Fighting malware with GPUs in real time

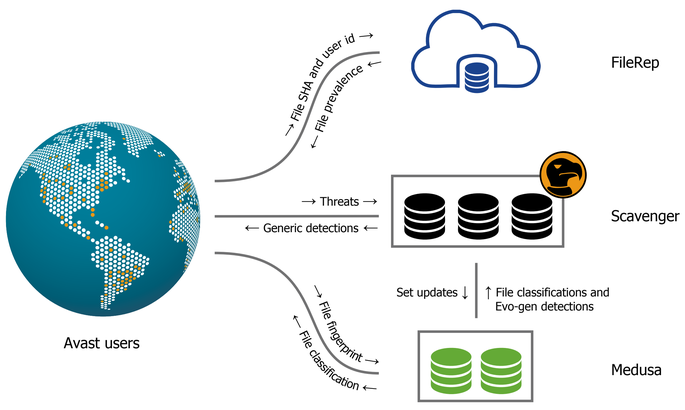
Every day several hundred thousand previously unseen executables appear in the machines of our users. But most of them are only variants of already known files. To classify such variants quickly and reliably we need to effectively leverage their similarity with the known files. This helps our analysts to focus on the truly new threats.

Libor Morkovský, Peter Kováč
AVAST Software s.r.o.

Visit Peter's talk (S5612) for more details about Medusa.

System Setup

There are four main parts in our system. First there are few hundred million user machines, which serve as a sensor network for our system, while being protected from emergent threats in return. Next there is a huge database containing user counts for all the files ever tested by Avast called **FileRep**. Then there is an internal system in Avast called **Scavenger.** It keeps track of all the threat samples that ever entered the company systems and of all the work that the virus analysts did to classify and detect those threats. Last there is the GPU powered system presented in this poster, that leverages all the information to automatically classify as much samples as possible. We call it **Medusa**.



Synchronization

Medusa has to be kept up to date with all the changes that happen in Scavenger – new files are arriving, files are classified as malware or benign or removed from the sets which are relevant for classification. A durable message queue in RabbitMQ is used to ensure that no updates are lost even during the Medusa server updates and maintenance. Additionally the consistency between Medusa and Scavenger is checked with complete dumps of object identifiers on schedule.

Real time deployment

One instance of Medusa is currently deployed as a real time classifier, which is called whenever any single Avast Antivirus has trouble deciding if a newly found file could be malicious or not (see the Customer protection part). Given the 200 million users of Avast there is a potential for a huge load. To reduce the potential load we implemented a caching proxy between the clients and the Medusa cluster. A file can be classified differently, as new information arrives from Scavenger, so the TTL of the cached decisions is set to few minutes. Despite this, the cache hits in almost 40 % of the requests.

Cluster setup

Every Medusa node in our deployment uses two or four Nvidia GPUs. A Medusa cluster has one master node that is aware of all the sets and several slave nodes, which contain some parts of the sets.
The classification needs a lot of clean and malware samples. The Evo-gen generator uses also a set with unclassified samples. Because of a big difference in usage patterns we keep the different sets separately (think separate database tables). Clean set samples are the most important because of the inherently high costs of a false positive. Thus the clean set takes most of the space and is proportionally the slowest to scan. To increase the throughput we keep the clean set mirrored. The sets with recent malware and unclassified samples take up only a fraction of the space needed for clean set (roughly 1/10).

Machine Learning

Some form of machine learning is necessary to effectively use all the information collected over the years in our internal systems. We chose instance based learning because it has a lot of desired properties: • re-learning the model is only a matter of adding or removing a sample to or from the correct set • it is simple to find the reasons for particular decisions • it is easy to fine tune the false positive rate • the decision boundary is not expected to be very smooth for this kind of problem.

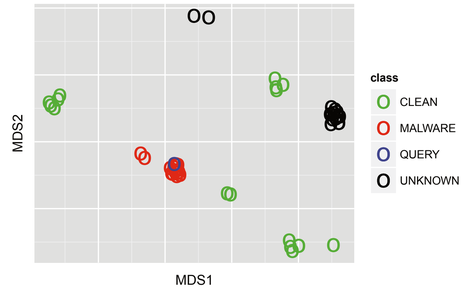
Custom distance function

Each sample is represented by a constant-sized feature vector consisting of ~100 attributes. We keep the exact composition of the feature vector secret. The obvious candidates like section table data in the Portable Executable format are also included, for example. In general there are static and dynamic features, categorized as offsets, sizes, checksums, factors, bit flags and generic numbers.
In our proof-of-concept implementation we started with a simple Hamming distance. It worked surprisingly well, but there was enough room for improvement. Taking into account the nature of the attributes we ended up with several distance operators and a weighting scheme, that equalizes the importance of the attributes. The following table contains a sample of operators we're using.

Distance operator	Field types	Description
EQUAL_RET32	CHECKSUM, VALUE	return 32 when values are equal
HAM	STRING	Hamming distance
HAM_MUL32	BITFIELD	Hamming distance multiplied by 32 - each flag change is as important as maximal change of one feature
LOG	LENGTH, OFFSET	base 2 logarithm of a difference
ORDER	LENGTH, OFFSET, VALUE	difference of base 2 logarithms
RETZ	*all*	ignore the feature, return 0 for all values

kNN classifier

The most common approach for instance based learning is the nearest neighbour classification. To fine tune our classifier we built a tool that displays nearest neighbours of a given query sample (called **Pythia**). It uses a dimensionality reduction method (NMDS) to display the neighbours in 2D space and also displays a lot of additional metadata for the selected samples. This information can be used by a human to decide whether it is plausible to distinguish between malware and clean neighbours in current case.



The goal was to create a fully autonomous system – that means high precision at the cost of lower recall. After some experimenting we added few thresholds: for minimal allowed distance to clean files, maximal allowed distance to malware files and a weighting term that shifts the balance between clean and malware sets.

Real world data

The redundancy in real world data is quite significant. Our internal systems handle around 250,000 new PE files every day. Out of those, 150,000 can be directly assigned to one of 20,000 clusters using very strict clustering criteria (low threshold distance and complete linkage). Each cluster can then be classified as a whole. That means 130,000 less decisions to make. And the total number of clusters does not grow by 20,000 every day, the clusters overlap between days.

Customer Protection

There are several methods to transfer the classification to our customers. Each of them has its own pros and cons. The following table summarizes the main differences, the pros are in green, cons are in red.

Method	How many users	Delay in classification	How many file versions
Real time classification	*single user*	*no delay*	*one*
FileRepMalware	*all users*	*seconds*	*one*
Evo-gen	*all users*	*minutes*	*many*

Real time classification

Avast Antivirus checks every executable before it's executed in the customer's machine. When no signature from current threat database matches the file, the FileRep service is queried. If the returned user count (prevalence) is anomalously low, the executable ends up in Avast Sandbox. If the executable trace log does not match any known threat, the real time classifier is invoked.
Avast Antivirus extracts the feature vector, submits it to a cloud based service and waits for the response. Most of the low prevalence files are benign. Out of ~250,000 requests daily about 4,000 are classified as malicious.

FileRepMalware

Once a file is classified as malicious and our internal systems check that it is safe to detect this particular file worldwide, a simple flag is set in the FileRep service. Every Avast client that encounters that particular file instantly blocks it and reports it as FileRepMalware.

Evo-gen

The good thing about (the old) string based signatures is that when done right, they generalize to many variants of the threat. But string based signatures require a skilled analyst and time. We needed something that generalizes like string signatures, but does not need the human skill and time.
Enter Evo-gen – once we've got a set of very similar feature vectors thanks to the distance function, we can start to pick features that make them similar and build a rule set from those features. It is a bit similar to rule set generation in decision trees, but the objectives are different. To boost the generalization we can try to pick as few rules as possible, while keeping hits in clean set at zero. But there is a lot of ways to pick 20 rules from 100 possible ones – 5.36×10^{20} (536 billion billion) numerically. We're currently taming the combinatorial explosion with a stochastic approach (gives better results than greedy approaches). This is where the speed of the GPUs is very important again.
While trying to understand how the Evo-gen rule sets (blue) affect the signature 'ecosystem', we produced the following visualization. Each blob represents a different rule set or signature, the size of the blob is proportional to the number of detected variants.