WIF: Efficient Library for Network Traffic Analysis

Richard Plný
FIT CTU
Prague, Czechia
plnyrich@fit.cvut.cz

Karel Hynek
FIT CTU & CESNET
Prague, Czechia
hynekkar@cesnet.cz

Pavel Šiška
CESNET
Prague, Czechia
siska@cesnet.cz

***Abstract*—Network traffic classification and analysis are crucial for maintaining computer security. Nevertheless, the rise of encrypted traffic has made reliable threat detection increasingly challenging, requiring more complex algorithms such as heterogeneous ensembles. These types of algorithms proved to be effective in complex threat detection while maintaining high accuracy and explainability. However, their complexity and time-consuming development process limit their widespread adoption. Therefore, we created a new library called Weak Indication Framework (WIF) for the faster development of heterogeneous ensembles, which minimizes the time between attack discovery and detection capability. Moreover, WIF-based detectors are efficient enough to operate on large Internet Service Provider networks—a single detector can protect millions of users. We demonstrate the effectiveness of the WIF library through four different detectors (TOR, Cryptomining, IoT Malware, and Tunnel detector), each achieving outstanding performance and quick deployment times.**

***Index Terms*—network security, traffic analysis, threat detection, high-speed networks**

I. Introduction

Network traffic classification and analysis are essential approaches for maintaining computer security. Many intrusion detection and prevention systems (IDS/IPS), such as Suricata [1] and Zeek [2], are available for this purpose. Moreover, both are customizable, and users can develop their own detection rules. Nevertheless, with the rise of encrypted traffic, reliable threat detection has become as challenging as never before. Therefore, encrypted traffic analysis (ETA) approaches must employ more complex algorithms to maintain detection capabilities, such as Machine Learning (ML). Nevertheless, the accuracy and reliability of ML are not always sufficient. Therefore, Uhříček et al. [3] proposed a novel approach to heterogeneous ensemble.

These ensembles do not rely on a single detector but use a combination of various and simple heterogeneous detectors (modalities), each working on different principles—we denote them as weak indicators. The output of these weak indicators is then combined together for final detection. The use of multiple weak indicators then results in better detection performance, robustness, and explainability.

The main disadvantage of the weak indication principle is the complexity and timelines of developing novel detectors. Instead of designing a single monolithic detector, we must design multiple of them and then develop a results-fusion function. Therefore, we are presenting a Weak Indication Framework (WIF) to accelerate the development of heterogeneous ensembles and shorten the deployment time of emerging threat detectors.

WIF is a novel library containing the essential primitive blocks that can be reused in the detection pipelines. Apart from multiple classifiers (such as regular expression, Machine Learning, blocklist, and more) and results-fusion functions, WIF also has storage capabilities to store detection context for a longer time period. Despite the fact that WIF was developed for high-performance deployments such as Internet Service Provider (ISP) networks or data centers, it can also be easily used to protect smaller networks. The library has been released under an open-source license and is available on our GitHub[1]. Moreover, to showcase the usage of the WIF library, we have developed four different network threat detectors that utilize the weak-indication approach.

The rest of the paper is organized as follows: Section II provides an overview of related work and existing methods. Section III describes the proposed library. Section IV introduces possible use cases of the proposed library with performance evaluation. Section V concludes the paper.

II. Related Work

There are many approaches aiming for high-accurate network threat detection. We can divide the approaches into two distinctive categories: Network Traffic Analysis methods that mainly utilize Machine Learning and IDS/IPS that rely mainly on a set of rules provided by the operators.

A. Network Traffic Analysis

Many ETA methods had been developed, categorized by Pacheco et al. [4] into Payload inspection, Statistical-based techniques, Behavioral techniques, and ML techniques. However, this field is dominated by Machine Learning, mainly due to the current vast usage of encryption. As also mentioned by Pacheco et al. [4], ML is highly suitable for this task.

However, Possebon et al. [5] write that as such, a single detection and classification process is unlikely to be effective. Therefore, they emphasized the usage of meta-learning and ensemble techniques. The experiments described by Possebon et al. [5] focus on several supervised classifiers and meta-learning techniques for a combination of their outputs.

According to Buczak et al. [6], a *weak learner* is "one that consistently generates better predictions than random".

[1]https://github.com/CESNET/WIF

Then, ensemble methods use multiple weak learners to create a stronger one, with more reliable predictions. This is achieved by combining multiple hypotheses to form a better one than the best hypothesis alone. A *weak learning algorithm* can be trained to combine weighted results of weak learners. These methods can be used for attack detection (DoS, scanning), botnet detection, and anomaly detection [6].

Aceto et al. [7] proposed a Multi-Classification System (MCS) consisting of state-of-the-art (base) classifiers and an intelligent combination of their results. They created a robust system that is able to overcome the deficiencies of a single classifier and improve the overall outcome of the classification. It was shown that MCS achieved a 9.5% performance gain over the best base classifier.

Several heterogeneous systems for ETA were proposed. Uhříček et al. [3] proposed Botnet Analysis (BOTA) for detection of IoT-based malware. The BOTA system utilizes eight weak classifiers whose output is fed to a rule-based combination with 100% recall achieved. Furthermore, Plný et al. [8] proposed a heterogeneous system for the detection of cryptomalware and cryptomining in general. Three weak classifiers are combined together via a sophisticated meta-classifier.

However, traffic can change over time, such as the appearance of a new application or a change in the behavior of an existing application, as described by Pacheco et al. [4]. Moreover, only a change in distribution can cause ML models to worsen. Therefore, it might be necessary to retrain the model over time. Pešek et al. [9] proposed a system for continuous monitoring of deployed ML models and automatic retraining when the accuracy of the model drops. Therefore, the accuracy of deployed models is kept even over long periods.

B. IDS/IPS

Suricata IDS [1] is an open-source IDS with a high-performance and signature-based detection system. Moreover, it can be customized by Lua scripts. However, only basic scripts are supported, and more advanced detection methods cannot be implemented in this way.

Snort [10] is an open-source IPS, currently developed under Cisco. Similarly to Suricata, it also uses rule sets to look for suspicious traffic. In addition, Lua scripts can be used to extend Snort's functionality. But again, scripts are rather simple and do not allow the implementation of more complex detection methods.

Zeek [2] is an open-source network analysis framework for network security monitoring. It can be extended with user-defined scripts written in Zeek's own Turing-complete language. Therefore, support for new detection methods is possible. However, script performance might be insufficient for high-speed networks.

Cejka et al. [11] proposed a system for both online and offline network flow processing and analysis called NEMEA. The system consists of stand-alone modules for detection, traffic filtration, reporting etc. Moreover, libtrap and libunirec are libraries for used for module interconnection. The users can deploy only specified modules and interconnect them any way they want. In addition, NEMEA aims at operation on ISP-level networks. However, detectors for this framework must be developed from scratch.

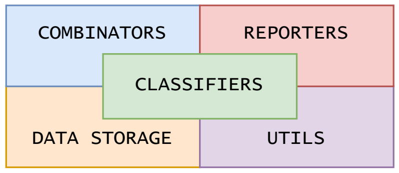


Fig. 1: Composition of the Weak Indication Framework

We did not identify any existing tool that would support the implementation of robust and complex detection methods while maintaining sufficient performance. However, the development of a highly tailored detector for the specified task is needed in most cases.

III. Weak Indication Framework

Weak Indication Framework (WIF) is a C++ library providing the most commonly used methods for network traffic analysis and threat detection. It uses a modular design and supports the development of heterogeneous classification and detection methods.

A. Design

Weak Indication Framework consists of several groups of objects, as depicted in Figure 1. Each group serves a specific purpose: *classifiers* provide methods for ETA, *combinators* provide methods for combination of weak indicators, *reporters* are used for additional information output, *data storage* contains types for data representation, and finally *utils* contain additional functionality used by other parts of the library, but are made available to the library users as well.

Moreover, classifiers, combinators, and reporters have their own base classes, which define a common interface for each group. Therefore, it is easy to replace a classifier or a detector in an existing method or to add a new one.

Classifiers and combinators are mainly used by ETA to build and implement complex detection methods. The interconnection of such instances is fully up to the user: multiple classifiers can be used and supplied to a combinator whose output can be processed through a classifier again.

B. Detection Capabilities

IP blocklist-based detection is a fundamental approach for the detection of network security threats and traffic classification. WIF contains `IpPrefixClassifier` implementing this highly efficient approach due to utilization of binary search.

Another essential method is pattern matching. WIF provides this method via `RegexClassifier` performing pattern matching via regular expressions. Regex backend from the boost library[2] is used to perform efficiently, up to 10 times faster than with the backend from the standard library.

Machine Learning is currently considered state-of-the-art when it comes to ETA, and scikit-learn library[3] is arguably the widely used tool used by researchers for this task. `ScikitMlClassifier` can be used to perform classification through basically any ML model from the scikit library. Classification of buffers (with at least 10 000 elements) is recommended to maintain sufficient throughput.

WIF addresses the problem of degrading ML models by the utilization of ALF [9]. `AlfClassifier` creates a connection with ALF and performs ML-based classification. It can detect a new ML model version on disk (dropped by ALF), perform model reload, and always use an up-to-date model. Therefore, used models are protected from degradation and maintain steady accuracy over time.

`AverageCombinator` can be used to obtain an average of weak results, `SumCombinator` to obtain the cumulative sum. `MajorityCombinator` provides a combination by finding the prevalent element in the supplied data. Moreover, `BinaryDSTCombinator` provides a combination via Dempster-Shafer Theory [12], [13], a special theory of probability.

Reporters can be used to send additional data from WIF-based detectors. `UnirecReporter` sends data to output unirec [11] interface. For example, `AlfClassifier` uses `UnirecReporter` to send needed data for the ALF.

C. Internals

Firstly, data must be prepared. WIF has its own structure for data representation called `FlowFeatures`. It can hold only supported types and a pre-defined number of values. This type can be passed to `classify()` method of classifier objects. However, source indexes to the `FlowFeatures` object must be initialized by `setSourceFeatureIDs()` before the first call of `classify()`. Classifier then only works with the set indexes of `FlowFeatures` object. This is important since we only want `RegexClassifier` to work with indexes holding `std::string` or `ScikitMlClassifier` to only use selected fields holding `double` as ML features.

The intended use is that data is received in a loop, transformed into a `FlowFeatures` object, and run through classifiers and combinators. In the end, either an alert is sent, or original data is sent out together with a new field(s) representing the classification output. `FlowFeatures` object should maintain its composition: same features on the same indexes throughout the process. That way, `setSourceFeatureIDs()` is called only once at the start, and then `FlowFeatures` can be passed periodically to the classifiers.

Classification results can be either binary (true/false; for example, the presence of an IP address on a blocklist) or a vector of numbers (array of probabilities for each class,

[2]https://boost.org
[3]https://scikit-learn.org

```
1  std::vector<WIF::IpPrefix> blocklist;
2  blocklist = load("blocklist.txt");
3  IpPrefixClassifier clf(blocklist);
4  clf.setSourceFeatureIDs({
5      SRC_IP_ID,
6      DST_IP_ID
7  });
8  // Processing loop
9  UnirecRecordView rec;
10 while (rec = inputIfc.receive()) {
11     auto flow = recordToFlowFeatures(rec);
12     auto result = clf.classify(flow);
13     if (result.get<double>() > 0) {
14         signalWarning();
15     }
16 }
```

Listing 1: Example of blocklist-based detection via WIF

from ML). Therefore, the return type of the `classify()` method is our own `ClfResult` class, which can hold either a `double` or `std::vector<double>`. Each classifier must define what value is stored in the `ClfResult` upon return and what its meaning is.

Combinators work with a vector of doubles. Their method `combine()` takes `std::vector<double>` and returns a single `double` value. No method must be called prior to the first call of `combine()`. All the initialization must be performed in the combinator's constructor.

D. Example

Listing 1 demonstrates how a simple blocklist-based detector would be implemented with the utilization of WIF. Firstly, a blocklist is loaded (line 2) and a new instance of `IpPrefixClassifier` is created and initilized (line 3). Method `setSourceFeatureIDs()` on the line 4 tells the classifier which fields should be processed. We propose to use the NEMEA system for receiving flows, specifically libtrap and libunirec. Records from the input unirec interface are received in a loop and transformed to WIF's internal data storage object (called `FlowFeatures`, line 11). Classification is performed on the line 12. If the result is positive, a method is called for signaling that blocklisted communication was detected (line 14).

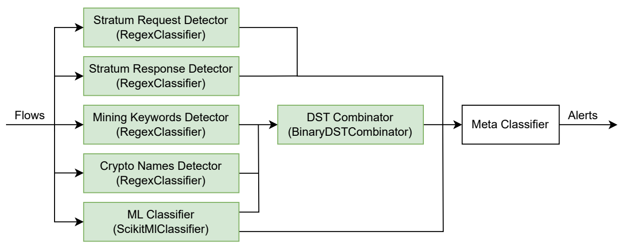
IV. Use Cases

We designed and implemented several detectors based on the new WIF library to demonstrate its usefulness. Detectors were based on the Weak Indication Framework and NEMEA libraries for interconnecting the modules.

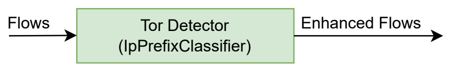
Figure 2 depicts high-level schemes of the implemented detectors. The green color indicates a component from the WIF and, therefore, a reused code that would have to be implemented otherwise.

A. Cryptomining Detector (DeCrypto)

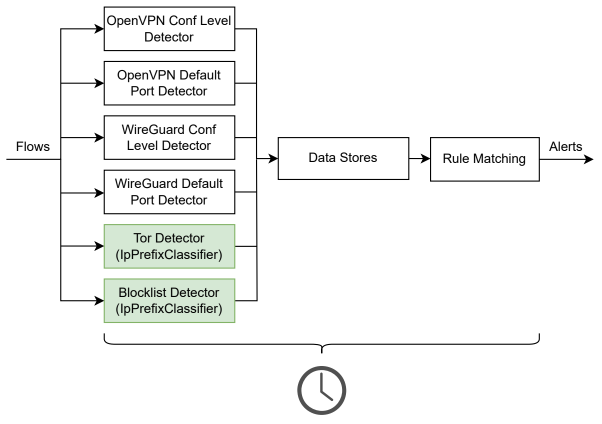
Cryptomining detector showcase the usage of ML and regex-based traffic classification. It is based on our previous work described in [8]. We reimplemented this detector in C++


Flows
Stratum Request Detector (RegexClassifier)
Stratum Response Detector (RegexClassifier)
Mining Keywords Detector (RegexClassifier)
Crypto Names Detector (RegexClassifier)
ML Classifier (ScikitMlClassifier)
DST Combinator (BinaryDSTCombinator)
Meta Classifier
Alerts

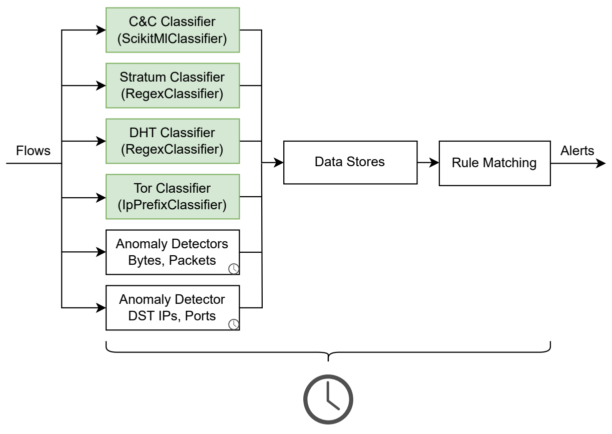
(a) Cryptomining detector



(b) Tor detector


Flows
OpenVPN Conf Level Detector
OpenVPN Default Port Detector
WireGuard Conf Level Detector
WireGuard Default Port Detector
Tor Detector (IpPrefixClassifier)
Blocklist Detector (IpPrefixClassifier)
Data Stores
Rule Matching
Alerts

(c) Tunnel detector


Flows
C&C Classifier (ScikitMlClassifier)
Stratum Classifier (RegexClassifier)
DHT Classifier (RegexClassifier)
Tor Classifier (IpPrefixClassifier)
Anomaly Detectors Bytes, Packets
Anomaly Detector DST IPs, Ports
Data Stores
Rule Matching
Alerts

(d) IoT malware detector

Fig. 2: Developed WIF-based detectors showcasing usability across various use-cases

rather than Python with the use of WIF. The scheme is depicted on Figure 2a. Basically, the whole module is composed of WIF components apart from the final meta classifier, which performs the fusion of weak indications and provides a final prediction.

Furthermore, the detector can utilize ALF. When a special CLI argument is set, the Cryptomining detector will send additional data required by ALF to the secondary unirec interface and will use retrained models, to prevent model aging. This setup is currently being evaluated on the national CESNET3 network.

B. Tor Detector (TorDer)

Tor detector demonstrates a blocklist-based classification. It is composed of a single WIF component: `IpPrefixClassifier`. It is depicted on Figure 2b. Both Tunnel and IoT Malware detectors would perform detection of Tor communication, therefore a stand-alone module was implemented which serves as a prerequisite to the other two.

Moreover, the detector can track modification changes of the blocklist source file. When a file change is detected, a reload is performed to obtain the newest version. Therefore, the detector uses up-to-date blocklists without the need to restart.

C. Tunnel Detector (TunDer)

The Tunnel detector is an example of a method that aggregates and observes weak indicators in flow data obtained externally rather than performing the detection itself. Therefore, flow contains all the needed information and results, and no complex detection is required. The indicators are observed for a specific time interval. The final combination and detection is performed at the end of the time window.

The Tunnel detector looks for covert communication tunnels such as Tor and Virtual Private Networks (VPNs). The scheme is shown in Figure 2c. The user defines IP ranges that will be observed, and the detector then aggregates weak indications for each IP address from the defined ranges. The detector works in 15-minute intervals by default.

Weak indicators used by this detector come from several sources. The first is the output of the TOR detector. Then, confidence levels are observed, produced by the ipfixprobe VPN plugins[4] (`openvpn` & `wireguard`). Port-based detection is performed by looking for default values: 1194 for OpenVPN and 51820 for WireGuard. Finally, users can provide their own blocklist for their specific use cases.

Detection is performed at the end of the time window. Firstly, intermediate results are obtained from each detector resulting in True/False values. Such results are then passed to the defined boolean rules. An alert is sent to the output unirec interface for each satisfied rule with information about the IP address, a string representation of the matched rule, intermediate results, and their explanations.

[4]https://github.com/CESNET/ipfixprobe

D. IoT Malware Detector (MalDer)

IoT malware detector demonstrates full-scale usage of the WIF library. It is based on the BOTA system proposed by Uhříček et al. [3]. The detector implements a complex and robust classification method of IoT-based malware, such as Mirai and Gafgyt. Similar to the Tunnel detector, target IP ranges are defined, and the detector observes IP addresses from these ranges. Moreover, the Malware detector works in 5-minute intervals as the original BOTA system.

WIF components were successfully used for the following indicators: detection of Command and Control communication via ML, pattern-matching discovery of DHT and Stratum protocols, and detection of Tor. Anomaly detectors had to be implemented manually because WIF does not yet support anomaly detection. However, the WIF library dramatically eased the development of this module. It is also more efficient and achieves higher throughputs because it is implemented in C++.

E. Performance

The throughput of each detector was evaluated in the experimental environment on anonymized data captured directly on the national CESNET3 network. The tests were performed on a virtual machine running Oracle Linux 8 with a 4-core CPU (Intel 6226R@2.90GHz) and 5.75 GB RAM in a scenario very close to the deployment environment. Each detector was executed 100 times and the results were averaged.

Results are summarized in the Table I. TorDer and TunDer achieved very high performance and are capable of operation on ISP-level networks without any problems. The Cryptomining detector still achieves high performance but is lower than the two previously mentioned. This is caused by the increased complexity of the detector. The last evaluated detector, MalDer, showed significantly less throughput. The detector is far more complex than the others. Moreover, four separate instances of anomaly detectors have a significant impact on the throughput. Despite that, the performance is still sufficient for deployment to ISP-level networks.

TABLE I: Single-thread performance overview

	Flows per second			
Detector	**Min**	**Average**	**Median**	**Max**
TorDer	395 430.58	723 882.8	725 806.45	747 508.31
DeCrypto	99 009.9	109 070.28	109 890.11	113 122.17
TunDer	641 025.64	673 755.35	679 758.31	700 934.58
MalDer	16 055.37	17 926.86	18 192.11	18 792.28

V. Conclusion

The rise of encrypted traffic demands increasingly complex network security detectors, which are time-consuming to develop. This paper introduced the Weak Indication Framework (WIF), a library designed to streamline the development process and reduce the time between the occurrence of network threats and the deployment of detectors. WIF includes state-of-the-art methods for (encrypted) traffic classification and threat detection. It is implemented in C++ for high efficiency. This allows modules built using WIF to be deployed in national ISP-level networks with high-speed lines. WIF supports a wide range of detection methods, from simple single-classifier approaches to advanced methods processing multiple indicators over defined periods. The library is publicly available on GitHub[1] and is free for use. Its modular design allows users to implement custom classifiers and combinators tailored to specific needs, integrating them seamlessly into WIF.

We demonstrated WIF's capabilities with four distinctive detectors: Cryptomining detector, TOR detector, and Tunnel detector, as well as an IoT malware detector. All achieved high flow throughput, making them suitable for deployment in large ISP networks.

Future work includes extending the library with combination methods such as the Behavior-Knowledge Space method [14]. We also plan to develop additional detectors for community use.

Acknowledgment

This work was supported by the Ministry of the Interior of the Czech Republic, grant No. VJ02010024: "Flow-Based Encrypted Traffic Analysis," and also by the Grant Agency of the Czech Technical University in Prague, which is funded by the Ministry of Education, Youth and Sports of Czech Republic, grant No. SGS23/207/OHK3/3T/18.

References

[1] The Open Information Security Foundation, "Suricata," https://suricata.io, 2024, [cit. 31-07-2024].

[2] The Zeek Project, "Zeek," https://zeek.org, 2024, [cit. 31-07-2024].

[3] D. Uhříček, K. Hynek, T. Čejka, and D. Kolář, "Bota: Explainable iot malware detection in large networks," *IEEE Internet of Things Journal*, vol. 10, no. 10, 2023.

[4] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, 2019.

[5] I. P. Possebon, A. S. Silva, L. Z. Granville, A. Schaeffer-Filho, and A. Marnerides, "Improved network traffic classification using ensemble learning," in *2019 IEEE symposium on computers and communications (ISCC)*. IEEE, 2019.

[6] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, 2016.

[7] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *Journal of Network and Computer Applications*, vol. 103, 2018.

[8] R. Plný, K. Hynek, and T. Čejka, "Decrypto: Finding cryptocurrency miners on isp networks," in *Nordic Conference on Secure IT Systems*. Springer, 2022.

[9] J. Pešek, D. Soukup, and T. Čejka, "Active learning framework for long-term network traffic classification," in *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, 2023.

[10] Cisco, Inc, "Snort," https://snort.org, 1998, [cit. 31-07-2024].

[11] T. Cejka, V. Bartos, M. Svepes, Z. Rosa, and H. Kubatova, "Nemea: A framework for network traffic analysis," in *2016 12th International Conference on Network and Service Management (CNSM)*, 2016.

[12] A. P. Dempster, "Upper and Lower Probabilities Induced by a Multivalued Mapping," *The Annals of Mathematical Statistics*, vol. 38, no. 2, 1967. [Online]. Available: https://doi.org/10.1214/aoms/1177698950

[13] G. Shafer, *A Mathematical Theory of Evidence*. Princeton University Press, 2021. [Online]. Available: https://doi.org/10.1515/9780691214696

[14] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2014.