Malheur Version 0.5.3
— User Manual —

Konrad Rieck

December 24, 2013

Contents

1 NAME

malheur - automatic analysis of malware behavior

2 SYNOPSIS

malheur [**-hrvV**] [**-m** *maldir*] [**-o** *outfile*] *action dataset*

3 DESCRIPTION

malheur is a tool for the automatic analysis of malware behavior (program behavior recorded from malicious software in a sandbox environment). The tool has been designed to support the regular analysis of malicious software and the development of detection and defense measures. **malheur** allows for identifying novel classes of malware with similar behavior and assigning unknown malware to discovered classes. It supports four basic actions for analysis which can be applied to reports of recorded behavior:

Extraction of prototypes.

From a given set of reports, **malheur** identifies a subset of prototypes representative for the full data set. The prototypes provide a quick overview of recorded behavior and can be used to guide manual inspection.

Clustering of behavior.

malheur automatically identifies groups (clusters) of reports containing similar behavior. Clustering allows for discovering novel classes of malware and provides the basis for crafting specific detection and defense mechanisms, such as anti-virus signatures.

Classification of behavior.

Based on a set of previously clustered reports, **malheur** is able to assign unknown behavior to known groups of malware. Classification enables identifying novel and unknown variants of malware and can be used to filter program behavior prior to manual inspection.

Incremental analysis.

malheur can be applied incrementally for analysis of large data sets. By processing reports in chunks, the run-time as well as memory requirements can be significantly reduced. This renders long-term application of **malheur** feasible, for example for daily analysis of incoming malware programs.

A detailed description of these techniques as well as technical background on analysis of malicious software is provided in the following articles:

Automatic Analysis of Malware Behavior using Machine Learning. Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Journal of Computer Security (JCS), 19(4) 639-668, 2011.

A Malware Instruction Set for Behavior-Based Analysis. Philipp Trinius, Carsten Willems, Thorsten Holz, and Konrad Rieck. Technical report TR-2009-07, University of Mannheim, 2009

The input of **malheur** is a *dataset* containing reports of malware behavior. The *dataset* is provided either as a directory or a compressed archive containing the reports. **malheur** supports the following formats for compressed archives: tar.gz, zip, pax and cpio. A *report* is a textual document describing the recorded activity of a malware program, where individual events are separated by delimiter characters, such as white space or carriage return. The events in a report are expected to be in sequential order if n-grams with n $>$ 1 are extracted. If the behavior is represented using the malware instruction set (MIST) further options may be selected.

The result of an analysis is written to *outfile*, a textual file containing columns corresponding to particular analysis results. By default *outfile* is set to *malheur.out*.

The configuration and internal state of **malheur** are stored in the directory *maldir*. If this directory does not exist, it is created and the system wide configuration is copied. A detailed description of the malheur configuration is provided in §5. By default *maldir* is set to *~/.malheur*.

4 ACTIONS & OPTIONS

malheur supported different *actions* for analysis of a *dataset*. For all actions the reports are first mapped to a high-dimensional vector space, such that each report is represented as a feature vector. Recorded events or n-grams of events are reflected in different dimensions and the dissimilarity of behavior can be assessed geometrically by computing distances and angles.

distance

If this action is specified, **malheur** computes a distance matrix for *dataset*. The entry (i,j) of this matrix reflects the distance (dissimilarity) of the reports i and j. The distance values lie in the range 0 to sqrt(2). The distance matrix is written to *outfile*.

prototype

If this action is specified, **malheur** determines a set of *prototypes* representing *dataset*. The prototypes are selected from the contained reports, such that the distance from any report to its nearest prototype is minimized. The prototype assignment of *dataset* is written to *outfile*.

cluster

If this action is specified, **malheur** performs a *clustering* of *dataset*. The clustering is first determined on prototypes and then propagated to all reports. Small clusters with too few members are merged in a *rejection cluster*. The prototypes representing accepted clusters are stored as internal state of **malheur** for later classification and incremental analysis. The clustering of *dataset* is written to *outfile*.

classify

If this action is specified, **malheur** performs a *classification* of *dataset*. Each report is either assigned to the nearest prototype of a known cluster or rejected as unknown. This action requires that a clustering has been performed beforehand and an internal state of **malheur** exists. The classification of *dataset* is written to *outfile*.

increment

If this action is specified, **malheur** performs an *incremental analysis* of the reports. The reports are first classified to known clusters as in the action *classify*. Reports rejected from classification are then clustered as in the action *cluster*. The prototypes of the accepted clusters and the rejected reports are written to the internal state of **malheur** for further incremental analysis. The classification and clustering of *dataset* are written to *outfile*.

protodist

If this action is specified, **malheur** computes a distance matrix for prototypes. The entry (i,j) of this matrix reflects the distance (dissimilarity) of the prototypes i and j. The distance values lie in the range 0 to sqrt(2). This action requires that either a prototype extraction, a clustering or an incremental analysis has been performed beforehand. The distance matrix is written to *outfile*.

malheur also supports the following command-line *options* which are used to further control the analysis process

-m ***maldir***

This option specifies the malheur directory *maldir* which holds the configuration and internal state of **malheur**. If the directory does not exist, it is created and the system wide configuration is copied.

-o ***outfile***

This option specifies the output file *outfile* for analysis. The file is created during analysis and the results are stored in textual form.

-r

This option resets the internal state of **malheur**. Prototypes of clusters and rejected reports from previous runs of **malheur** are removed.

-v

This option is used to increase the verbosity of **malheur** during analysis, where the verbosity level corresponds to the number of "-v" options.

-h

This option prints a brief help screen.

-V

This option prints a version and copyright string.

5 CONFIGURATION

The configuration file *malheur.cfg* in *maldir* determines how the malware behavior in a data set is analysis. The configuration contains five groups of settings which are described in the following.

All parameters of the configuration can be also specified on the command line. That is, if a parameter is defined in the configuration as **xx = "yy";** in the group **zz**, it can be alternatively supplied as a command-line option by **--zz.xx "yy"** to **malheur**.

input = {

format = "mist";

This parameter specifies the input format. Supported values are "text" for textual and XML reports, and "mist" for reports using the malware instruction set (MIST).

mist_level = 2;

This parameter specifies the MIST level. If the input format is set to "mist", this parameter controls the analysis level of MIST instructions, otherwise it is ignored.

mist_rlen = 0;

This parameter specifies the report truncation length. If the input format is set to "mist", this parameter controls the truncation of MIST reports, otherwise it is ignored. If set to 0 the parameter is ignored in all cases.

mist_tlen = 0;

This parameter specifies the thread truncation length. If the input format is set to "mist", this parameter controls the truncation of MIST threads, otherwise it is ignored. If set to 0 the parameter is ignored in all cases.

};

features = {

ngram_delim = "%0a%0d";

This parameter defines characters for delimiting events in report files. The characters can be either specified as regular bytes or as hexadecimal numbers prefixed by "%". If no characters are specified, the reports are analyzed at byte-level, as if each byte would reflect one event.

ngram_len = 2;

This parameter specified the length of n-grams. If the events in the reports are not sequential, this parameter should be set to 1. In all other cases,

it determines the length of event sequences to be mapped to the vector space, so called n-grams.

vect_embed = "bin";

This parameter specifies how the feature are embedded in the vector space. Support values are "bin" for associating each dimension with a binary value or "cnt" for associating each dimension with a count value for the occurrences of features.

lookup_table = 0;

This parameter is used to enable an optional feature lookup table. The table can be used during debugging and verbose output for tracing dimensions in feature vectors back to events. For performance reasons it should be disabled by default.

hash_seed1 = 0xc0cac01a;

hash_seed2 = 0xadd511fe;

To enable efficient comparison of feature vectors, **malheur** internally represents string features as 64 bit hash values using MD5. These two parameters allow to change the seed of the MD5 hash and should be initialized to random values, which protects from targeted collision attacks. The remaining risk of collisions is minimal: (a) the number of unique features per report is limited to several thousands, and (b) in case of a collision the respective features can not be predicted.

};

prototypes = {

max_dist = 0.65;

This parameter specifies the maximum distance to a prototype. During analysis prototypes are selected in a way such that the distance from each report to its nearest prototype is below this value. The parameter lies in the range 0 to sqrt(2). If set to 0 all reports are considered as prototypes.

max_num = 0;

This parameter defines the maximum number of prototypes. During analysis prototypes are selected until this value is reached. If too many prototypes are determined, this parameter can be used to reduce computational costs at the price of a coarser approximation. If set to 0 this parameter is ignored.

};

cluster = {

link_mode = "complete";

This parameter specifies the clustering mode. Supported values are "complete" for complete-linkage clustering, "average" for average-linkage clustering and "single" for single-linkage clustering.

min_dist = 0.95;

This parameter defines the minimum distance between clusters. The clustering operates in a bottom-up manner. That is, clusters are successfully

merged until the minimum distance between the closest pair of clusters is above this value. The parameters lies in the range 0 to sqrt(2).

reject_num = 10;

This parameter specifies the minimum number of members in a clusters. Small clusters containing less members than this value are rejected. The corresponds reports are assigned to a global rejection cluster. If set to 0, all clusters are accepted.

shared_ngrams = 0.0;

This parameter allows to extract shared n-grams for each clusters. The shared n-grams are determined by merging the members in each cluster and identifying all n-grams shared by at least the given ratio of members. The resulting list of shared n-grams is appended to *outfile*. If set to 0.0, this feature is disabled. Note that if shared n-grams are enabled, a feature lookup table is maintained which consumes extra memory.

};

classify = {

max_dist = 0.68;

This parameter defines the maximum distance to prototypes during classification. Reports that are closer to the nearest prototype than this value are assigned to the cluster represented by prototype, whereas reports that are farther away than this value are rejected from classification. The parameter lies in the range 0 to sqrt(2). If set 0 all reports are classified, irrespective of the distance to a prototype.

};

6 FILES

/etc/malheur.cfg

The system wide configuration file of **malheur**. See §5 for further details.

~/.malheur/malheur.cfg

Per user configuration file of **malheur**. See §5 for further details. If this file does not exist, it is automatically created using the system wide configuration as template.

~/.malheur/prototypes.zfa

~/.malheur/rejected.zfa

Internal state files of **malheur** containing compressed feature vector array (zfa) of **prototypes** and **rejected reports**. The feature vectors are used for classification and incremental analysis. See §4 for further details.

7 EXAMPLES

Distances of program behavior. The first example demonstrates how a distance matrix is computed for the archive *dataset.zip* containing reports of program behavior. The matrix is written to the file *out.txt*.

```
malheur -o out.txt -v distance dataset.zip
```

The distance matrix reflects the dissimilarity of behavior for each report in the archive. The entries of the matrix range from 0 to sqrt(2), where small values indicate similar behavior and larger values deviating behavior. The matrix can be used as the basis for several analysis and data mining techniques, such as hierarchical clustering, nearest-neighbor classification or multi-dimensional scaling. It is a generic starting point for research on analysis of malware behavior.

Extraction of prototypes. Manual inspection of several behavior reports is tedious and annoying. The second example illustrates how prototypical reports are extracted from the dataset *dataset.zip*. The prototypes are written to the file *out.txt*.

```
malheur -o out.txt -v prototype dataset.zip
```

From all the reports of program behavior, a small subset is selected which is representative for the full data set. The elements of this subset are referred to as prototypes. Prior to further analysis of a large data set, a quick inspection of prototypes enables an overview of contained behavior and shows patterns typical for the data set.

Clustering and classification. This example demonstrates how clustering and classification are applied for analysis of two data sets, *dataset1.zip* and *dataset2.zip*. The clustering and classification results are written to *out1.txt* and *out2.txt* respectively.

```
malheur -o out1.txt -v cluster dataset1.zip
malheur -o out2.txt -v classify dataset2.zip
```

First, reports in the archive *dataset1.zip* are clustered into groups of similar behavior. The groups can be used to discover novel malware classes or identify behavioral patterns shared by several malware instances. Each cluster is represented by a small set of prototypical reports, such that manual inspectation can usually be restricted to prototypes. Second, the reports in *dataset2.zip* are assigned to the discovered groups. This classification can be used to filter out variants of classes contained in *dataset1.zip*, such that novel malware in *dataset2.zip* can be identified.

Incremental analysis. In the next example, Malheur is applied for incremental analysis of a larger data set split into three archives, namely *dataset1.zip*, *dataset2.zip* and *dataset3.zip*. Results of this analysis are written to the files *out1.txt*, *out2.txt* and *out3.txt*.

```
malheur -o out1.txt -v -r increment dataset1.zip
malheur -o out2.txt -v increment dataset2.zip
malheur -o out3.txt -v increment dataset2.zip
```

First, the archive *dataset1.zip* is processed using incremental analysis. The extra option **-r** is used to reset the internal state of Malheur, such that results from previous incremental runs are discarded. Then, the files *dataset2.zip* and *dataset3.zip* are analyzed where for each archive first known behavior is identified using classification and novel groups of malware are discovered using clustering. The intermediate results for each archive are stored in the Malheur home directory, by default *~/.malheur*. The incremental analysis allows to process large data sets efficiently, where run-time and memory requirements are significantly reduced in comparison to batch analysis.

Debugging. The reports of malware behavior are embedded in a vector space where each report is represented by a sparse feature vector. To understand this representation and trace down problems, a lookup table can be enabled in the features setting of *malheur.cfg*.

```
malheur -o /dev/null -vvv prototype dataset.zip
```

The above command extracts prototypes from the provided data set. However, it also present a lot of verbose information on the reports and extracted prototypes. In particular, for each prototype the corresponding feature vector is displayed. If the lookup table is enabled, the dimensions of this vector are printed with respective instruction n-grams (substrings composed of n instructions).

8 BUGS

The reports for analysis need to be textual documents. Although non-printable characters may be contained in the report files, no occurrences of the NUL character (0x00) are allowed. The behavior of **malheur** is undefined in this case.

The vectorial analysis underlying **malheur** does not handle null vectors, as they can not be scaled to a fixed norm. Consequently, empty files are discarded during extraction of feature vectors.

Depending on the linked version of libarchive, **malheur** may support different types of archive formats. For example, zip archives of version 2 are not generally supported by libarchive. As a fallback, the use of tar archives is recommended.

9 COPYRIGHT