2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 978-1-6654-3786-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SANER53432.2022.00116

Phishing Kits Source Code Similarity Distribution: A Case Study

Ettore Merlo,
Mathieu Margier
Department of Computer
and Software Engineering,
Polytechnique Montreal,
Montreal, Canada
Email: ettore.merlo@polymtl.ca
Email: mathieu.margier@polymtl.ca

Guy-Vincent Jourdan
School of Electrical Engineering
and Computer Science
University of Ottawa,
Ottawa, Canada
Email: gjourdan@uottawa.ca

Iosif-Viorel Onut
IBM Centre for Advanced Studies
Ottawa, Canada
Email: vioonut@ca.ibm.com

***Abstract*— Attackers ("phishers") typically deploy source code in some host website to impersonate a brand or in general a situation in which a user is expected to provide some personal information of interest to phishers (e.g. credentials, credit card number). Phishing kits are ready-to-deploy sets of files that can be simply copied on a web server and used almost as they are. In this paper, we consider the static similarity analysis of the source code of 20 871 phishing kits totalling over 182 million lines of PHP, Javascript and HTML code, that have been collected during phishing attacks and recovered by forensics teams. Reported experimental results show that as much as 90% of the analyzed kits share 90% or more of their source code with at least another kit. Differences are small, less than about 1000 programming words – identifiers, constants, strings and so on – in 40% of cases. A plausible lineage of phishing kits is presented by connecting together kits with the highest similarity. Obtained results show a very different reconstructed lineage for phishing kits when compared to a publicly available application such as Wordpress. Observed kits similarity distribution is consistent with the assumed hypothesis that kit propagation is often based on identical or near-identical copies at low cost changes. The proposed approach may help classifying new incoming phishing kits as "near-copy" or "intellectual leaps" from known and already encountered kits. This could facilitate the identification and classification of new kits as derived from older known kits.**

I. Introduction

Despite years of research to combat it, phishing is still very much an active and in fact a growing problem. The last quarterly report from the Anti-Phishing Working Group[1] (APWG), published in May 2021 and covering January to March 2021, reported that over 611 000 phishing attacks have been detected during that period, with January 2021 having the highest reported number of attacks of APWG's records, 245 771 during that month alone.

The general life cycle of a phishing attack is well understood. The attacker (the "phisher") must have the source code for a functional website available. That website is typically designed to impersonate a brand or in general a situation in which the user is expected to provide information of interest to the phisher (e.g. credentials, credit card number). For each instance of the attack, the phisher needs to have somewhere to deploy the site. It may be a web hosting service, and a server they own, or someone else's website that they have compromised. Once the code of the phishing site has been deployed on the host, victims have to be lured to it. This usually involves a spamming campaign to distribute the URL of the phishing site, over email or other messaging systems. When a victim discloses some information to the phishing site, the phisher needs to have a way to exfiltrate the data, and then to somehow monetize it.

Some costs are clearly involved during that life cycle, but some technical skills are also required. In practice, many phishers do not design the attack from scratch and instead use what is known as a "phishing kit" [1], [2].

A phishing kit is a ready-to-deploy set of files that can be simply copied somewhere on a web server and used almost as is. It typically uses PHP as the back-end programming language, ensuring that it will work on most servers. It contains all the CSS, HTML, JavaScript, etc. files required to present the web page to the victim, and to process the data being submitted. In the vast majority of the cases, the data provided by the victim is simply sent by email to a "drop address" [2]–[9]. The only thing that a phisher really needs to do with a phishing kit is to configure the drop email address to be that of some account they control, and the deployed attack is ready to serve its purpose. Some kits may have some more advanced features, such as other means to exfiltrate the data, encrypt the data before exfiltration, URL randomization [10], built-in cloaking mechanisms both client- and server-side [6], [11] etc. Phishers can purchase premade kits or have them customized, copy existing ones, modify them to change the appearance or the target, or add and remove features. The code can be obfuscated, presumably to evade detection. Some kits will even have some obfuscated code that does exfiltrate the collected data to a secondary location, perhaps as a means for some phishers to collect other phisher's loot [1], [10], [12].

The vast majority of the literature on anti-phishing focuses either on detecting phishing emails sent to the victims

[1]https://docs.apwg.org/reports/apwg_trends_report_q1_2021.pdf

978-1-6654-3786-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SANER53432.2022.00116

(e.g. [13]–[18] and many more) or on detecting phishing web pages themselves (e.g. [19]–[27] and many more). By comparison, much less academic literature is dedicated to the server-side of the attack. One reason for this might be that actually getting access to real phishing kits used in real attacks is more difficult than having access to phishing emails or to the client side of a phishing site. Han et al. [10] acquire their phishing kits by setting up a honey pot, but in general, academic authors that have had access to a large number of phishing kits used online have been granted access to the database by private security firms [5], [6]. This is also the case for our study. We have been provided access to the source code of 20 871 phishing kits that have been collected during phishing attacks and recovered by forensics teams over a period of about three years, from March 2018 to mid 2021.

In this paper we consider the static analysis of the PHP, Javascript and HTML code found in phishing kits. We assume that, while some kits are original, some kits are cloned and propagated with little to no modifications, under the development assumption that the least effort needed to evade detection is performed in terms of source code changes. We also assume that the links between high-similarity kits constitute some sort of "lineage" of evolution among kits and we want to reconstruct the "lineage" graph among kits, based on the similarity of the PHP, Javascript, HTML source code. Thus, a path in the lineage can represent a plausible sequence of modification steps to obtain a kit from another one. This is very similar to the "origin" problem in source code clone detection [28] where a known evolution tree/graph of an application is used to analyze when and what code was involved in duplication, near-duplication, and merging of code fragments.

Current works on detecting the origin of source code clones [29] are based on existing repository of committed versions of an application. The goal is often to analyze sequences and lifetime of changes in similar source code fragments over several commits and releases. Some studies in "late propagation" of software clones [30] investigate the non-synchronous delayed over time propagation and occasional reversal of changes in similar code fragments during software evolution as documented in the historical information in a repository. Clone genealogy extraction from repository [31] examines the duplication and propagation of changes in similar source code fragments to compute the historical paths in the repository commits that have lead from a set of similar fragments to a later set of modified version of the similar starting set, to investigate deviations and evolution of clones.

Phishing kits analysis present additional issues with respect to the "origin" and "genealogy" problems in clone detection. First, the underlying evolution tree/graph is unknown, as we do not have access to repositories of committed versions. This is in fact what we want to discover or reconstruct. Second, contrary to conventional software evolution, in which we have all the commits, the repository of kits not only is not in our possession, but it may not even exist. The collected phishing kits constitute a sparse sample with evolutionary holes between related kits. In other words, we have no guarantee of having a sample for each no-matter-how-small-difference between modified kits. It is also possible that different kits may become similar in some parts, because of information sharing between hackers. It is also possible that parts of kits are reverted to previous versions, for functional or code scrambling reasons. Third, again in opposition to conventional software, since we do not have the complete history of changes and evolution, we do not have an "oracle" to validate the extracted "lineage" against the real one. Last, again as a difference with respect to conventional software, no precise dating is reliably associated with collected kits, beside the discovery date. So, time information cannot reliably be used to establish a possible order among close changes in similar code fragments from different kits. Furthermore, unlike conventional software, permutations of components with the sole objective of maintaining the functionality, while perturbing the surface look of kits are easy to perform at low cost of developers' effort. Several structures in PHP, JavaScript, and HTML that are in sequence in a file, can very easily be permuted without changing the functionality, in an attempt to disrupt and evade detection. For example, functions in a file, methods in a class, or attributes in a HTML tag, can be permuted at will to change the surface look, without changing the semantics.

The objective of this paper is to analyze the similarity of PHP, Javascript, HTML source code in different phishing kits, and to combine in a multi-language similarity approach the results from the three languages. Based on the similarity, the goal is the optimal reconstruction of neighborhood links that minimize the overall modification effort among kits on the database. This would correspond to a "plausible" lineage of kits. Because with the previously mentioned limitations, the real evolution of kits may be different. Nevertheless, if we assume that kits are cloned and propagated with little to no modification, under the development assumption that the least effort needed to evade detection is performed in terms of source code changes, similar kits may really be close in evolutionary terms.

Larger kit-changes are occasional, abrupt and could be motivated by "economy": attacks may no longer be very effective because of high detection rate and therefore require modifications. These higher cost modifications and changes give rise to a proliferation and distribution of low-cost identical copies derived from the modified root.

In this paper we investigate the reconstruction of this intertwined web of kits similarity and the relevance and frequency of larger changes over exact duplications or small changes. The inferred lineage is also useful for forensic examination of kits, allowing us to explore close links of multi-language and single-language similarity.

The paper is organized as follows: we first provide all the details of our approach in Section II. In Section III, we discuss the details of our experiments, and we give the results in Section IV. We discuss possible threats to the validity of our results in Section V. We review some related work in Section VI and we conclude in Section VII. Additional

data and figures for this paper is made publicly available at https://ssrg.eecs.uottawa.ca/PKSourceCodePaper/.

II. Approach

Token-based clone detection was first proposed by Kamiya et al. [32]. Other authors [33], [34] have since used tokens with different algorithms and tokens-based clone detection is now a family of clone detection tools. Code metrics based clone detection was introduced by Meyrand et al. [35] and developed by Merlo et al. [36], [37]. The idea of code metrics clone detection is to chose software syntactic features, such as statements, branching instructions, function calls, etc., and to build a vector for which each dimension is a selected feature. The value of each vector component is the frequency of the corresponding feature. Syntactic analysis is first done to compute the frequencies and build the vectors. These are then compared using a similarity criterion, such as the euclidean distance, cosine distance, and so on. The original technique of [35] used space discretization for clone clustering.

Tools such as CCFinder [32] allow to find clones in different languages, independently. Merlo et al. [38] have applied this multi-language technique to large industrial source code in Java and C/C++. Kraft et al. [39] were the first to propose cross-language clone detection, resolving the issue of finding a fragment cloned across different languages. Such techniques have been implemented in other tools: for the .NET framework by Al-Omari et al. [40] and for more languages in LICCA [41] by Vislavski et al.

Clone analysis involves finding similar code fragments in source code as well as interpreting and using the results to tackle design, testing, and other software engineering issues [42]–[44].

There are four main similarity types of clones, defined in the literature [45]: Type-1 includes lexically identical code fragments, Type-2 identifies syntactically identical fragments, Type-3 characterizes fragments with structural modifications such as changed, added, or removed statements, and Type-4 indicates fragments that perform similar computations but implemented using different syntactic variants. In the literature, these are referred to as "semantic" clones. A finer classification of clone differences has been proposed in the context of object oriented refactoring [42] and a set of mutation based clone differences has been investigated in the context of clone-detection tool evaluation [46]. In this paper, we refer to Type-2 clones as "parametric" clones, excluding Type-1 clones, unless otherwise indicated. We use Type-3 fragment similarity based on token lexical type, while Type-4 fragment similarity, much harder to detect, is ignored.

The algorithm used in this paper computes metrics based on tokens in fragments and builds vectors of token type frequencies [47], [48]. Similarity distances based on Type-3 similarity are more robust than those computed using Type-1 or Type-2 similarity. They are more independent of common camouflage techniques such as source code layout changes, systematic identifier renaming, constants replacements involving strings or other values, refactoring of blocks and functions, and so on. In this paper, we focus our attention on function and method bodies, initially treating each language independently. The process to analyze kit similarity – in a single-language perspective – is illustrated Figure 1.

After the source code files of the 3 languages have been identified during the pre-processing phase, the first step is to extract the tokens from the software source code using a lexical and syntactic analyzer for each language (PHP, Javascript and HTML). The parser creates an Abstract Syntax Tree (AST) that is used to identify fragments at the appropriate level of granularity – function and method level for this paper. The parsing phase uses the extracted tokens from functions and methods to build the frequency vectors of metrics.

The analysis steps are applied for each language independently. First, fragments are partitioned in equivalent classes of identical clones, or Type 1 (respectively parametric clones, or Type 2), using their sequence of token images (respectively types). Then, since we are interested in tracking which specific codes the kits share, rather than the actual number of repetitions of them, we consider kits as sets of equivalent classes of fragments. After equivalent classes are computed, the next step is the computation of distances. In this paper, we use uni-grams of token types to compute the frequency vectors of token types in fragments[2]. Similarity of fragments is computed during the analysis phase, using the Manhattan distance (see equation 1) between metric vectors. If we interpret metric vectors as bags of token types, then the Manhattan distance between two fragments' metric vectors is the symmetrical difference between their bags.

The Manhattan distance – or l_1 norm – between two vectors u and v of size k is defined as:

$$l_1(u, v) = \sum_{i=1}^{k} |u_i - v_i| \tag{1}$$

We now move on to the distance between kits, for every pair of (parametrically distinct) kits. Beforehand, a metric vector is computed for each of those kits. This kit metric vector is defined in equation 2, where $\text{pdf}(k)$ is a set of parametrically distinct fragments of kit k. Since parametric fragments have the same metric vector, it does not matter which fragment of a parametric class is chosen. Then, the similarity of kits is defined as the Manhattan distance (see equation 1) between those kit metric vectors. Summing the fragment metric vectors is a quick way to compute the distance between all pairs of kits, stored in a matrix of distance. Ignoring parametric kit clones reduce drastically the size of the matrix, without losing any information.

$$v_k = \sum_{f \in \text{pdf}(k)} \text{metricVector}(f) \tag{2}$$

The fragment granularity, although ignored in this step, is taken into account later with the detailed comparison of kits.

[2]Although the presented approach can use n-gram token sequences to build metrics, in previous experiments [47], [48] using n-grams did not improve accuracy significantly, but required substantial additional computational effort.

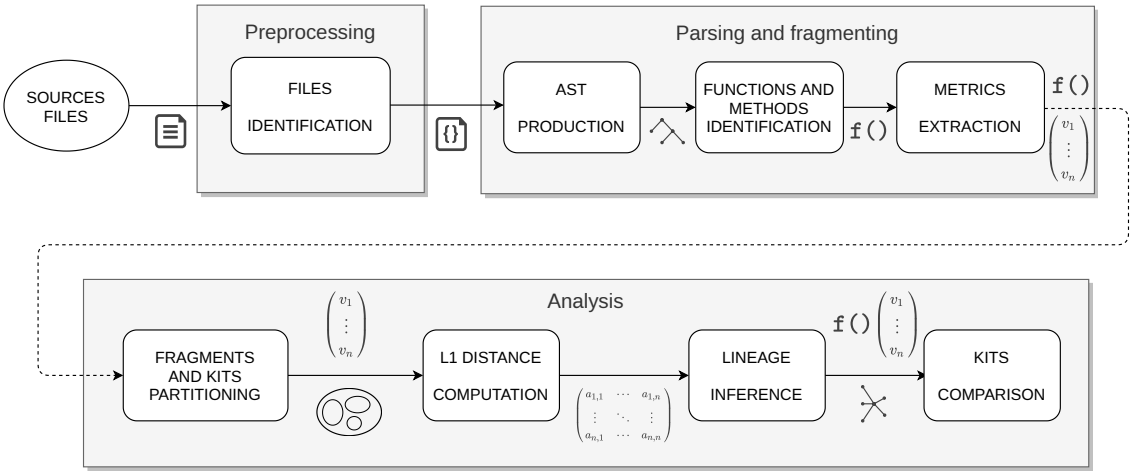

Fig. 1. Single-language analysis process

The distance of matrix allows us to compute the Minimal Spanning Tree (MST) of the associated complete graph. Edges in the MST can be interpreted as the most likely derivation by similarity of one kit from another. Paths in the MST may be interpreted as likely derivation paths in kit propagation. Therefore we can interpret the MST as a plausible, yet not exact, undirected "lineage" of phishing kits. It does not necessarily represent the temporal evolution of kits, but rather minimizes the global modification effort.

We then perform a detailed comparison of kits for each edge of the inferred lineage. The l_1 distance between the kit metric vectors is sensitive to distortion: by permuting some instructions between fragments, kits are different but have the same metric vectors. Therefore, the goal is to produce a more accurate distance, by working at the fragment level, and to establish a mapping between two kits' fragments. Since it is more computationally heavy than the Manhattan distance between kit metric vectors, this comparison is only performed for the edges of the lineage. Each fragment of a first kit is associated with another fragment of a second kit if they are identical (type 1), parametric (type 2), or similar clones (type 3), with this priority. When multiple similar clones are possible, the priority is given to the lowest l_1 distance. If a fragment in one kit does not have any clone in the other kit, it is considered original. We say that two fragments, with their metric vectors v_1 and v_2, are similar if and only if $l_1(v_1, v_2) \leq \alpha \min(\|v_1\|_1, \|v_2\|_1)$. We choose $\alpha = 0.3$ as recommended by Lavoie et al.'s experiments [48].

Comparing all pairs of n fragment vectors by Manhattan distance would asymptotically require $\mathcal{O}(n^2)$ operations in the worst, best, and average cases. To reduce the execution time and memory complexity, we use the metric tree indexing [47], [48] to store and access kit vectors. Metric tree indexing allow the efficient computation of distances between a vector and its nearest neighbor (NN) without loss of information. The average complexity in practice for a search in metric trees is $\mathcal{O}(log(n))$. That reduces the average complexity of computing the NN of a kit to $\mathcal{O}(n \cdot log(n))$, that is a lower order of growth than all pair comparison. Other matching approaches [35], [36] are asymptotically linear on the number of kits ($\mathcal{O}(n)$), but create small distortions to distances and some loss of information and precision.

l_1 satisfies the metric axioms (non-negativity, symmetry, identity, and triangle inequality) required by the metric tree indexing.

This multi-language distance d is the sum of the l_1 distances of each language, as defined in equation 3 for two kits k_1 and k_2. In our construction, fragments cannot belong to multiple programming languages and therefore they represent a partition of a kit into three mutually exclusive sets of fragments. The overall total difference can be computed as the sum of the three language-based differences. Then, a multi-language lineage is inferred by computing the MST of this matrix. Finally, the comparison of kits is performed on this new lineage for each language, and the results are merged. We now have multi-language more accurate distances for the edges of the lineage, as well as the mapping of fragments for those pairs.

$$d(k_1, k_2) = d_{PHP}(k_1, k_2) + d_{JS}(k_1, k_2) + d_{HTML}(k_1, k_2) \quad (3)$$

III. Experiments

The experimental setup uses an 8-core Intel(R) Core(TM) i7 CPU 950 @ 3.07GHz, with 16 GB of RAM under Fedora 31. The data analyzed here is a set of 20 871 phishing kits. Table I shows the distribution of files, according to their filename extension, except for PHP. The method used to detect PHP files is to search for text files whose content has a PHP opening tag. PHP, Javascript and HTML are the main languages found in the dataset, therefore we focus our analysis on those 3 languages.

Other files, although not analyzed in this paper, can also take part in the logic of phishing attacks. Kits also include graphics files, which are mainly PNG (54.4% of graphics files), GIF (23.8%), JPEG (11.6%) and SVG (10.2%) files. The most common types of other files are CSS (35.2% of other files), .download (8.6%), TXT (7.9%) and .htaccess (6.9%).

File type	# files	# files (%)	LOC	LOC (%)
PHP	338 985	17.95%	55 765 042	30.63%
JS	201 219	10.66%	82 169 018	45.13%
HTML	119 970	6.35%	44 152 496	24.25%
Graphics	669 146	35.44%	-	-
Other	558 930	29.60%	-	-
Total	1 888 250	100%	182 086 556	100%

TABLE I
DISTRIBUTION OF FILE TYPES OF PHISHING KITS DATASET

A custom top-down parser written in JavaCC is used to parse PHP files. For Javascript, we use the parser `acorn`[3] For HTML, the fault-tolerant parser `htmlparser2`[4] is used. Since PHP code can be intertwined with both HTML and Javascript code, the latter can become syntactically incorrect. Consequently, it is important to use fault-tolerant parsers.

Statistics about parsing are reported in Table II, for the phishing kits . It reports the number of fragments found in a specific language, as well as the number of kits having at least one fragment in this language. Due to the large number of Javascript fragments, only the fragments whose size is above 50 tokens (around 10 LOC) are kept. In total, 20 871 kits have fragments in at least one of the languages, PHP being present in almost all the kits.

Language	PHP	JS	HTML	Multi
# Kits	20 867	14 413	19 341	20 871
# Kit fragments	1 089 639	18 250 741	223 726	-

TABLE II
PARSING STATISTICS OF PHISING KITS DATASET

For the kits, 0.90% of detected PHP files do not have a .php extension. Files whose name ends with a .html, .txt, .gif or .asp for instance, can also contain PHP code. Those files make up 2.5% of lines of code of analyzed PHP files. This shows that phishing kits use other types of files to store PHP code, presumably in order to hide their code.

To measure the similarity between kits, we perform several experiences on the multi-language analysis[5]:

First, we study the kit equivalent classes. We measure the duplication rate by counting how many kits have identical clones, parametric clones, or no clone. We also reports statistics about the classes sizes. We then investigate to what extent most of kits belong to few classes by plotting the cumulative number of classes.

Second, we visualize the MST of all kits on PHP, JS and HTML. This visualization emphasizes the reconstruction of kit propagation tree by minimal distance. We use Multi-Dimensional Scaling (MDS), so the visual proportional distance between kits has been respected in the visualization. Still, with this number of kits, MDS introduces some distance distortions, and some similar kits can be positioned far away. Therefore, the MST is recomputed on the 2D-space euclidean distance for the visualization, while the actual MST is used for the analysis.

Third, we measure the absolute similarities along the MST, using the distances provided by the comparison of kits. We also measure the relative similarities using kit metrics vectors, in two ways. If we want to compare absolute token differences to kit sizes, we have to take into consideration that similar kits may not be of the same size. A common measure is the Jaccard Coefficient JC reported in Equation 4.

$$JC(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A| + |B| - |A \triangle B|}{|A| + |B| + |A \triangle B|} \quad (4)$$

where $\triangle$ is the symmetric difference between A and B and corresponds to the Manhattan distance between kit vectors.

The Jaccard Coefficient is symmetric, although the size of kits may be different ($JC(A,B) = JC(B,A)$). Note that JC is often smaller than an intuitive percentage. For example, two sets sharing half of their elements with one another will have a JC of 0.333.

We can also compute and show the asymmetric percentages of sharing for both kits of each NN MST edge. Equation 5 computes the Relative Difference (RD) between two kits.

$$RD(A,B) = \frac{|A - B|}{|A|} \quad (5)$$

RD is not symmetric: RD(A, B) may be different from RD(B, A), because of possible different sizes of A and B.

Fourth, for a given kit, we visualize the multi-language lineage centered on this kit, its neighbors in the other lineages, and all intermediate kits from the central kits and those neighbors. In this visualization, far fewer kits are represented, thus we can use MDS directly. This can be useful for forensic investigation, when we need to focus on a kit and see kits that are similar to it.

Fifth and last, we compute the correlation between distances of each language on the multi-language lineage, to understand the relation between languages. To avoid bias, parametric duplication of kits is taken into account: if an edge links a cluster of n kits and another of m kits, the associated distance is counted $n \times m$ times. Distances do not follow a normal distribution, therefore we use Spearman's method to compute the correlation.

[3]https://github.com/acornjs/acorn/. When a file cannot be parsed by the regular parser, we use its fault-tolerant variant `acorn-loose`. Javascript codes embedded in HTML's `script` tag are parsed, as well as files with `.js` extension.

[4]https://github.com/fb55/htmlparser2/. HTML static code block from PHP files are parsed, as well as files with `.html` or `.htm` extension.

[5]Experiences on each single-language analysis have also been performed but cannot all fit in this paper. Results on PHP, HTML and Javascript can be found at https://ssrg.eecs.uottawa.ca/PKSourceCodePaper/.

IV. Results

In this section we report results from the multi-language analysis, unless otherwise indicated. Single-language results can be found in our website.

language	php	js	html	multi
% with identical clone(s)	15.7	85.7	67.5	12.8
% with parametric clone(s)	61.0	1.2	6.5	56.1
% without clone	23.3	13.0	26.0	31.1
# total kits	20 867	14 413	19 341	20 871

TABLE III
KITS DUPLICATION DISTRIBUTION

Table III shows the distribution of kit clones for the 3 combined languages, and for each single language.

When taking into account all 3 languages, a small portion (12.8%) of the kits have identical clones, even though every kit in our dataset is unique. This is easily explained: some changes in the source code (e.g. permutations of functions) are not impacting our analysis, and some of the files (e.g. images) are not part of the analysis. Authors can also make changes to the file structure of the kit (renaming or duplicating files) that we do not consider as differences. More than half (56.1%) of the kits have parametric clones. This measure shows that a lot of kits are reused (potentially by different attackers) for many attacks, changing only some parameters. Finally, nearly a third (31.1%) of the kits do not have any parametric clone. Those might be highly similar between one another, or be totally original. This issue is investigated with the analysis of the multi-language lineage, which removes all parametric clones.

The proportion of kits that are identical clone for their PHP component (15.7%) is slightly higher than the multi-language identical clones. This is expected, since all but very few kits do include PHP code. If two kits are identical in all languages, they must therefore be identical in PHP. The same is true for parametric clones (61.0%).

This is different for HTML and JS, which are not found in all the kits. Indeed, the kit similarity distribution does not match the multi-language one for these languages: the majority of kits that contain these languages have HTML (67.5%) and JS (85.7%) code identically cloned with other kits. For JS, this can be explained by the fact that kits can ship common libraries, such as `JQuery`. For HTML, the same template can be used by different attackers.

equivalent classes	n	mean	stdev	min	max	med.
identical kits	1 110	2.4	1.2	2	17	2
parametric kits	2 476	5.2	8.5	2	150	3

TABLE IV
STATISTICS OF EQUIVALENT CLASSES' SIZES

Table IV reports the statistics on the equivalent classes sizes of kits in all languages (PHP, JS and HTML). Since we only consider the equivalent class of size > 1, meaning we ignore kits without clone, the minimum size is always 2. Median size is either 2 or 3, suggesting that a large part of kits are being reused a small number of times. In average, a kit has twice as many parametric clones as identical clones. The biggest identical class has a size of 17, while the biggest parametric class is made of 150 cloned kits. This supports the hypothesis that kits requiring to change only some parameters are widespread.

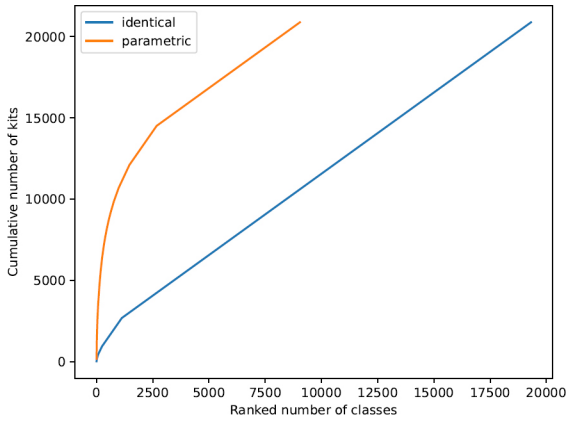

Fig. 2. Cumulative size of equivalent classes - Multi-language

Figure 2 plots the cumulative number of kits against the ranked number of equivalent classes, either identical or parametric (in the broad sense). The curves end up with a straight line of slope 1, caused by the kits without clones. Because few kits have identical clones, the identical curve is almost a straight line. Yet, the parametric curve quickly grows at the beginning, showing that as much as 70% of the kits belong to less than 30% of the parametric equivalent classes. If we consider each class as a potential source of kits, this would mean that most of the kits come from a restricted number of sources.

Language	Number of kits
multi	9 043
php	7 210
js	3 023
html	6 931

TABLE V
NUMBER OF PARAMETRICALLY UNIQUE KITS, BY LANGUAGE

Table V reports the number of parametrically distinct kits (ie number of Type-2 equivalent classes) found for each language. JS is the language with the least number of parametrically unique kits, while PHP and HTML have similar numbers. The highest number is reached for multi-language analysis, since the number of unique kits can only increase when considering new languages.

Figure 3 presents the visualisation of the MST using MDS. Since MDS scaling represents kits with their proportional mutual distances, very similar kits are often very close in the graphical rendering and some details have been reported by zooming on some dense areas of the graph.

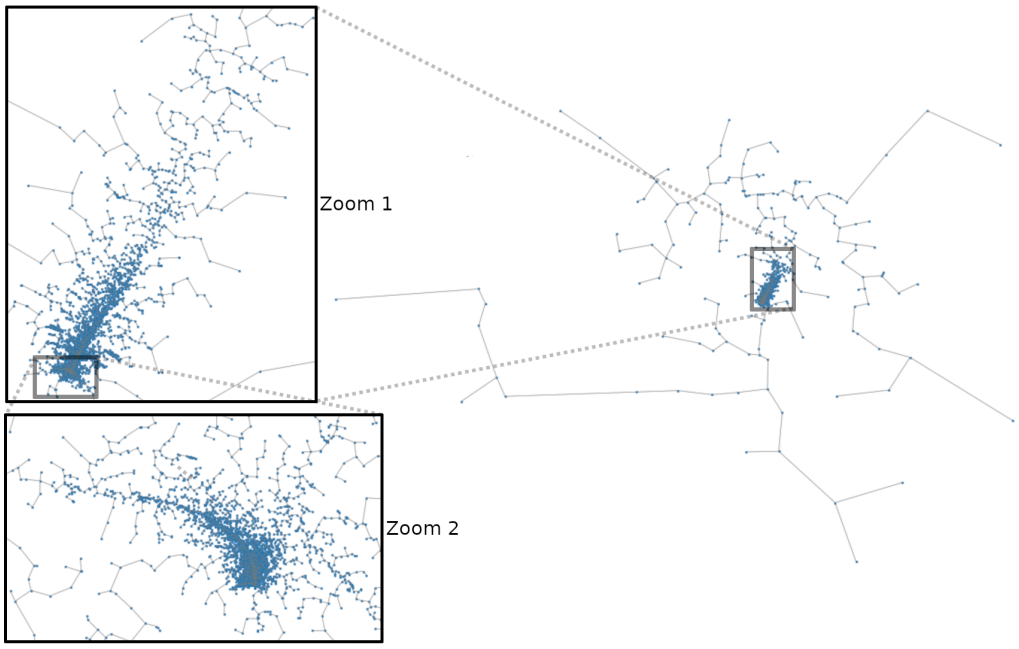

Fig. 3. Kit Similarity NN MST

Figure 4 presents the curve of absolute distances, in logarithmic scale, for NN MST edges, ranked in decreasing order. In this last figure, we can observe that as much as 40% of the classes differ by less than about 1000 tokens of different types. If these tokens appeared contiguously in the code and were all from, say, PHP, they would represent only a few hundreds LOCs of difference. Still, we notice some outliers, which are at a large distance from any other classes: one such example was a PHP file containing a very large array of millions of strings, increasing drastically the number of different tokens.

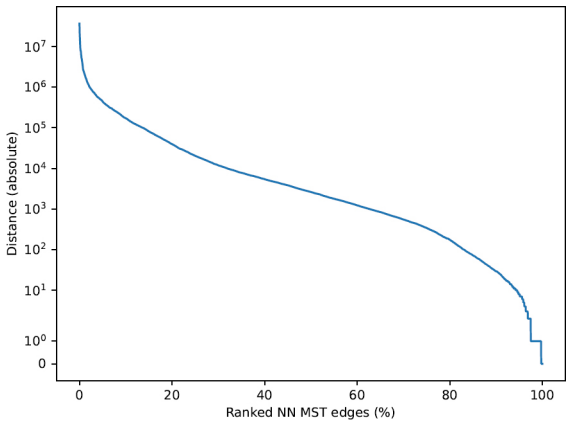

Fig. 4. Ranked NN MST absolute similarity

Figure 5 shows the JC and RD similarity of NN MST edges ranked in increasing order. Both diagrams are skewed towards high frequencies of high-similarity edges. For example, in Figure 5, roughly 90% of edges propagate kit classes with a JC of about 0.8. In this figure, RD indicates that about 90% of propagated kit classes share 90% of their code with at least one other class. Kit classes show a much more concentrated RD similarity around high levels than JC, because each kit is considered with its own size only for RD similarity, while JC takes into account the different sizes of the compared kits.

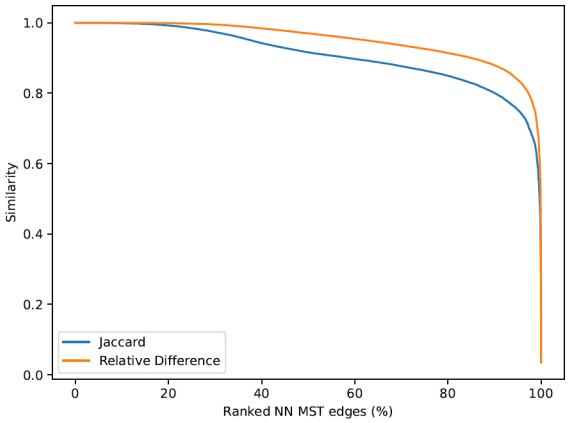

Fig. 5. Ranked NN MST similarity

Kits similarity distribution is consistent with our hypothesis about kits propagation model: often, the propagation is based on identical or near-identical copies at low cost changes.

Table VI reports the number of Type-2 equivalent classes

RD	Similar kits	Percentage
1	97	1.1 %
0.9	6 180	68.3 %
0.8	8 270	91.5 %
0.7	8 825	97.6 %
0.6	8 968	99.2 %

TABLE VI
PERCENTAGE OF KIT SIMILARITY

type	mean	stdev.	min	med.	max
identical	115 343	698 700	0	857	15 305 024
parametric	4 027	22 789	0	67	791 794
similar	30 926	203 000	0	1 246	7 015 328
original	126 670	896 830	0	2 481	35 641 887

TABLE VII
STATISTICS ON FRAGMENT SIZES GROUPED BY SIMILARITY TYPE ON THE MST EDGES

whose degree of similarity with at least another class is higher than the specified threshold. For example, 97 classes (1.1%) have neighbors at the maximum level of similarity. Since we only consider parametrically unique kits, it shows that a small percentage of kits have the same metric vector without begin parametric. This can happen if instruction inside one fragment are permuted for instance. Other 6 180 classes (68.3%) are structurally and lexically similar to at least another class at 0.9 RD level, and so on. Only few classes seems to be highly different from any other class (RD less than 0.7).

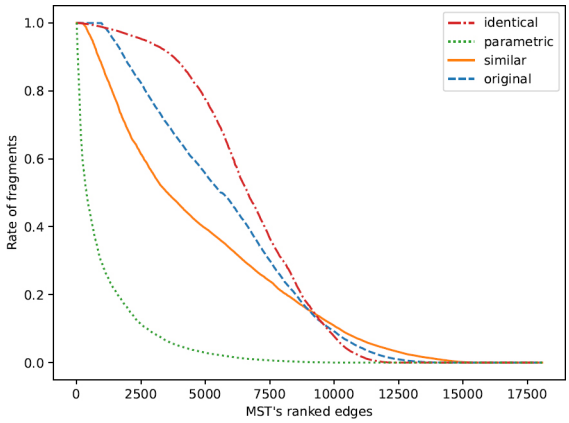

Fig. 6. Distribution of fragments along the lineage

Figure 6 shows the proportion of each category of fragments (identical, parametric, similar or original) between kits along the multi-language lineage, weighted by the fragment sizes counted in number of tokens. The proportion is the number of fragments for a category, divided by the total number of fragment of the Type-2 equivalent class' kit representative. Note that this gives two values for one edge of the lineage. Identical fragments have the highest rates, suggesting that many components are reused between similar kits. In comparison, parametric fragments are much less frequent. This can be explained in part by the fact that parametric kit clones are removed from the lineage. Also, parametric changes are the most restricted type of modification, whereas identical clones do not need any action other than being copied. Similar and original categories still represent a large part of the fragments, showing that nontrivial changes are performed between kits.

Table VII reports statistics on total size of fragments grouped by their similarity type, for each lineage's edge. For example, we can see that in average, identical fragments between two kit classes of a lineage's edge add up to nearly 115 000 tokens. The minimal values are always 0, meaning that there is at least one edge where kit classes do not have any fragment for a given clone type. The maximal values are high, leading to both high standard deviations and large gap between means and medians. Similar clones have also a high median, suggesting that many kits integrate substantial modifications. Like for the proportions in Figure 6, we can see that parametric clones are much less frequent than identical clones.

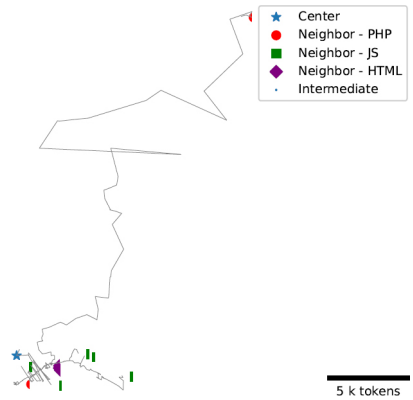

Fig. 7. Focused lineage visualisation

We also show an example of what we call a "focused lineage.". It starts from a central kit class (shown with a star in the figure), and includes all of that central kit class direct neighbors for all the lineages (i.e. that kit class one-edge neighbors in the PHP, JS, HTML, multi-language lineages). Because the one-edge neighboring kit classes are usually different in different lineages, we also include all the intermediate kit classes from the central kit class to the neighboring kit class along the all-language lineages. This kind of focused view is useful for forensics analysis, when more information is required about a specific kit. In the example of Figure 7, the central kit can be found on the bottom-left corner of the figure. HTML (square) and JS (diamond) neighbors are all clustered next to the central kit, where almost of all of them have a difference of less than 5 000 multi-language tokens. Thus, when a small change is made in either HTML or JS, few modifications are done in the other languages. However, there are both close and far away PHP (circle) neighbors: small changes in PHP can either imply little or big changes in the

other languages.

	multi	php	js	html
multi	1	0.8437092	0.4980513	0.3926228
php	0.8437092	1	0.3525423	0.1363826
js	0.4980513	0.3525423	1	0.3951109
html	0.3926228	0.1363826	0.3951109	1

TABLE VIII
LINEAGE DISTANCE MATRIX CORRELATION

Table VIII reports the correlation between the distances of the global lineage in PHP, JS, HTML, and multi-language. For each cell, the associated p-value is very low ($p < 2.2e-16$). Therefore, matrix results are significant ($p < 0.05$) without having to adjust the p-value for multiple comparisons. PHP distances are the most correlated with multi-language distances: chances of predicting the similarity between two kits by analyzing only the PHP code are high. PHP and HTML distances are weakly correlated, suggesting that both languages are independently modified. Indeed, one can change the look and feel of web pages (HTML) while leaving as is the code handling the logic of the attack (PHP).

V. THREATS TO VALIDITY

The set of 20 871 phishing kits that we have analyzed might include websites that are not really phishing kits. To avoid including unrelated websites as much as possible, we have leveraged the exfiltration feature that is an inherent part of a phishing kit. All of the kits have been individually deployed in a secured container and programmatically interacted with to detect their expected user-inputs and provide it. We then detected whether or not the kit was attempting to exfiltrate the provided data (e.g. by sending it in an email). We excluded from our analysis any kit for which we could not trigger such an automated exfiltration mechanism. The 20 871 phishing kits that we included have all successfully passed this test.

Our experiments suffer from selection bias of the samples. This bias is unknown and depends on the set of tools used to identify kits and on the variability of examined systems. Therefore, the reported results relate to the similarity of selected kits. The existence of large high-similarity clusters may therefore have been produced by a selection bias. A different set of kits may produce different similarity distributions. Since the kits, in a sense, suffer from the same selection bias, a manifold cross validation may not be sufficient to address the selection bias issue, but several additional samples taken with different tools and different systems should be necessary. Nevertheless, the observed similarity distribution is consistent with the assumed model of phishing kit evolution and a high kit similarity may also occur in other samples. Also, our method is independent of the data collection mechanisms, which reduces the impact of this bias.

Some kits uses obfuscation techniques (e.g. using `eval` on base64-encoded strings), which can bias the similarity measure between source files, since this code is not included in our analysis. De-obfuscating that code to analyze it is part of our future work, but less than 25% of all the kits use `eval` at all, and apart from a few outliers the code being dynamically evaluated is not very large and should not impact our general results significantly.

The reported results depend on the choice of Manhattan distance of token types to compute similarity as described in section II. Other similarity measures exist and may produce different similarity values and distributions. Nevertheless, since Manhattan distance is a close approximation of the optimal alignment [47], [48], results obtained with other techniques may not be much better than the reported ones.

No oracle is available for the evolution of kits. So, we cannot assess the validity of the inferred NN lineage. To mitigate this limitation, the same approach to extract the lineage has been performed and compared with the known evolution of Wordpress over 235 versions. The comparison between the automatically extracted genealogy and the documented one from the repository of Wordpress has been performed and manually validated by the authors and judged satisfactory. Details of that experiment is available on our website.

Similarity between kits has been computed using the PHP, JS, HTML code extracted from the kits and parsed using a modified open source PHP parser and unmodified ones for JS and HTML. Since the definition of lexical tokens is not unique among parsers, computation of fragment metrics may slightly vary, when different parsers be used. Similarity levels should still be close in values, regardless of the parsing environment.

The presented approach does not take into consideration other programming languages and structures. Therefore, kits that differ because of, for instance, CSS may have erroneously been detected as highly similar because of PHP, JS, HTML code. This issue will be addressed as future research to integrate additional sources of information and revise the similarity distributions accordingly. Other work on HTML based similarity have been reported in the literature [21], [49], [50].

VI. RELATED WORK

Analysis of malware has been mostly performed on binaries. Consequently, malware similarity and propagation has been investigated mostly for binaries as well.

A search engine for binary code [51] has been proposed by Khoo et al. Several papers address the investigation of genealogy and lineage of malware. Jang et al. [52] proposed an automatic software lineage inference approach. Canfora, Cimitile et al. [53], [54] investigated Android malware phylogenesis using model checking. Dumitras et al. [55] investigated the provenance and Lineage of malware. Khho et al. presented [56] a phylogenetic-inspired approach for reverse engineering and detection of malware families using execution traces.

Malware evolution has been investigated by Gupta et al. in an empirical study perspective [57] and by Darmetko from the structural point of view [58]. Lindorfer et. al. presented insights into the malicious software industry [59].

Binary similarity of malware and binary clones have been discussed by Sæbjørnsen et al. [60]. Dynamic similarity of

binary functions for security [61] has been investigated by Egele et al., while Ming et al. propose a semantic diff for binaries [62], [63].

Source code similarity is also called "clone detection" and so it is referred in the literature. Clone detection state of the art includes different techniques. For identical code fragments ("type-1", see Section II) and syntactically identical code fragments ("type-2", see Section II), AST-based detection has been introduced by [64]. Other methods for type-1 and type-2 include metrics-based clone detection [35], suffix tree-based clone detection [65], and string matching clone detection [66]. For a detailed survey of clone detection techniques, a good portrait is provided by Roy et al. [67] and Rattan et al. [68].

Several studies have addressed the problem of finding clone genealogy, given the underlying software repository commits. Otkin et al. proposed a clone origin detection approach [28]. Saha et al. introduced a clone genealogy extractor [31] and Godfrey et al. investigated software artifact provenance [29] using clone detection.

The technique used in this paper uses a fact commonly found in the literature: the Levenshtein distance is effective at finding meaningful clones.

Cordy and Roy have introduced NiCAD [34], a clone detection tool based on many prefiltering algorithms with a final filtering step using the length of the longest common subsequence (LCS). The LCS is dual to the Levenshtein distance and both can be viewed as roughly equivalent since the Levenshtein distance naturally provides a lower bound on the LCS length. The CLAN tool [35] by Merlo et al. also suggests to use LCS to filter results for better precision. The difference between NiCAD and CLAN is the number and the complexity of each step before the LCS filtering: NiCAD uses many lexical and syntactic preprocessing steps and CLAN uses a single metric projection step. The Levenshtein distance has also been suggested as a reference to benchmark clone detection. Tiarks et al. [69] created a small reference set inspired by the Levenshtein distance. An exhaustive set of all pairs satisfying a chosen Levenshtein distance threshold [47] was proposed by Lavoie et al. Without claiming the Levenshtein distance produces the absolute clone reference, it is still reasonable to assume it produces good results. Consequently, there is a motivation to use an approach which produces results similar to those computed with the Levenshtein distance.

Cova et al. [1] analyzed free phishing kits available on internet, executing them in a secured environment to observe emails sent to the attackers. In PhishEye [10], Han et al. set up a honeypot to collect phishing kits. By letting attackers execute their kits in a sandbox, the authors were able to analyze in-the-wild phishing attacks. In our paper, we do not execute any phishing kit, as we only perform static analysis.

Other authors performed analysis on the kits' files, without executing them. Imperva [7] carried out a static analysis of more than 1 000 kits, extracting from the source code useful information such as author's signature, email content, and so on. Mccalley et al. [2] studied the kits of a hacker group, revealing the presence of backdoors inside their kits. They were used to ex-filtrate the attack results to the authors, without the kit owner knowing it. Oest et al. [6] investigated kits' `.htaccess` files, which are used to block some IP addresses for instance. Thus, they described the means used by kit authors to avoid detection. Here, we are considering only source code files, not to extract information about attacks behaviour, but to measure the similarity between phishing kits.

CrawlPhish [11] is a framework to detect and categorize cloaking techniques in client-side JS scripts of known phishing sites. It also relies (partly) on source code similarity, but only for client-side JS, while we consider the server-side source code in PHP, JS and HTML in this paper.

To the best of our knowledge, source code similarity analysis have not been applied to server-side phishing kits so far.

VII. Future Work and Conclusions

In the paper, we perform the first multi-language analysis over a large set of over 20 000 actual phishing kits corresponding to about 180 MLOC of code, and provided a scalable solution to infer lineage amongst kits in a realistic context: absence of baseline and missing versions.

Kits similarity distribution seems consistent with the assumed hypothesis about kits propagation model. So, the kits evolution process can be interpreted as intertwined sequences of higher cost intellectual changes and lower-cost surface changes. Intellectual changes are more differentiated and possibly aim at camouflage, detection avoidance, and introducing technical changes. They may become "stem" kits for further propagation. Identical or near-identical surface changes exploit the ease of propagation of "successful" kits.

The proposed approach may help classifying new incoming phishing kits as "near-copy" or "intellectual leaps" from known and already encountered kits. This could facilitate the identification and classification of new kits as derived from older known kits.

This analysis could be further enhanced using extra information: using the date of collection, extracting brand from source code, analysing source code comments and obfuscation patterns are several ways to improve it. Other techniques can be applied, using ASTs instead of tokens for instead, to better take into account structure of source code. We are also planning on adding de-obfuscation of code to our analysis, and add support for other languages (e.g. CSS).

The composition of the hierarchical structure of a kit has been computed by the sum of the metrics of the components. Other ways of computing the composition of similarity should be investigated and compared to the reported one.

New kits are detected at a fast pace. The reported approach is global in the sense that all samples have to be processed together to determine the distribution. This may not be appropriate when the size of the available kit information grows large and increases fast. We plan to introduce incremental analysis strategies of kits and of similarity to address the scalability issue of kits analysis.

REFERENCES

[1] M. Cova, C. Kruegel, and G. Vigna, "There Is No Free Phish: An Analysis of" Free" and Live Phishing Kits," in *2nd Conference on USENIX Workshop on Offensive Technologies (WOOT)*, vol. 8, San Jose, CA, 2008, pp. 1–8.

[2] H. Mccalley, B. Wardman, and G. Warner, "Analysis of back-doored phishing kits," in *IFIP International Conference on Digital Forensics*. Springer, 2011, pp. 155–168.

[3] S. Zawoad, A. K. Dutta, A. Sprague, R. Hasan, J. Britt, and G. Warner, "Phish-net: Investigating phish clusters using drop email addresses," in *2013 APWG eCrime Researchers Summit*, Sep. 2013, pp. 1–13.

[4] PhishLabs, "How to Fight Back against Phishing," https://info.phishlabs.com/hs-fs/hub/326665/file-558105945-pdf/White_Papers/How_to_Fight_Back_Against_Phishing_-_White_Paper.pdf, 2013.

[5] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki *et al.*, "Data breaches, phishing, or malware?: Understanding the risks of stolen credentials," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1421–1434.

[6] A. Oest, Y. Safei, A. Doupé, G. Ahn, B. Wardman, and G. Warner, "Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis," in *2018 APWG Symposium on Electronic Crime Research (eCrime)*, May 2018, pp. 1–12.

[7] Imperva, "Our Analysis of 1,019 Phishing Kits," https://www.imperva.com/blog/our-analysis-of-1019-phishing-kits/, 2018.

[8] EC-Council, "How Strong is your Anti-Phishing Strategy?" https://blog.eccouncil.org/how-strong-is-your-anti-phishing-strategy/, 2018.

[9] Q. Cui, G.-V. Jourdan, G. V. Bochmann, and I.-V. Onut, "Proactive detection of phishing kit traffic," in *Applied Cryptography and Network Security*, 2021, pp. 257–286.

[10] X. Han, N. Kheir, and D. Balzarotti, "Phisheye: Live monitoring of sandboxed phishing kits," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1402–1413.

[11] P. Zhang, A. Oest, H. Cho, Z. Sun, R. Johnson, B. Wardman, S. Sarker, A. Kapravelos, T. Bao, R. Wang, Y. Shoshitaishvili, A. Doupé, and G.-J. Ahn, "Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing," in *42nd IEEE Symposium on Security & Privacy*, 2021.

[12] D. Manky, "Cybercrime as a service: A very modern business," *Computer Fraud & Security*, vol. 2013, p. 9–13, 06 2013.

[13] I. Fette, N. Sadeh, and A. Tomasic, "Learning to detect phishing emails," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 649–656.

[14] R. Verma, N. Shashidhar, and N. Hossain, "Detecting phishing emails the natural language way," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 824–841.

[15] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi, "An evaluation of machine learning-based methods for detection of phishing sites," in *International Conference on Neural Information Processing*. Springer, 2008, pp. 539–546.

[16] I. R. A. Hamid and J. Abawajy, "Hybrid feature selection for phishing email detection," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2011, pp. 266–275.

[17] M. Chandrasekaran, K. Narayanan, and S. Upadhyaya, "Phishing email detection based on structural properties," in *NYS cyber security conference*, vol. 3. Albany, New York, 2006.

[18] G. Stringhini and O. Thonnard, "That ain't you: Blocking spearphishing through behavioral modelling," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 78–97.

[19] W. Liu, G. Liu, B. Qiu, and X. Quan, "Antiphishing through Phishing Target Discovery," *IEEE Internet Computing*, vol. 16, no. 2, pp. 52–61, 2012.

[20] G. Ramesh, I. Krishnamurthi, and K. S. S. Kumar, "An efficacious method for detecting phishing webpages through target domain identification," *Decision Support Systems*, vol. 61, no. 1, pp. 12–22, 2014.

[21] Q. Cui, G.-V. Jourdan, G. V. Bochmann, R. Couturier, and I.-V. Onut, "Tracking phishing attacks over time," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 667–676.

[22] G.-G. Geng, X.-D. Lee, W. Wang, and S.-S. Tseng, "Favicon - a clue to phishing sites detection," in *eCrime Researchers Summit (eCRS), 2013*, Sept 2013, pp. 1–10.

[23] E. H. Chang, K. L. Chiew, S. N. Sze, and W. K. Tiong, "Phishing detection via identification of website identity," in *2013 International Conference on IT Convergence and Security, ICITCS 2013*. IEEE, 2013, pp. 1–4.

[24] Y. Zhang, J. Hong, and C. Lorrie, "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th International Conference on World Wide Web*, Banff, AB, 2007, pp. 639–648.

[25] A. P. E. Rosiello, E. Kirda, C. Kruegel, and F. Ferrandi, "A layout-similarity-based approach for detecting phishing pages," in *Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks, SecureComm*, Nice, 2007, pp. 454–463.

[26] T.-C. Chen, S. Dick, and J. Miller, "Detecting visually similar web pages: Application to phishing detection," *ACM Trans. Internet Technol.*, vol. 10, no. 2, pp. 5:1–5:38, Jun. 2010.

[27] S. Afroz and R. Greenstadt, "Phishzoo: Detecting phishing websites by looking at them," in *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*. IEEE, 2011, pp. 368–375.

[28] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2005.

[29] M. W. Godfrey, "Understanding software artifact provenance," *Sci. Comput. Program.*, vol. 97, no. P1, p. 86–90, Jan. 2015. [Online]. Available: https://doi.org/10.1016/j.scico.2013.11.021

[30] L. Barbour, F. Khomh, and Y. Zou, "Late propagation in software clones," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 273–282.

[31] R. Saha, C. Roy, and K. Schneider, "gcad: A near-miss clone genealogy extractor to support clone evolution analysis," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, 09 2013, pp. 488–491.

[32] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multi-linguistic token-based code clone detection system for large scale source code," vol. 28, no. 7. IEEE Computer Society Press, 2002, pp. 654–670.

[33] H. Basit, S. Pugliesi, W. Smyth, A. Turpin, and S. Jarzabek, "Efficient token based clone detection with flexible tokenization," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2007.

[34] C. Roy and J. Cordy, "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *International Conference on Program Comprehension*. IEEE Computer Society Press, 2008, pp. 172–181.

[35] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*, Monterey, CA, Nov 1996, pp. 244–253.

[36] E. Merlo, G. Antoniol, M. D. Penta, and F. Rollo, "Linear complexity object-oriented similarity for clone detection and software evolution analysis," in *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*. IEEE Computer Society Press, 2004, pp. 412–416.

[37] E. Merlo and T. Lavoie, "Detection of structural redundancy in clone relations," Ecole Polytechnique of Montreal, Tech. Rep. EPM-RT-2009-05, 2009.

[38] E. Merlo, T. Lavoie, P. Potvin, and P. Busnel, "Large scale multi-language clone analysis in a telecommunication industrial setting," in *2013 7th International Workshop on Software Clones (IWSC)*, 2013, pp. 69–75.

[39] N. A. Kraft, B. W. Bonds, and R. K. Smith, "Cross-language clone detection," in *20th International Conference on Software Engineering and Knowledge Engineering, SEKE 2008*, San Francisco Bay, CA, United states, 2008, pp. 54 – 59.

[40] F. Al-Omari, I. Keivanloo, C. K. Roy, and J. Rilling, "Detecting clones across microsoft.net programming languages," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, Kingston, ON, Canada, 2012, pp. 405 – 414. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2012.50

[41] T. Vislavski, G. Rakić, N. Cardozo, and Z. Budimac, "Licca: A tool for cross-language clone detection," in *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 512–516.

[42] M. Balazinska, E. Merlo, M. Dagenais, B. Laguë, and K. Kontogiannis, "Advanced clone-analysis as a basis for object-oriented system refactoring," in *Proc. Working Conference on Reverse Engineering (WCRE)*. IEEE Computer Society Press, 2000, pp. 98–107.

[43] F. Deissenboeck, B. Hummel, E. Juergens, B. Schaetz, S. Wagner, S. Teuchert, and J. F. Girard, "Clone detection in automotive model-based development," in *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society Press, 2008.

[44] J. Guo and Y. Zou, "Detecting clones in business applications," in *Proceedings of the Working Conference on Reverse Engineering*, 2008.

[45] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470–495, May 2009.

[46] C. K. Roy and J. R. Cordy, "A mutation/injection-based automatic framework for evaluating code clone detection tools," in *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ser. ICST '09. IEEE Computer Society, 2009, pp. 157–166.

[47] T. Lavoie and E. Merlo, "Automated type-3 clone oracle using levenshtein metric," in *IWSC11 Proceedings of the 5th International Workshop on Software Clones*, 2011, pp. 25–32.

[48] ——, "An accurate estimation of the levenshtein distance using metric trees and manhattan distance," in *Proceedings of the 6th International Workshop on Software Clones*, ser. IWSC '12. New York, NY, USA: ACM, 2012, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/1985404.1985411

[49] Q. Cui, G.-V. Jourdan, G. V. Bochmann, I.-V. Onut, and J. Flood, "Phishing attacks modifications and evolutions," in *European Symposium on Research in Computer Security*. Springer, 2018, pp. 243–262.

[50] S. Le Page, G. v. Bochmann, Q. Cui, J. Flood, G. Jourdan, and I. Onut, "Using ap-ted to detect phishing attack variations," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 2018, pp. 1–6.

[51] W. M. Khoo, A. Mycroft, and R. J. Anderson, "Rendezvous: a search engine for binary code," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*. IEEE Computer Society, 2013, pp. 329–338. [Online]. Available: https://doi.org/10.1109/MSR.2013.6624046

[52] J. Jang, M. Woo, and D. Brumley, "Towards automatic software lineage inference," in *Proceedings of the 22nd USENIX Conference on Security*, ser. SEC'13. USA: USENIX Association, 2013, p. 81–96.

[53] G. Canfora, F. Mercaldo, A. Pirozzi, and C. A. Visaggio, "How I met your mother? - an empirical study about android malware phylogenesis," in *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT, Lisbon, Portugal, July 26-28, 2016*. SciTePress, 2016, pp. 310–317. [Online]. Available: https://doi.org/10.5220/0005968103100317

[54] A. Cimitile, F. Martinelli, F. Mercaldo, V. Nardone, A. Santone, and G. Vaglini, "Model checking for mobile android malware evolution," in *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, 2017, pp. 24–30.

[55] T. Dumitraş and I. Neamtiu, "Experimental challenges in cyber security: A story of provenance and lineage for malware," in *Proceedings of the 4th Conference on Cyber Security Experimentation and Test*, ser. CSET'11. USA: USENIX Association, 2011, p. 9.

[56] W. M. Khoo and P. Lio', "Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware families," in *First SysSec Workshop 2011, SysSec@DIMVA 2011, Amsterdam, The Netherlands, July 6, 2011*. IEEE, 2011, pp. 3–10. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SysSec.2011.24

[57] A. Gupta, P. Kuppili, A. Akella, and P. Barford, "An empirical study of malware evolution," in *Proceedings of the First International Conference on COMmunication Systems And NETworks*, ser. COMSNETS'09. IEEE Press, 2009, p. 356–365.

[58] C. Darmetko, S. Jilcott, and J. Everett, "Inferring accurate histories of malware evolution from structural evidence," in *Proceedings of the Florida Artificial Intelligence Research Society Conference*, 2013. [Online]. Available: https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS13/paper/view/5884/6082

[59] M. Lindorfer, A. Di Federico, F. Maggi, P. M. Comparetti, and S. Zanero, "Lines of malicious code: Insights into the malicious software industry," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 349–358. [Online]. Available: https://doi.org/10.1145/2420950.2421001

[60] A. Sæbjørnsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su, "Detecting code clones in binary executables," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ser. ISSTA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–128. [Online]. Available: https://doi.org/10.1145/1572272.1572287

[61] M. Egele, M. Woo, P. Chapman, and D. Brumley, "Blanket execution: Dynamic similarity testing for program binaries and components," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14. USA: USENIX Association, 2014, p. 303–317.

[62] J. Ming, D. Xu, and D. Wu, "Memoized semantics-based binary diffing with application to malware lineage inference," in *ICT Systems Security and Privacy Protection*, H. Federrath and D. Gollmann, Eds. Cham: Springer International Publishing, 2015, pp. 416–430.

[63] ——, "Malware hunt: Semantics-based malware diffing speedup by normalized basic block memoization," *Journal of computer virology and hacking techniques*, Aug 2017. [Online]. Available: http://par.nsf.gov/biblio/10066920

[64] I. Baxter, A. Yahin, l. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees." in *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*, 1998, pp. 368–377.

[65] N. Göde and R. Koschke, "Incremental clone detection," in *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2009, pp. 219–228.

[66] S. Ducasse, O. Nierstrasz, and M. Rieger, "On the effectiveness of clone detection by string matching," *International Journal on Software Maintenance and Evolution: Research and Practice - Wiley InterScience*, no. 18, pp. 37–58, 2006.

[67] C. Roy and J. Cordy, "A survey on software clone detection research," School of Computing, Queen's University, Tech. Rep. Technical Report 2007-541, November 2007.

[68] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Information and Software Technology*, vol. 55, no. 7, pp. 1165 – 1199, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584913000323

[69] R. Tiarks, R. Koschke, and R. Falke, "An assessment of type-3 clones as detected by state-of-the-art tools," in *Workshop on Source Code Analysis and Manipulation*. IEEE Computer Society Press, 2009, pp. 67–76.