Anomaly Detection

Applications: network throughput analysis

Paul Irofti
Andrei Pătrașcu
Cristian Rusu

Computer Science Department

Faculty of Mathematics and Computer Science

University of Bucharest

Email: `first.last@fmi.unibuc.ro`

Outline

- Preliminaries
- Ensembles and voting methods
- Abnormal traffic analysis
- Netalert: network throughput analysis
- DeDDoS: anti-DDoS solution (on-going)

The course main reference is Irofti, Pătrașcu, and Hîji 2022.

Preliminaries

Algorithms Overview

Models use ensembles consisting of the following algorithms

- Isolation Forest (IF)
- k-Nearest Neighbors (k-NN)
- Lightweight Online Detector of Anomalies (LODA)
- Local Outlier Factor (LOF)
- One-Class SVM (OC-SVM)
- Autoencoder (AE)

We will briefly discuss each and their parameters in existing implementations such as Graphomaly, PyOD, and Prophet.

Isolation Forest

- ensemble method consisting of multiple iTrees
- each iTree at node n selects a random feature f and a split value v
- signals are categorized at node n on the left hand side or the right hand side based on their f attribute: $x(f) > v \lor x(f) < v$?
- the tree leafs represent a single signal
- the longer the path towards the leaf the harder it is to isolate the signal
- we usually use around 100 iTrees in the ensemble

Q: How do we mark a signal as anomalous?

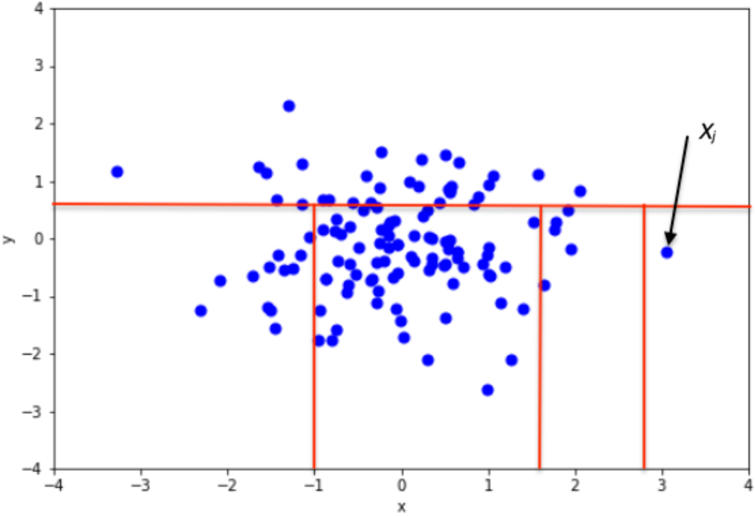
Isolation Forest

- ensemble method consisting of multiple iTrees
- each iTree at node n selects a random feature f and a split value v
- signals are categorized at node n on the left hand side or the right hand side based on their f attribute: $x(f) > v \vee x(f) < v$?
- the tree leafs represent a single signal
- the longer the path towards the leaf the harder it is to isolate the signal
- we usually use around 100 iTrees in the ensemble

Q: How do we mark a signal as anomalous?

A: Short paths indicate that the signal is far from the majority of points. It is anomalous!

Example: Isolation Forest



Source: https://en.wikipedia.org/wiki/Isolation_forest

Isolation Forest Parameters

- `contamination`: percentage of outliers
- `n_estimators`: number of iTrees
- `max_samples`: number of samples from the training set per iTree
- `max_features`: number of features from the training set per iTree
- `bootstrap`: sampling with replacement?
- `n_jobs`: parallelization level

k-Nearest Neighbors Parameters

For each signal perform the following:

- choose the k closest signals to the current signal
- 'closest' is decided on a distance metric (e.g. euclidian, ℓ_1, ℓ_ε, minkowski)
- distance to the k-nearest neighbor estimates the isolation score of the current signal
- the larger the distance to its neighbors the highest its outlier score
- score can be computed based on neighbors distance in various way: maximum, mean, median

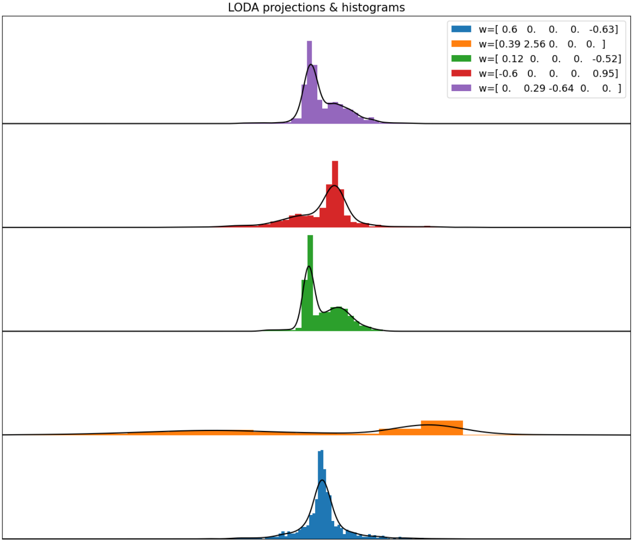
k-Nearest Neighbors Parameters

- `contamination`: percentage of outliers
- `n_neighbors`: k value
- `method`: outlier score based on
  - `largest`: use distance to kth neighbor
  - `mean`: use the average of all k neighbors
  - `median`: use the median of distance
- `metric`: how to compute the distance between two sigals

Lightweight online detector of anomalies

- ensemble method based on a set of subspaces (usually if $x \in \mathbb{R}^n$ then subspace is $\sqrt{n}$)
- generate sparse subspaces based on signal size
- choose random coefficients for the resulting vectors
- project each signal on the generated subspaces
- generate 1D histograms for each projection based on all the projected signals
- use each histogram to associate a probability for each signal
- for every signal use the probabilities resulted from each histogram to generate the anomaly score
- the lower the probability (the less frequent a bin is) the more chances the signal is an outlier
- the more histograms label the signal as an outlier, the higher the outlier score

Example: LODA



Source: https://anlearn.readthedocs.io/en/latest/loda.html

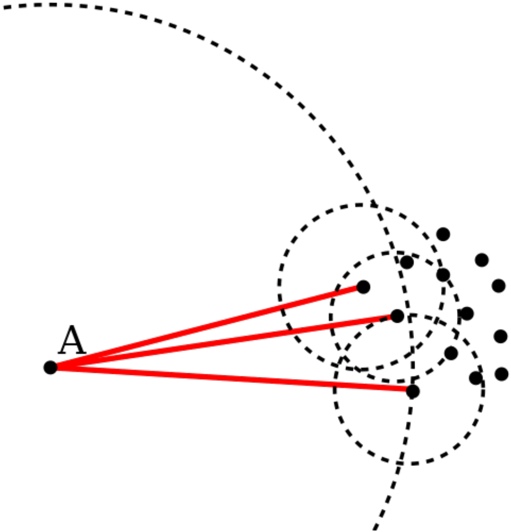
LODA Parameters

- `contamination`: percentage of outliers
- `n_random_cuts`: number of subspaces in the ensemble
- `n_bins`: number of bins in the histograms

Local Outlier Factor (LOF)

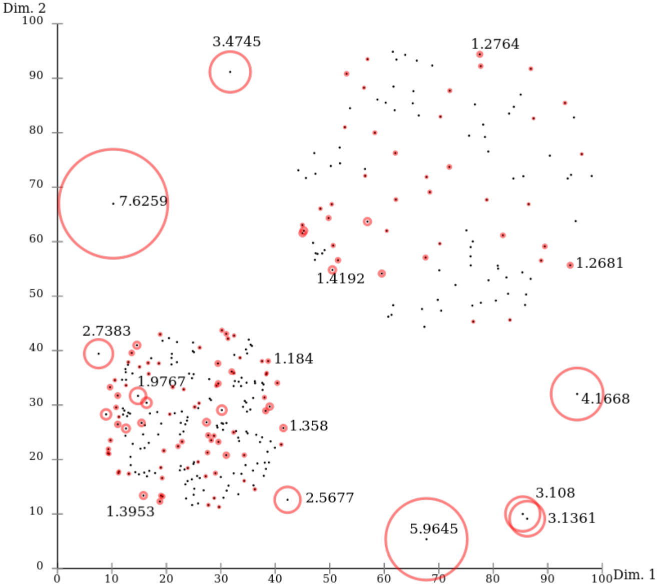
- similar to k-NN
- estimates the local density of a signal based on its k nearest neighbours
- next it computes the local density of its selected neighbors
- the distance to its kth neighbor creates a radius, everything within that radius is considered the signal's neighbor
- if the neighbors have a higher density than the current signal, then the signal is isolated
- based on that LOF builds the outlier score (anomalous if $\text{LOF}(x) > 1$)

Example: LOF local density



Source: https://en.wikipedia.org/wiki/Local_outlier_factor

Example: LOF anomaly detection



Source: https://en.wikipedia.org/wiki/Local_outlier_factor

LOF Parameters

- `contamination`: percentage of outliers
- `n_neighbors`: *k* value
- `algorithm`: method used to obtain the *k* nearest neighbors; how to inflate the hypersphere?
- `leaf_size`: tied to the `algorithm`, helps reduce complexity
- `metric`: how to compute the distance between two sigals
- `n_jobs`: parallelization level

Autoencoder: Architecture

Neural network architecture with an imposed two part structure.

Goal: Obtain a subspace where most of the signals lie.

Encoder.

- starts with a layer of neurons of the same size as the signal dimension
- next layers reduce the number of neurons
- last layer has the smallest number of neurons in the network; also called the latent space

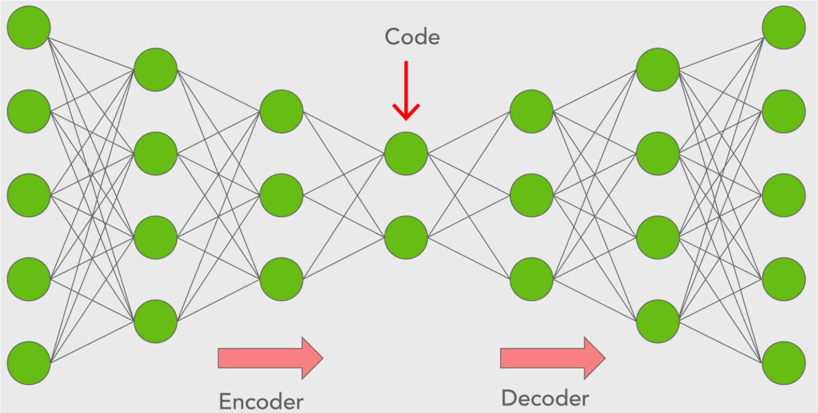
Decoder.

- start with a layer connected to the latent space; this new layer has more neurons
- next layers are always larger than the last
- last layer has the same size as the original signal's dimension

Autoencoder: Properties

- usually not very deep, a few hidden layers suffice
- most common activation function is ReLU
- the last layer produces a reconstructed signal
- outiler score is based on the representation error between the original and reconstructed signal
- normal signals should have a small error due to the latent space (subspace) that was trained for them
- contamination rate dictates the anomaly percentile

Example: Autoencoder



Source: https://www.mygreatlearning.com/blog/autoencoder/

AE Parameters

- `contamination`: percentage of outliers
- `activation_function`
- `l2_regularizer`: control weights growth during backpropagation
- `encoder_neurons`: vector where each element represents an encoder layer and its value the number of neurons in that layer
- `decoder_neurons`: same as above for the decoder
- `optimizer`: backpropagation algorithm
- `learning_rate`: gradient decent step
- `epochs`: number of times to go through the training set
- `loss`: how to measure the approximation quality

Ensembles and Voting Methods

In practice we create ensembles of the algorithm listed above in its models.

Ensembles.

- each algorithm labels the outlier in the dataset
- a label is like a verdict, it is a binary value
- every outlier has an outlier score (see above)
- the outlier score represents a confidence score of the verdict

Voting.

- hard voting takes a majority vote on the resulting labels from the ensemble algorithms
- soft voting uses an algorithm to combine the outlier scores and then produces the final verdict based on the sums of the resulting probabilities

Ensembles and Voting Methods

In practice we create ensembles of the algorithm listed above in its models.

Ensembles.

- each algorithm labels the outlier in the dataset
- a label is like a verdict, it is a binary value
- every outlier has an outlier score (see above)
- the outlier score represents a confidence score of the verdict

Voting.

- hard voting takes a majority vote on the resulting labels from the ensemble algorithms
- soft voting uses an algorithm to combine the outlier scores and then produces the final verdict based on the sums of the resulting probabilities

Q: Why do we need a combine algorithm?

Ensembles and Voting Methods

In practice we create ensembles of the algorithm listed above in its models.

Ensembles.

- each algorithm labels the outlier in the dataset
- a label is like a verdict, it is a binary value
- every outlier has an outlier score (see above)
- the outlier score represents a confidence score of the verdict

Voting.

- hard voting takes a majority vote on the resulting labels from the ensemble algorithms
- soft voting uses an algorithm to combine the outlier scores and then produces the final verdict based on the sums of the resulting probabilities

Q: Why do we need a combine algorithm?

A: We have distances, densities and probabilities as outlier scores.

Netflow

- flow_id
- src_ip
- src_port
- dst_ip
- dst_port
- protocol
- timestamp
- Duration
- PacketsCount
- FwdPacketsCount
- BwdPacketsCount
- TotalPayloadBytes
- FwdTotalPayloadBytes
- BwdTotalPayloadBytes
- PayloadBytesMax
- PayloadBytesMin
- PayloadBytesMean
- PayloadBytesStd
- PayloadBytesVariance
- PayloadBytesMedian
- PayloadBytesSkewness
- PayloadBytesCov
- PayloadBytesMode
- . . .

https://github.com/ahlashkari/NTLFlowLyzer

Unsupervised Abnormal Traffic Detection through Topological Flow Analysis

Based on work from Irofti, Pătrașcu, and Hîji 2022

Motivation

NetAlert: Automatic tools for detecting abnormal behavior in computer networks.

- unsupervised anomaly detection
- using only layer-3 information (no payload!)
- main goal is detecting abnormal traffic
- secondary goals are related to specific targeted attacks
- builds on open-source Graphomaly library resulted from PED research project (available on pypi)
- `https://cs.unibuc.ro/~pirofti/netalert.html`
- work supported by grant PN-III-P2-2.1-SOL-2021-0036

Network Topology

Main steps of our method reduce to

- embedding of the network flows into a directed graph
- extraction of several statistical node features from the graph and expand the original feature set.
- fixed IPs within a network mapped into integers
- graph $\mathcal{G} = \{V, E, W\}$ – V computers, E connections, W weights

Result: $X \in \mathbb{R}^{m \times N}$ network data, with m features and N flows. For two IPs let

$$(i, j) \equiv (\text{source\_}IP, \text{destination\_}IP)$$

then $(i, j) \in E$ if there exists a flow between IPs mappings (i, j). The value w_{ij} on ith column and jth line in matrix W defines some summable feature (e.g. packets transmitted)

Flow to Graph

Conversion of data from flow format into graph format:

- source, destination addresses (i, j)
- particular attributes which represents the weight w_{ij}.

Remark: Attribute may be any real-valued summable feature in the original data X.

Multi-graph: Multiple flows occur multiple times between the same pair of nodes:

- we get multiple weights w_{ij}^t, where t is time counter.
- we sum over t these weights in order to obtain a final weight

$$w_{ij} = \sum_t w_{ij}^t$$

.

Graph to Features

1) Fetch Egonets

1') Perform Random Walks

2) For any $i \in [n]$, extract p features of the egonet/random walk instance $\mathbb{E}_i$. Denote $z_i \in \mathbf{R}^p$ the vector of these features.

3) Output matrix $Z \in \mathbf{R}^{p \times n}$, as the array containing all egonet/random walk features.

Perform a single step of the two alternatives 1) or $1'$).

Notice that the random-walk $\mathbb{E}_i$ computed in scenario $1'$) is not limited to the egonet neighborhood of node i.

Egonets and Random Walks

Egonets:

1) Extract all the egonets and stack them into $\mathbb{E}$, where each $\mathbb{E}_i \in \mathbb{E}$ is the egonet associated to node $i \in V$.

Random Walks:

1') Extract a random-walk of size ℓ for each node.

1'1) Denote $\mathbb{E}_i \in \mathbb{E}$ as the random-walk associated to node $i \in V$.

1'2) Starting at node i, for at most ℓ iterations, a neighbor of the current node is randomly chosen and its associated edge is added to $\mathbb{E}_i$.

1'3) Chosen node becomes the current node, reiterate

1'4) If either the node i or the walk length ℓ are reached, stop

Graph Features

The statistical features computed in step 2), after step 1):

- dimension of egonet,
- the number of out-links,
- the number of in-links.

For scenario $1'$) they include:

- the weight on the first leg of the walk
- the weight transferred all the way from the first node to the last one of the walk.

The full description of all features can be found in Graphomaly.

Egonets

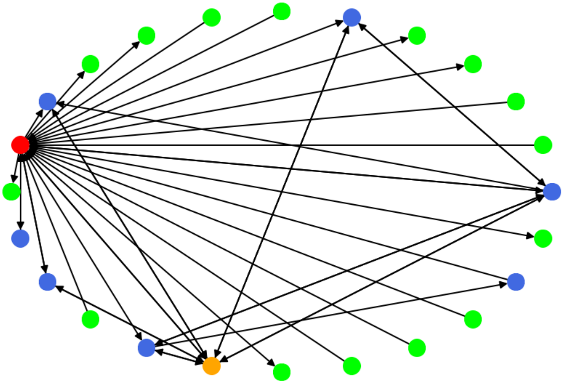


Figure: Graphomaly Egonet example (Dumitrescu, Băltoiu, and Budulan 2022)

Feature Expansion

We expand the original data by adding the columns of Z as prolongation of columns in X.

For a given flow $x_t \in X$ from source i to destination j, we form:

$$\hat{x}_t = \begin{bmatrix} x_t \\ z_i \\ z_j \end{bmatrix} \in \mathbf{R}^{m+2p}.$$

The matrix $\hat{X}$ containing columns $\hat{x}_t$ for $t \in [N]$ is the output.

Experiments

In our simulations we use:

- One-Class SVM (OC-SVM)
- Local Outlier Factor (LOF)
- Isolation-Forest (IForest)
- an Ensemble that includes the above.

In the implementation of the Ensemble we use voting methods.

Preliminary tests with autonecoder and variatonal autoencoder architectures that have not yet shown promising results.

The experiments were performed using public datasets CIC-IDS2017 and UNSW-NB15.

Obtaining Parameters

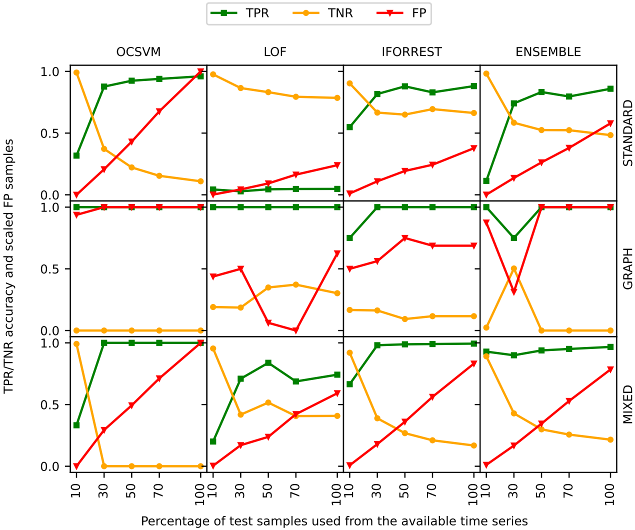
Assumption: We assume access to a small initial dataset depicting the normal state of the network through their recorded layer-3 traffic.

Normality: In our experiments, we extract the first 1% of samples from each dataset and assume that this is known data with known labels.

Model parameters: We perform grid-search for both databases when using the standard, graph and mixed features.

Balanced Accuracy Results

Prediction results for the models trained on 1% on UNSW-NB15.



Retraining on more Data

Testing: We use the obtained parameters to train the models on the next 10% of available data from the time-series.

Remark: We see a clear degradation in the balanced accuracy compared to the tuned experiments: larger dataset, new attacks, model parameters are not optimal.

Dataset	Method	(m,N,outliers)	OC-SVM		LOF		IForest		Ensemble
CIC-IDS2017	standard	(87, 45883, 928)	0.3724	242.30s	**0.4811**	529.12s	0.4132	0.35s	**0.4996**
	graph	(48, 1999, 1)	0.4997	0.43s	0.4705	0.07s	**0.4750**	0.06s	0.4874
	mixed	(183, 45883, 928)	**0.5955**	601.00s	0.4805	86.42s	0.4222	0.32s	0.4821
UNSW-NB15	standard	(59, 44004, 5148)	0.6542	358.21s	0.5096	147.76s	0.7259	5.09s	0.5474
	graph	(48, 46, 4)	**0.7829**	0.01s	**0.6382**	0.01s	0.3289	0.03s	0.5119
	mixed	(155, 44004, 5148)	0.6619	579.19s	0.5775	750.70s	**0.7926**	0.82s	**0.9103**

Using the Ensemble

Dataset	Attack	Detected	Total
standard	Exploits	163	2088
	DoS	79	1014
	Fuzzers	29	516
	Worms	0	7
	Backdoor	11	138
	Analysis	9	123
	Shellcode	2	52
	Reconnaissance	31	548
	Generic	256	662
mixed	Exploits	1933	2088
	DoS	911	1014
	Fuzzers	502	516
	Worms	7	7
	Backdoor	124	138
	Analysis	109	123
	Shellcode	47	52
	Reconnaissance	506	548
	Generic	644	662

Netalert

Based on the Soluții 2021 Grant PN-III-P2-2.1-SOL-2021-0036
https://cs.unibuc.ro/~pirofti/netalert.html

Netalert: Goals

Goal: NetAlert aims to create a hardware-software sensor solution for detecting anomalies in computer networks based on the monitoring and analysis of data packets.

The network-mounted sensor will provide real-time alerts on abnormal traffic behaviors using two complementary approaches:

(i) static analysis based on rules and behavioral patterns

(ii) machine learning (ML) analysis without prior expert knowledge

Netalert: Recorded Traffic

Netalert records, monitors and analyzes traffic:

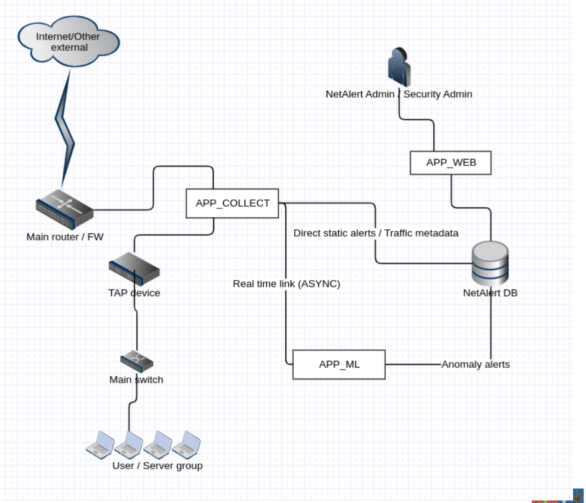
- network traffic to the Internet
- network traffic passing through Proxy
- internal network traffic
- TCP, UDP, ICMP traffic
- network traffic associated with DNS and DHCP
- Domain Controller (Kerberos) network traffic
- SMTP network traffic
- Samba (SMB) network traffic
- TOR communications

Netalert: Detected Behaviors

- transferred files and data internally and externally
- DNS requests and DNS traffic anomalies
- device operating system
- proxy server identification
- cryptocurrency mining activity
- applications specific to APT attacks
- communications with the TOR network
- dynamic DNS
- port and network scanning
- abnormal behavior per device
- loss of connections or packets
- Kerberos authentications
- lateral movement
- bruteforce attacks
- Heartbleed attacks
- invalid SSL certificates
- port hijacking
- connecting non-standard ports

Netalert: Architecture

- Network traffic collection (`APP_COLLECT`)
- Machine Learning and Alerting Service (`APP_ML`)
- Web Application for Admin and Agents (`APP_WEB`)
- Database (`NetAlert DB`)



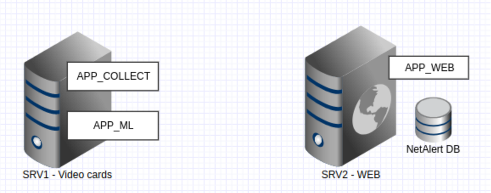
Software Modules and Hardware Architecture

SRV1 contains:

- 2xAMD EPYC 7502 2x32 cores or 2x2x32 threads
- 8xNVIDIA RTX A6000, 10.752 cores, 48GB mem
- 512GB system memory
- 1.92TB NVMe secondary memory
- 2x network cards

SRV2 contains:

- 2xIntel Xeon Platinum 8200 2x24 cores or 2x2x24 threads
- 512GB system memory
- 2x960GB secondary memory
- 2x10/100/1000 and 2x1Gbe/10Gbe SFP+



Netalert Models

Netalert uses the following behavior models:

- TCP: a model for each device and one for the entire network
- UDP: a model for each device and one for the entire network
- kerberos authentication (brute force)
- lateral movement
- network scan
- DNS
- SMTP

Technologies used: Graphomaly, Prophet, PyOD

TCP

Network.

- trained on TCP traffic from the entire network
- ensemble based on IF, KNN, LODA, LOF, OCSVM, AE
- soft voting
- no graph preprocessing (egonets, random walks)

Device.

- trained on device TCP traffic for each network device
- based on MAC address list
- same configuration as for the network model

UDP

Network.

- trained on UDP traffic from the entire network
- ensemble based on IF, KNN, LODA, LOF, OCSVM, AE
- soft voting
- no graph preprocessing (egonets, random walks)

Device.

- trained on device UDP traffic for each network device
- based on MAC address list
- same configuration as for the network model

Autoencoder is smaller than for the TCP models.

Brute Force

- trained on sliding-window traffic
- ensemble based on IF, KNN, LODA, LOF, OCSVM
- no AE, small dataset
- soft voting
- graph preprocessing: egonets and random walks
- trained only on graph features (not mixed)

Lateral Movement

- trained on sliding-window network traffic
- small models available also per device
- detection based on MAC and IP
- ensemble based on IF, KNN, LODA, LOF, OCSVM
- no AE, small dataset
- soft voting
- graph preprocessing: egonets and random walks
- trained only on graph features (not mixed)

Network Scan

- trained on sliding-window network traffic
- looking for ICMP traffic
- taking into account TCP RST and SYN traffic
- also the number of packets in relation with these flags
- detection based on IP
- ensemble based on IF, KNN, LODA, LOF, OCSVM
- no AE, small dataset
- soft voting

DNS and SMTP

Two models with similar setup each handling its designated traffic.

- uses the Prophet forecasting model from Facebook
- adapted for anomaly detection
- scans and identifies time series for anomalous peaks
- sliding-window with large daily and weekly seasonality (can also do yearly)
- based on historical behavior statistics such as
  - maximum different IPs and MACs
  - minimum successful requests and percentages
  - average TTL
  - average content and header size
  - targeted MACs count and their header and content size together with the average TTL

DeDDoS

On-going Soluții 2024 Grant PN-IV-P6-6.3-SOL-2024-2-0197
https://cs.unibuc.ro/~pirofti/deddos.html

References

Dumitrescu, Bogdan, Andra Băltoiu, and Ștefania Budulan (2022). "Anomaly detection in graphs of bank transactions for anti money laundering applications". In: *IEEE Access* 10, pp. 47699–47714.

Irofti, Paul, Andrei Pătrașcu, and Andrei Hîji (2022). "Unsupervised abnormal traffic detection through topological flow analysis". In: *2022 14th International Conference on Communications (COMM)*. IEEE, pp. 1–6.