Provenance Graph:

Convert a system log to provenance graphs

Under real-world conditions, user endpoints host a variety of applications, where each application typically produces hundreds of behavior events every minute. Similar to previous works [17], [23], [30], [24], we use provenance graphs as an intermediate data representation. EAGLEEYE builds prove nance graphs by creating a node for each behavior event and edges for capturing the parent-child relationships between the different events. We represent provenance graphs as follows: we store all behavior information in nodes, and edges do not contain any information. Nodes contain information about both the action taken (e.g. read, write, execute) as well as the involved system entity (e.g. file name, path, file access mode, etc). A provenance graph's head node represents the bootstrap ping process. All its child nodes are actions taken directly by this process, including the creation of new processes. Child processes, in turn, will point to new nodes containing actions initiated by them. Clearly, the provenance graph thus created is a DAG (directed acyclic graph). EDR solutions gather various details for each behavior event, beyond just process names and call hierarchies. We use only a subset of all available information, which we store as raw features in nodes of the corresponding graph. Such raw features can be simple in nature, like file names and process names, which are also used in other works [23]. We also store other particulars like file access flags and command-line flags. Every operating system has its own bootstrapping pro cesses, which are involved during startup, and later used for orchestration of compute loads. On Windows OS, user processes are started directly or indirectly by a handful of boot strapping processes, like winlogon.exe, wininit.exe, and logonui.exe. Since different applications like note pad.exe and outlook.exe should be analyzed separately, we do not include the bootstrapping processes as a common ancestor, and instead create disjoint provenance graphs for the user applications.

To construct provenance graphs from a system log, we select a random behavior event, and iteratively move up the Processes/threads Miscellaneous Suspicious Process start Socket Process termination File access Thread creation Stack pivot Memory protection File permission Hook injection Thread suspension Thread resumption Registry key RPC call Process hollowing Token manipulation parent-child hierarchy until we reach a native OS process. This corresponding child node (with a native OS process as parent) serves as the head of the provenance graph. The head node will only have outgoing edges and no incoming ones. We now perform breadth-first search from the graph head. For each process, we add all child processes as child nodes to the graph. Moreover, for each process, we search for all initiated kernel calls, like file accesses and network connections, and add these events as child nodes to the corresponding process node. The resulting provenance graph contains a subset of all behavior events from the system log, as there are multiple disjoint applications running concurrently on a given system. Next, we pick another behavior event which did not get added to the previous set of provenance graphs, and proceed to create another provenance graph in the same manner. We continue building provenance graphs until each behavior event is either represented in one of the graphs, or was initiated from the operating system and is thus ignored. Thanks to this grouping of data into different graphs, we achieve separation of data, which can then be analyzed independently. Each graph gives an account of where the application interactions originate from. In the example of the Raccoon Infostealer (Figure 1), the provenance graph explains where the malicious

persistence RealtekSb.lnk comes from, and thus the head process VVV.exe from the graph can be flagged as suspicious. Our data processing pipeline has several built-in measures to control the size of the resulting graphs. First, we do not include processes from the operating system. Next, we do not include all existing behavior event types, but only the most relevant ones from a security perspective. Table I gives a list of behavior events captured by most EDR systems; however we limit EAGLEEYE to only use four event types (see Table VII in the Appendix). Finally, we set a maximum duration for each provenance graph, and start a new graph after the timer expires. While a longer duration helps to capture more dependencies and relationships among the different events, the resulting larger graph also creates computational and storage challenges. Furthermore, the goal of EAGLEEYE is to detect malicious software as early as possible, before any harm is inflicted on the endpoint.

TABLE I: Behavior event types collected by EDR solutions

Processes/threads	**Miscellaneous**	**Suspicious**
Process start	Socket	Stack pivot
Process termination	File access	Memory protection
Thread creation	File permission	Hook injection
Thread suspension	Registry key	Process hollowing
Thread resumption	RPC call	Token manipulation

Datasets:

TABLE II: Overview of datasets

REE-2023 dataset	**Benign**	**Malicious**
Graphs	5,580	7,160
Events	6.14 million	23.1 million
Windows of 200 events	9,140	9,140
DARPA-5D dataset	**Benign**	**Malicious**
Graphs	721	91
Events	130,000	22,800
Windows of 200 events	2,370	2,520

1) Real-world Enterprise EDR (REE-2023) dataset:

Dataset link: Malware_dataset_7k_graphs_REE-2023.zip - Google Drive
This proprietary dataset contains data which we collected from an enterprise environment. To monitor the behavior of both benign and malicious applications, we leverage a commercial EDR tool, which uses ETW [42] and Sysinternals [43] to monitor kernel calls. The complete dataset is 15 GB in size and contains 29.2 million behavior events and 12,700 provenance graphs. The benign part of the dataset was gathered during four months in 2022 and 2023, from four different enterprise users located in three different countries in Asia and Europe. The dataset contains behavior from hundreds of applications running on Windows

OS, used by the users while carrying out their daily work. For the malicious part of the dataset, we deployed a safe sandbox environment to execute malware and collect the corresponding data, thus ensuring that no damage is inflicted by malware on real endpoints. All malware samples were downloaded from VirusTotal in 2022 and 2023, are in ex ecutable format for Windows, and were executed on a special CAPE sandbox. We worked together with security engineers to tune the CAPE sandbox, to make it hard for the malware to detect the sandbox environment. We performed sanity checks to see if the samples included malicious activities, and that they were not malfunctioning. Since the automatic start of malicious samples in the sandboxing environment always happens in the same way, we do not include the first process start in the provenance graph. Lest class-related artifacts are left in the data, we apply the same logic to benign provenance graphs. On average, the benign and malicious provenance graphs of the REE-2023 dataset have a size of 2,300 behavior events, and the graphs have an average depth of 3.5 nodes. For the REE-2023 dataset, we use four event types, namely: new process creations, file access operations, network connections, and Windows registry events. For the benefit of the research community, we release the malware dataset via our GitHub repository. However, the benign dataset is private and has sensitive user information; for ethical reasons, we do not release the benign dataset.

2) DARPA FIVEDIRECTIONS (DARPA-5D) dataset:

DatasetLink:
https://drive.usercontent.google.com/download?id=1Jz0ZuiZlUEZdAgqlnfmpN2_X0Cms6Sl8&export=download

The Transparent Computing (TC) program of DARPA released multiple public datasets [44] containing system logs from APT attacks (advanced persistent threat). The TC program held engagements number 3 and 5, in 2018 and 2019, respectively. Both engagements consisted of a red team trying to penetrate a target network and exfiltrate sensitive information. Since attacks were carried out manually, the malicious parts of both datasets are limited in size and variability, and we therefore choose to combine both datasets to form one larger dataset. We focus on the attacks tracked by the FIVEDIRECTION team; they targeted hosts running Windows 10. The red team used a variety of attack vectors, including Firefox backdoors and phishing emails with malicious MS Office macros. Labeling of the provenance graphs (as malicious and benign) requires manual work, as the public version of the DARPA TC dataset has no labels attached to behavior events. We use the same la beled dataset as ProvNinja [45], which includes the following behavior events: file creations, process creations, and network socket creations.

E DARPA-5D DATASET

The DARPA TC program released system log tracing data for engagements number 3 and 5. During engagement 3, attackers used Firefox to download and execute a backdoor named DRAKON as part of the APT group's toolset; moreover attackers used phishing emails with malicious MS Office macros to install malware beacons. For engagement 5, attackers used a Firefox backdoor to download and execute the payload DRAKON and MICRO backdoor.

Since the official DARPA dataset is not labeled at the event level, we describe here how we performed labeling, preprocessing of the data, and provenance graph creation. We thank the authors of ProvNinja [45] for providing us with a labeled version of the DARPA TC-3 and TC-5 dataset. For creation of labeled graphs, the publicly available red team reports are leveraged. With help of the reports, malicious system entities used during an attack are located, and the entities are used as starting points to grow malicious provenance graphs. The growing of the provenance graphs is done via breadth-first search with a maximal depth of 8. Malicious system entities can include malicious URLs, sensitive files, abused living-off-the-land binaries, and dropped binaries.

Benign behavior happens concurrently with attacks. Such behavior includes normal web browsing, use of Office applications to create documents, and reading of emails. Note that benign and malicious provenance graphs can overlap in time. Once malicious graphs are created, all remaining unused behavior events are leveraged for the creation of benign graphs. Attacks in the TC program start with either Firefox or Microsoft Excel. Both applications also get used regularly for benign activity. To make the malware detection task as realistic as possible, both benign and malicious graphs contain at least one Firefox application; additionally, most graphs also contain a Microsoft Excel application.

Note, our labeled version of the dataset might differ slightly from other works, as separating benign from malicious activity is a manual process and entails various design decisions. In particular, our dataset consists of *provenance graphs*, where each node has exactly one incoming edge, namely from its parent process.