Baltic J. Modern Computing, Vol. 6 (2018), No. 3, 217-234
https://doi.org/10.22364/bjmc.2018.6.3.01

Towards a Robust Method of Dataset Generation of Malicious Activity for Anomaly-Based HIDS Training and Presentation of AWSCTD Dataset

Dainius ČEPONIS, Nikolaj GORANIN

Vilnius Gediminas Technical University, Vilnius, Lithuania

dainius.ceponis@vgtu.lt, nikolaj.goranin@vgtu.lt

Abstract. Classical signature-based attack detection methods demonstrate stagnation and inability to fight the zero-day and similar attacks, while anomaly-based detection methods are still characterized by huge numbers of false-positives. The progress achieved in recent years in the area of deep learning techniques provide a potential for renewing investigations on anomaly-based intrusion detection system training. While network-based intrusion detection systems have datasets for training, host-based intrusion detection systems researchers lack this component. Most datasets are created for Linux OS and the latest Windows OS dataset was introduced in 2013 and included only minimal collection of system calls' features. In this article we propose a method for automated system-level anomaly dataset generation that is to be used in further artificial intelligence-based host-based intrusion detection systems training as well as our generated exhaustive collection of Windows OS malware-based system calls, that also includes additional information on malware activity. Main characteristics of the dataset are presented.

Keywords: intrusion detection, system calls, HIDS, dataset, anomaly-based

1. Introduction

By definition intrusion detection systems (IDS) (Bace and Mell, 2001) is: "the process of monitoring the events occurring in a Computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network". Generic network intrusion detection system is shown on Fig. 1. Main components are: anomaly detection engine, alarm module, human analyst and security manager.

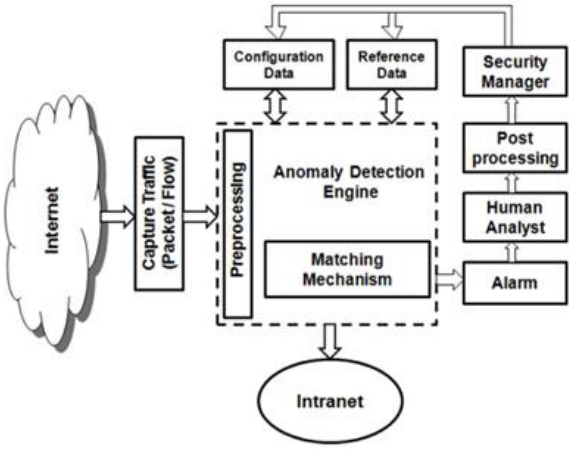

Fig. 1. Generic intrusion detection system (Bhattacharyya and Kalita, 2013)

IDS can be classified into two types (Garcia-Teodoro et al., 2009).

- Network-based intrusion detection system (NIDS) is designed to detect the intrusion before it happens by analyzing computer network traffic (Hay et al., 2008). On suspicious network activity –security personal is notified about possible attempt to commit intrusion. Those systems can also be not only intrusion detection, but in the same time had an intrusion prevention module. Despite the notification, they also try to prevent intrusion – discard resources or take other actions. That system is mainly installed on one network point where all network traffic is visible and can be easily controlled. Observing agent is installed on that point and connected to main server.
- Host-based intrusion detection system (HIDS) is located on end-point user machine and monitors user and host operating system behavior. HIDS can provide the following functions: file integrity checking, registry monitoring, rootkit detection, policy monitoring, log analyzing and system calls analysis (Hay et al., 2008). File integrity monitoring (FIM) basically deploys simple alteration detection on sensitive system files by collecting cryptographic hash values of critical files. Later HIDS checks if that hash value is changed. If the result is a positive one – an alert to system administrator is raised (Hay et al., 2008). Registry monitoring also provides valuable information about user actions executed on system. HIDS can detect rootkits conducting signature-based scans or finding anomalies in the results of different system calls (Bace and Mell, 2001). Additional layer is inserted between operating system and applications to monitor system calls. HIDS can analyze system calls sequences and look for suspicious events. It can collect all sort off information: active applications, memory and CPU usage, outgoing and ingoing internet traffic. All that information, in real time, is transferred to the main server. Main server analyses information from the hosts and decides whether to notify security staff on suspicious behavior on the host machine. Suspicious activity can be: abnormal CPU and RAM usage or text editing application attempt to modify system password file. File integrity is also monitored besides activity on the host machine i.e. HIDS can be seen as an agent which monitors system and checks if any other agent violates security policy.

The first intrusion detection model was introduced in 1987 (Denning, 1987). Three intrusion detection methods have evolved since that time (Buczak and Guven, 2016):

- Misuse or Signature-based
- Anomaly-based
- Hybrid

Signature-based detection is designed to detect known attacks and has a small number of false-positives. It scans for patterns associated with known attacks against computer system. Those patterns can be: hardware-related parameters collection (CPU and RAM usage), cryptographic hash value of rootkit or error log generated by an attack (Hay et al., 2008). A regular database update on attack pattern is necessary to have a fully functional system. This method is not effective against new (zero-day) attacks until they are added to the database (Xie and Hu, 2013).

The idea of anomaly-based intrusion detection is predicated on a belief that an intruder's behavior is noticeably different from that of a legitimate user and that many unauthorized actions are detectable. This type of intrusion detection should be effective against zero-day attacks. Another advantage for this type is that a – detection algorithm can be tailored for a specific company, network or a user, making it challenging task for the attacker to select effective and non-detectable intrusion actions. Numerous machine learning methods of clustering and classification were applied for anomaly detection (Agrawal and Agrawal, 2015). The main disadvantage of this type of intrusion detection are the high false positives rates. Large sets of training data are required to construct normal behavior profile (Aydin et al., 2009). It is also possible that malicious activity is included into the legitimate activity training data – in that case the intrusive activity will be legit in later detection process (Tan et al., 2002). Due to this reason it is extremely important to ensure creation of "sterile" datasets that would separate legitimate and malicious actions.

Many IDS systems usually make use of a hybrid method which combines signature and anomaly-based techniques. Such combination provides small amount of false positives for unknown attacks and raises detection rate on known intrusions (Depren et al., 2005).

So far NIDS systems are dominating the field. However, HIDS systems are receiving more attention due to the fact that they provide more information about intrusion and can prevent from significant damage (e.g., the alteration of important system files) as well as offering an additional layer of security.

While NIDS have sufficient amount off open data for training, HIDS researchers lack this important ingredient, since most of datasets are created for Linux OS and the latest publicly available Windows OS dataset was introduced in 2013 and only included minimal collection of system calls features, although systems call with additional information can provide valuable information on suspicious process behavior evaluation.

A system call is typically a function in the kernel that services I/O requests from users; it is implemented in the kernel because only a high-privilege code can manage such resources (Dang et al., 2014). Linux system calls are well documented and their description can be easily found and system calls traces can be easily collected by invoking "strace" followed by the program and its command-line arguments (Mitchell and Oldham, 2001). These are the main reasons why Linux system calls are more widely used than ones based on Windows OS, despite the majority of attacks and malware being Windows-oriented. Unfortunately, descriptions of Windows OS system calls are very limited and a full list of them for x86 and x64 can be found in (Jurczyk, n.d.). Microsoft MSDN only provides information about a system call if you know its exact name. No defined special list for them is available (Dang et al., 2014). Plus, system call traces collections on Windows can only be executed with third party applications (Canzanese et

al., 2015). Still, a method for system calls monitoring layer to kernel mode with minimal impact on system performance was introduced earlier (Battistoni et al., 2004).

Therefore, we propose a robust method of malicious activity generation on a Windows OS platform. We also present our generated new collection of Windows OS malware-based system calls that can be used for training anomaly-based HIDS systems. Main characteristics such as structure, composition, distribution of malware used for system call generation of the dataset are discussed. The generated dataset provides possibility for better HIDS training, since as well as offering a more in-depth view on malware used system calls, it also provides information on other malware performed actions, like files modified.

The paper contains six main sections: introduction, related work on datasets generated to train anomaly-based IDS systems, presentation of the proposed method for new dataset generation, presentation of the attack-caused Windows OS system calls traces dataset and conclusions.

2. Related Work

Signature-based intrusion detection is showing better results on detecting known attacks, but it fails to report new and unknown attacks. For that reason anomaly-based intrusion detection methods are getting more attention (Azad and Jha, 2013; Hu, 2010) – more than 60% research papers are focused on anomaly detection. However, despite of significant progress in anomaly-based intrusion detection methods, they still show higher false-positive detection rate than signature-based methods. The progress achieved in recent years in the sphere of deep-learning artificial intelligence techniques provide a potential for renewing the research on the topic specified.

A key factor in machine learning, which forms a basis for anomaly detection algorithms, is the quality of data. Most of the recent research on intrusion detection has been done using DARPA and KDD Cup 99 datasets. So far 42 % KDD cup dataset, 20 % DARPA dataset and 38 % other datasets have been used to verify proposed new methods for anomaly detection (Azad and Jha, 2013). KDD Cup 99 dataset was collected in 1999 by processing the tcpdump portions of the 1998 DARPA Intrusion Detection System (IDS) Evaluation dataset, created by Lincoln Lab under contract to DARPA (Brugger, 2007; Lippmann et al., 1999). Those datasets contain various information collected on simulating attacks against a network. Four main attack types have been used against a simulated US Air Force LAN (Mukkamala et al., 2005):

- Probing. Probing is a class of attacks where an attacker scans a network to gather information or find known vulnerabilities. Attacker can find related exploits if network machines map with corresponding services is available.
- Denial of service attacks. DoS is a class of attacks where an attacker makes some computing or memory resource too busy or too full to handle legitimate requests, thus denying legitimate users access to a machine.
- User to Root attacks. Attacker starts on local normal user account, and after some commands and related exploits usage – gains root user account control.
- Remote to User attacks. On this type off attack, attacker sends commands to the target machine (one uses already known exploits for that machine) and illegally gain local access as a user.

Therefore, DARPA-related datasets have a data associated to a network and are perfect to apply in NIDS research (Sahu et al., 2014). It is necessary to stress that

DARPA, which has been used as a de facto standard for anomaly-based NIDS training, present the simulated and not the real attack data. Nevertheless, it is still considered by experts as a valuable dataset.

The research done in the sphere of anomaly dataset generation for HIDS training is minimal despite the fact of the growing need for anomaly-based HIDS systems.

Some information has been collected during the KDD dataset assembly. At first it was a UNIX-type host systems information. Later, in January 2000, Windows NT hosts data was collected on similar circumstances (Korba, 2000). It contains not only tcpdump provided data, but also the Windows NT event log audit data. Despite the provided collections, KDD Cup-related datasets lack host machines-related information and only NIDS researches use it (Haider et al., 2016).

Some attempts have been made to generate novel public datasets for the Windows operating system. Windows audit logs analysis method was introduced and collected data was prepared for public usage by (Berlin et al., 2015). Audit logs have been produced by running malware on a target machines. The proposed audit logs analysis method yields high detection rate. Still, audit logs have some disadvantages. One of them – it cannot detect thread injection, which is a main tool in malicious tactics (Berlin et al., 2015).

One of the latest datasets related to the host-based intrusion detection is the ADFA-IDS dataset. In an experiment a zero day attack was simulated and system calls in Windows and Linux operating systems have been collected (Haider et al., 2016). Two comprehensive Windows operating system-based datasets (ADFA-WD and ADFA-WD: SAA) were introduced for the research community. Prior to the Windows OS datasets, Linux related collection was introduced (Creech, 2014; Creech and Hu, 2013). Windows OS datasets contains core dynamic link library (dll) name and called function address. Linux dataset contains sequence of numbers. Those numbers are representing a corresponding system call. An example of collected Windows OS system calls and their extracted feature vector can be seen in Fig. 2.

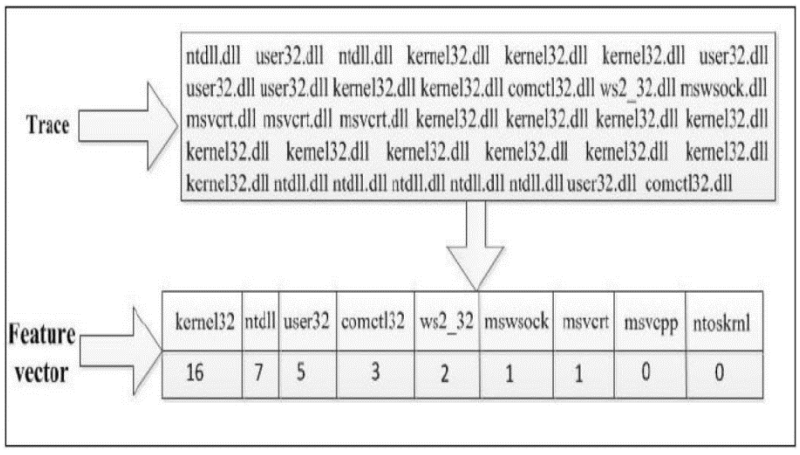

Fig. 2. Windows OS system calls and their feature vector (Haider et al., 2016)

However, ADFA family datasets have just a minimal data required for intrusion detection since it contains only system call identification: system dynamic link library

(dll) file name and the called function name. Twelve known vulnerabilities were analyzed for ADFA-WD dataset and three stealth attacks (Doppelganger, Chimera, and Chameleon) for ADFA-WD: SAA (Haider et al., 2016). As can be seen in Fig. 3, 6636 malicious system call traces were collected in total:

Data Sets	Normal Training	Normal Validation	Attack
ADFA-WD	356 traces	1828 traces	5773 traces
ADFA-WD:SAA	same as above		863 traces

Fig. 3. System Calls traces results (Haider et al., 2016)

But even authors of (Haider et al., 2016) agree, that ADFA Windows datasets are incomplete. Only basic information was collected and insufficient amount of vulnerabilities was used to generate malicious activity.

3. The Proposed Robust Dataset of Malicious Activity Generation Method

3.1. Method description

The following nonfunctional requirements were formulated for the malicious activity dataset generation method: the system has to be flexible (it must allow adapting new data collection in the future), easy to configure (no special tools must be required to change system parameters), and based on open-source software only. The target operating system for malicious activity collection chosen – was Windows, because it is still the most widely used operating system in the world, although the method can be easily adapted for any other OS.

For reasons of simplicity and proof of concept, only openly available malware samples were used to generate malicious activity samples. The method can be easily automated: any malware samples can be downloaded, prepared according to the requirements, and used. Malware samples contain items for all well-known operating systems, so malicious activity sets are generated for any operating system.

The proposed method (Fig. 4) has six following steps:

1. **Malware samples preparation.** At first, malware must be obtained from available sources. Later, malware of Windows OS executables type should be extracted and added to a separate collection for use.
2. **Host machine preparation.** Hypervisor must be installed and configured on the selected host machine. Malware samples must be copied to the host machine for later execution.
3. **Guest machines preparation.** Template for a virtual guest machine must be added and configured on the host machine. Later, the required number of guest machines (copies) should be created for malware execution. Execution of malware samples can be performed in parallel and is dependent on the number of guest machines available.

4. **Data collection server preparation.** Server storage for the malware execution log information (such as anomaly samples in form of logs, network activity, system calls, etc.) must be prepared.
5. **Malware samples execution and data collection.** When all machines are prepared – main execution script is started. The main script will upload a malware sample to the target guest machine and will start the operating system. Later, a script on the guest machine will execute malware on OS startup. Malware-generated activity log will be automatically collected and uploaded to the data collection server.
6. **Collected data preparation and analyzation.** When all samples are executed, collected data can be transformed to the XML format and analyzed.

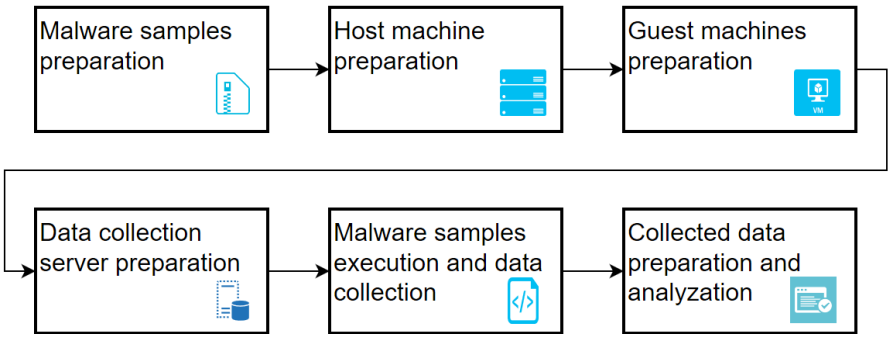

Fig. 4. Main steps of the proposed method of dataset generation.

3.2. Method implementation

The proposed method implementation (architecture) can be seen on

Fig. 5. The virtualization technique, based on a free ProxMox hypervisor, was selected to simulate quest machines that will be used for running malicious actions. ProxMox VE is a completely open-source platform for enterprise virtualization, a built-in web interface that allows management of VMs and containers, software-defined storage and networking, high-availability clustering, and multiple out-of-the-box tools in a single solution (Kovari and Dukan, 2012). ProxMox is running QEMU - a generic and open source machine emulator and virtualizer and is based on Debian operating system. According to the results of the latest research, QEMU has a less detectable virtualization through basic detection techniques (Miller et al., 2017), which maximizes the malware execution rate.

A main bash script is executing all commands required to collect data: a guest machine is prepared, started and stopped by that script. The main bash script has only one parameter – a folder that contains prepared malware samples. ProxMox firewall is enabled on the Host machine to manage network flow and minimize the risk of malware propagation. Only one-directional flow to the remote HIDS server was allowed – all other connections were blocked. All data sent to that server was stored on LOG server for later analysis.

An anomaly data collection was done by three tools: Dr. Memory provided system call tracer for the Windows OS, OSSEC (open source HIDS (Timofte, 2008)) for file

integrity monitoring and WinDump for the network-related information. Dr. Memory tool provides not only system call name, but also passed parameters list and return value. All that information can be used to detect earlier mentioned thread injection, which is missing in method provided by (Berlin et al., 2015).

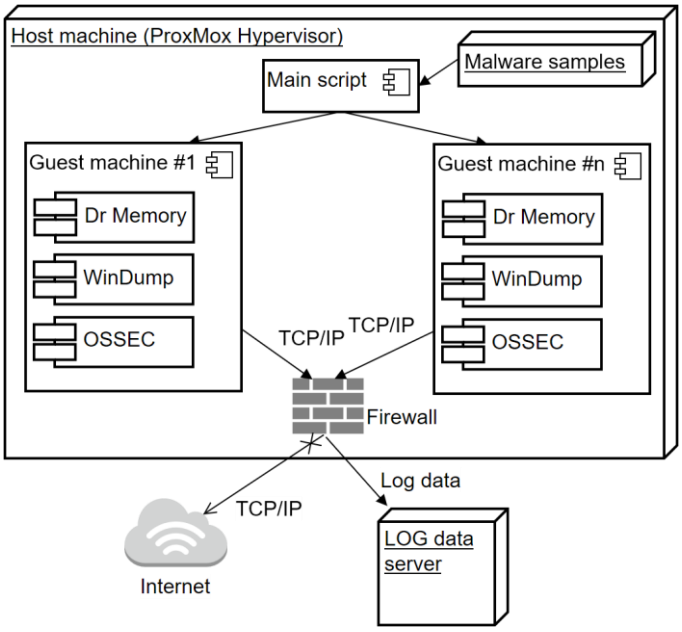

Fig. 5. Malware execution components scheme

Open malware collections were used to generate malignant activity on guest machines. Malware execution was conducted on a Windows operating system. For simplicity reasons, during the first step, only malware of executable type was used, in order to minimize dependence on third party applications (e.g. office suites, utilities, viewers or any other). Malware samples were taken from the freely available database provided by VirusShare (WEB, c) (For this paper, VirusShare_00289.zip package, created on 2017-05-07, was used) and theZoo (WEB, b). VirusShare provides malware packages in a form of password-protected zip files with the usual 'infected' password or any other file types. As a result, every package file type must be analyzed, because there is no file extension provided. Every package can contain various types of malicious files that can target different operating systems: Linux, Windows, Mac, Android and iOS. For that reason, each package must be analyzed and only Windows OS-executable malicious samples have been selected in our case. VirusShare samples were combined with theZoo malware collection, that holds most popular and controversial malware samples. theZoo database already contains password protected archives with executable files. For that reason, no special preparation procedure was required.

Malware sample preparation is presented on Fig6.

Usually the first byte of a file is holding information about the file type. If that is already a Windows-executable file – corresponding file extension is added to it and the file is packed to the archive with a password “infected”. If the analyzed file is an archive

– it is extracted for further analysis and, if executable files are found, they are added to the password protected archive. All other files are skipped.

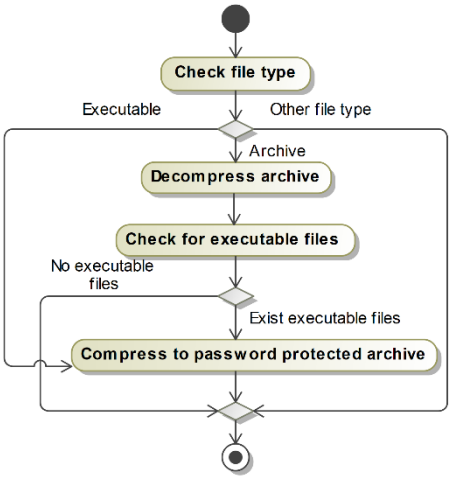

Fig. 6. Malware file preparation

3.3. The Process of Malware Loading to the Guest Machine

Malware transfer to the guest machine was implemented with the help of ProxMox VE, which provided the capability to attach an additional virtual drive and copy the file straight to it. It is not dependent on any other third-party software and firewall configuration has no impact on file transfer.

The number of malware samples that can be executed in parallel, thus influencing the dataset generation speed, depends on the number of running guest machines, that is directly related to the available hardware resources.

For our experiments tests were performed on the HP ProLiant DL 380 G6 server with the following specifications: 2x Xeon E5520 CPU, 8 GB of RAM and 4x146 GB HDD's connected to RAID 5. Six guest machines were running in parallel.

A bash script on the host machine was used to control guest machine's state (startup and shutdown) and malicious file transfer to the corresponding virtual drive. Virtual drive preparation for the guest machine also was implemented via bash script. To ensure such method on the newest ProxMox VE – a thin provisioning must be turned off. After that, every guest machine drive is represented on a hypervisor as a simple local file. Most importantly, it can be mounted on a hypervisor system and updated with required malware file. Main actions performed by the bash script on the host machine are:

1. Copying guest machines disks from prepared templates.
2. Mounting virtual disk for every guest machine, copying prepared malware, unmounting disk.
3. Starting the guest machine.
4. Pausing script for defined time to provide the malware the possibility to reveal all functionality and features. The default pause time in tests was equal to 30 minutes but can be optimized for generating sets of specific malware types (e.g. longer for botnet and shorter for worm).

5. Stopping the guest machines. The Stop command will halt the machine immediately. Shutdown process is not initiated.

Basically, all steps combine simple commands: copy, machine start and machine stop. However, step number 2 requires more sophisticated knowledge of virtual disk handling commands.

Guest machine images were also prepared. Each guest machine was running Windows 7 OS and Dr. Memory, OSSEC agent and WinDump. A malware execution script was added to the Windows task scheduler. Task scheduler provides all required privileges for an unimpeded application/malware startup. Then defined archive file is extracted, malware is executed by a run command for every executable in the extracted folder. The anomaly data gathered (list of modified/accessed files, system calls with related information and network data) was sent to the LOG server for analysis.

All actions required for implementing malware samples execution are presented on Fig.7.

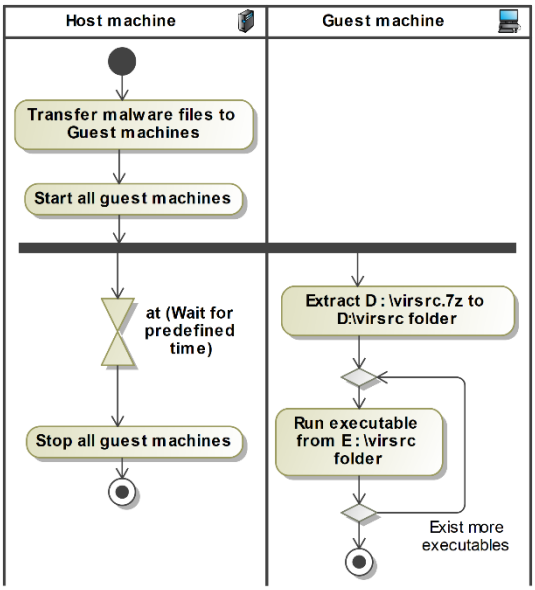

Fig. 7. Activity diagram of single malware sample batch execution

Malware samples are executed in a batch manner. Every batch has a number of files identical to the number of available guest machines. It can be seen, that host machine waits for the predefined time while a script on guest machine is executing the provided malware sample. This pause is needed to collect anomaly activities in case malware manifests itself after some delay after infecting the machine.

4. Attack-Caused Windows OS System Calls Traces Dataset

While developing the dataset (further referred to as Attack-Caused Windows System Calls Traces Dataset or AWSCTD) the following three main objectives were targeted for the dataset generation process:

1. Malware/attack tools have to be publicly available in order to assure easy renewal of dataset in future database renewal and independent verification of results.
2. The dataset should contain the following information:
    a. All possible information about the system call (function name, passed arguments list and values of them, return value).
    b. List off changed files affected by malware attack/tools.
    c. Network traffic generated by malware attack/tools.
3. Dataset should be based on a relatively high number of system calls, generated by a wide selection of malware/attack tools.

4.1. Dataset Size and Structure

A total of 12110 executable malware samples in a form of password protected archives were prepared from VirusShare provided packages and used for dataset generation. All samples were tested in a two months period (2017.07-2017.08) and 89.34% (10820) of them were executed successfully, i.e. the selected malware sample has infected the custom-made test system with Windows OS running on it. Further verification has revealed that not all malware samples acquired could be considered to be 100% malicious. Some of them did not have a proper amount of positive detection rate reported on VirusTotal.com site (WEB, d). Because of that, only samples with 15 and more positive detections were selected for the dataset.

Table 1. Most common malware categories and families used in dataset generation

Category	Count	Family	Count
AdWare	5139	AdWare.Win32	4655
Trojan	2353	Trojan.Win32	2326
Downloader	853	Downloader.Win32	830
WebToolbar	659	WebToolbar.Win32	654
DangerousObject	137	AdWare.MSIL	412
Trojan-Ransom	101	DangerousObject.Multi	137
Backdoor	79	Trojan-Ransom.Win32	96
RiskTool	55	Backdoor.Win32	75
Trojan-Downloader	45	AdWare.NSIS	71
Trojan-Spy	37	RiskTool.Win32	46
Packed	34	Trojan-Downloader.Win32	44
Virus	17	Packed.Win32	34
Trojan-PSW	16	Trojan-Spy.Win32	34
Trojan-Dropper	15	Downloader.NSIS	19
Trojan-Clicker	5	Trojan.MSIL	18

Distribution of selected malware is presented in Table 1 (“DangerousObject” category according to Kaspersky: Malicious software is detected by KL Cloud Technologies. This verdict used for samples that were not classified exactly.). Malware category and family information is based on VirusTotal.com classification. Only malware detected by at least 15 antivirus vendors was analyzed further – this rule has allowed us to select 10276 malware samples from 10820 tested.

Collection of system calls traces was performed with the help of *drstrace* tool, developed by Dr. Memory (WEB, a). It allowed to gather all required system calls

information: system call name, passed parameters information (parameters count and values), return values and execution result (success or false). A sample system call is presented on Fig. 8.

```
NtQueryValueKey
    arg 0: 0x35a (type=HANDLE, size=0x4)
    arg 1: 26/28 "DescriptionID" (type=UNICODE_STRING*, size=0x4)
    arg 2: 0x2 (type=int, size=0x4)
    arg 3: 0x02c5daa0 (type=<struct>*, size=0x4)
    arg 4: 0x90 (type=unsigned int, size=0x4)
    arg 5: 0x02c5da7c (type=unsigned int*, size=0x4)
    failed (error=0xc0000034) =>
    arg 3: <NYI> (type=<struct>*, size=0x4)
    arg 5: 0x02c5da7c => 0x0 (type=unsigned int*, size=0x4)
    retval: 0xc0000034 (type=NTSTATUS, size=0x4)
```

Fig. 8. System Call sample from drstrace tool

```
for (every line in log file){
    if (line[0] == '\t') {
        if (line[1] == 'a') {
            //Processing a argument line
        }
        else {//'result'
            //Processing a result line
        }
    }
    else if (line[0] == ' ') {
        //If previuos line was result
        //then get succeded parameter value
    }
    else {
        //We are processing new system call entry
    }

    if (New system call detected){
        //Write current system call to database.
        nCallNumber++;
        if (nCallNumber > 10000) {//Only first 10000 traces are transfered
            break;
        }
    }

    if (System call name) {
        //Extract system call name
    }

    if (Argument line){
        //Extract argument info
    }

    if (Result argument line){
        //Extract result argument info
    }

    if (None flags has been assigned){
        //Extract system call return value
    }
}
```

Fig. 9. Part of code of system calls log transfer to database

All system calls were recorded in that format to the log files. A special logs conversion into a better defined formats (JSON and SQLITE database) was implemented

in separate C++ application. A part of code, presenting logs conversion to SQLITE database is shown on Fig. 9.

A total of 112.56 million system calls traces for generated by 10276 malware samples were recorded to the database. Such amount of data had a massive impact on the database size – the generated SQLITE file consumes 39.1 GB of storage space. Database contains not only system calls, but also metadata about the malware (database scheme can be seen on

Fig10).

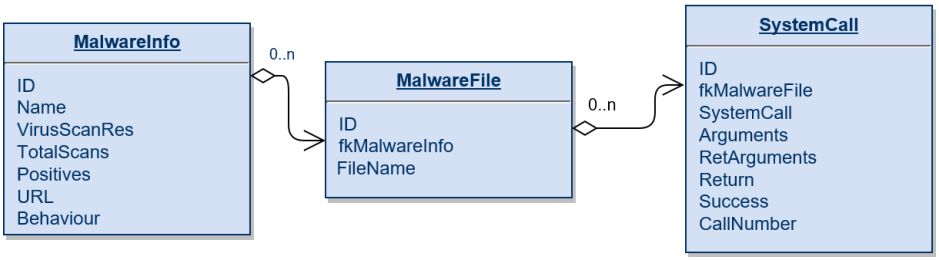

Fig. 10. Generated malware activity database scheme.

Part of that information (Malware info) was imported with the help of Academic API provided by Virus Total. The information for every malware record has included:

- Scan engines (antivirus applications) that provided information about the detected threat: e.g. malware type;
- Positive scan results value;
- Web page to malware description page;
- Malware behavior information:
  - File system action;
  - Network communication;
  - Loaded modules (dll files)

A sample of exported record from the database is provided below in CSV format with | symbol as fields separator (first row – fields names, second – values):

Table “Malware info”:

ID|Name|VirusShare|VirusScanRes|TotalScans|Positives|URL|Behaviour

2|000600ee5aedc6e5d4ca946b99f3c924|"{""Kaspersky"":{""detected"":true,""version"":""15.0.1.13"",""result"":""AdWare.Win32.MultiPlug.nnnn"",""update"":""20171024""}|67|54|https://www.virustotal.com/file/6abbf5200f267e482b363c4634db9b7213c746ef03cae20ff65da7b8c14d0866/analysis/1508868909/|{Virus Behavior information in JSON format}

Table “MalwareFile”:

ID|fkMalwareIndo|FileName

1|2|drstrace.VirusShare_000600ee5aedc6e5d4ca946b99f3c924.exe.02320.0000

Table “SystemCalls”:

ID|fkMalwareFile|SystemCall|Arguments|RetArguments|Return|Success|CallNumber

1|1|NtQueryPerformanceCounter|"[""0x002ff8f4 (type=LARGE_INTEGER, size=0x4)"","0x002ff8d8 (type=LARGE_INTEGER*, size=0x4)""]"|"[""0x1f4690a2 (type=LARGE_INTEGER*, size=0x4)"","0x989680 (type=LARGE_INTEGER*, size=0x4)""]"|0x0 (type=NTSTATUS, size=0x4)|1|1*

As it can be seen from Table 2, AWSCTD exceeds the only currently available dataset by the number of tested malware and generated system calls count by order.

Table 2. ADFA-IDS dataset comparison to our dataset.

Dataset	Executed malware samples	Collected System Calls count
ADFA-IDS	15	6636
AWSCTD	10276	112.56 million

It is also important to note, that AWSCTD includes additional information missing in ADFA-IDS. Of course, the researcher is free to choose only those dataset parameters that are relevant to his specific task.

4.2. Dataset Characteristics

According to the (Miao et al., 2006) there are about 949 native calls (284 key APIs from Ntdll.dll and 665 less important from Ntoskrnl.dll) in the already discontinued Windows XP operating system. In our tests performed on the basis of Windows 7 OS, 645 distinct system calls were captured. The most commonly used system calls are presented on Fig11. The dominating part of calls generated by malware were related to registry querying. The next dominating group of calls was implementing the file processing functions (reading and writing). The system calls success rate parameter obtained was very high – 99% of all executed calls have returned the desired result. Dr. Memory was used to evaluate the success rate. The calls with the highest success rate were NtQueryValueKey and NtOpenKeyEx. The lowest rate was demonstrated by NtYieldExecution (this function stops execution of thread calling and switches to any other currently running thread) and NtCallbackReturn (this function finishes execution of User-Mode callback).

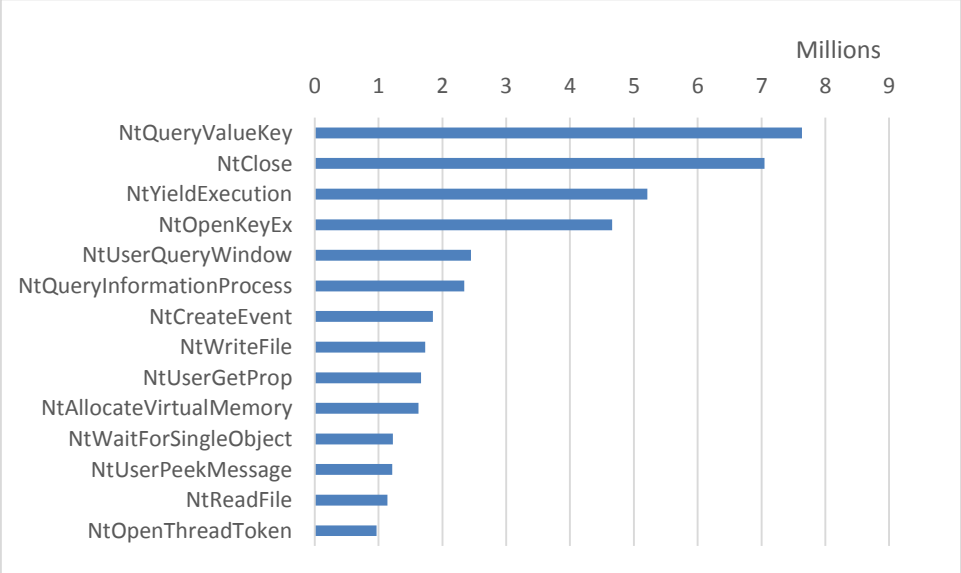

Fig. 11. Most frequently requested system calls.

As stated earlier, some characteristics of malware behavior (mainly related to file access) was obtained from Virus Total. The most frequently requested files are listed in Table 3. In this category, not only critical Windows system files can be noticed, but also various services: such as driver responsible for maintaining persistent drive letters and volume names (MountPointManager), or remote procedure calling (lsarpc) used by the applications based on the client-server architecture.

Table 3. The most frequently requested files.

File path	Count
\\.\PIPE\lsarpc	1080
C:\WINDOWS\Registration\R000000000007.clb	766
\\.\Ip	681
\\.\MountPointManager	652
c:\autoexec.bat	611
C:\WINDOWS\system32\shdocvw.dll	326
C:\WINDOWS\system32\msi.dll	248
C:\WINDOWS\system32\stdole2.tlb	221
C:\WINDOWS\WindowsShell.manifest	218
C:\Program Files\Internet Explorer\iexplore.exe	210

AWSCTD dataset provides additional system call information: a list of parameters, which were not included in previous publicly available datasets. These parameters can provide additional information on malware behavior while performing ML methods training. A sample parameters combinations, together with statistics of the NtCreateFile system call, are provided in Table 4 (parameter CreateOptions) and Table 5 (parameter ShareAccess).

Table 4 NtCreateFile parameter "CreateOptions" most frequently used combinations.

Parameter combinations	Count
FILE_SEQUENTIAL_ONLY|FILE_NON_DIRECTORY_FILE	8434
FILE_SEQUENTIAL_ONLY|FILE_NON_DIRECTORY_FILE|FILE_OPEN _REPARSE_POINT	8431
FILE_NON_DIRECTORY_FILE|FILE_OPEN_NO_RECALL	374
FILE_SYNCHRONOUS_IO_NONALERT|FILE_NON_DIRECTORY_FILE |FILE_OPEN_NO_RECALL	221
FILE_SYNCHRONOUS_IO_NONALERT	129
FILE_SEQUENTIAL_ONLY|FILE_SYNCHRONOUS_IO_NONALERT| FILE_NON_DIRECTORY_FILE|FILE_OPEN_NO_RECALL	10
FILE_OPEN_FOR_BACKUP_INTENT	6
FILE_SEQUENTIAL_ONLY|FILE_SYNCHRONOUS_IO_NONALERT	4
FILE_SEQUENTIAL_ONLY|FILE_SYNCHRONOUS_IO_NONALERT| FILE_OPEN_FOR_BACKUP_INTENT	1
FILE_WRITE_THROUGH|FILE_SYNCHRONOUS_IO_NONALERT| FILE_NON_DIRECTORY_FILE	1

Table 5. NtCreateFile parameter "ShareAccess" most frequently used combinations.

Parameter combinations	Count
FILE_SHARE_READ	340030
FILE_SHARE_READ|FILE_SHARE_WRITE	233424
FILE_SHARE_READ|FILE_SHARE_WRITE|FILE_SHARE_DELETE	92609
FILE_SHARE_READ|FILE_SHARE_DELETE	85237
0x0	27297
FILE_SHARE_WRITE	914
0xffffff90	3

The parameter combinations used with system calls usually vary for legal applications and malware, therefore this information can be used as one of distinguishing characteristics of malware behavior. The dataset includes parameter combinations in the same form as presented for all system calls included in the dataset.

5. Conclusions

- The performed analysis has shown that there is an increasing requirement for the development and training of anomaly-based HIDS solutions, which is currently being slowed down due to the lack of available and suitable host-level anomaly datasets.
- The method for host-level anomaly dataset generation was proposed. The proposed method is based on malware execution in a sterile, isolated virtual machine environment with further anomaly activity collection and data representation in an SQLite database format.
- The method was implemented and tested only with free or open-source tools and freely available malware samples. The tests performed have proved the method stability and method suitability for host-level anomaly dataset generation. Automated anomaly generation allows flexible training data-set expansion, response to the new attack types and generation of specific on-demand datasets. No interruptions or errors related to the malware execution were noticed which is an advantage against well-known tool for such task - Cuckoo sandbox. According to Miller et al. – it has stability issues that cause Cuckoo samples results to be inconsistent between runs (Miller et al., 2017).
- AWSCTD was generated for 10276 malware samples. The dataset size exceeds the size of previously known datasets by order and includes much wider representation of malware types and system calls. AWSCTD has additional advantage against existing datasets because of parameters (system call arguments list and return value) that allow more in-depth HIDS training.

- An expansion of the generated dataset is being planned for creating a more comprehensive host-level anomalies dataset for HIDS training. The expansion is planned via inclusion of non-executive type malware samples, non-malware attacks and optimizing the pause interval for better feature assembly of delayed malware activities.

References

Agrawal, S., Agrawal, J. (2015). Survey on anomaly detection using data mining techniques, In *Procedia Computer Science*.

Aydin, M. A., Zaim, A. H., Ceylan, K. G. (2009). A hybrid intrusion detection system design for computer network security, *Computers and Electrical Engineering* 35(3): 517–526. Retrieved from http://dx.doi.org/10.1016/j.compeleceng.2008.12.005

Azad, C., Jha, V. K. (2013). Data Mining in Intrusion Detection: A Comparative Study of Methods, Types and Data Sets, *International Journal of Information Technology and Computer Science* 5(8): 75–90. Retrieved from http://www.mecs-press.org/ijitcs/ijitcs-v5-n8/v5n8-8.html

Bace, R., Mell, P. (2001). *NIST special publication on intrusion detection systems*, *Nist Special Publication*. Retrieved from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA393326

Battistoni, R., Gabrielli, E., Mancini, L. V., Informatica, D. (2004). A Host Intrusion Prevention System for Windows Operating Systems, *ESORICS* : 352–368.

Berlin, K., Slater, D., Saxe, J. (2015). Malicious Behavior Detection using Windows Audit Logs, *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security - AISec '15* : 35–44. Retrieved from http://dl.acm.org/citation.cfm?doid=2808769.2808773

Bhattacharyya, D. K., Kalita, J. K. (2013). *Network Anomaly Detection: A Machine Learning Perspective*, Chapman and Hall/CRC.

Brugger, T. (2007). KDD Cup'99 dataset (Network Intrusion) considered harmful, *KDnuggets newsletter* 7(18): 15.

Buczak, A. L., Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection, *IEEE Communications, Survveys & Tutorials* 18(2).

Canzanese, R., Mancoridis, S., Kam, M. (2015). System Call-Based Detection of Malicious Processes, *Proceedings - 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015* : 119–124.

Creech, G. (2014). Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks : 215.

Creech, G., Hu, J. (2013). Generation of a new IDS test dataset: Time to retire the KDD collection, *IEEE Wireless Communications and Networking Conference, WCNC* : 4487–4492.

Dang, B., Gazet, A., Bachaalany, E. (2014). *Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation*, John Wiley & Sons.

Denning, D. E. (1987). An Intrusion-Detection Model, *Ieee Transactions on Software Engineering* 13(2): 222–232.

Depren, O., Topallar, M., Anarim, E., Ciliz, M. K. (2005). An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks, *Expert Systems with Applications* 29(4): 713–722.

Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., Vazquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges, *Computers & Security* 28(1): 18–28. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S0167404808000692

Haider, W., Creech, G., Xie, Y., Hu, J. (2016). Windows based data sets for evaluation of robustness of Host based Intrusion Detection Systems (IDS) to zero-day and stealth attacks, *Future Internet* 8(3).

Hay, A., Cid, D., Bary, R., Northcutt, S. (2008). *OSSEC Host-Based Intrusion Detection Guide*, *OSSEC Host-Based Intrusion Detection Guide*. Retrieved from http://www.sciencedirect.com/science/article/pii/B9781597492409000053

Hu, J. (2010). Host-based anomaly intrusion detection, *Handbook of Information and Communication Security*: 235–255. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-04117-4_13

Jurczyk, M.. (n.d.). Microsoft Windows x86 System Call Table (NT/2000/XP/2003/Vista/2008/7/8/10). Retrieved December 27, 2017, from http://j00ru.vexillium.org/syscalls/nt/32/

Korba, J. (2000). Windows NT Attacks for the Evaluation of Intrusion Detection Systems* Windows NT Attacks for the Evaluation of Intrusion Detection Systems.

Kovari, A., Dukan, P. (2012). KVM & OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE, *2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics, SISY 2012* : 335–339.

Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., Zissman, M. a. (1999). Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation, *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*: 12–26 vol.2. Retrieved from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA526274

Miao, W., Cheng, Z., Jingjing, Y. (2006). Native API based windows anomaly intrusion detection method using SVM, *Proceedings - IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing* 2006 II: 514–519.

Miller, C., Glendowne, D., Cook, H., Thomas, D., Lanclos, C., Pape, P. (2017). Insights gained from constructing a large scale dynamic analysis platform, *Digital Investigation* 22: 48–56. Retrieved from http://dx.doi.org/10.1016/j.diin.2017.06.007

Mitchell, M., Oldham, J., Samuel, A. (2001). Advanced Linux Programming, Retrieved from http://portal.acm.org/citation.cfm?id=558873

Mukkamala, S., Sung, A., Abraham, A. (2005). Cyber security challenges: designing efficient intrusion detection systems and antivirus tools, *Vemuri, V. Rao, Enhancing Computer Security with Smart Technology.(Auerbach, 2006)* : 125–163.

Sahu, S. K., Sarangi, S., Jena, S. K. (2014). A detail analysis on intrusion detection datasets, *Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014* : 1348–1353.

Tan, K. M. C., Killourhy, K. S., Maxion, R. A. (2002). Undermining an anomaly-based intrusion detection system using common exploits, In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 2516, pp. 54–73.

Timofte, J. (2008). Intrusion Detection using Open Source Tools, *Architecture* 2(2): 75–79. Retrieved from http://www.revistaie.ase.ro/content/46/Timofte.pdf

Xie, M., Hu, J. (2013). Evaluating host-based anomaly detection systems: A preliminary analysis of ADFA-LD, *Proceedings of the 2013 6th International Congress on Image and Signal Processing, CISP 2013* 3(Cisp): 1711–1716.

WEB (a). DynamoRIO. Dr. Memory Memory Debugger for Windows and Linux. https://github.com/dynamorio/drmemory

WEB (b). theZoo aka Malware DB by ytisf. Retrieved December 27, 2017 from http://thezoo.morirt.com/

WEB (c). VirusShare.com. Retrieved December 27, 2017 from https://virusshare.com/

WEB (d). VirusTotal. Retrieved December 27, 2017 from https://www.virustotal.com/

Received May 30, 2018, accepted June 5, 2018