataset was generated by executing advanced attack scenarios that mimic the Tactics, Techniques, and Procedures (TTPs) of sophisticated threat actors. The simulation followed a linear kill-chain progression, moving from initial access (e.g., payload delivery via downloaders) to execution, persistence establishment, privilege escalation, and credential access. This structured approach ensures that the dataset captures the sequential dependencies and causal relationships between different forensic artifacts.

Injection of Advanced Evasion Techniques: To reflect the complexity of modern threats, the simulation included techniques specifically designed to minimize forensic footprint and bypass static detection. This included the use of "Living off the Land" (LotL) binaries, fileless execution methods (e.g., reflective DLL injection, in-memory .NET assembly loading), and obfuscation of command-line arguments. These techniques ensure that the resulting dataset challenges the detection capabilities of the model beyond simple signature matching.

Temporal Annotation (Ground Truth Generation): Crucial to the supervised learning pipeline, precise timestamps were recorded for every distinct phase of the attack simulation. These time windows constitute the "ground truth," defining the exact periods during which malicious artifacts were introduced to the system. These annotations are subsequently used in the data processing pipeline to label the supertimeline segments, allowing for the calculation of precision and recall metrics during model evaluation.

Forensic Data Acquisition: Upon completion of the simulation scenarios, the state of the virtual environment was frozen, and a full forensic acquisition was performed. This process yielded a raw bitstream image of the non-volatile storage (disk image). These raw data sources serve as the primary input for the forensic extraction framework (e.g., Athena), linking the physical simulation phase to the logical data processing phase described in the subsequent section.

3.2 Attack techniques used in the creation of the dataset

The base layer of network and system activity was generated using the Cymulate platform, which was used to create a realistic background and simulate standard intersection vectors.

Within the Cymulate instrument, an advanced scenario was carried out, simulating the activities of the APT-19 group.

3.2.1 APT-19 attack preparation stages

The advanced APT-19 scenario begins with service-level execution: an attacker installs and runs a PowerShell-based service, prepares a payload using Invoke-WebRequest, and runs Base64-encoded PowerShell code for stealth.

- **Downloader: Invoke-WebRequest (PowerShell)**
  This attack simulation downloads a file from a specified URL to a specific path on your computer using the Invoke-WebRequest cmdlet in PowerShell. This cmdlet allows the user to retrieve the contents of a web page or file from a web server and can be used to download the file from a remote server to a local machine.

- **Execute Base64-Encoded PowerShell Command**
  This attack simulation creates malicious PowerShell code that is encoded with Base64. This type of attack is commonly used by attackers to gain access to the system. Once successfully executed, the PowerShell code executes an encoded command that outputs the message "Hey, Atomic!" to standard output.

- **Executing a Command as a Service**
  This attack simulation creates a service that allows any command to be executed. When you try to run a command such as PowerShell, the service will report a failed completion, even if the code is executed correctly. If the command is successful, the sc.exe create command creates a new service and Powershell.exe creates a new file named art-marker.txt.

- **Process Injection: Reflective DLL Injection**
  Reflective DLL injection is a method of attacking a system by injecting malicious code into a running process. This technique allows an attacker to inject a dynamic library (DLL) into the target process without the need to first write the DLL to disk. Instead, the DLL is loaded directly from memory, bypassing the traditional security mechanisms that may be deployed. Once a malicious DLL has been inserted, it can be used to execute malicious code or gain access to sensitive information.

- **Service Installation Using PowerShell**
  This attack simulation installs an on-premises service using PowerShell. Once successfully launched, PowerShell will download the AtomicService.exe from Github. It then uses the New-Service and Start-Service commands to start the service.

- **Hidden Window**
  This simulation of the attack triggers a hidden PowerShell window that executes the executable file without the user's knowledge. This is accomplished by passing the -

WindowStyle Hidden argument to the PowerShell command, setting the WindowStyle parameter to hidden. This effectively hides the PowerShell window and its associated processes from the user's view.

- **Modifying SecurityHealth Registry with CMD**
  This attack simulation involves modifying the RUN key in the Local Machine registry to change the Windows Defender executable to run at system startup. This can only be done when Command Prompt (CMD) is run with administrator privileges.

- **Registry Modification: Hide File Extensions**
  This command sets a registry entry under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced, which instructs Windows Explorer to hide the extensions of all files.

- **Registry Run Keys: Persistence**
  This attack simulation creates new registry keys in sites that are commonly used to maintain persistence (persistent presence), such as the Windows registry. The values of these keys are then set to refer to the payload that the attacker wants to run. This allows the payload to remain permanently present in the system even after the system is restarted.

- **Hijack Execution Flow: Path Interception (PATH)**
  This attack simulation test downloads a malicious binary to the specified path. It then sets the PATH environment variable to redirect the alias (net.exe) to the path to the malicious binary. This will set the new path with a higher priority than any other paths in the PATH value. When the net command is used, a malicious binary is executed instead of the original binary. The default binary should output the string "This is a test binary by Cymulate".

- **LOLBIN Download - Remote execution with ADS**
  The command-line interpreter in Windows adds content to an alternate data stream (ADS). Example of a harmful use case: It can be used to bypass defensive countermeasures or to hide as a persistence mechanism.

3.2.2 APT-19 attack execution stages

Specific execution sequences focused on persistence, privilege escalation, and identity theft (credential access/theft) were then manually injected into this environment. The attack scenario took place in the following stages:

1. **Persistence:** SharPersist (a C# toolkit for persistence in Windows) was used to maintain persistent access, which in this case was run from C:\Windows\Tasks\SharPersist.exe. With this tool, persistence mechanisms were created using Scheduled Tasks, services,

and registry keys. The created persistence ensured the presence of a backdoor, located in C:\Windows\Tasks\mssvc\mssvc.exe.

2. **Environment enumeration and privilege escalation:** After ensuring persistent access, a vulnerability investigation of the system was conducted. Two complementary instruments have been used for this purpose:

   - **SharpUp** (C:\Windows\Tasks\SharpUp.exe): A C# implementation designed to quickly identify configuration errors (e.g., modifiable services) that allow escalation to the administrator level.
   - **winPEAS.ps1** (C:\Windows\Tasks\winPEAS.ps1): A complex PowerShell script that performed an in-depth analysis of the system. This tool generates a significant number of artifacts (reading files, accessing registers), creating a specific "noisy" pattern in the dataset.

   Both tools perform an extensive set of checks (e.g., service configurations, access rights, registers, file system) in normal use, which typically translates into an increased number of system queries and a "noisier" profile in the logs compared to minimalist manual enumeration.

3. **Credential Access:** SafetyKatz and SharpKatz were used to extract authentication material (NTLM and Kerberos keys/tickets) from the LSASS process.

   - C:\Windows\Tasks\SafetyKatz.exe – A C# tool from the GhostPack suite that acts as a **wrapper around the well-known Mimikatz hacking tool.** SafetyKatz creates a minidump of the LSASS process's memory using **the MiniDumpWriteDump** function to *C:\Windows\Temp\debug.bin*. It then **loads the modified Mimikatz** via PELoader, executes sekurlsa::logonpasswords and sekurlsa::ekeys over the dump, and deletes the file when finished.

   - C:\Windows\Tasks\SharpKatz.exe – C# port of the selected Mimikatz commands.

4. **Defence Evasion:** The SharpKiller tool (port "AMSI-Killer" to .NET) was used to bypass AMSI (AntiMalware Scan Interface) by patching the in-memory check so that AMSI does not block malicious content.
   - C:\Windows\Tasks\SharpKiller.exe

3.3 Threat Landscape

The created dataset is relevant to current research in the field of cybersecurity, as it reflects a shift in the paradigm of attacks from traditional executables to techniques referred to as **"Living off the Land" (LotL)** and **".NET Tradecraft".**

3.3.1 Detection of "Fileless" and .NET threats

Traditional antivirus systems have long focused on file signatures on disk. The tools used in our dataset (SharpUp, SharpKatz, SharPersist) are often loaded directly into memory via *the Reflective DLL Injection* or *Assembly Loading technique*, minimizing the footprint on the disk. For ML models, this presents a challenge to identify anomalies in process behaviour and API calls, rather than relying on static file attributes. The dataset provides training data for detecting malicious code running in the context of a legitimate common language runtime (CLR).

3.3.2 Abuse of Trusted Paths and Camouflage

The placement of tools in the C:\Windows\Tasks directory is not accidental. This directory is often browsed by administrators because it contains a lot of legitimate system files, and at the same time, ordinary users also have write privilege to it in certain configurations. Combining this site with names like mssvc.exe simulates the sophisticated cloaking that is typical of Advanced *Persistent Threat (APT*) groups. An ML model trained on this data must learn to distinguish between a legitimate process and its spoofed counterpart based on contextual metadata (parental process, start time, absence of digital signature), not just by name.

3.3.3 Bypass of modern protectors

The inclusion of the AMSI bypass (Sharp-Killer) technique in the dataset is essential for simulating a sophisticated adversary. If the ML model cannot detect the telemetry disabling act (AMSI) itself, it becomes blind to the subsequent phases of the attack. This dataset allows you to train models to detect so-called "pre-exploit" activities, where an attacker manipulates the integrity of the monitoring system itself.

3.3.4 Practical applicability

The techniques described are currently actively used not only by state-sponsored actors, but also by Ransomware-as-a-Service operators (e.g., Conti groups, LockBit), who use tools such as *SharpUp* and *Mimikatz* variants to move laterally in the network before the data is encrypted. Research based on this dataset therefore has a direct impact on improving detection capabilities against the most widespread cyber threats today.

4 Method for forensic dataset creation from CTF

This chapter presents possible data sources and methodologies for creating datasets for digital forensic analysis. The chapter describes what types of sources—including CTF competitions, reference dataset portals, and synthetic security datasets—are suitable for obtaining relevant artifacts. It also discusses two approaches to dataset creation: using forensic tools and creating a dataset using embedded data.

4.1 Possible sources of data

As mentioned above, **CTF competitions** are a suitable source for creating datasets for digital forensic analysis, as access to real data from security incidents can be problematic and limited. Data from real security incidents is often not publicly available and obtaining it can be difficult due to the need for authorisation, legal restrictions, or personal data protection.

For the purpose of creating datasets, combined sources are therefore used to simulate or directly access relevant artifacts and scenarios. The following data sources are practical and proven options that are commonly used in digital forensic analysis research and teaching:

CTF and training datasets

- **DFIR CTFs Archive** – a collection of various CTF tasks and datasets for DFIR training (disk images, artifacts) – clearly available through portals such as DFIR Training / DFIR CTF archives [6].
- **Digital Corpora** – freely available collection of disk images, scenarios, and artifacts, including various CTF scenarios and M57 Patents scenario (disk images, memory, PCAPs) [7].
- **Honeynet Project Forensic Challenges** – competitions and forensic challenges (challenges and dataset packages for practicing forensic techniques) [8].

Reference dataset portals and projects

- **EVIDENCE Project** – a European project focused on the exchange and standardization of digital evidence in practice; background and framework materials for working with real evidence (access to data may be limited) [9].

4.2 Dataset creation using forensic tools

A dataset for research in the field of digital forensic analysis can be created directly using specialized forensic tools. Data extracted from disk images of individual CTF scenarios can be processed using the **Kroll Artifact Parser and Extractor (KAPE)** tool [10]. KAPE is one of the widely used tools in the field of digital forensic analysis and enables systematic, repeatable, and automated extraction and parsing of forensic artifacts from the Windows operating system. Thanks to the support of modular profiles (Targets and Modules), it is possible to precisely define which artifacts are to be retrieved from the system and how they are to be processed into a structured form suitable for further analysis.

In addition to the KAPE tool, separate forensic tools from the **Eric Zimmerman Tools** [11] set can also be used to create a dataset. These tools are often used in practice either directly or as parsing modules integrated into KAPE. These tools enable detailed analysis of specific types of forensic artifacts and generate outputs most often in the form of CSV files, which are suitable for subsequent processing using data analysis and machine learning methods.

The process of creating a dataset using forensic tools involves several systematic steps that ensure the extraction, parsing, and evaluation of forensic data. Each step is designed to preserve the integrity of the original data and to ensure that the resulting dataset contains analytically relevant information suitable for further digital forensic analysis and experiments with data analysis or machine learning.

The following steps were applied to create the dataset:

1) mounting the disk image,
2) extracting forensic artifacts using forensic tools (e.g., KAPE),
3) parsing forensic artifacts,
4) cleaning and preparing output data,
5) automated and manual enrichment of output data (silver and gold labeling)

In the following subchapters, we provide more detailed comments on each section.

4.2.1 Mounting the disk image

The data processing consisted of several steps. In the first step, the disk image was mounted in read-only mode using disk image mounting tools (e.g., Arsenal Image Mounter [12]). This approach ensures the integrity of the original data and eliminates the risk of unintentional modification during analysis, which is a key principle of digital forensic methodology.

4.2.2 Extraction of forensic artifacts using forensic tools

After successfully mounting the disk image, the KAPE tool is launched on the mounted file system. During the extraction phase, it is possible to use built-in "targets" that allow for the targeted extraction of relevant forensic artifacts. Specifically, it is advisable to use targets for the following artifacts:

- $MFT (Master File Table),
- $J (USN Journal),
- SRUM (System Resource Usage Monitor),
- Registry hives,
- Amcache,
- LNK files and Jump Lists,
- Prefetch files,
- Windows Event Logs,
- Recycle Bin, and

- Windows Timeline.

These artifacts are essential sources of information about user activity, application launches, file operations, and system events, and are therefore crucial for incident reconstruction and analysis of potentially malicious activity. More information about forensic artifacts on the Windows operating system can be found in deliverable - D12 Method for automated collection of digital evidence.

4.2.3 Parsing forensic artifacts

The following KAPE tool modules and relevant tools from Eric Zimmerman can be used for the above-mentioned forensic artifacts:

- **AmcacheParser.mkape** – using the **AmcacheParser** tool [13].
- **AppCompatCacheParser.mkape** – use of the **AppCompatCacheParser** tool [14].
- **EvtxECmd.mkape** – use of the **EvtxECmd** tool [15].
- **JLECmd.mkape** – using the **JLECmd** tool [16].
- **LECmd.mkape** – using the **LECmd** tool [17].
- **MFTECmd.mkape** – using the **MFTECmd** tool [18].
- **PECmd.mkape** – using the **PECmd** tool [19].
- **RBCmd.mkape** – using the **RBCmd** tool [20].
- **RECmd_AllBatchFiles.mkape** – using the **RECmd** tool (processing all batch files) [21].
- **SBECmd.mkape** – using the **SBECmd** tool [22].
- **WxTCmd.mkape** – using the **WxTCmd** tool [23].
- **SrumECmd.mkape** – using the **SrumECmd** tool [24].

These parsers transform raw forensic artifacts into a structured form, resulting in CSV files that represent individual artifact types and their attributes.

4.2.4 Cleaning and preparing output data

The output files are CSV files containing selected forensic artifacts, whose structure and number may vary depending on:

- the version of the Windows operating system,
- system settings,
- the number of user accounts (more users lead to more output files),
- the availability of specific mechanisms (e.g., SRUM is not present in all OS versions).

Console logs, which are generated by the KAPE tool during runtime by default, should be removed from the outputs as they do not represent analytically relevant data for the purposes of creating a dataset. It is also advisable to exclude the PECmd parser output in the form of a timeline, as the standard CSV output of this tool contains more detailed and analytically valuable information.

4.2.5 Automated and manual enrichment of output data

The extraction and parsing of forensic artifacts is followed by a phase of enriching the output data, which is based on identifying the time windows during which malicious activity took place. The time windows for individual datasets are determined primarily on the basis of publicly available descriptions of CTF scenarios (CTF write-ups), which specify the course and time frame of the incident, or on the basis of manual forensic analysis of specific CTF cases.

All outputs from Eric Zimmerman Tools are then automatically analyzed. As part of this analysis, attributes containing timestamps (so-called silver labeling) are identified in each CSV file. Based on their comparison with the defined incident time window, each row of data is evaluated as follows:

- value **1** if at least one timestamp in the row falls within the identified security incident time window,
- value **0** if no timestamp in the row belongs to this time window.

When evaluating individual datasets, other important time milestones known from CTF scenarios are also taken into account. Typically, these include the time of operating system installation, the start and end of the incident (incident timeframes), and the time of system seizure or digital evidence collection. Information about the installation of the system is used to trim the data, as records prior to this time should not be present in the system and their timestamps are highly likely to have been modified. From the time of system installation until the start of the incident, records are evaluated as 0, during the incident as 1, and after its end until the time of securing again as 0. Records after the system was secured are removed from the dataset again, as no new legitimate artifacts should be added to the system after this point and their timestamps would inevitably be changed.

Examples of timeframes for some CTF case studies are shown in Tab. 1.

Case	Install OS	Incident timeframes	Collection of evidence
NIST Data Leakage Case	2015-03-22 14:34:26	2015-03-23 13:29 - 2015-03-23 16:43, 2015-03-24 09:26 - 2015-03-24 17:06, 2015-03-25 10:46 - 2015-03-25 11:30	2015-04-23 10:58:22
SSS - DC	2020-09-17 16:43:59	2020-09-19 02:19 - 2020-09-19 02:35	2020-09-19 04:51:27
SSS - Desktop	2020-09-18 05:47:03	2020-09-19 02:35 - 2020-09-19 08:52	2020-09-18 22:18:11
Magnet CTF 2019	2018-07-28 07:27:53	2019-03-18 18:35:00 - 2019-03-18 19:08:57	2019-03-20 21:29:33
Magnet CTF 2020	2020-02-14 02:10:21		2020-04-22 21:55:30

Magnet CTF 2022	2022-02-04 07:05:47	2022-02-10 18:23 - 2022-02-12 02:20	2022-02-13 15:52:20

Table 1 – Time frames for selected CTFs

Exceptions to this procedure are outputs that do not contain any time attributes (e.g., some CSV files extracted from Windows registries). In these cases, it is not possible to determine the relationship of the record to the incident based on time, and therefore these records are assigned a value of **N/A (not applicable)**.

In addition to automated time evaluation, it is advisable to supplement the dataset with manual labeling (so-called gold labeling). In the case of CTF scenarios, a detailed manual forensic analysis is performed, based on which records that are directly related to a security incident with a high degree of certainty are marked with a value of 1. Other records are not clearly evaluated, as it is not possible to assign a value of 0 (clearly unrelated to the security incident) to some of them, while others need to be labeled "unknown" because it was not possible to determine with certainty their relationship to the security incident.

This means that records can be rated in several ways: either automatically based on their classification within the time frame of the incident (silver labeling) or manually based on an assessment of whether a specific record from the forensic output is relevant or irrelevant to the security incident in question (gold labeling).

4.3 Creating an embedding dataset

At the input of the embedding-based dataset construction pipeline, we use the output of the **Plaso forensic framework** [25], specifically the **supertimeline**, which represents a chronologically ordered aggregation of forensic artifacts extracted from the disk image. The supertimeline integrates events from multiple sources (e.g., file system metadata, registry entries, event logs, browser artifacts), providing a unified temporal view of system activity.

To make this raw forensic timeline suitable for machine learning–based analysis, a multi-stage transformation process was applied, consisting of the following steps:

1) marking data according to the incident time windows listed in the table (ADD),
2) trimming the system installation and image backup time,
3) creating a delta for every two records,
4) column selection,
5) combining selected columns into a text field,
6) filtering the scaler on all deltas,
7) training the tokenizer on all texts,
8) dividing into windows,
9) creating embeddings from text in windows, and
10) returning embeddings, deltas, labels as triples.

4.3.1 Marking data according to incident time windows (ADD)

Each supertimeline record was annotated based on predefined **incident time windows**, which were derived from prior knowledge of attack execution times (ground truth). These time windows represent periods during which malicious activity is known to have occurred. Records falling within these windows were labeled accordingly, enabling supervised or semi-supervised learning. This step establishes a direct temporal linkage between forensic artifacts and attack phases.

4.3.2 Trimming system installation and image backup artifacts

To reduce noise and prevent bias in the dataset, we removed timeline segments corresponding to **operating system installation**, initial configuration, and **forensic image acquisition or backup activities**. These phases typically generate dense but non-relevant forensic artifacts that can dominate the timeline and distort temporal patterns unrelated to attacker behavior.

4.3.3 Delta computation with logarithmic scaling

For each pair of consecutive records in the supertimeline, we computed a **temporal delta**, defined as the difference between their timestamps. This delta captures the temporal dynamics of system activity rather than absolute time values.
To mitigate the effect of extreme timestamp gaps (e.g., system inactivity, shutdown periods), the deltas were transformed using **logarithmic scaling**, which stabilizes variance and improves robustness for downstream ML models.

4.3.4 Feature (column) selection

From each Plaso record, we selected a subset of semantically meaningful attributes commonly used in forensic interpretation:

- **user** – the user context associated with the event
- **host** – the system or hostname
- **desc** – human-readable description of the artifact
- **MACB** – file activity semantics (Modified, Accessed, Changed, Birth)
- **sourcetype** – origin of the artifact (e.g., NTFS, Registry, EVTX)
- **type** – artifact or event type

This selection balances contextual richness with dimensionality reduction.

4.3.5 Textual fusion of selected attributes

The selected columns were concatenated into a single **textual representation** per timeline record. This transformation converts heterogeneous forensic metadata into a unified text format, making it compatible with **natural language processing (NLP)–based embedding models**.

At this stage, records originating from all disk images and experiments were merged into a **single consolidated dataset**, ensuring consistent tokenization and embedding space across all samples.

4.3.6 Fitting the delta scaler

A global scaler was fitted on all computed delta values across the entire dataset. This scaler was later applied uniformly during model training and evaluation, ensuring consistent normalization of temporal features across different samples and scenarios.

4.3.7 Training the tokenizer on forensic text

A tokenizer was trained on the complete corpus of forensic text representations. Training the tokenizer on domain-specific forensic language (e.g., registry paths, executable names, event descriptions) allows the embedding model to better capture the semantics of digital forensic artifacts compared to generic tokenizers.

4.3.8 Windowing of the supertimeline

The continuous supertimeline was segmented into **fixed-length windows**, each representing a short temporal context of system activity. Windowing enables the model to learn local behavioral patterns and temporal correlations between events, rather than treating each record in isolation.

Each window can be interpreted as a micro-sequence of forensic activity corresponding to a potential attack phase or benign system operation.

4.3.9 Embedding generation from windowed text

For each window, embeddings were generated from the aggregated textual content using the trained tokenizer and embedding model. These embeddings encode the semantic and contextual information of forensic events into dense vector representations suitable for downstream ML tasks such as classification, clustering, or anomaly detection.

4.3.10 Output construction

The final dataset consists of **triples**:

- **Embeddings** representing semantic event context,
- **Temporal deltas** capturing timing dynamics,
- **Labels** derived from incident window annotations.

This structured output enables hybrid modeling approaches that jointly exploit **semantic**, **temporal**, and **supervisory** information, providing a robust foundation for advanced forensic analysis and attack detection experiments

5 Bibliography

[1] Grajeda, C., Breitinger, F., & Baggili, I. (2017). Availability of datasets for digital forensics—and what is missing. *Digital Investigation, 22*(Suppl.), S94–S105. https://doi.org/10.1016/j.diin.2017.06.004

[2] Luciano, L., Conti, M., Dragoni, N., Massacci, F., & Smith, C. (2018). Digital forensics in the next five years. In *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018)*. ACM. https://doi.org/10.1145/3230833.3232813

[3] Breitinger, F., & Jotterand, A. (2023). Sharing datasets for digital forensics: A novel taxonomy and legal concerns. *Forensic Science International: Digital Investigation, 45*, 301562. https://doi.org/10.1016/j.fsidi.2023.301562

[4] Horsman, G., & Lyle, J. R. (2021). Dataset construction challenges for digital forensics. *Forensic Science International: Digital Investigation, 38*, 301264. https://doi.org/10.1016/j.fsidi.2021.301264

[5] Cymulate Ltd. (n.d.). *Cymulate platform*. Available at https://cymulate.com/platform/ (Accessed December 15, 2025).

[6] DFIR Training. (n.d.). *DFIR CTFs archive: Test images*. Available at https://www.dfir.training/downloads/test-images (Accessed December 15, 2025).

[7] Digital Corpora. (n.d.). *Disk images*. Available at https://digitalcorpora.org/category/disk-images/ (Accessed December 15, 2025).

[8] The Honeynet Project. (n.d.). *Forensic challenges*. Available at https://www.honeynet.org/challenges/ (Accessed December 15, 2025).

[9] EVIDENCE Project Consortium. (n.d.). *EVIDENCE project*. Available at https://www.evidenceproject.eu/ (Accessed December 15, 2025).

[10] Kroll, LLC. (n.d.). *Kroll Artifact Parser and Extractor (KAPE)*. Available at https://www.kroll.com/en/services/cyber/incident-response-recovery/kroll-artifact-parser-and-extractor-kape (Accessed December 15, 2025).

[11] Zimmerman, E. (n.d.). *Eric Zimmerman tools*. Available at https://ericzimmerman.github.io/#!index.md (Accessed December 15, 2025).

[12] Arsenal Recon. (n.d.). *Arsenal Image Mounter*. Available at https://arsenalrecon.com/products/arsenal-image-mounter (Accessed December 15, 2025).

[13] Zimmerman, E. (n.d.). *AmcacheParser*. GitHub repository. Available at https://github.com/EricZimmerman/AmcacheParser (Accessed December 15, 2025).

[14] Zimmerman, E. (n.d.). *AppCompatCacheParser*. GitHub repository. Available at https://github.com/EricZimmerman/AppCompatCacheParser (Accessed December 15, 2025).

[15] Zimmerman, E. (n.d.). *EvtxECmd*. GitHub repository. Available at https://github.com/EricZimmerman/evtx (Accessed December 15, 2025).

[16] Zimmerman, E. (n.d.). *JLECmd*. GitHub repository. Available at https://github.com/EricZimmerman/JLECmd (Accessed December 15, 2025).
[17] Zimmerman, E. (n.d.). *LECmd*. GitHub repository. Available at https://github.com/EricZimmerman/LECmd (Accessed December 15, 2025).
[18] Zimmerman, E. (n.d.). *MFTECmd*. GitHub repository. Available at https://github.com/EricZimmerman/MFTECmd (Accessed December 15, 2025).
[19] Zimmerman, E. (n.d.). *PECmd*. GitHub repository. Available at https://github.com/EricZimmerman/PECmd (Accessed December 15, 2025).
[20] Zimmerman, E. (n.d.). *RBCmd*. GitHub repository. Available at https://github.com/EricZimmerman/RBCmd (Accessed December 15, 2025).
[21] Zimmerman, E. (n.d.). *RECmd*. GitHub repository. Available at https://github.com/EricZimmerman/RECmd (Accessed December 15, 2025).
[22] Zimmerman, E. (n.d.). *SBECmd*. Available at https://download.ericzimmermanstools.com/SBECmd.zip (Accessed December 15, 2025).
[23] Zimmerman, E. (n.d.). *WxTCmd*. GitHub repository. Available at https://github.com/EricZimmerman/WxTCmd (Accessed December 15, 2025).
[24] Zimmerman, E. (n.d.). *SrumECmd*. GitHub repository. Available at https://github.com/EricZimmerman/Srum (Accessed December 15, 2025).
[25] Plaso Team. (n.d.). *Plaso documentation*. Available at https://plaso.readthedocs.io/en/latest/ (Accessed December 15, 2025).