International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

Exploring Prototype-Based Clustering for Malware Detection: Insights from MutantX-S

Indravadan Patel
CGI Inc.

ABSTRACT: Computers and internet-based technologies are an essential aspect of modern life. Numerous network architectures are used to connect computers, and occasionally, it's feasible for a particular network or machine to be attacked by malicious software, or malware. Numerous negative outcomes, such as system damage, data theft, performance deterioration, spamming, and more, might arise from these attacks. Malware comes in a variety of forms, including as viruses, worms, spyware, rootkits, and many more. Every year, millions and millions of new malware samples are sent to antivirus research firms. The ever-increasing number of malware samples makes it impossible to examine each one separately. This results in a low detection rate of fresh malware samples due to a delay in the propagation of malware signatures. Researchers from Symantec Labs created Mutant X-S, a scalable malware categorization framework, to address this problem. MutantX-S is able to efficiently group samples according to how similar they are to one another. This framework offers a scalable solution to handle the enormous volume of malware that exists in the wild. The Mutant X-S is designed to enhance current dynamic behavior-based systems rather than replace them in order to improve malware program coverage and clustering accuracy [1].

KEYWORDS: Clustering, Hashing, MutantX-S, Malware, N-gram

INTRODUCTION

Computers and internet-based technologies are widely used in government, business, and personal activities in today's digitally connected world. This greater dependence results in a greater susceptibility to harmful software, or malware. These dangers—which include worms, viruses, spyware, rootkits, and more—can jeopardize systems by stealing confidential information, impairing system functionality, permitting illegal access, or starting mass spam operations.

Malware is increasing in both variety and volume at a startling rate. Over 403 million malware samples were gathered in 2011 alone, a startling 41% increase over the year before, according to Symantec's research. The time needed for manual analysis and signature updates makes it difficult for traditional signature-based detection systems to keep up with the exponential increase of new malware strains. Because of this, a lot of recently discovered threats avoid prompt identification.

Researchers at Symantec Labs developed MutantX-S, a scalable and automated malware classification framework, to get around these restrictions. MutantX-S uses prototype-based clustering approaches to effectively group structurally and behaviorally similar malware samples rather than evaluating individual samples. Faster categorization, higher detection rates, and more efficient use of resources for in-depth research are made possible by this method, opening the door for proactive and flexible cybersecurity defenses.

ARCHITECTURE

Recommended font sizes are shown in Table 1. Figure 1 above shows a generalized representation of the Mutant X-S design. A collection of unprocessed malware samples is fed into MutantX-S. To ascertain whether malware files are handled using runtime packers, it makes use of pre-existing technologies like PeID, a runtime packer identifier. After that, it disassembles them using an interactive disassembler like IDA Pro and unpacks them using a generic algorithm created by the developers of MutantX-S. After that, n-gram analysis is carried out, feature vectors are created for every sample, and features are retrieved from opcodes obtained from the deconstructed machine-level instructions. A prototype-based clustering technique is then used to group these feature vectors into families according to their Euclidean distance to a functionally defined set of prototypes. Regarding scalability, MutantX-S increases the speed of vector distance computation at the lowest possible cost in clustering accuracy by compressing the feature vectors using a hashing method.

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

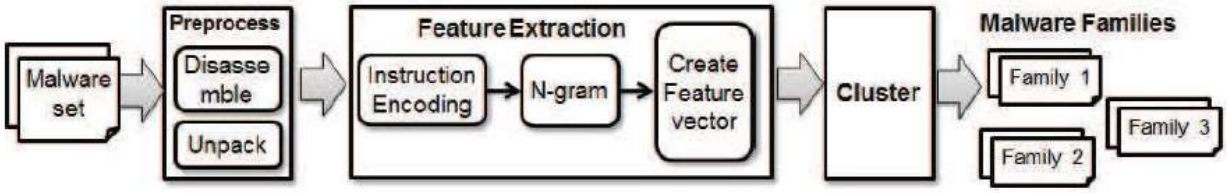

Figure 1: Mutant X-S Architecture

MALWARE PREPROCESS

Because of the level of obfuscation used on malware, preprocessing is essential to performing static analysis on it. Symantec Research estimates that a variety of runtime packing strategies conceal about 80% of malware variants discovered in the wild. Figure 2 shows how the packing is done.

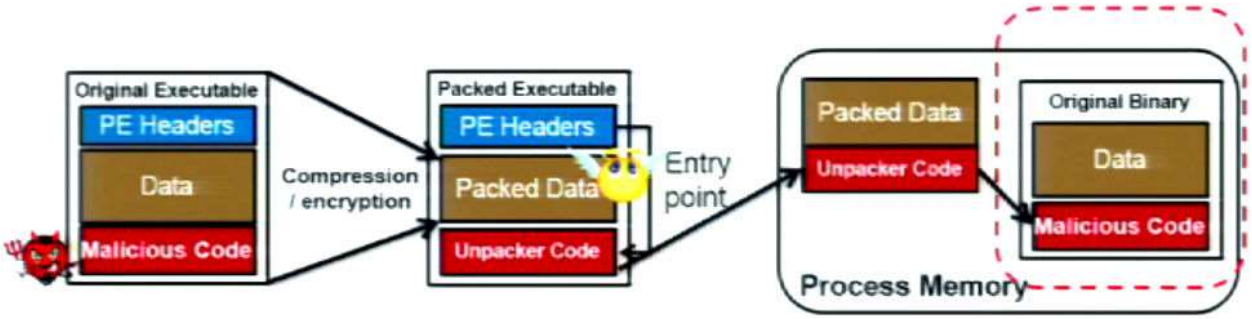

Figure 2: Malware Runtime-Packing based Obfuscation

The most common type of malware that is unobfuscated is a binary executable. This binary executable file has its own PE headers and is known as a portable executable (PE) file. Often called "fluff code," the malware source contains a portion of code designed to appear like innocuous data. The harmful code itself appears either after or inside the fluff code. Using a runtime packer or compiler, the entire original PE file is compressed or encrypted into a new PE file with distinct headers, and unpacker code is added after the packed contents. The data in process memory is unpacked by this unpacker code, which also provides a point of entry to the malicious code in its raw form. The pathogen can maintain its original functionality while disguising itself as innocuous code in this fashion. Processing-wise, unpacking is costly. As long as the original instructions can be examined and features can be retrieved, Mutant X-S does not need to ensure that the unpacked programs are executable. In order to address the issue of obfuscation through runtime packing, Mutant X-S provides a general unpacking mechanism.

GENERIC UNPACKING ALGORITHM

MutantX-S's Generic Unpacking Algorithm takes advantage of an unpacking process feature that allows it to write the original program to memory and run it. Therefore, we are able to identify pages that might contain the malicious instructions that are the subject of additional investigation by continuously monitoring memory excess and identifying those memory pages that are altered and then executed.

Mutant X-S will first load the packed malware into the memory as part of this unpacking method, after which it will mark every memory page as executable but not writeable. The original code will then begin to be written into the memory by the unpacker code. At the same time, a write exception happens because of the page marking. Mutant X-S designates the page as "dirty" and modifies the permission to writable but not executable when the exception occurs. After then, execution keeps going in this manner until the unpacked code is finished and begins to switch to freshly created code. Because the "Dirty" page does not currently have executable permissions, an execute exception occurs. When this exception happens, Mutant X-S adds dummy PE headers and dumps the memory image containing the original malicious code into a freshly made binary. Additionally, the address where this execution exception occurs is changed as the entry point. The reconstructed binary is then passed to the disassembler by mutant X-S, where it is broken down into a series of machine-level instructions that will be utilized to extract features.

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

Unpacking accomplishes two goals. It executes the packed binary, finds the proper original entry point (OEP) for the dumped image, and dumps the running process's memory image at the right moment when the binary is probably going to finish unpacking. Correct disassembly and feature extraction depend on locating the OEP. Given that there is no other reference to this "skipped" section of the code, an incorrect OEP results in the disassembler missing all of the instructions between the original and incorrectly recognized entry points.

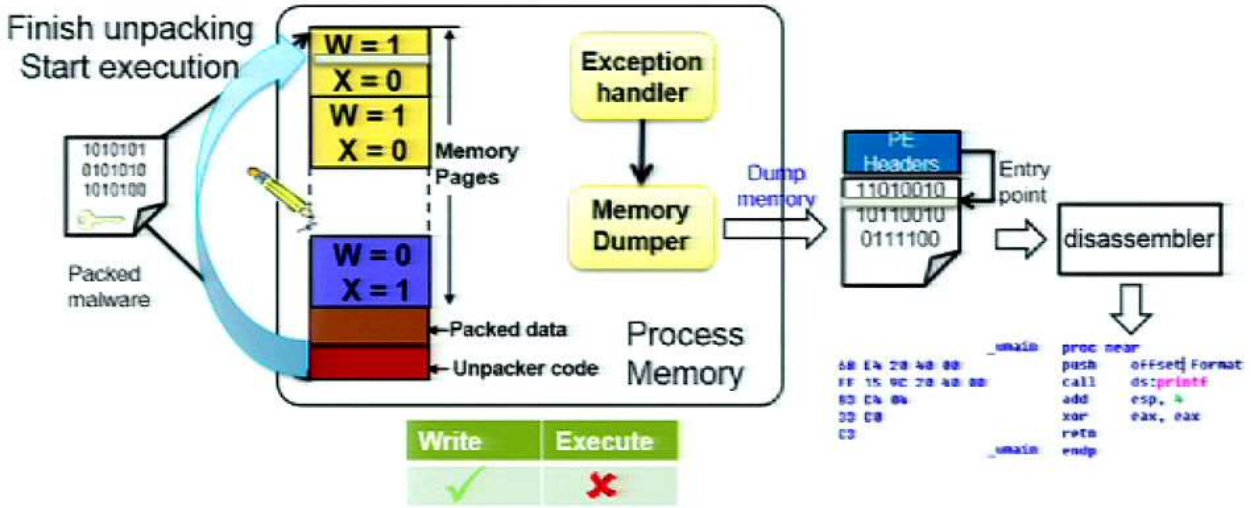

Figure 3: Generic Unpacking Algorithm

FEATURE EXTRACTION

MutantX-S uses the dumped memory pictures to create a new portable executable file after the unpacking algorithm has accurately identified the OEP. An appropriate starting point for disassembly instructions is established when the OEP is accurately identified. MutantX-S breaks down a malicious program into a set of machine instructions by using IDA Pro, an interactive disassembler. The feature extraction procedure then uses these instructions after they have been translated into their corresponding opcodes. Comparing the similarities between the many malware samples that were gathered and deconstructed by examining their individual machine instructions is the core feature of MutantX-S. One issue with machine instructions is that some mnemonics have different functions, and these mnemonics (mov, jmp, etc.) might be ambiguous when used for feature extraction. Malware is dynamically changed for a variety of reasons, including mutation, polymorphism, and obfuscation. Semantically comparable instructions are frequently substituted for one another in these modifications. This implies that there is no tolerance for differences in the syntax of the instructions in order to ensure accuracy when analyzing and comparing them.

On the other hand, correctness is seriously jeopardized if all kinds of variants are to be accepted. By taking advantage of the x86 instruction format and using the opcodes of the instructions as a representation of the semantics, MutantX-S strikes a compromise between the two extremes. There are several advantages to using opcodes rather than the generic mnemonics of machine instructions. First, opcodes—more especially, certain opcode sequences—can be generalized to represent malware variations within a given malware family. The majority of malware on the internet nowadays is variations on existent forms of malware created either by manually altering components of existing malware or, more frequently, by malware-generating tools. This is because it is challenging to design spyware that is both efficient and effective. Malware variants of a same archetype are regarded as belonging to the same family. Because they are inherited from the same code, members of the same malware family frequently have similar instructions. Instructions can be changed between variations by relinking, rebinding, and rebasing operands. MutantX-S exhibits some resistance to low-level mutation while still accurately representing the underlying semantics since it employs opcodes and disregards operands. This feature of MutantX-S enables it to represent a specific malware sample in a way that accurately reflects its functioning. Opcode sequences provide a more accurate representation of semantics than machine instruction sequences, according to the evaluation of MutantX-S. Machine instruction mnemonics frequently oversimplify the processor actions they represent. As a result, commands with quite different interpretations seem to be similar. The mnemonic "move" can be used to

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

illustrate this. For example, a critical operating system function, such turning on or off debugging or employing interrupt controls, may be the cause of a specific value being moved using the "mov" command to a control or debug register. Moving values between ordinary registers (e.g., eax, edx, …) should not be compared to this. Moving data from a register to memory (memory load vs. memory store) should ideally be different from moving data from main memory to a register. Nonetheless, both operations are represented by the mnemonic "mov." This may lead to the deceptive appearance of similarity between these two quite diverse activities. Opcodes, on the other hand, provide a finer level of differentiation between these semantically distinct commands. Better clustering accuracy is thus made possible by the use of opcodes in such analysis.

MutantX-S describes the substance of malware programs by analyzing n-grams, which are sequences of opcodes of n length. Using a fixed-length window across a string of words—or in our instance, opcodes—and taking into account a subsequence of length N at each place is known as n-gram analysis. The resulting N-grams are brief instruction patterns that naturally capture the program's fundamental meaning. MutantX-S builds a feature vector V in a |S|-dimensional space where |S|=|O|^ after creating these n-grams where O is a collection of every conceivable outcome. The number of instances of a specific n-gram is represented by each dimension in the feature vector space. MutantX-S may use this to determine how similar two malware programs are by calculating the Euclidean distance between their feature vectors in vector space [3] . Explicit feature representation is made possible by using geometric computation of vector similarities rather than other metrics. Tracing each n-gram's contribution to the clustering process back to the source code patterns is made possible by explicit feature representation. These trends might be related to the intrinsic traits of certain malware families, particularly those that are extremely common for samples that belong to the family but uncommon for samples that do not. Therefore, the code segments that these patterns reflect can be utilized as signatures to identify more recent malware variants.

CLUSTERING ALGORITHM

Following the creation of these feature vectors, MutantX-S organizes samples according to shared characteristics. To keep up with the speed at which new malware is created and distributed, MutantX-S aims to process remarkably huge sets of malware samples. Traditional clustering techniques, such K-Means clustering and other hierarchical and partitioning-based clustering algorithms, don't scale well since their time complexity increases at least quadratically with the number of samples. To get around the time complexity problem, MutantX-S uses a prototype-based clustering method with near-linear runtime complexity and a hash kernel that efficiently compresses the high-dimensional vector into a low dimensional space.

HASHING KERNEL

Kernel methods eliminate the need to calculate the coordinates of the data in a high-dimensional feature space [4]. This is especially useful when the input data can be linearly split in a high-dimensional feature space but has a non-linear decision boundary. The original space in MutantX-S has a very high number of dimensions, which presents the opposite issue. The computing complexity of the vector distance depends on the number of dimensions, and D in the n-gram grows exponentially with N. Specifically, the number of dimensions D=|O|^N, where |O|>200 and |O| is the number of distinct opcodes. As a result, a distributed feature vector with more than 8 million dimensions will be produced by even a comparatively small n-gram, such one of size 3. With a high number of malware samples, this problem becomes unmanageable because the n-gram size needs to be at least 3 or 4 in order to be considerably descriptive. To get over this problem, MutantX-S uses a hashing technique that uses the mapping function $\phi: X \rightarrow {}^m$ to hash the high dimensional input vector $x \in {}^n$ into a lower dimensional feature space ^m. The hashing approach reduces the feature vector to a compact representation since $m \ll n$, which makes it possible to handle huge datasets with a significantly lower investment of resources like memory and processing power. When employing dimensionality reduction, the hash kernel incurs a logarithmic complexity yet almost maintains the vector distance. The uniform hash function H: {N-gram}→[1..m], which is the hash function that MutantX-S uses hashes N-gram straight into a position in a feature vector of length m. When a collision occurs—which is defined as when two or more N-grams map to the same location - MutantX-S adds up all of the collisions' counts to use as the new vector's value. Let v and v' stand for the original feature vector that was taken from the opcode sequences for malware M and M', and let ξ represent the mapping from the N-gram $(o_1, o_2, \ldots, o_N) \in S$ to the index in vector v. After that, the hash feature map ϕ can be described as

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

$$\phi_i(v) = \sum_{l:H(o)=i, l\epsilon S} v(\xi(o))$$

Hence, the definition of the Euclidean distance between malware samples M and M' is

$$d_\phi(M, M') = \left|\left|v - v'\right|\right|_\phi = \left|\left|\phi(v), \phi(v')\right|\right|$$

A trade-off between computational difficulty, storage overhead, and clustering accuracy determines the low dimensional vector's length, m. A smaller m is associated with a shorter vector length, which speeds up distance computation and reduces memory usage. However, it also increases the likelihood of collisions, which can result in over-compression and worse cluster accuracy.

PROTOTYPE-BASED CLUSTERING

The complexity of traditional clustering methods, such the widely used hierarchical clustering, is usually more than linear with respect to the amount of the input data. For instance, the complexity of the k-means and hierarchical clustering algorithms is O(n^kd) and O(n^2 log⁡ 〖n)_, respectively. When trying to cluster a big sample amount of data, this leads to an unreasonable computation time. Empirical evidence has demonstrated the effectiveness of prototype-based clustering algorithms, while MutantX-S specifically uses a prototype-based method with a nearly linear complexity. First, a set of representative prototypes for every family is extracted using prototype-based clustering techniques. The closest prototype in the feature space is linked to the remaining data points, or those that aren't specifically designated as prototypes. By restricting the clustering to the tiny portion of the original data that are designated as prototypes, prototype-based algorithms are able to reduce computing time. Prototype-based clustering techniques consist of two primary steps: Clustering with Prototypes and Prototype Extraction.

Prototype Extraction: The representational quality of the clusters produced by the algorithm is significantly impacted by the prototype selection. Prototypes can precisely depict the distribution of the input data when positioned correctly. This makes it possible to draw precise class borders in the feature space. Gonzalez has proposed an approximate algorithm to solve the NP-Hard problem of determining the number of prototypes [9]. The datapoint that is most far from the previous prototypes is chosen as the next prototype by this algorithm, which selects prototypes iteratively. In this algorithm, the initial prototype is selected at random. Until every prototype is within a threshold distance P_max from every other prototype, the iterative procedure is repeated. This approach has the advantage of having a near-linear runtime complexity, precisely O(kn), where k is the number of prototypes the algorithm selects. The number of malware families is represented by the value of k, which depends on the data distribution. When k and P_max are chosen optimally, the algorithm's complexity increases linearly with the size of the input data sample (n). Clustering with prototypes: The algorithm's effectiveness can be ascribed to the fact that it exclusively uses the prototypes for hierarchical clustering. The prototypes are initially singleton clusters, and the two closest clusters are iteratively merged. The process stops when the distance between the two closest clusters exceeds a predetermined threshold, Min_d. The algorithm's propagation stage has a linear complexity. The complexity of standard hierarchical clustering techniques is usually O(n^2 log⁡ 〖n 〗) . The speed-up achieved by the method in MutantX-S is at least (n/k)2. As a result, the complexity O(n^2 log⁡n) becomes O(k^2 log⁡k), where n is the size of the input data and k is the number of prototypes selected from that data. This is because the method is only run on the prototypes.

EXPERIMENTAL EVALUATION

Two data sets were used to assess the accuracy and efficiency of MutantX-S. The reference set, also known as set (1), consists of 4832 malware files with labels created by security professionals working for a sizable anti-virus organization. As a result, the labels are deemed to be reasonably trustworthy. Although there are 132,234 malware samples in the set known as (2), the labels are very speculative because they were obtained from anti-virus scanners. Samples from 20 different families are included in Set (1); Table 2 shows their distributions. MutantX-S's clustering algorithm's heuristic parameters can be configured, assessed, and adjusted due to the labelling's dependability in set (1), the reference set. The algorithm's scalability is assessed using the larger set (2).

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

Table 1: Malware families of the reference data set: set (1)

Family	#	Family	#	Family	#
Pilleuz	500	Bredolab	301	Tidserv	59
Koobface	496	Vundo	249	Waledac	34
Silly	489	Almanahe	241	Ackantta	32
Fakeav	489	Sasfis	199	Mebroot	26
Zbot	459	Graybird	166	Hotbar	21
Banker	449	Gammima	126	Qakbot	17
Virut	361	Mabezat	107		

EFFECTIVENESS OF UNPACKING ENGINE

Eight widely used packing algorithms were employed to pack a malware sample in order to evaluate the unpacking procedure. PEcompact, EXECryptor, EXEStealth, VMprotect, ASprotect, UPX, NSPack, and Armadillo were specifically utilized algorithms. The MutantX-S generic unpacker was used to unpack the sample once the packing methods had been applied. The freshly unpacked samples were then compared to the original samples to see how comparable they were. Ideally, there would be no difference between the two samples. It is not possible for the samples to be identical since MutantX-S does not recreate the import table and instead dumps the unpacker code into a file for memory. Furthermore, clustering does not require that the samples be identical [2]. Given this, the evaluation's similarity metrics are the Euclidean distance between the samples' N-gram feature vectors (NG) and the difference in the instruction count (IC), as these two variables have a direct impact on clustering accuracy. MutantX-S typically recovered the majority of the original binaries with only a 1-6% increase in ICs. The inclusion of unpacking processes in the leaked memory image is the cause of this rise. MutantX-S was also rather successful in terms of the feature vectors. In general, they were determined to have a distance of less than 0.1. The distance values were normalized between 0 and 1, where 0 indicates they were identical and 1 indicates they were completely unlike. However, MutantX-S completely failed on some packers. A packed version of the binary was frequently still present in the Armadillo packer sample, according to a memory dump of the sample. An analysis of Armadillo's functioning revealed that it unpacks an intermediate executable onto the disk and launches a new process to execute it. As a result, the original virus instructions were still missing from the memory dump of the file that contained Armadillo. Additional causes for failure included the malware's occasional refusal to operate on a virtual computer and the fact that some unpacking procedures took longer than the predetermined time limit. Table 2 displays the unpacking algorithm's effectiveness.

Table 2: Unpacking Effectiveness (IC: Instruction Count; NG Dist: N-gram Difference)

Packer	Difference in IC	NG Distance
PEcompact	0.88%	0.068
EXECryptor	3.20%	0.176
EXEStealth	0.88%	0.071
VMprotect	2.50%	0.10
ASprotect	6.70%	0.133
UPX	0.88%	0.068
NSPack	0.87%	0.069
Armadillo	-	-

MALWARE CLUSTERING ACCURACY

Testing and calibrating MutantX-S on the reference set (1) was the initial stage in determining the accuracy of the malware clustering component of the program. The evaluations conducted by MutantX-S's creators were conducted on a computer running Ubuntu 10.4 with an i7 3.0G CPU and 12GB of RAM. The primary measures utilized to assess MutantX-S's accuracy were precision and recall. Assume that n input malware samples can be categorized into a collection of clusters O where O={O_1,O_2,…,O_o } according to the labels given in the reference data set (1). Since C={C_1,C_2,…,C_C}, it is assumed that MutantX-S has an output set of clusters

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

C. The exactness of the clusters, or how closely each cluster resembles the original classes, will subsequently be measured by precision P. The recall R, which can be thought of as a metric characterizing the completeness of each cluster, describes how dispersed the classes or families are across the clusters. Precision P and recall R have the following formal definitions:

$$P = \frac{1}{n}\sum_{i=1}^{c} \max(|C_i \cap O_i|, |C_i \cap O_2|, \dots, |C_i \cap O_o|)$$

$$= \frac{1}{n}\sum_{j=1}^{o} \max(|O_j \cap C_1|, |O_j \cap C_2|, \dots, |O_i \cap C_C|)$$

Only when every sample in each cluster C_i belongs to the same family does P=1. Even though there might not be just one family in the cluster, R=1 only occurs if all malware samples from that family are included in it. Figure 4 displays the clustering's precision, recall, and runtime at different P_max and Min_d criteria. MutantX-S's developers employed a 4-gram with N=4 and 12 hash bits in their experiment, which means that the 4-gram is mapped into 2^12 hash bins.

As can be shown in Figure 4, MutantX-S cluster samples with an average precision of 0.80 and a range of 0.72 to 0.89. It was demonstrated that two of the methods employed in previous studies, notably Bayer U. et al. on Scalable behaviour-based malware clustering in 2009 and Rieck K. et al. for the Berlin Institute of Technology in 2009, had higher precision, 0.996 and 0.984, respectively. MutantX-S's creators assume that the discrepancy in accuracy results from various malware sets and possibly inaccurate labelling applied throughout the testing. They assume that "high-level generalization of behaviour at the cost of running time and limited coverage" is the reason for dynamic-behaviour techniques' improved accuracy. But in contrast to the other methods, MutantX-S offers far greater scalability without sacrificing a respectable level of accuracy and precision. Figure 4 serves as an example of this, demonstrating that the clustering process for the complete reference data-set (1) takes less than 30 seconds. K-means and hierarchical clustering, on the same dataset, took 32.3 and 51.3 seconds, respectively, with a precision of 0.75 and 0.82. MutantX-S's recall is found to be between 0.3 and 0.4 overall. The high degree of variation among variants, which may be brought about by poorly labelled samples, unrecognized packers, or highly obfuscated binaries, may be the source of the low recall level. MutantX-S frequently divides families into sub-families as a result of this variability, which lowers recall. Indeed, it was demonstrated during testing that MutantX-S could generate over 50 clusters for the set (1), which included 20 malware types. These findings also demonstrate that P_max significantly affects clustering speed. The algorithm is forced to select more prototypes when P_max is small, which lengthens the computation time. Clustering accuracy was found to be significantly impacted by min_d. The merger process might be stopped by a smaller inter-cluster distance. However, this can be advantageous because it reduces the likelihood of unrelated prototypes merging. However, this comes at the cost of over-fitting clustering, as the algorithm will likely produce multiple small clusters.

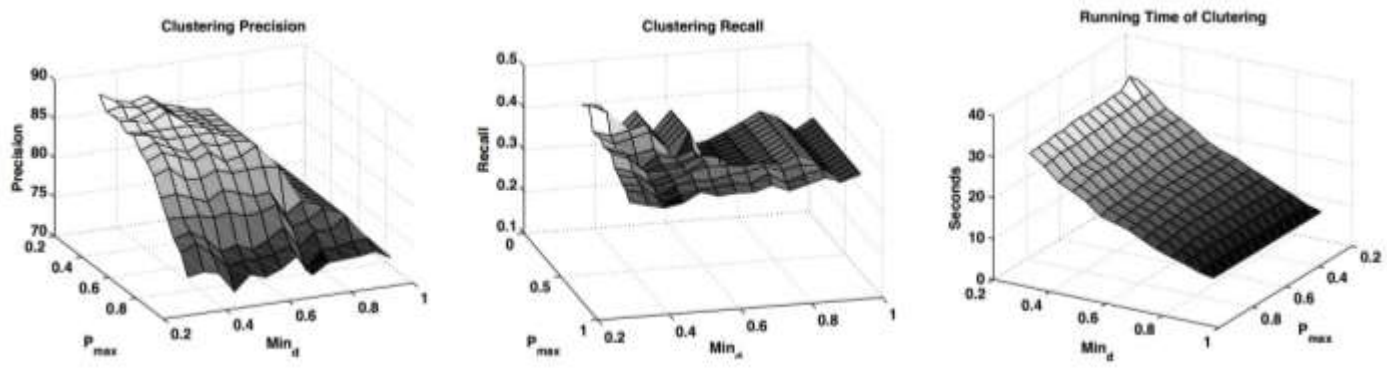

Figure 4: Precision, Recall, and Runtime of MutantX-S

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

VALIDITY OF THE HASHING TRICK

Regarding the hashing trick's appropriateness, the possibility of information loss as a result of dimensionality reduction brought about by compression is a problem. The reference data set was clustered using varying numbers of hash bins in the experimental evaluation of the hashing method (1). MurmurHash 2.0 is the hashing function used in MutantX-S. This function has a high throughput, a uniform value distribution, and good collision resistance. MutantX-S was also run on the original feature vectors without the hashing method for assessment purposes. The outcome of that specific run served as a standard for the best outcome with no data loss. Figure 5 compares the precision, clustering time, and memory needs for different hash sizes, including the hash-free benchmark.

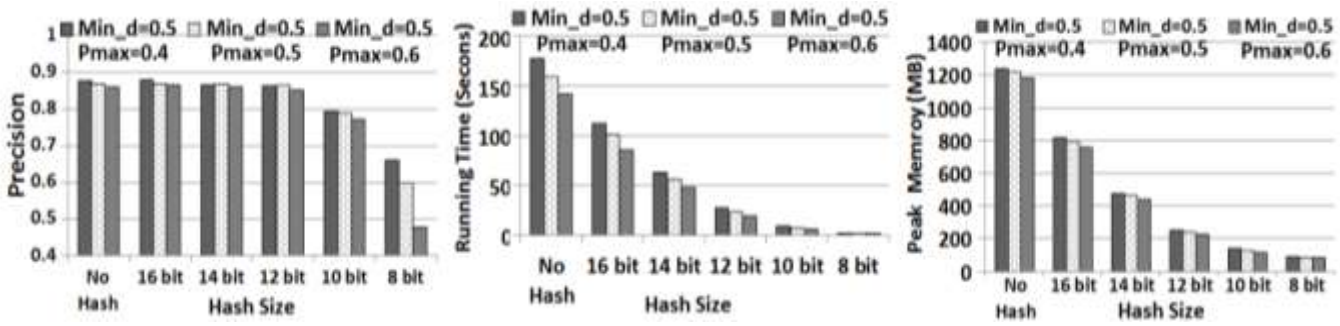

Figure 5: Precision, time, and peak memory with hash size from 2^8 to 2^{16} and with no hash trick

The bars in Figure 5 show the outcomes produced by different parameter combinations. As hash size increases, precision improves, according to the precision chart in the picture. This is explained by a decrease in the likelihood of collisions. Actually, the hashed feature vectors perform similarly to the un-hashed vectors after a specific hash size threshold, which depends on other factors. The clustering precision is 0.864 and the un-hashed feature precision is 0.0868 when there are more than 212 hash bins. The precision falls to less than 0.5 for some parameter combinations as the hash size is reduced, especially when the number of bins is reduced to 28. The collision of numerous crucial features, specifically those indicating that various families are mapped into the same hash bin, is the cause of this drop in precision. As a result, having a big hash size is usually preferred. On the other hand, Figure 5's middle and right plots demonstrate how a modest hash-size significantly lowers the algorithm's memory and runtime needs. This is due to the fact that shorter feature vectors translate into fewer calculations needed to determine the Euclidean distance between them. According to the charts, a reduction in runtime from two minutes to less than ten seconds and a drop in memory requirements for the calculations from 800 Mb to less than 100 Mb are the reasons for the hash size going from 16 bits to 8 bits. A 12-bit hash function strikes a fair mix between precision, runtime, and memory requirements, based on the tests conducted by the MutantX-S developers. The trials also demonstrate that a hash function is required for many malware strains since the memory need becomes an unmanageable issue.

IMPACT OF N-GRAM ON PERFORMANCE

The algorithm's descriptiveness is directly correlated with the size of the N-gram [6]. However, the dimensionality of the feature vector space, m^N, where m is the total number of different opcodes, increases exponentially with larger N-gram sizes. Generally speaking, N-grams of sizes 3 and 4 provide adequate descriptiveness without requiring an excessive amount of memory or computing time. MutantX-S's hashing technique enables dimensionality reduction and, thus, performance evaluation of a large N. Figure 6 displays the evaluation's findings.

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

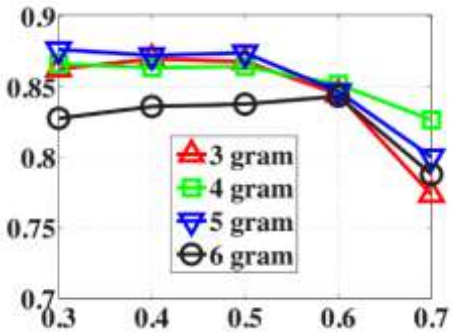

Figure 6: Precision of clustering with varying size N-grams

SCALABILITY OF MUTANTX-S

MutantX-S's scalability was assessed utilizing data set (2), which included more than 130,000 samples. The full set was subjected to many runs of MutantX-S with different parameters. Figure 7 illustrates the outcomes of these experiments.

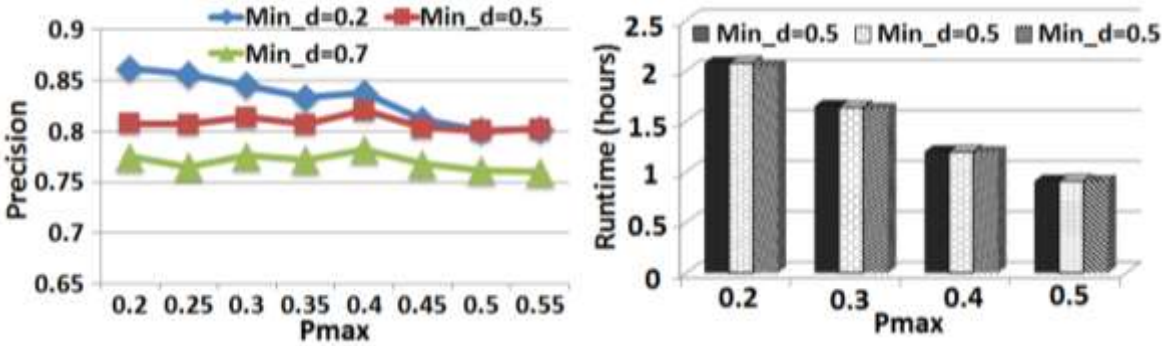

Figure 7: Plotted results of scalability of MutantX-S

The entire amount of time needed to cluster each set is shown in the figure. The runtime of MutantX-S is significantly impacted by the value of P_max; for example, a difference of 0.5 and 0.2 in P_max is one hours and half an hour, respectively. A lower runtime is the outcome of a higher P_max value. However, increasing P_max can drastically reduce accuracy. Precision can be reduced by over 10% for every 0.5 to 0.2 decrease in P_max. This is due to the fact that prototypes created within the space will be able to cover a vast region if P_max is large. This makes it more likely that unrelated samples will be included in the prototype-related cluster. The full 130,000+ samples in data set (2) can be clustered in less than an hour and a half with a precision of close to 0.82 if appropriate parameters are chosen, such as Min_d=0.5 and P_(max)=0.4. Unfortunately, because of the previously indicated sub-partitioning of malware families into smaller clusters, the recall for this identical set of parameters is approximately 0.25. This configuration's peak memory use was approximately 3.6GB. This demonstrates that MutantX-S may offer a way around the scaling problem that heuristic clustering techniques present [5]. It also shows that MutantX-S may be able to keep up with the rate at which new malware strains are found.

PREDICTING LABELS OF UNKNOWN MALWARE

Using pre-labelled data, the previously mentioned tests evaluate MutantX-S's performance. MutantX-S evaluating incoming malware and forecasting family labels would be a plausible scenario. The malware that enters the system is linked to and identified as its closest relative, or more precisely, the data point that is nearest to it in terms of Euclidean distance. Malware samples need to be organized based on when they were created in order to replicate such a situation. The IMAGE_FILE_HEADER, a common PE header, is used to extract the malware's creation time. The compiler sets the timestamp in this PE header during compilation. Figure 8 illustrates how one year's worth of malware was grouped specifically using this timestamp.

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

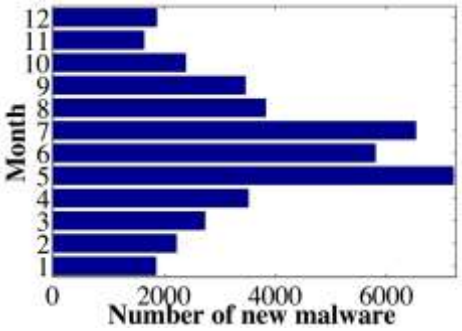

Figure 8: Number of new samples each month used to evaluate prediction capability

Based on development time, the malware program is divided into training and testing sets. This is done to mimic a real-world situation in which an antivirus vendor tries to label new malware samples (testing set) after having previously examined malware (training set). Samples from July through December make up the testing set. Three training sets are selected: the first contains samples from January through one month before the selected training month; the second contains samples from just the six months before the testing month; and the third, which serves as the experiment's control, is chosen from the first six months regardless of the testing month. Based on the labels of "constituent" malware samples, MutantX-S generates a set of clusters C_i (i=0,1,…,n), each of which has a label L(C_i). Next, MutantX-S labels the malware L(x_j) in the test month according to the label of the closest cluster after identifying the family of a recently received sample x_j. The following is a description of a generalized example:

$$L(x_j) = L(C_i)\ where\ d(x_j, C_i) = \min\left(d(x_j, C_k)\right) \forall k = 0, 1, \dots, n$$

The label produced by this procedure is then contrasted with sample x_j's initial label. Figure 9 shows a plot of the simulation's outcomes.

As the green line in the figure illustrates, previously acquired information becomes outdated very quickly. Regardless of the testing month, this instance is consistent with the third test set, which specifically uses the first half of the year. In the other two lines, the data is dependably accurate, creating a stark contrast. The enormous volume of malware being created and distributed makes it impractical to save all of the malware samples that have already been gathered. However, the chart shows a slight decrease in accuracy between the red and blue lines, which represent the cases when all prior months are used and the case where only the previous six months are used, respectively.

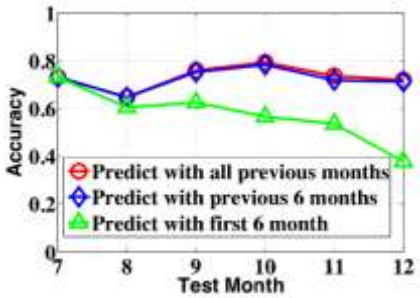

Figure 9: Accuracy in applying MutantX-S to predict family labels for unknown malware

LIMITATIONS AND IMPROVEMENTS

The MutantX-S has a number of exploitable flaws. MutantX-S is susceptible to code obfuscation, which happens at the instruction level, because it is based on static analysis. Furthermore, the Generic Unpacking Algorithm of MutantX-S is ineffective against sophisticated packers. Advanced obfuscation techniques, such driver-level protection, or anti-debug and anti-emulation measures, like the Armadillo runtime packer's methodology, are used by some runtime packers. A number of anti-disassembly strategies are also present; Yason M. discusses many of these in The Art of Unpacking. By confounding the disassembler, these strategies undermine the functionality of interactive diassemblers like IDA Pro. Among other things, the strategies may involve combining

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

data and code, creating impossible conditional leaps to the middle of a future instruction, and more. The current version of MutantX-S does not incorporate the methods that have been suggested to get around these issues, such as those put forth by Kruegel C. et al. in the 2004 conference paper Static Disassembly of Obfuscated Binaries. Heavy machine-level obfuscation can also be implemented by using instructions that are syntactically different but semantically similar. Techniques like the ones suggested by Raber, J. et al. in the work Deobfuscator and Udupa, S. K. et al. in Reverse engineering Obfuscated Code can be used to enhance the pretreatment step by adding a normalization of the malware codes. Despite not having these drawbacks, dynamic approaches to malware clustering have scalability issues and are susceptible to their own set of evasion tactics. The purpose of MutantX-S is to support dynamic approaches to malware clustering, not to replace them. Similarity-based clustering techniques, both static and dynamic, are comparable to the situation of parasitic malware that inserts itself into host executables. More research is needed on these kinds of parasitic malware.

CONCLUSION

A solution to static clustering that is both easily scalable and reasonably accurate is MutantX-S. It's a clever and useful method of reducing the computing time usually required by heuristic clustering algorithms is which use prototypes to group malware variants according to their similarities. Its use of N-gram analysis on opcode sequences makes it possible to depict malware samples in a concise but informative manner. Its generic unpacking algorithm maximizes its ability to analyse the actual instructions found in malware source code by effectively fighting the several kinds of runtime packing algorithms utilized by malware samples. The algorithm's potential for scalability is also significantly increased by the hashing kernel's implementation for dimensionality reduction through compression. It has proven that it can handle 132,234 malware samples in a matter of hours, thanks to all these characteristics. When combined with dynamic malware analysis, MutantX-S offers a workable way to deal with the rapid release of new malware samples into the wild.

REFERENCES

1. Faruki, P., Bhan, R., Jain, V., Bhatia, S., El Madhoun, N., & Pamula, R. (2023). A Survey and Evaluation of Android-Based Malware Evasion Techniques and Detection Frameworks. Information 2023, 14, 374.
2. Chhabra, A., Masalkovaitė, K., & Mohapatra, P. (2021). An overview of fairness in clustering. IEEE Access, 9, 130698-130720.
3. Poornima, S., & Subramanian, T. (2022). Evolution of Deep Quantum Learning Models Based on Comprehensive Survey on Effective Malware Identification and Analysis. Artificial Intelligence, Machine Learning and Blockchain in Quantum Satellite, Drone and Network, 83-105.
4. Nadim, M., Lee, W., & Akopian, D. (2023). Kernel-level rootkit detection, prevention and behavior profiling: A taxonomy and survey. arXiv preprint arXiv:2304.00473.
5. Haq, I. U., & Caballero, J. (2021). A survey of binary code similarity. Acm computing surveys (csur), 54(3), 1-38.
6. Balande, B. B., Kolte, D. M., Manza, R. R., & Revate, S. S. (2023, November). Literature Review on N-Gram Text Classification Models for Hotel Reviews Sentiment Analysis. In International Conference on Computational Intelligence (pp. 641-655). Singapore: Springer Nature Singapore.
7. BAYER, U., COMPARETTI, P., HLAUSCHEK, C., KRUEGEL, C., AND KIRDA, E. Scalable, behavior-based malware clustering. In Proc. of the 16th NDSS (2009).
8. HU, X., BHATKAR S., GRIFFIN K., SHIN K. G., MutantX-S: Scalable Malware Clustering Based on Static Features. USENIX Annual Technical Conference (2013)
9. KRUEGEL, C., ROBERTSON, W., VALEUR, F., AND VIGNA, G. Static disassembly of obfuscated binaries. In Proceedings of the 13th conference on USENIX Security Symposium (2004).
10. RABER, J., AND LASPE, E. Deobfuscator: An automated approach to the identification and removal of code obfuscation. Reverse Engineering, Working Conference on (2007)
11. RIECK, K., TRINIUS, P., WILLEMS, C., AND HOLZ, T. Automatic analysis of malware behavior using machine learning. Tech report, Berlin Institute of Technology (2009)

International Journal of Current Science Research and Review
ISSN: 2581-8341
Volume 08 Issue 07 July 2025
DOI: 10.47191/ijcsrr/V8-i7-22, Impact Factor: 8.048
IJCSRR @ 2025
www.ijcsrr.org

12. UDUPA, S. K. DEBRAY, S. K., AND MADOU, M. Deobfuscation: Reverse engineering obfuscated code. Reverse Engineering, Working Conference on (2005)
13. YASON, M. The Art of Unpacking, http://www.blackhat.com/presentations/bh-usa-07/yason/whitepaper/bh-usa-07-yason-wp.pdf
14. Murmurhash 2.0. http://sites.google.com/site/murmurhash
15. PEiD 0.95 Packer,Cryptor,Compiler detector http://www.peid.info , 2008
16. IDA Pro Interactive Disassembler https://www.hex-rays.com/products/ida/

Cite this Article: Patel, I. (2025). Exploring Prototype-Based Clustering for Malware Detection: Insights from MutantX-S. International Journal of Current Science Research and Review, 8(7), pp. 3337-3348. DOI: https://doi.org/10.47191/ijcsrr/V8-i7-22