MCRIT: The MinHash-based Code Relationship & Investigation Toolkit

Daniel Plohmann[1], Manuel Blatt[1], and Daniel Enders[1]
[1] *Fraunhofer FKIE*

This paper was presented at Botconf 2023, Strasbourg, 11-14 April 2023, www.botconf.eu
It is published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr/ojs

Abstract

As the number of malware attacks continually rises, malware analysts are facing an ever-increasing workload. The growing complexity of malware families and the sheer volume of new threats make it challenging for analysts to keep up with their analysis tasks.

Code similarity analysis offers high potential in this regard, helping analysts to orient themselves and to speed up analysis. While being a very active research field with many recent publications, only few of these focus on malware or support immediate practical usage, as they are rarely accompanied by public code releases.

In this paper, we present the MinHash-based Code Relationship & Investigation Toolkit (MCRIT). MCRIT is intended to serve as a framework for code similarity analysis, mainly focusing on One-to-Many (1:N) comparisons and with the ability to recognize and filter out library code. We publish MCRIT as open source, including a dockerized setup for easy deployment.

Keywords: malware analysis, reverse engineering, code similarity.

1 Introduction

As the information security is constantly evolving, malware analysts continue to face a constant influx of samples from a wide array of malware families. Recent phenomena such as the frequent rebranding observable for multiple ransomware strains or the occasional retooling of APTs make it increasingly hard to keep track.

Code similarity analysis is a means with high potential in this regard to help analysts navigate the space and accelerate their analysis. It has become a popular research topic in recent years, easily explained by its multi-dimensional use in security context, such as vulnerability discovery and malware analysis. Especially when diving deep into malware families, focusing on their internal evolution and relationship among each other, there is a crucial need to have this capability available.

In 2019, Haq and Caballero [1] published a survey, listing about 50 academic works on the topic that have been proposed since 2010. Sadly, many of the presented approaches have not been evaluated on larger malware data sets and few are focused on use cases like malware clustering, lineage analysis, or library recognition. They are also not tailored towards immediate practical usage, as they are rarely accompanied by public code releases. Other established tools like BinDiff [2] and Diaphora [3] focus only on direct comparison of single files with each other.

With this paper, we present and publish MCRIT, the MinHash-based Code Relationship & Investigation Toolkit. MCRIT is intended to serve as an open source framework for code similarity analysis, mainly focusing on One-to-Many (1:N) comparisons and with the ability to recognize and filter out library code. MCRIT was originally developed since 2018 as a pure research-oriented evaluation framework [4] with support for quasi-identical and fuzzy code matching. Over the last two years, it was extended towards becoming applicable in practical malware research context, with internal improvements to efficiency and also the addition of a user friendly web front-end and dockerized deployment.

We envision four primary use cases for MCRIT. First, the toolkit is intended to aid with malware family and library code differentiation. Being able to dis-

cern these characteristics can provide massive benefits that accelerate triage and analysis, as analysts can direct their focus on code relevant to their analysis goals. Second, MCRIT provides the capability to isolate code only found in a given sample or family. This allows deriving code artifacts that could be useful for hunting of similar specimen but also for the generation of identification rules, similar to our previously presented results on YARA Signator in 2019 [5]. Third, MCRIT is intended to serve as a system for lead generation when aiming for the discovery of potentially unknown links across samples and families. In this case, MCRIT can filter and search for matches between pairs or small sets of families that still exhibit a high matching score, directing an analyst's attention towards potential connections in their code. Fourth, as MCRIT is also able to store meta data for functions, it can be used for the transfer of function labels. This is a frequent operation when wanting to speed up analysis by reusing existing analysis results, e.g. during the investigation of multiple samples of the same family or when exchanging information in a team.

In summary, our primary contributions with this paper are:

- We present MCRIT, a framework that enables efficient One-to-Many code comparisons using quasi-identical and fuzzy code matching.
- We discuss a series of case studies to underline the benefits that such a system can bring to malware analysis.
- We provide an implementation of MCRIT as open source, including a dockerized setup for easy deployment [6].

The remainder of this paper is structured as follows. We provide an overview of related work in Section 2. The main content of the paper is divided in two parts: Section 3 describes the matching methodology and framework design of MCRIT and Section 4 provides a series of case studies to illustrate the four primary use cases we currently envision for MCRIT. To conclude the paper, Section 5 discusses limitations and future work while Section 6 summarizes the paper.

2 Related Work

As already outlined in the introduction, (binary) code similarity analysis provides immense practical value to various application contexts. As a consequence, research on the topic has become highly popular in recent years. Haq and Caballero [1] have provided a comprehensive survey on the topic that we recommend for study.

According to their survey, the first two binary code similarity approaches were published in 1999. Baker et al. [7] presented ExeDiff, an approach intended to capture secondary changes introduced by the compiler during compilation of the same source code in order to reduce patch size, and Wang et al. [8] published BMAT, a tool to propagate profiling information between sequential releases of Windows DLLs.

In the following decade, among the seven papers on the topic are the two seminal publications by Dullien and Rolles. Dullien introduced an approach that uses structural properties of functions and the call graph for matching of functions between programs [9] and they both proposed using the Small Primes Product (SPP) algorithm for deriving a representation of basic blocks that is robust against instruction reordering [10]. Both of these techniques are key methodologies used in Zynamics' BinDiff [2], which has been a de-facto standard plugin for applied binary diffing using IDA Pro [11]. Notably related to the scope we cover with this paper, Xin et al. [12] published their method SMIT, which was the first known One-to-Many matching approach intended to find similar malware in a database, given a reference sample.

In the period of 2010-2019, Haq and Caballero counted 52 publications on code similarity, documenting its rise in popularity. Generally since this time, the focus shifted primarily towards the search of binary code with the intent of (re-)discovering vulnerabilities, potentially across CPU architectures. This was accompanied with wide-spread adoption of machine learning techniques. Related to this paper, Jin et al. [13] were the first to apply MinHashing in order to enable scalable code comparisons. Ding et al. [14] presented Kam1n0, another approach using a Locality-Sensitive Hashing (LSH) approach, which also enables matching on basic block level.

In practical context, Zynamics apart from BinDiff also published VxClass [15] and BinCrowd [16], systems based on the techniques from BinDiff to enable One-to-Many approaches for code similarity. Kornau-von Bock und Polach et al. [17] presented concepts for large-scale code similarity analysis within Google. Koret published Diaphora, another plugin for IDA which reimplements the techniques used in BinDiff in pure Python but also provides a variety of additional methods and features to enable effective diffing [3].

3 MCRIT

In this section, we now provide an overview of the MinHash-based Code Relationship & Investigation Toolkit (MCRIT). The code similarity analysis and matching approach used in MCRIT was developed in the context of the PhD thesis by Plohmann [4], which includes a detailed discussion and evaluation of design choices, parameterizations, and matching algorithm performance.

In the following, we therefore limit ourselves to a shortened summary of the core algorithms and rather provide a detailed description of the components that have been developed to turn MCRIT into a usable framework as well as the interfaces it provides for interaction.

3.1 Motivation

One of the central drivers motivating the creation of MCRIT is our ongoing malware collection and preservation project Malpedia [18]. While Malpedia has already become a widely used resource in the cyber threat intelligence context, we always believed that especially the curation of the underlying data set of accurately identified and unpacked reference malware samples holds significant value that will be eventually unleashed in the future. In that sense, a lot of the work on Malpedia was and is forward-looking, believing that the data set will unfold its full potential as novel analysis methods and tools become available that allow benefiting from the effort spent in the past. We believe that examples for success of this philosophy are e.g. our own research on WinAPI usage in malware using ApiScout [19] and automated generation of code-based YARA signatures [5], but also studies like the one by Oosthoek and Doerr [20] who used Malpedia to map out prevalance of TTPs as captured in the MITRE ATT&CK framework [21].

Revisiting the survey on binary code similarity by Haq and Caballero, we noticed that apart from the work by Alrabaee et al. on FOSSIL [22], there is an underrepresentation of papers with studies using many different malware families and/or how to use reference library code to ease malware analysis. Similarly, we generally perceived that there is a lack in freely available tooling, especially as open source to address these aspects.

For this reason, we started our work on MCRIT, intended to serve as an open source framework for code similarity analysis, mainly focusing on One-to-Many comparisons and with the ability to recognize and filter out library code. As requirements for MCRIT, we defined that we wanted to have interpretable similarity measures including a simple and efficient representation for indexed code as well as scalability into millions of functions.

The core idea of MCRIT is to reuse known, proven techniques for code similarity analysis and use it to explore novel approaches for result interpretation, specifically when incorporating information about match origin (e.g. from benign software/library) and when aggregating matching results, being able to assign frequency weights for popularity of code across the whole database.

3.2 Code Similarity Analysis

MCRIT is built around two core matching techniques, applied to code similarity analaysis: PicHash and MinHash.

Position-Independent Code Hashing (short: PicHash) has been proposed by Cohen and Havrilla [23] and is a method that allows the projection of units of disassembled code such as a function or a basic block into a hash. Prior to hashing, the code is transformed by replacing its concrete addressing with wildcards, thus making the code position-independent as suggested by the name. The technique was demonstrated to be able to massively reduce redundancy of functions in binary data sets, for example by a factor of 40 on their test set of 4 million executables.

Since look-ups using this technique can be done as exact hash matches, it can be implemented in algorithmic complexity $\mathcal{O}(1)$ using a hash map or $\mathcal{O}(\log n)$ using a B-tree database index, which makes it incredibly fast even in large data sets. As the method is prone to collisions in small functions, MCRIT uses PicHash to capture functions with ten or more instructions, as well as basic blocks of size four instructions and larger. The PicHashes used in MCRIT are the numerical representation of the first 64 bits of the SHA256 hash calculated over the wildcarded instruction bytes (cf. [4]).

MinHash on the other hand has its roots in text document similarity analysis. Originally described by Broder [24] as a fast approximation of Jaccard set similarity, it has served as the foundation for the web search engine AltaVista but also found application in other contexts such as genome matching. Combining MinHash with locality-sensitive hashing (LSH) methods has been documented to provide great scalability and is able to provide efficient lookups for single entities in $\mathcal{O}(\log n)$ when accepting a certain degree of error. This implies that full pair-wise comparisons (i.e. on malware sample level) can be conducted in $\mathcal{O}(n \log n)$, opposite to $\mathcal{O}(n^2)$ when building off a system that only supports One-to-One instead of One-to-Many queries. MinHash was first applied to code similarity by Jin et al. [13] and found use in several follow-up works.

The expected accuracy of using minhashing for code similarity is directly tied to the features used for the description of the code to be indexed. The overall MinHash indexing procedure of MCRIT is summarized in Fig. 1. MCRIT uses token-based and metrics-based features to capture the characteristics of functions.

The token-based features abstract concrete instructions into their semantic operation (mnemonic) and operand classes, which are then sorted to turn them into permuted code n-grams. This methodology has been successfully applied before by Karim [25] as well as Walenstein et al. [26] who both used n-grams and n-permutations (short: n-perms) as representation for code comparisons, while Adkins et al. [27] presented a similar concept for instruction abstraction. Metrics-based features capture statistical properties of the functions, a concept that equally has been proposed before, e.g. by Eschweiler et al. [28], who used it as input for kNN clustering of code in their system discovRE. The data points used in MCRIT are the size of their largest basic block, the number of calls, the count and share of instructions belonging to the most common semantic instruction classes, as well as the stack size. Note that the parameter selection has been optimized to work well within the same bitness but does not work well when comparing 32 bit with 64 bit code. The reason for this is that Malpedia consisted mainly (90%+) of 32 bit samples when we started the evaluation and wanted to optimize for that over generaliza-

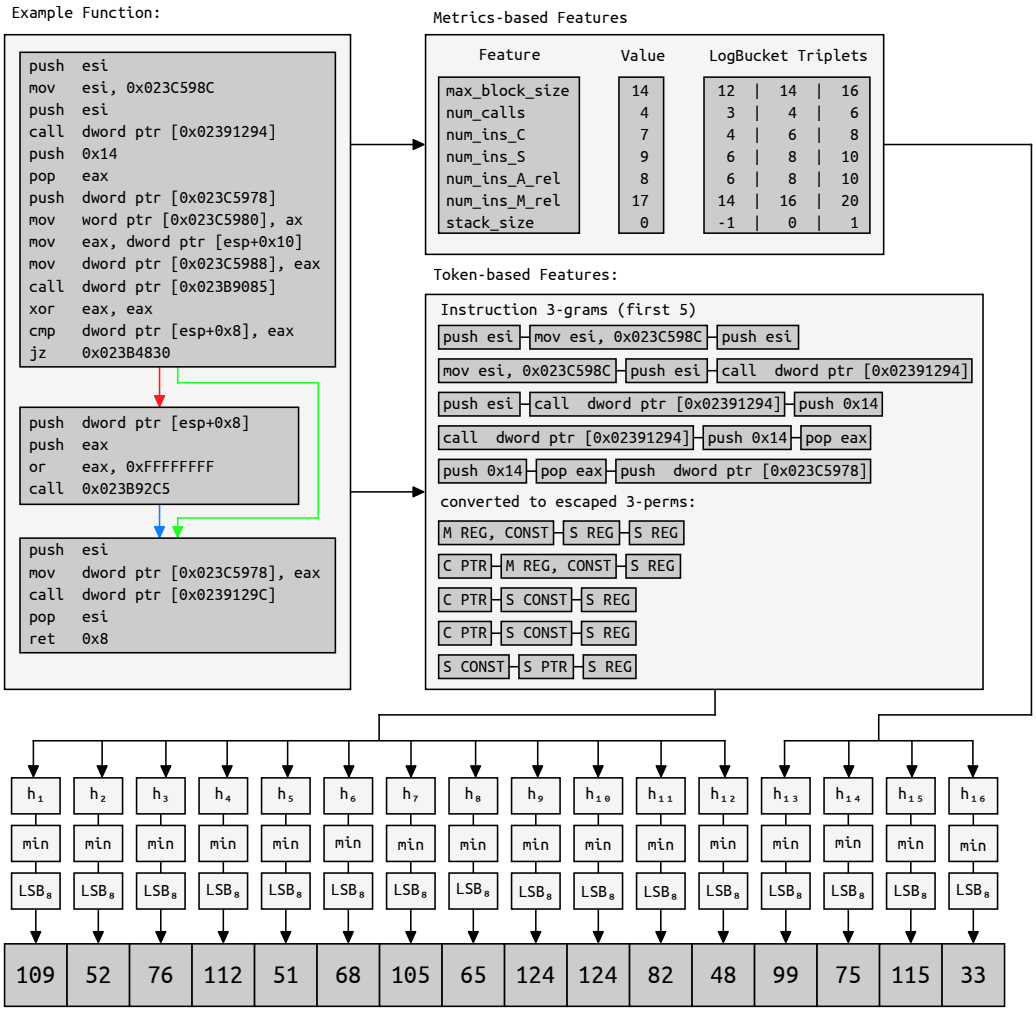

Figure 1: A complete example for calculation of an MCRIT MinHash signature, segmented into 12 token-based and 4 metrics-based entries [4].

tion. As mentioned, the design and derivation of these features are extensively explained and evaluated in [4], alongside an optimization of the overall system configuration and the meta parameters used in MCRIT. For this paper, we omit the full details and refer to the thesis for all further information (cf. [4], p. 131f.).

For matching, MinHash sequences as shown at the bottom of Fig. 1 are directly compared with each other. For this, the number of matching entries at same offsets (e.g. both MinHashes having a 109 in the first position etc.) is counted and divided by the signature length, resulting in a matching score between 0 and 1. The MinHash configuration used in MCRIT currently uses a signature length of 64 bytes. To allow efficient matching, candidates are identified using the so-called banding technique [29] and processed in parallel.

As outcome of this operation, all matches with their respective MinHash similarity score are calculated for each input function, which can then be aggregated to sample level. Note that for effective operation of the MCRIT system, it allows and essentially requires to store meta data indicating a malware family label as well as a boolean flag if a sample or family is considered benign library code.

Now during result aggregation, MCRIT has the ability to incorporate knowledge about reference library code and to perform occurrence frequency analysis with respect to all other samples and families identified during matching, which can also additionally serve as a generic heuristic for the detection of common code, such as third party library code. In general, the occurrence frequency analysis during matching allows filtering and focusing on functions with a specific interest and from multiple viewpoints, enabling what we consider the primary use cases of the system.

3.3 Framework Overview

We now continue with a description of how the above code similarity analysis methodology has been implemented into a practically usable framework, as shown in Fig. 2. Overall, Python has been used as the primary programming language.

The previously described code similarity analysis methodology is divided into two core components, which share a common codebase [30]: MCRIT Server and MCRIT Worker. The MCRIT Server provides an interface to all functional aspects of the system, which

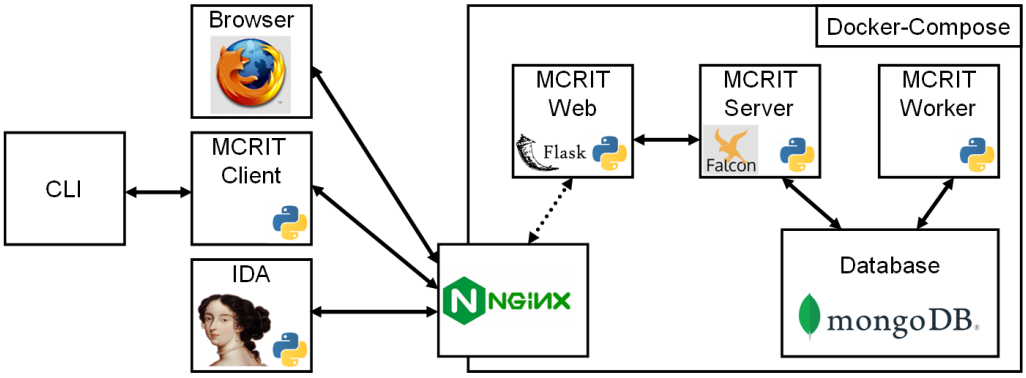

Figure 2: Overview of the components of the MCRIT framework.

is exposed in the form of a REST API, implemented using the Falcon framework [31]. Most API endpoints that just respond to simple information queries, e.g. on families, samples, or functions are realized synchronously, while tasks that are potentially tied to longer run time, such as disassembly, indexing, and similarity matching lead to the creation of jobs, which are then stored in a queue and handled asynchronously. This is the primary role of the MCRIT Worker, which takes these jobs from the queue and processes them. As disassembler, SMDA [32] is used, which had been created to provide better code coverage and accuracy for the analysis of memory dumps, which are a common representation of unpacked code in Malpedia. Because of SMDA, the system is currently limited to x86/x64 Intel code, but we expect expansion to further architectures in the future, e.g. support of .NET code has been tested internally already and is in development. The matching of MinHashes is done using NumPy [33] and parallel processing to boost system performance. It is possible to use multiple MCRIT Workers in parallel as well, as their coordination is handled implicitly through the queue.

To support larger-than-memory operations and provide persistence, MCRIT relies on a database that stores the once processed disassembly and meta data for all samples and the functions they contain. We have chosen MongoDB, as SMDA's output is already in JSON and we saw additional synergies in data to object mapping in other places of the system.

To increase usability of the system, we used Flask [34] to create a browser front-end called MCRIT Web [35]. MCRIT Web is implemented against the REST API of MCRIT Server. Apart from visualization of contents and matching results stored in the system, it adds user management and access control.

All the previously mentioned components have been containerized using Docker [36] in order to simplify setup and deployment of the framework. Docker-MCRIT [6] additionally adds NGINX [37] including default configurations to quickly enable the transfer of large files and add TLS support.

While MCRIT Web is one primary way to interact with the system, we anticipate that users may want to integrate MCRIT into existing processing pipelines or use the service on a machine-level. For this reason, we have implemented an MCRIT Client, which allows direct interaction with all endpoints of MCRIT Server's REST API directly using Python. In deployments where MCRIT server is not directly exposed to the outside, an API pass-through is possible for a subset of endpoints via MCRIT Web. A console application using MCRIT Client has been added to the package [30] as well.

As one example how MCRIT can be directly used from within analysis tools, we recently started developing an IDA Pro plugin. The current state of the plugin allows sending the disassembly from the IDA database IDB) including all function labels to MCRIT, as well as querying MCRIT for similar functions and importing labels tied to the matched functions.

We expect to further increase the usability of MCRIT as we will continue development and hopefully receive feedback from its future users.

4 Case Studies

To illustrate the capabilities of MCRIT in its current state, we will now present a number of case studies, covering different functional aspects of the system. These case studies are structured along the primary use cases we currently envision for MCRIT:

- Malware family identification and library code differentiation to accelerate triage and analysis
- Isolation of unique family code to provide means for hunting towards their characteristics
- Lead generation for discovering potentially unknown links between samples and families
- Label transfer between samples during in-depth analyses of malware families.

For the following experiments and discussion, we use an MCRIT database populated with the entire Malpedia data set as of 2022-12-09 (commit: `4ea73337`), and a collection of reference code for the Microsoft Visual C++ runtime (MSVCRT) for all versions between 6 and 2019 in multiple compilation settings [38]. In total, this leads to a database of unpacked code for about 6.200 malware samples, associated with 1.500 distinct malware families and totalling 7 million functions.

	Version		SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct		Frequency	
win.remcos	2022-02-09-v3.4.0_Pro	4501	6e33193f	cfd29c1ac5...0x00400000	32	2456	1816	1668	729	92	89	49	67
win.tclient		4398	f7a756a7	f7a756a7ca...c_unpacked	32	2592	870	545	598	35	16	8	7
win.bit_rat	2021-06-30	3959	8b964655	ba47f657a4...0x00400000	32	30723	917	678	695	41	14	5	2
win.smominru		4721	a42c7ea0	f1c36aebdc...0x00400000	32	13173	894	618	671	39	13	5	1
win.pay2key	2020-11-28	2242	d98a8ab5	93347a4779...0x00400000	32	7622	871	653	689	39	11	5	1
win.defray		468	bee81c9d	08cf8ed94c...4_unpacked	32	3508	864	632	677	38	11	5	1
win.slub	2019-02-22	1948	407c4dae	43221eb160...0x00840000	32	3541	803	639	649	38	12	5	1
win.adkoob	2017-12-26	4414	6a6260cb	e383582413...6_unpacked	32	4133	866	651	676	38	11	5	1
win.r980	2016-07-22	1692	5aee480f	32c1ddede5...b_unpacked	32	9580	834	614	662	37	11	5	1
win.webmonitor	2018-12-20	4200	969d4891	be535a8c32...0x00400000	32	7479	823	593	660	37	11	4	1

Figure 3: Best Family Match Result View; after submitting the `win.remcos` v3.8.0 test sample to the demo MCRIT database.

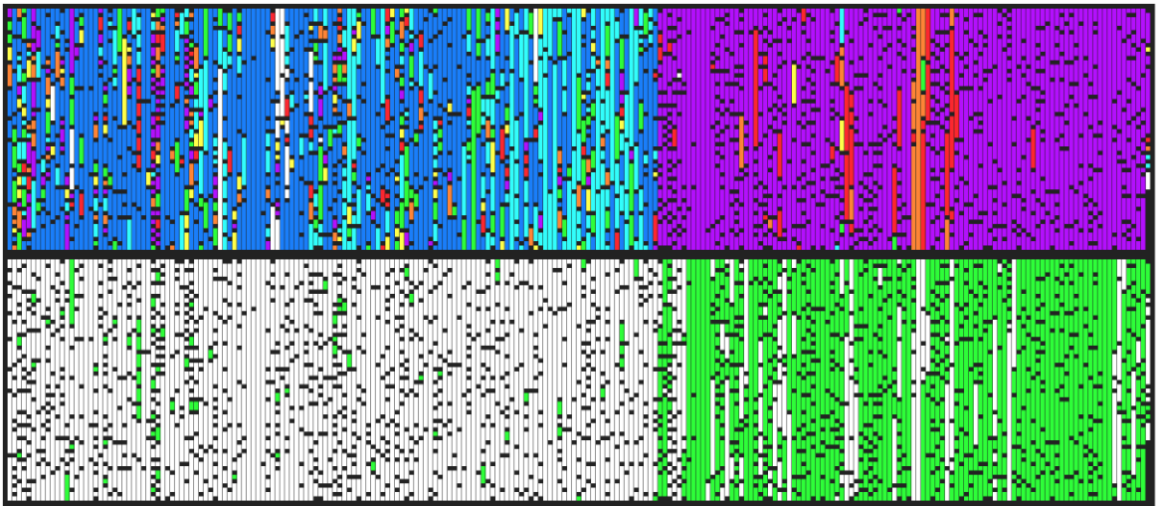

Figure 4: MCRIT Match Diagram for the `win.remcos` v3.8.0 test sample. The upper row shows function occurrence frequency, the lower row recognized library functions. The section on the right side indicates that this sample contains about 40% library functions, in this case from MSVCRT.

4.1 Malware Family Identification and Library Code Differentiation

As our first primary use case, MCRIT is intended to aid with malware family and library code identification. Generally, being able to discern these characteristics can provide massive benefits that accelerate triage and analysis, as analysts can direct their focus on code relevant to their analysis goals. On the one hand, functions immediately recognized as library code can be discarded from the scope of analysis and their usage in the program's context is often immediately understood. On the other hand, it also allows to highlight and weigh function matches pointing to code from known malware families when trying to quickly understand what changed in samples of new versions of a given malware family.

We now investigate a sample[1] of the RAT `win.remcos`, which was submitted to Malware Bazaar [39] on 2022-12-07 and assume we had not been able to identify it by other means. It has an internal version v3.8.0, while the highest version known to our database at this time is v3.4.0.

Upon submission to MCRIT, we are presented with the result shown in Fig. 3. The table shows the best matched sample per family, with the columns *Direct* and *Frequency* showing the matching score in percent. Here, the left number are scores without removing the share of recognized library functions while the right number is a matching score only calculated across the remaining non-library functions. We notice multiple things. First, as anticipated, with regard to *Direct* scores, `win.remcos` is the best matched family, with the closest sample matching 92%. We can also see that multiple other families match with up to 41%, when not using library filtering or frequency-based weights. However, looking at the last two columns under *Frequency*, we see that we still have strong matches against `win.remcos`, with 67% especially when filtering out library functions while all others drop notably.

With regard to library recognition, we developed a specific type of diagram for MCRIT that is able to visu-

[1] ae07807f71e0584e2651db6ac5ba04db40923066375ed1977ac9b5ac65f5af44

	Version		SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct		Frequency	
win.remcos	2022-02-09-v3.4.0_Pro	4501	6e33193f	cfd29c1ac5...0x00400000	32	2456	1816	1668	729	92	89	49	67
win.remcos	2022-02-09-v3.4.0_Pro	4561	cfd29c1a	cfd29c1ac5...2_unpacked	32	2514	1816	1668	729	92	89	49	67
win.remcos	2021-10-01-v3.3.0_Pro	845	fc705f6a	2ad10b64da...7_unpacked	32	2515	1664	1090	720	75	65	35	46
win.remcos	2021-10-01-v3.3.0_Pro	1968	e699445f	2ad10b64da...0x008e0000	32	2543	1664	1100	720	75	65	35	46
win.remcos	2018-06-12-2.0.5_Pro	4575	5f4ed313	c75ba39173...0x00400000	32	478	61	2	18	0	0	0	0
win.remcos	2019-09-20_2.5.0_Pro	5159	e8efb538	fe12c01213...0x00400000	32	504	61	2	18	0	0	0	0
win.remcos	2019-09-20_2.5.0_Pro	5562	0898de8a	af1cf57215...b_unpacked	32	502	61	2	18	0	0	0	0
win.remcos	2018-02-10-2.0.2_Pro	5894	aab0bb63	e2a0eadc10...0x00400000	32	467	58	0	15	0	0	0	0
win.remcos	2019-09-20-v2.5.0_Light	4683	a3e2dd92	d8b6ce5544...0x00400000	32	384	58	2	17	0	0	0	0
win.remcos	2017-11-15-1.9.3_Pro	510	b5cef2e8	0ca47d6924...0x00400000	32	440	52	0	15	0	0	0	0

Figure 5: Best Sample Match Result View, filtered to family `win.remcos`; after submitting the `win.remcos` v3.8.0 test sample to the demo MCRIT database.

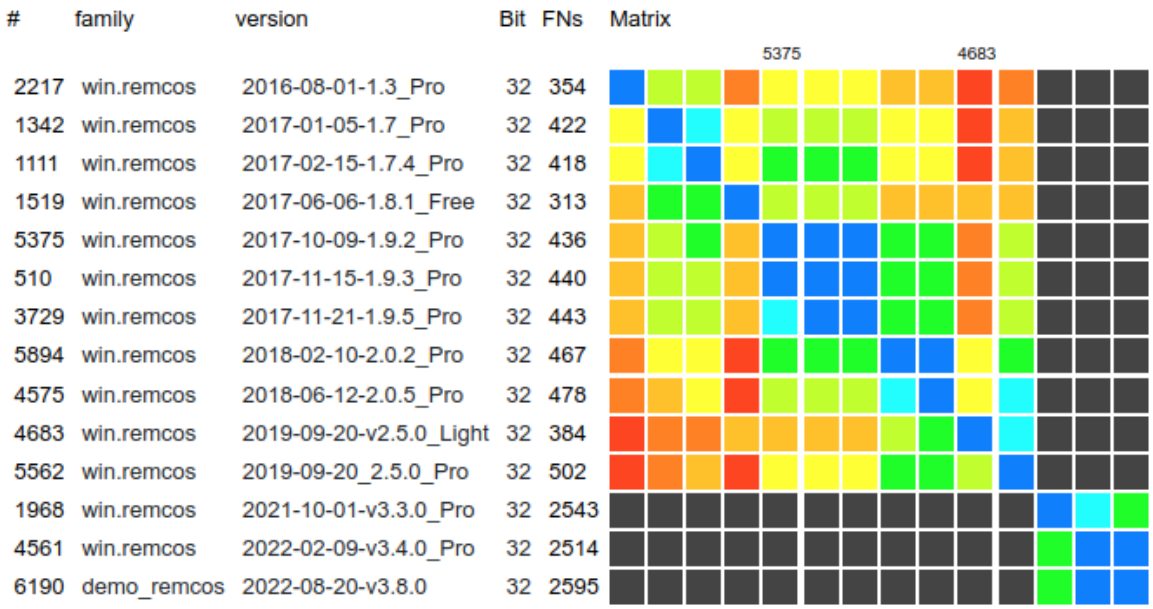

Figure 6: Cross-Compare Result Matrix for family `win.remcos`, sorted by compilation timestamps. Two clusters for major versions 2 and 3 are clearly recognizable.

alize the occurrence frequency of functions across all families and which functions have been recognized as library.

The diagram for our `win.remcos` sample is shown in Fig. 4. Generally, both rows divided by the black bar in the middle provide the same view on the whole executable but contain different information. From left to right, all functions larger than ten instructions sorted linearly by their virtual address are shown within the small columns, being divided by the little black squares. The size of the bars also indicates the function size.

The upper row shows the occurrence frequency per function across all families in the data set. The color scheme ranges from dark blue (1 family), to cyan (2-3), over green (4-7), yellow (8-15), orange (16-31), and red (32-63) to violet (64+). We can see a divide towards the right side, as in the beginning mostly functions with low occurrence in few families are found (blue), while in the "back" most matches are found in 64 and more families (violet). The second row of the diagram (only green is used for coloring) additionally indicates where known library functions have been identified in the executable. In the MCRIT demo instance, we currently only use `MSVCRT` as reference library code and it appears that about one third of the code contained in our `win.remcos` sample is associated with the standard runtime of Microsoft's Visual C++ compiler.

We also confirmed this result with IDA Pro, as the embedded FLIRT technique for library recognition produces very similar results. We expect that this in particular will become much more useful once an array of additional popular libraries such as `zlib` or `OpenSSL` have been added to an MCRIT database, as MCRIT can also serve as a label provider for function names.

As a further experiment, let us filter down into matches versus the target family `win.remcos` (Fig. 5). Furthermore, we can now see that there was apparently a significant development gap between the v2.x

```
rule mcrit_639358a4e0ff5413a77221e4 {
    meta:
        author = "MCRIT YARA Generator"
        description = "Code-based YARA rule composed from potentially unique basic blocks for the selected set of samples/family."
        date = "2022-12-09"
    strings:
        // Rule generation selected 20 picblocks, covering 19/19 input sample(s).
        /* picblockhash: 0xa4f6124d03ad806 - coverage: 15/19 samples.
         * 6a10   | push 0x10
         * 5a     | pop edx
         * 663bc2 | cmp ax, dx
         * 7505   | jne 0x405250
         */
        $blockhash_0xa4f6124d03ad806 = { 6a10 5a 663bc2 75?? }
```

Figure 7: Excerpt of an automatically generated YARA rule, based on characteristic basic blocks for 19 samples of family `win.remcos`.

and v3.x branches of `win.remcos`, as older samples almost appear unrelated.

To further understand the development of `win.remcos` over time, we can use MCRITs so-called Cross-Compare feature, in which all samples of a chosen set are compared against each other, with the result being rendered in a matching matrix. The result is shown in Fig. 6. Here, the strength of matches is indicated by color and decreases from blue (strongest) over green, to yellow, orange and red, with black being no meaningful match. Please note that while the Cross-Compare would automatically perform hierarchical clustering, we have re-arranged the order of samples as shown in Fig. 6 chronologically by their compilation timestamps.

In this matrix, we can see that indeed two bigger clusters with internal code relationship exist, one spanning versions prior to v3.x and the smaller one with the v3.x samples. We are generally presented with what appears a linear development history, meaning that matching scores get lower as version numbers increase. An interesting observation is that the two *Free* and *Light* versions each include about 100 functions (column *FNs*) less than their surrounding *Pro* counterparts, accompanied by expectedly lower matching scores. Our assumption in this regard is that some functionality has been excluded in the compilation of these versions.

Both the previously shown matching and Cross-Compare can be used for queries of arbitrary families, allowing the comparison of pairs or multiple samples at once to gain an overview of potential relationships.

With respect to computational performance, the processing time for disassembly of the 2.500 functions in our `win.remcos` v.3.8.0 sample and their matching against all of the 7 million functions of the 6200 samples in the demo MCRIT database took 4:26min on a machine with 8 virtual cores and 32GB RAM.

4.2 Isolation of Unique Malware Family Code

As a second primary use case, MCRIT provides the capability to isolate code only found in a single given family or a specific selection of samples. Similar to our presentation of and the results achievable with YARA-Signator in 2019 [5], MCRIT is able to filter down to code found only in a single family, but this time we are focusing on full basic blocks (which are already indexed with PicHashes anyway) instead of instruction n-grams.

Consequently, this provides us with a convenient way to derive code-based YARA rules. For this, we have to solve the multi-set multi-cover problem, in which we try to cover all samples of the family in question with a chosen number of basic blocks occurring in them. A minimal solution to this problem is algorithmically very hard to achieve but greedy approximation algorithms exist that make solutions feasible. The default setting here is based on our findings with YARA-Signator, where we want every sample to be covered with 10 signature strings or more.

Returning to our `win.remcos` example, we can ask MCRIT to differentiate the family against the remainder of the corpus and produce a result as shown in Fig. 7 In the reference collection, MCRIT is able find a total of 10,028 unique basic blocks across all `win.remcos` samples that are not found in any samples of other families and it produced a selection of blocks that covers all 19 input samples. We can see in the comment, that a total of 20 basic blocks were sufficient to cover all samples with at least 10 basic blocks each, as many blocks like the one shown here as example cover a multitude of samples at once (15/19). This is also in line with our previous observation that a major rewrite happened within `win.remcos` between v2.x and v3.x and MCRIT identifies only one single basic block that is found in all 19 samples.

To validate the quality of the automatically generated YARA rule, we perform a retrohunt against VirusTotal's goodware corpus [40], which shows that the generated rule has no false positives against that data set.

4.3 Lead Generation for Investigations of Code Relationships

As a third core feature, we believe that MCRIT is well suited for providing leads that can be used to find and investigate potential relationships between malware families. As an example, we want to highlight how MCRIT would have been able to aid in the analysis of `win.wannacry`. At the time, Neel Mehta

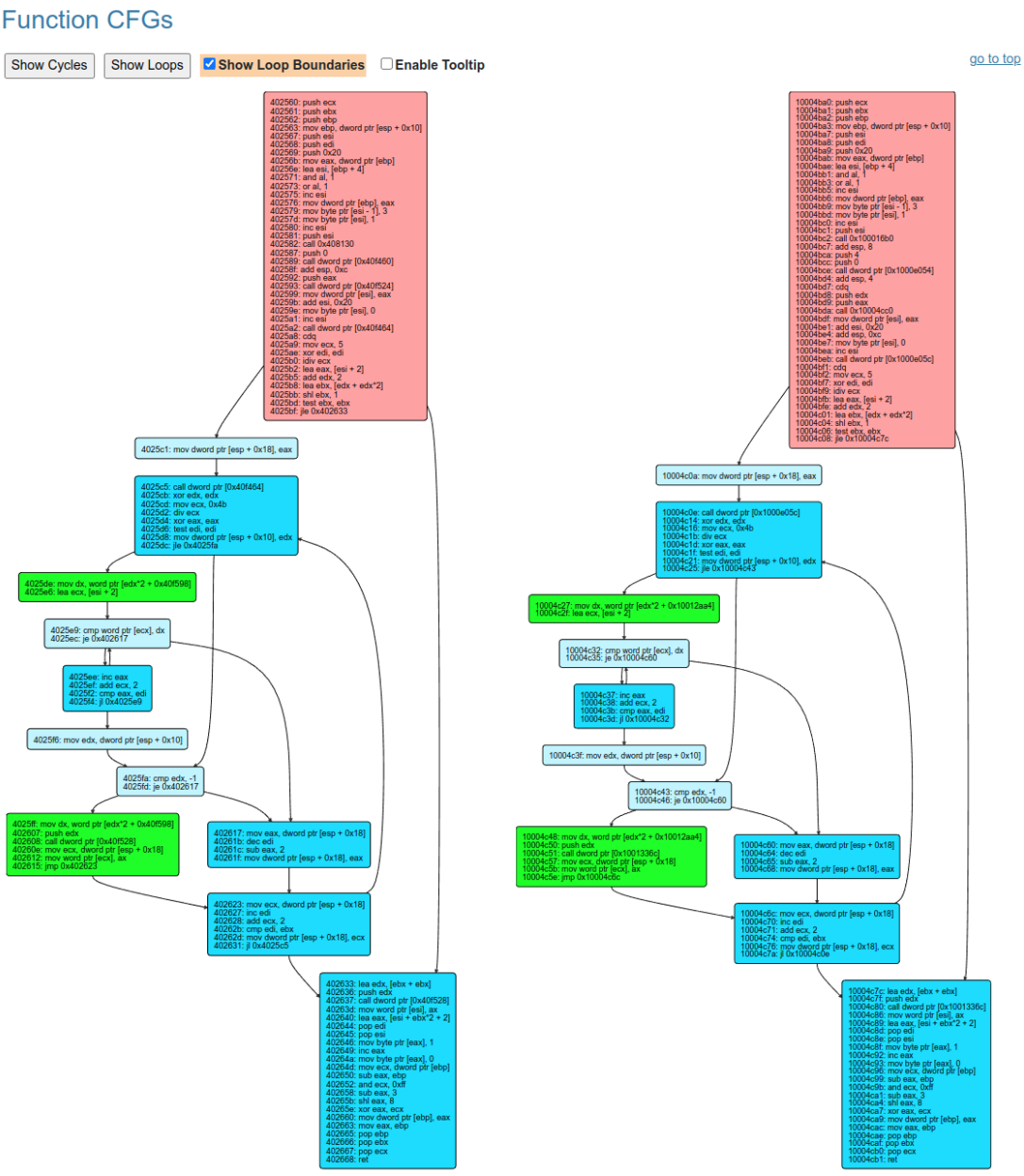

Figure 8: Control flow graph comparison feature of MCRIT Web. The functions shown are those from malware families `win.wannacry` and `win.contopee`, mentioned in the tweet [41] hinting about `win.wannacry` attribution.

tweeted out [41] two MD5 hashes and pairs of virtual addresses[2] which were among the first public attribution hints of this infamous ransomware worm towards the direction of North Korean threat actor group Lazarus.

We replicated the situation and show that MCRIT is able to find this exact same pair of similar functions between the respective samples of `win.wannacry` and `win.contopee`, based on which the latter was previously attributed to Lazarus by Symantec [42]. The functions in question are shown in Fig. 8, which also serves as demonstration of the current state of the built-in function control flow graph comparison feature of MCRIT Web. In this case, blue and green blocks are byte-identical (PicHash) and semantically-escaped matches (cf. Fig. 1), whereas the first basic block is not matched as well as the others, due to several smaller changes in its instructions and sequence. This function match in itself has an overall MinHash similarity of 67% but it stands out so well, as all of those matched basic blocks are (with respect to our demo database) indeed only found in those two families and almost undeniably the product of shared source code. Further hints towards Lazarus identified with MCRIT have been discussed in Plohmann's thesis (cf. [4] p. 162f.).

We believe that it should be possible to even isolate and rank such specific code similarity occurrences based on characteristics like how large matched functions are, how much code is identically reused and in how many other malware families it is found. MCRIT

[2] 9c7c7149387a1c79679a87dd1ba755bc @ 0x402560, 0x40F598
ac21c8ad899727137c4b94458d7aa8d8 @ 0x10004ba0, 0x10012AA4

already supports aspects of this hunting approach by allowing to filter the list of matched functions e.g. by score and number of families.

4.4 Label Transfer

The fourth feature of MCRIT that we want to highlight in this paper is the system's ability to store labels for functions, which opens possibilities for reusing and transferring these labels between analysis tools and projects. Label transfer, especially across analysis projects in both static and dynamic analysis tools, promises immense productivity gains, which partially explains why there have been many ventures towards providing solutions for this task [16, 43, 44, 45], out of which many unfortunately have been discontinued. The challenges for creating an effective solution in this context have been explained to be enormous by their respective authors, which is likely also influenced by different usage aspects which create competing demands. Particularly enabling real-time synchronization of work states, keeping history and bridging deviations in function definitions etc. appear to stand out as difficult aspects.

MCRIT does not aspire to solve these tasks and instead rather focuses on label collection and providing label candidates through its matching capabilities. As a proof of concept, we currently develop a plugin for IDA Pro, which can already query for function matches and push/pull function label information (cf. 3.3).

Generally, there exist multiple sources for such function labels. For example, the disassembler used in MCRIT, SMDA [32], is capable of extracting them via meta data information from binaries where available (e.g. Delphi, Go, DWARF; PE exports). Interfacing SMDA with another disassembler like IDA Pro (e.g. exporting IDBs) additionally allows access to IDA's full capabilities, including labels from functions recognized by IDA FLIRT [11] or user defined function labels.

As we see value in a collection of labels independent from the technology intending to benefit from them, we expect to allocate some time in the future for the creation of reference data for popular libraries that will allow the derivation of label collections also usable in MCRIT.

5 Limitations and Future Work

We will now briefly outline the limitations and what we want to address in future work on MCRIT.

The state released along this paper is considered as a first version. While it is fully functional as described, there are different aspects which we will address with usability improvements, especially once we receive feedback.

As explained in Section 3.2, MCRIT is currently limited to x86/x64 Intel assembly. Furthermore, it has not been optimized for cross-bitness matching, which we consider an increasingly impactful limitation moving forward. However, as the matching engine of the framework has been designed modularly, it allows for trivial addition or modification of feature shinglers. This will enable later extension to both support other CPU architectures or bytecode like .NET, as well as further optimization of overall matching performance.

Another limitation is that the framework in its current state has only limited options for exchanging data with other MCRIT instances and users. While basic import and export of data is supported, we are looking forward to improve these mechanisms to help reduce computational redundancy across deployments and to make it easier to provide reference data, e.g. a pre-processed version of Malpedia.

MCRIT currently only focuses on indexing of code using PicHashes and MinHashes but while wokring with it we noticed it might be of value to add capabilities for more detailed search in code, e.g. by requiring sets of specific WinAPIs used in a function, strings, or PE/ELF meta data.

6 Conclusion

In this paper, we presented MCRIT, the MinHash-based Code Relationship & Investigation Toolkit.

We first motivated the creation of MCRIT with the need for a freely available solution that enables efficient One-to-Many code comparisons for malware investigations. We continued with a summary of the two code similarity estimation methodologies used in MCRIT, enabling quasi-identical and fuzzy matching: PicHash and MinHash. This was followed by a characterization of the framework and its components, which is tied together with Docker for a painless setup.

The second part of the paper was dedicated to case studies demonstrating the use of MCRIT along the four primary use cases we envision for the project. This included an explanation of how MCRIT can aid in malware family and library code identification and how MCRIT isolates basic blocks unique to selected malware families to support YARA rule creation and hunting. We furthermore revisited the attribution of the `win.wannacry` case and showed MCRIT's view on the identified code relationship. Finally, we discussed the value of function labels and how we envision MCRIT aiding in transferring them between analysis projects.

We provide the code for all components of MCRIT as open source via Github repositories [30, 35, 6]:
`https://github.com/danielplohmann/docker-mcrit`.

Acknowledgments: The authors would like to thank the anonymous reviewers of Botconf for their valuable input and the various beta testers of MCRIT for their feedback. We would also like to express our gratitude to the Malpedia community for the continued work on a data set that ultimately inspired the creation of MCRIT.

Author details

Daniel Plohmann

Cyber Analysis & Defense Department
Fraunhofer FKIE
Zanderstrasse 5, 53177 Bonn
daniel.plohmann@fkie.fraunhofer.de

Manuel Blatt

Cyber Analysis & Defense Department
Fraunhofer FKIE
Zanderstrasse 5, 53177 Bonn
manuel.blatt@fkie.fraunhofer.de

Daniel Enders

Cyber Analysis & Defense Department
Fraunhofer FKIE
Zanderstrasse 5, 53177 Bonn
daniel.enders@fkie.fraunhofer.de

References

[1] I. U. Haq and J. Caballero, "A Survey of Binary Code Similarity," 2019. arXiv:1909.11424 [cs.CR].

[2] Zynamics, "BinDiff Manual," 2004. Website: https://www.zynamics.com/bindiff/manual/ [online; accessed April 2023].

[3] J. Koret, "Diaphora, a program diffing plugin for IDA Pro," 2015. Blog post: https://joxeankoret.com/blog/2015/03/13/diaphora-a-program-diffing-plugin-for-ida-pro/ [online; accessed April 2023].

[4] D. Plohmann, *Classification, Characterization, and Contextualization of Windows Malware using Static Behavior and Similarity Analysis*. PhD thesis, University of Bonn, 2022.

[5] F. Bilstein and D. Plohmann, "YARA-Signator: Automated Generation of Code-based YARA Rules," *The Journal on Cybercrime & Digital Investigations*, vol. 5, 2019.

[6] D. Plohmann, "Docker MCRIT," 2023. Github Repository: https://github.com/danielplohmann/docker-mcrit [online; accessed April 2023].

[7] B. S. Baker, U. Manber, and R. Muth, "Compressing differences of executable code," in *Proceedings of the 1999 ACM SIGPLAN Workshop on Compiler Support for System Software (WCSSS)*, 1999.

[8] Z. Wang, K. Pierce, and S. McFarling, "BMAT – A Binary Matching Tool for Stale Profile Propagation," *The Journal of Instruction-level Parallelism*, vol. 2, 2000.

[9] H. Flake, "Structural Comparison of Executable Objects," in *Proceedings of the 1st Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2004.

[10] T. Dullien and R. Rolles, "Graph-based comparison of executable objects," in *Symposium sur la sécurité des technologies de l'information et des communications (SSTIC)*, 2005.

[11] I. Guilfanov, "IDA Pro," May 1990. Company Website: https://hex-rays.com/ida-pro/ [online; accessed April 2023].

[12] X. Hu, T.-c. Chiueh, and K. G. Shin, "Large-scale malware indexing using function-call graphs," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, CCS '09, p. 611–620, 2009.

[13] W. Jin, S. Chaki, C. Cohen, A. Gurfinkel, J. Havrilla, C. Hines, and P. Narasimhan, "Binary Function Clustering Using Semantic Hashes," in *Proceedings of the 11th International Conference on Machine Learning and Applications (ICMLA)*, 2012.

[14] S. H. Ding, B. C. Fung, and P. Charland, "Kam1n0: MapReduce-Based Assembly Clone Search for Reverse Engineering," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[15] Zynamics, "VxClass: Clustering Malware, Generating Signatures," 2010. Presentation given at Inbot: https://static.googleusercontent.com/media/www.zynamics.com/de//downloads/inbot10-vxclass.pdf [online; accessed April 2023].

[16] S. Porst and C. Ketterer, "Zynamics BinCrowd IDA Pro Plugin," 2012. Github Repository: https://github.com/zynamics/bincrowd-plugin-ida [online; accessed April 2023].

[17] S. Zennou, S. K. Debray, T. Dullien, and A. Lakhotia, "Malware Analysis: From Large-Scale Data Triage to Targeted Attack Recognition (Dagstuhl Seminar 17281)," *Dagstuhl Reports*, vol. 7, 2017.

[18] D. Plohmann, M. Clauß, S. Enders, and E. Padilla, "Malpedia: A Collaborative Effort to Inventorize the Malware Landscape," *The Journal on Cybercrime & Digital Investigations*, vol. 3, 2018.

[19] D. Plohmann, S. Enders, and E. Padilla, "ApiScout: Robust Windows API Usage Recovery for Malware Characterization and Similarity Analysis," *The Journal on Cybercrime & Digital Investigations*, vol. 4, 2018.

[20] K. Oosthoek and C. Doerr, "SoK: ATT&CK Techniques and Trends in Windows Malware," in *Proceedings of the 15th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2019.

[21] MITRE, "MITRE Adversarial Tactics, Techniques and Common Knowledge," 2013. Website: https://attack.mitre.org/ [online; accessed April 2023].

[22] S. Alrabaee, P. Shirani, L. Wang, and M. Debbabi, "FOSSIL: A Resilient and Efficient System for Identifying FOSS Functions in Malware Binaries," *ACM Transactions on Privacy and Security*, vol. 21, 2018.

[23] C. Cohen and J. Havrilla, "Function Hashing for Malicious Code Analysis," tech. rep., SEI, CMU, 2009.

[24] A. Broder, "On the Resemblance and Containment of Documents," in *Proceedings of the Compression and Complexity of Sequences (SEQUENCES)*, 1997.

[25] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," *Journal in Computer Virology*, vol. 1, 2005.

[26] A. Walenstein, M. Venable, M. Hayes, C. Thompson, and A. Lakhotia, "Exploiting Similarity Between Variants to Defeat Malware "Vilo" Method for Comparing and Searching Binary Programs," in *Proceedings of BlackHat DC*, 2007.

[27] F. Adkins, L. Jones, M. Carlisle, and J. Upchurch, "Heuristic malware detection via basic block comparison," in *Proceedings of the 8th International Conference on Malicious and Unwanted Software (MALWARE)*, 2013.

[28] S. Eschweiler, K. Yakdan, and E. Gerhards-Padilla, "discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code," in *Proceedings of the 23rd Annual Network & Distributed System Security Conference (NDSS)*, 2016.

[29] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2011.

[30] D. Plohmann, M. Blatt, S. Enders, and P. Hordiienko, "MinHash-based Code Relationship & Investigation Toolkit (MCRIT)," 2023. Github Repository: https://github.com/danielplohmann/mcrit [online; accessed April 2023].

[31] K. Griffiths, J. Vrbanac, V. Liuolia, and N. Zaccardi, "The Falcon Web Framework," 2013. Website: https://falcon.readthedocs.io/en/stable/ [online; accessed April 2023].

[32] D. Plohmann, "SMDA: A minimalist recursive disassembler library," 2023. Github Repository: https://github.com/danielplohmann/smda [online; accessed April 2023].

[33] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[34] A. Ronacher, "Flask (A Python Microframework)," 2017. Website: http://flask.pocoo.org/ [online; accessed April 2023].

[35] D. Plohmann, M. Blatt, and D. Enders, "MCRITweb," 2023. Github Repository: https://github.com/fkie-cad/mcritweb [online; accessed April 2023].

[36] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[37] W. Reese, "Nginx: The high-performance web server and reverse proxy," *Linux J.*, vol. 2008, sep 2008.

[38] D. Plohmann, "Empty MSVC," 2019. Github Repository: https://github.com/danielplohmann/empty_msvc [online; accessed April 2023].

[39] Abuse.ch, "MalwareBazaar," 2023. Website: https://bazaar.abuse.ch/[online; accessed April 2023].

[40] VirusTotal, "VirusTotal Malware Intelligence Services." Website: https://www.virustotal.com/intelligence/ [online; accessed April 2023].

[41] N. Mehta, "Attribution hints for WannaCrypt," 2017. Tweet: https://twitter.com/neelmehta/status/864164081116225536 [online; accessed April 2023].

[42] A. L. Johnson, "SWIFT attackers' malware linked to more financial attacks ," 2016. Blog post for Symantec: https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=8ae1ff71-e440-4b79-9943-199d0adb43fc&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments [online; accessed April 2023].

[43] C. Eagle and T. Vidas, "collabREate: IDA Pro Collaboration/Synchronization Plugin," 2014. Github Repository: https://github.com/cseagle/collabREate [online; accessed April 2023].

[44] A. Chailytko and A. Trafimchuk, "Labeless: multipurpose IDA Pro plugin system," 2015. Github Repository: https://github.com/a1ext/labeless [online; accessed April 2023].

[45] B. Amiaux, F. Grelot, J. Bouetard, M. Tourneboeuf, M. Pinard, and V. Comiti, "YaCo - Collaborative Reverse-Engineering for IDA," 2018. Github Repository: https://github.com/DGA-MI-SSI/YaCo [online; accessed April 2023].