Ransomware-as-a-Service and Its Evolution: Lessons from the Babuk, GandCrab, and Sodinokibi Families

Claude Fachkha*, Salwa Razaulla*, Christine Markarian*, Amjad Gawanmeh*, Benjamin C. M. Fung†, Chadi Assi‡
*College of Engineering and IT, University of Dubai, UAE
Email: {cfachkha, srazaulla, cmarkarian, agawanmeh}@ud.ac.ae
†School of Information Studies, McGill University, Canada
Email: ben.fung@mcgill.ca
‡Concordia Institute for Information Systems Engineering, Concordia University, Canada
Email: assi@ciise.concordia.ca

***Abstract*—Ransomware-as-a-Service (RaaS) has transformed cybercriminal operations by enabling threat actors to launch ransomware attacks using ready-made tools provided by developers. This study analyzes the evolution of several RaaS samples from Babuk, GandCrab, and Sodinokibi collected over several years. We introduce a novel learning model that uses contrastive techniques to understand the underlying patterns in the malware's binary code, enabling better capture of the subtle changes they undergo as ransomware evolve. Our method, which focuses on concept drift, demonstrates strong capability in identifying different variants, achieving an accuracy of almost 80% in distinguishing their forms from other ransomware types. These findings shed light on the difficulties RaaS malware presents for standard defenses and emphasize the need for sophisticated analytical tools in cybersecurity to mitigate their threats.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Ransomware has evolved rapidly in recent years. Attackers have shifted from broad, automated attacks to more focused, targeted campaigns against businesses. This shift has made ransomware more disruptive and profitable, attracting well-organized cybercriminal groups.

Ransomware-as-a-Service (RaaS) has emerged as a dominant force in the cybercrime landscape, dramatically lowering the barrier to entry for attackers and enabling widespread, scalable ransomware campaigns. Unlike traditional ransomware operated by a single group, RaaS separates development from deployment by offering customizable ransomware kits to affiliates in exchange for profit sharing. Notorious families like GandCrab, Sodinokibi (REvil), and Babuk have exemplified this model, with GandCrab alone responsible for over 50% of ransomware infections at its peak [1]. This paper explores the evolution of RaaS through these case studies, highlighting characteristics, detection, concept drift, classification, and tracing lineage, with particular focus on Babuk, GandCrab, and Sodinokibi, which are complex examples of RaaS malware. Our study makes the following main contributions:

This work is supported by the University of Dubai under the internal research fund.

1) **A contrastive learning approach for ransomware detection, including classification and trace lineage:** We present a self-supervised model that learns meaningful patterns from binary assembly code, enabling us to detect subtle structural changes in ransomware samples and better handle RaaS behavior.
2) **Dataset Availability and Support for Future Research:** To support future work in this domain, we provide the research community with access to our dataset of ransomware MD5 hash values[1].

The remainder of this paper is organized as follows. Section II provides the background information about concept drift and outlines the scope of the problem. In addition, it emphasizes the significance of this study. Section III elaborates on the working of the proposed approach. In Section IV, we discuss our findings and results, list the limitations, and highlight future directions. We discuss some related works in Section V. Finally, Section VI concludes our study.

II. Background and Problem Scope

In this section, we introduce concept drift and RaaS in the context of cybersecurity and malware activities [2]. Additionally, we outline the motivation for our approach.

A. Concept Drift

Concept drift occurs when the distribution of test data changes over time compared to the training data, causing machine learning models to lose accuracy [3]. This challenge is common in cybersecurity due to rapidly evolving threats. While supervised models are particularly vulnerable, even self-supervised systems can be affected [4]. Multi-class classification typically exhibits two types of drift [5]. Type 1 drift introduces samples from previously unseen families, while Type 2 drift involves known families with significantly changed behavior. Our study considers both drift types in the context of ransomware detection.

[1]https://github.com/malXhunter/ransomware

B. RaaS Malware

Ransomware-as-a-Service (RaaS) has revolutionized the ransomware ecosystem by providing a subscription-like model where developers offer ready-made ransomware tools to affiliates in exchange for a share of the ransom profits. This business model significantly lowers the technical barrier for launching ransomware attacks, enabling a wide range of threat actors to participate in sophisticated cyber extortion campaigns. RaaS platforms typically include customizable payloads, affiliate dashboards, and payment infrastructure, allowing rapid deployment and easy management of attacks. Recent research highlights that the rise of RaaS has led to a dramatic increase in ransomware incidents worldwide, complicating attribution and mitigation efforts. Curran et al. [6] provide a comprehensive analysis of RaaS operations and emphasize the challenges it poses to traditional cybersecurity defenses. As RaaS continues to evolve, understanding its operational and economic models is critical to developing effective countermeasures.

C. Concept Drift and RaaS Malware

Ransomware-as-a-Service (RaaS) malware is constantly evolving as developers update their code and affiliates customize attacks to avoid detection. This rapid change creates a challenge known as concept drift, where security tools trained on older ransomware samples become less effective over time. As RaaS families reuse code but frequently add new features or change behaviors, models and detection systems struggle to keep up with these shifts. This makes analyzing and investigating RaaS samples difficult because what worked to identify ransomware yesterday might not work today. Understanding concept drift is essential to improving adaptive detection methods and staying ahead of evolving ransomware threats.

D. Problem Scope

In cybersecurity, new threats emerge faster than researchers can prepare labeled datasets. Since labeling requires expert knowledge and is both time-consuming and expensive, it is not always practical at scale. To overcome this challenge, we adopt a contrastive learning-based autoencoder approach, drawing inspiration from models like BERT [7]. Instead of relying on labeled data, our method learns by comparing different augmented views of the same binary sample as positive pairs while treating unrelated samples as negative pairs [8].

This approach helps the model recognize when a new sample behaves differently from what it has seen during training—flagging it as a potential drift. Over time, as more of these drifting samples are collected, they can be used to refine and retrain the model, gradually improving its ability to detect evolving threats. The next section outlines the details of our framework.

III. Approach

We propose the framework shown in Figure 1, to illustrate both intra- and inter-classification of Windows ransomware

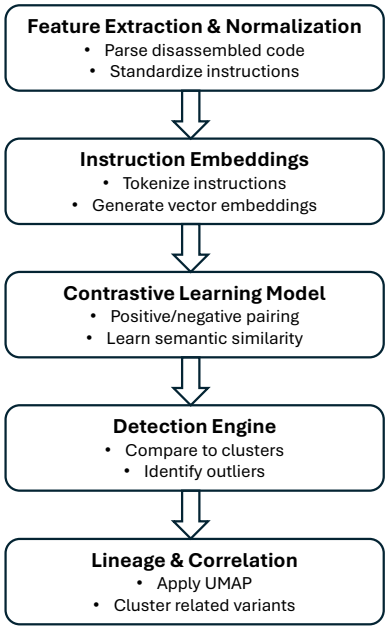


Fig. 1. Proposed workflow of our approach

[9] and to address the problem of concept drift. This can be addressed through two main steps: First, we accurately detect drifting binaries using an effective distance function to measure their deviation from training samples. Second, once the drifting samples have been identified, we must find meaningful reasoning as to why these samples deviate from the rest. We followed a five-stage approach starting with the extraction and normalization of assembly instructions and using an effective distance function to measure the similarity/dissimilarity between the samples. We then proceeded to learn a good representation of the meaning behind these instructions using contrastive learning. Our methodology also includes two main components: the detection module that detects drifting samples that do not belong to any training sample clusters. Finally, we provide a graph that traces the lineage of ransomware families and variants to show the evolution of these samples.

In the following sections, we discuss our proposed design in greater detail.

A. Feature Extraction & Instruction Normalization

The process begins with collecting training samples of ransomware binaries. These binaries are run through a disassembler, which converts them into readable assembly instructions. Because raw assembly code can be noisy and inconsistent, we apply instruction normalization to clean and standardize the instructions. This step ensures consistency across different samples, making them easier to analyze. Using sequences of assembly code as features in our machine learning model is similar to how natural language processing (NLP) works. We treat each instruction like a word in a sentence, allowing us to apply NLP techniques to understand the meaning and patterns in these instruction sequences. We used Ghidra [10]

to extract assembly instructions from the binaries. We selected this tool because it is open-source and provides powerful features including plugins and a robust decompiler [11].

For instruction normalization, we followed the method described in [12], where string literals are kept as they are, rather than being replaced with a generic token like in [13]. Keeping these strings intact helps distinguish different parts of the assembly code more clearly. Similarly, we chose not to replace all immediate operands with a single placeholder, since these values can represent many things—like strings, memory addresses, and more. However, retaining these raw values can cause issues, such as the out-of-vocabulary (OOV) problem mentioned in [14], where the model encounters instructions it has not seen before.

B. Instruction Embeddings

Once the assembly code is ready, it's turned into embeddings—structured numerical representations that machine learning models can understand. These embeddings are generated at multiple levels: token-level (individual instructions), sentence-level (instruction blocks), and positional (order of instructions). We create two different views of the same sample (e.g., by applying dropout) to prepare for contrastive learning. To achieve our goal, we leverage BERT, short for "Bidirectional Encoder Representations from Transformers," which is a multi-layer transformer encoder that draws inspiration from the implementation detailed in [15]. By employing a multi-head self-attention mechanism, the transformer model learns vector representations for input words and sentences [7]. The overall process involves two key phases: pretraining and fine-tuning. During the pretraining phase, the model is trained on an unlabeled dataset, focusing on two unsupervised tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). In the subsequent fine-tuning phase, the pre-trained model parameters were further adjusted using the labeled data from downstream tasks. The BERT model provides notable advantages over alternative models such as Word2Vec [16] and Glove [17]. Unlike Word2Vec and Glove, which generate context-independent word embeddings, regardless of where the words occur in a sentence and regardless of the different meanings they may have, BERT takes the word order into account.

Algorithm 1: Generating Instruction Embedding Pairs

Input: Assembly instructions for N samples: $s_1, s_2, \ldots, s_N$
Output: Embedding pairs (e_i^1, e_i^2) for each sample s_i

Set dropout rates: $d_1 = 0.1$, $d_2 = 0.5$
foreach s_i *in input samples* **do**
  tokens $\leftarrow$ **Tokenizer**(s_i)
  $e_i^1 \leftarrow$ **Model**(tokens, d_1)
  $e_i^2 \leftarrow$ **Model**(tokens, d_2)
  Save pair (e_i^1, e_i^2)
return *All* (e_i^1, e_i^2) *pairs*

Algorithm 1 outlines the steps we follow to generate the embeddings of the assembly codes. This algorithm takes in a list of assembly instructions—one for each ransomware sample. For each sample, it 1) tokenizes the instructions, breaking them down into a form that the machine learning model can understand; 2) generates two different embeddings (representations) of the same sample using two different dropout settings (a technique that helps the model generalize better); and 3) saves both embeddings as a pair, which will later be used to train the model to recognize similar or changing patterns in malware behavior. In other words, this algorithm turns each ransomware sample into two slightly different vector representations so the model can learn to spot similarities and differences, even when the code changes.

C. Contrastive Learning Model

At this stage, we feed the embeddings into a contrastive learning model. It learns by comparing different versions of the same malware sample (positive pairs) and unrelated samples (negative pairs). The goal is to teach the model how to recognize variations in the same sample versus differences across distinct ones. This helps the system understand what counts as similar or unusual behavior. Algorithm 2 lists the steps involved in the aforementioned model.

Algorithm 2: Contrastive Learning for Malware Embedding

Input: Embedding pairs $\{(e_i^1, e_i^2)\}_{i=1}^N$ for N malware samples
Output: Learned representation space for distinguishing similar and dissimilar malware

for $i \leftarrow 1$ **to** N **do**
  Positive pair: (e_i^1, e_i^2)
  Negative pairs: $\{(e_i^1, e_j^1), (e_i^1, e_j^2) \mid j \neq i\}$
  Compute similarity score: $s_i^+ = \text{sim}(e_i^1, e_i^2)$
  foreach $j \neq i$ **do**
    Compute similarity scores: $s_{ij}^- = \text{sim}(e_i^1, e_j^1)$, $s_{ij'}^- = \text{sim}(e_i^1, e_j^2)$
  Compute contrastive loss:

$$\mathcal{L}_i = -\log \frac{\exp(s_i^+/\tau)}{\exp(s_i^+/\tau) + \sum_{j \neq i} \left[\exp(s_{ij}^-/\tau) + \exp(s_{ij'}^-/\tau)\right]}$$

return *Learned encoder parameters minimizing total loss:* $\sum_i \mathcal{L}_i$

D. Detection Module

Now we are ready to evaluate new (testing) ransomware samples. The model checks whether a given sample fits into any known cluster formed during training. If it fits, it is marked as a non-drifting sample—meaning it behaves similarly to what we have seen before. If not, it is flagged as a drifting sample, possibly representing a new or evolved variant of

ransomware. Algorithm 3 provides a step-by-step explanation to this process.

Algorithm 3: Detecting Drifting Ransomware Samples

Input: Training data with C known clusters
Testing data samples
Output: Each test sample is either:
- Assigned to the closest known cluster, or
- Flagged as a drifting (unfamiliar) sample

for *each cluster* $i = 1$ *to* C **do**
  Compute centroid c_i of cluster i
  Compute standard deviation σ_i of samples in cluster i
Project all testing samples into the same embedding space
for *each testing sample* x_j **do**
  for *each cluster* i **do**
    Compute distance $d_{ij} = \text{distance}(x_j, c_i)$
  Find the minimum distance $d^j_{min} = \min_i d_{ij}$ and its corresponding cluster index i^*
  if $d^j_{min} < \sigma_{i^*}$ **then**
    Assign x_j to cluster i^*
  else
    Mark x_j as a drifting (novel) sample

This algorithm is designed to spot unusual or drifting ransomware samples—those that don't resemble anything the system has seen during training. First, it finds the center and spread (standard deviation) of each known ransomware group. Then, it checks each new (test) sample by comparing it to these groups. If the new sample is close enough to a known group, it's assigned to that group. If it's too far from all known groups, it's flagged as unfamiliar—possibly a new variant or previously unseen behavior. This helps researchers track malware evolution and catch novel threats early.

E. Tracing Lineage

For the drifting samples, we try to trace their lineage—basically mapping out their connection to other known ransomware families or variants. This is done by comparing how closely their behaviors or code structures relate. The goal is to understand how the malware may have evolved and whether it shares ancestry with older threats. In order to achieve this task, we utilized the Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP), a widely used dimensionality reduction technique [18], to visualize the relationships between strings and API calls embeddings between different variants within a ransomware family. Algorithm 4 explains how UMAP work in the context of visualizing ransomware variant relationships. By converting complex high-dimensional data like API call patterns into a 2D space, UMAP helps us see how closely related different variants are. Samples with similar behavior appear near each other, making it easier to trace lineage, detect evolution, and identify outliers among ransomware families in general and RaaS in particular.

Algorithm 4: Visualizing Ransomware Variant Relationships with UMAP

Input: Set of ransomware samples $S = \{s_1, s_2, ..., s_n\}$,
Embeddings for each sample $E = \{e_1, e_2, ..., e_n\}$,
UMAP hyperparameters (neighbors k, min_dist, metric)
Output: 2D projection $P = \{p_1, p_2, ..., p_n\}$ of embeddings for visualization
Normalize all embeddings in E
Construct a high-dimensional graph:
  for *each sample* e_i **do**
    Identify k nearest neighbors
    Compute edge weights using distance metric
Convert the neighbor graph into a fuzzy topological representation.
Optimize a low-dimensional layout P using stochastic gradient descent:
  for *each epoch* **do**
    Update 2D positions to minimize divergence from high-dimensional graph
return Projected coordinates P

IV. Evaluation Results

In this section, we assess the influence of contrastive representation learning on binary clustering and evaluate the effectiveness of our detection module in identifying drifting ransomware samples.

A. Dataset

Our dataset consists of ransomware samples collected from reputable malware repositories, including VirusTotal, VirusShare, and Malware Bazaar. We gathered a substantial collection of 107,000 malware samples within a 5-year period. First, to focus specifically on ransomware, we utilized VirusTotal's analysis report of these samples and then employed the widely used AVClass tool to label and categorize the samples. AVClass employs the majority rule method, which considers the labels assigned by numerous antivirus engines [19]. By considering the reported labels from these engines, AVClass determines the most probable family name associated with each sample. Thus, we identified approximately 10,000 ransomware samples, with 4,456 unique ones. In this study, we focus only on RaaS malware, namely, Babuk, GandCrab, and Sodinokibi. GandCrab was found to be one of the largest families in our dataset, using a set of 1,485 unique samples. Based on our manual analysis, these three ransomware were the only ransomware among the ones we examined that showed clear RaaS behavior. An overview of the collected ransomware samples is shown in Table I.

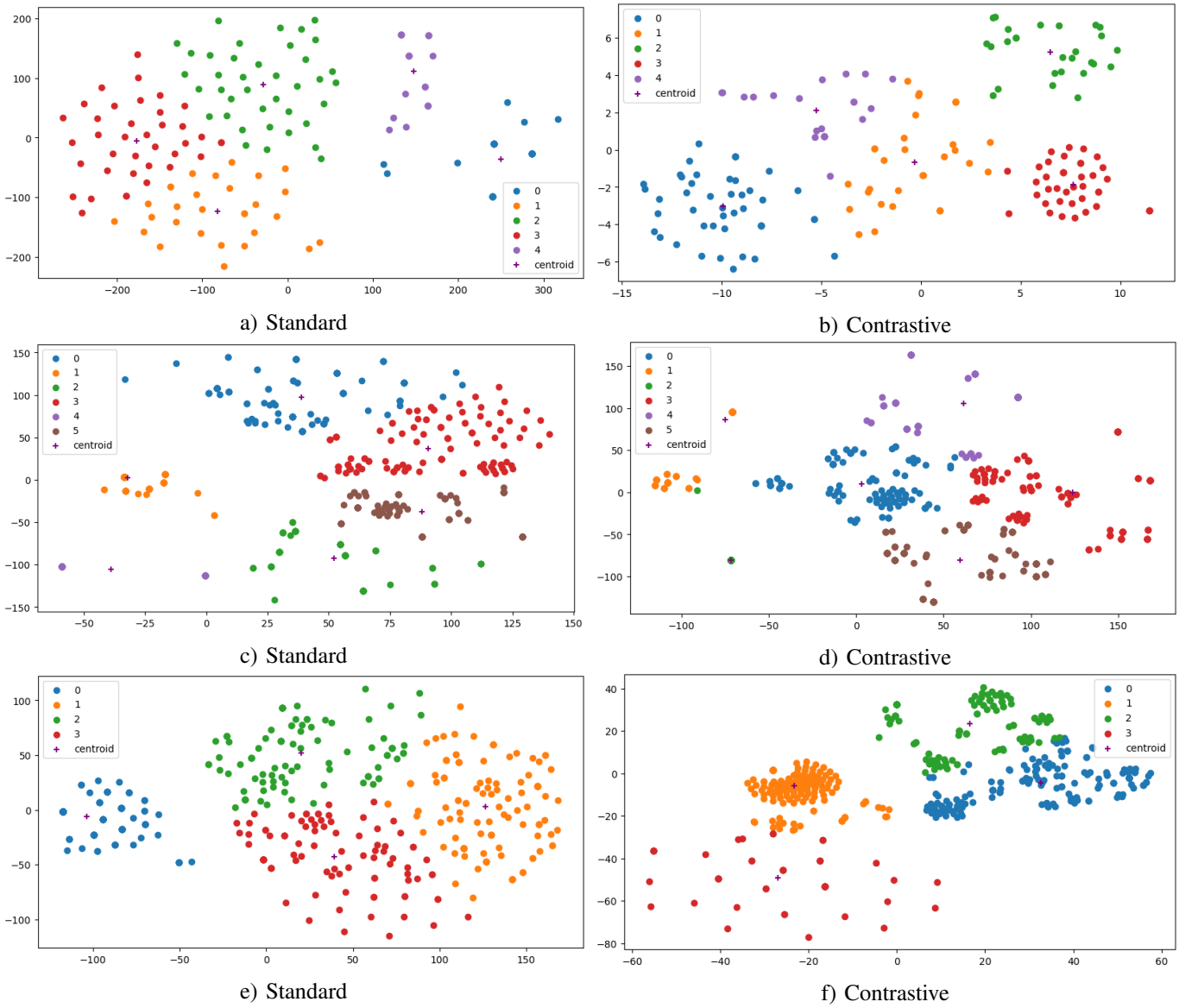


Fig. 2. t-SNE plots comparing Standard and Contrastive Representations of four ransomware families: Babuk (a-b), GandCrab (c-d), and Sodinokibi (e-f) (intra-class)

TABLE I
RANSOMWARE FAMILY SAMPLE DISTRIBUTION

Family	No. of samples
Babuk	167
Sodinikibi	244
Cerbu	531
Lamer	968
Virlock	1061
Gandcrab	1485
Total	**4456**

To make sure our dataset was clean and reliable, we first removed any corrupted files. We also filtered out packed samples since we needed to unpack them to study them properly. Most were unpacked using the UPX [20] tool, but some couldn't be unpacked and were removed. Then, we used the Ghidra disassembler to break down the remaining malware files. However, some files couldn't be processed by Ghidra, so we left those out too. In the end, we were left with 1,896 valid and usable RaaS samples.

Subsequently, to assess the performance of the proposed detection module, we employed a comprehensive evaluation approach. The primary objective of this evaluation was to accurately identify drifting samples from the unseen family during the testing phase. By systematically testing the ability of the detection module to recognize previously unseen ransomware families, we can effectively assess its performance and measure its effectiveness in detecting emerging ransomware variants.

B. Contrastive Illustration

In this section, we assess how contrastive learning improves the clustering of RaaS ransomware samples. Using t-SNE for visualization, Figure 2 compares embeddings from a standard model and a contrastive learning model. To find the optimal number of clusters, we used the Elbow method [21], which helps determine the best fit by identifying where adding more clusters no longer significantly improves separation. The results show that contrastive learning creates more compact and well-separated clusters, making it easier to spot drifting samples. In contrast, the standard model produces more loosely grouped clusters.

C. Detection of Drifting Samples

In this section, we examine how well our detection module performs by testing it across six different runs—each time leaving out one ransomware family as unseen data. We trained the model on five families using an 80:20 train-test split and evaluated it on both known and unseen samples. The unseen family was treated as drifting (negative), while the rest were considered non-drifting (positive).

Family	Accuracy (%)	Precision (%)
Babuk	65.2	31.9
Cerbu	87.5	73.4
GandCrab	78.5	83.5
Lamer	88.8	87.6
Sodinokibi	60.4	35.0
Virlock	88.1	84.1

TABLE II
DETECTION PERFORMANCE METRICS FOR RANSOMWARE FAMILIES

To measure performance, we used accuracy, precision, and recall. Accuracy reflects the overall correct predictions, precision measures how many of the flagged drifting samples were truly drifting, and recall shows how well the model caught all actual drifting samples.

As shown in Table II, the model achieved poor performance for Babuk and Sodinokibi but better accuracy for GandCrab with 78.5% ± 2.3% (95% Confidence Interval). The limited number of samples we could obtain for Babuk (167) and Sodinokibi (244) clearly impacted our detection results, leading to disappointing accuracy rates of 65.2% and 60.4% respectively. This challenge highlights a frustrating reality in cybersecurity research: newer and more sophisticated RaaS families are inherently harder to study because fewer samples are available in public repositories, and these families often employ advanced evasion techniques that make sample collection more difficult. Fortunately, our experience with GandCrab tells a more encouraging story. In fact, with 1,485 samples available, we achieved a much more promising 78.5% accuracy, demonstrating that our approach can deliver strong results when given sufficient training data.

D. Explaining drifting samples

Despite clustering the families and variants by utilizing the assembly codes of the binaries, we have still not been able to know what makes these ransomware families distinct, what makes them similar, what links them together, and how these variants have evolved over time. To address these questions, we dig deeper by extracting the strings and API calls of these binaries in this section. All analyzed ransomware families follow the basic operational pattern established 30 years ago by the AIDS virus [22], the first known ransomware. The operational methods of the ransomware families we analyzed remain relatively unchanged. As discussed in Section III, we leverage the UMAP, a widely used dimensionality reduction technique [18], to visualize the relationships between strings and API calls embeddings between different variants within a ransomware family as shown in Figure 3.

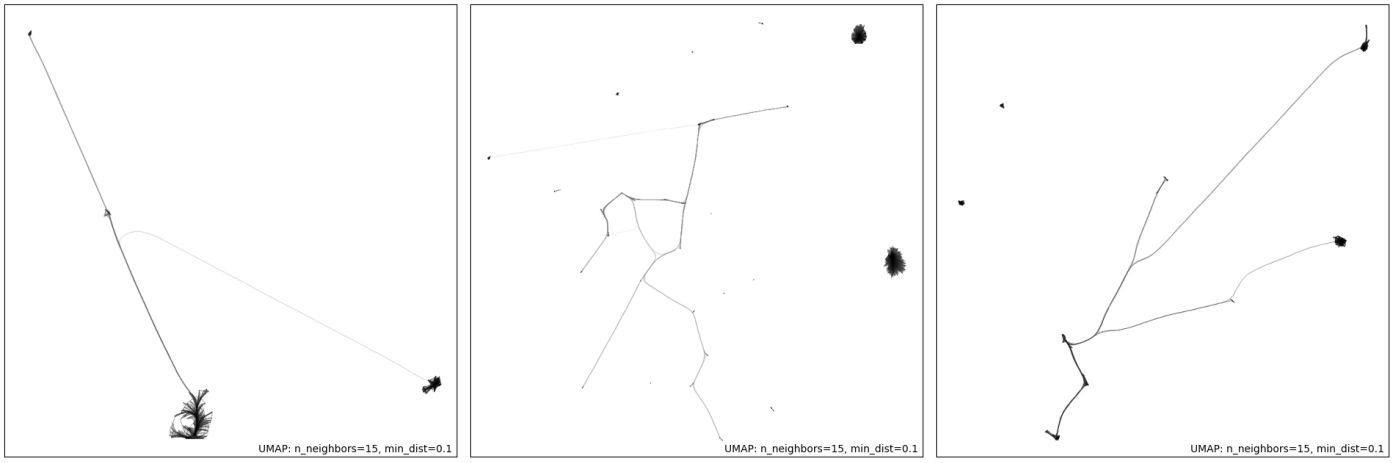
Among the different RaaS samples, we noticed specific patterns in the sequences of their API calls. Many basic and crucial API calls are commonly used across various variants. These include calls such as *NtOpenFile*, which opens files or directories and allows the program to work with their data; *NTQueryInformationFile*, which provides information about a specified file; *NtProtectVirtualMemory*, which changes the memory protection in the program's virtual address space; and *NtTerminateProcess*, which unconditionally terminates and modifies a process. Furthermore, less frequently used API calls were also present, including *FindResourceA, StartService, RegCreateKeyExW, CreateToolhelp32Snapshot* and *IsDebuggerPresent*. We found that some samples in cluster #4 used the Process Environment Block (PEB) to determine debugger presence. These samples aim to evade detection by employing *IsDebuggerPresent* to spot debuggers and *ExitProcess* to avoid analysis [23]. The *IsDebuggerPresent* API call checks whether the program is being debugged by a user-mode debugger. Another technique seen in cluster #2, involved using *CreateToolhelp32Snapshot* to identify known debugger names within the parent process [24]. These API calls are employed to examine system artifacts and to identify whether a debugger is involved.

We noticed an interesting observation while analyzing the Gandcrab samples. While the extracted strings of these samples did not contain URLs associated with mainstream platforms such as Microsoft or Symantec, the majority of the Gandcrab samples displayed a distinct inclination toward invoking services designed for anonymous communication. Two such platforms, *www.torproject.org* and *tox.chat*, have been prominently featured. Additionally, a recurrent link that surfaced across numerous Gandcrab samples was *http://gdcbghvjyqy7jclk.onion*. A lookup of this domain on *whois*[2] revealed its non-existence.

E. Discussions

Deployment in a Real-Life Environment: Our approach leverage both static and dynamic malware analysis. Although dynamic malware analysis uses sandbox technologies that simulate real-life environments, deploying our detection system in the real world brings up several challenges. The system needs to handle constant new samples while keeping false alarms

[2]WHOIS: https://www.whois.com/



a) Babuk c) Gandcrab e) Sodinokibi

Fig. 3. UMAP graph representing the connection between different variants of each RaaS family

low and running efficiently. There's also the ongoing cat-and-mouse game where attackers will try to figure out how the detection works and then create new variants to beat it. We acknowledge that integrating this with existing security tools would require careful optimization and translation of technical results into actionable information for security teams.

F. Limitations and Future Work

Our study has several limitations. First, the dataset we used consisted of only 4,456 unique ransomware samples, which may not be large enough to draw highly generalizable conclusions. Although we accessed over 100,000 malware samples, only 10,000 were ransomware, and many were packed or encrypted, complicating analysis. As a result, our findings may be affected by the dataset's limited size and diversity.

Another limitation is that our approach assumes the training data is correctly labeled. In practice, mislabeled data is common and can negatively impact model performance. Our current study does not account for this issue. Additionally, we relied on a standard deviation-based method to assign testing samples to clusters. While simple, this approach can be sensitive to outliers and requires manual tuning, which may impact clustering accuracy.

For future work, we aim to expand our dataset by sourcing more ransomware samples across various families. We also plan to analyze the behavior of common packers to improve unpacking and sample quality. Lastly, we intend to explore alternative clustering methods, such as Gaussian Mixture Models (GMM), to improve detection accuracy and robustness.

V. Related Work

Ransomware-as-a-Service is rapidly evolving, changing the ransomware scene through decentralized, affiliate-driven operations. While earlier research mainly explored the economic models and lifecycle of RaaS, recent studies have offered deeper insights into its organizational setup and technical complexity. In this section, we review related work and touch on RaaS analysis and ransomware detection techniques.

Ruellan et al. [25] studied leaked communications from the Conti RaaS group and uncovered a well-structured, corporate-like organization with clearly defined roles, from developers to negotiators. This maturity makes it harder for law enforcement to attribute attacks or dismantle these groups. Detecting RaaS variants remains challenging due to their constant evolution. Mishra and Stamp [26] proposed a clustering-based method to spot concept drift in malware data, helping track changes over time. Similarly, Ispahany et al. [27] developed an adaptive detection system that uses real-time data and incremental learning to update ransomware classifiers as behaviors change. Willie [28] highlighted the growing role of AI in RaaS, showing how automation and generative tools lower entry barriers and boost ransomware's effectiveness. These recent insights build on earlier work like Kharraz et al.'s UNVEIL system [29], which detects ransomware based on runtime behaviors—a method still useful against today's RaaS threats. On the economic and operational side, Anderson et al. [30] analyzed RaaS platforms' profit-sharing and affiliate roles, explaining how this division complicates tracking and law enforcement. Harrop et al. [31] conducted a large-scale study tracing code reuse and infrastructure overlaps across GandCrab, Sodinokibi, and Babuk, emphasizing the value of malware lineage tracking. Additionally, Dib et al. [14] and Yang et al. [32] focused on malware evolution within and between classes. Unlike them, our work covers both intra-class and inter-class evolution, offering a more complete view. Building on our previous work [33], we leverage three diverse feature sets—assembly instructions, strings, and API calls—to enhance ransomware family attribution, thereby increasing the reliability and robustness of our approach. Finally, while much research focuses on individual RaaS families, our study takes a broader approach. By analyzing Babuk, GandCrab, and Sodinokibi together, we explore both their static and dynamic characteristics. This lets us identify shared code patterns,

affiliate-driven customizations, and evidence of concept drift. Our comparative analysis provides fresh insights into how RaaS continues to evolve and highlights the need for detection methods that adapt over time.

VI. Conclusion

In this study, we examine the evolution of three RaaS malware families using a contrastive learning-based autoencoder to detect samples that deviate from the original training distribution. By visualizing the learned representations with t-SNE, we show that contrastive learning captures more meaningful patterns compared to a standard model. Our detection module effectively identifies drifting samples, offering insights into the evolving nature of these RaaS malware. To trace variant relationships, we analyzed assembly-level instruction similarities and incorporated additional features such as strings and API calls. The results reveal strong evidence of code reuse across its variants, highlighting patterns of evolution and adaptation over time.

References

[1] Europol, "Gandcrab ransomware service shut down through global operation," https://www.europol.europa.eu/newsroom/news/gandcrab-ransomware-service-shut-down-through-global-operation, 2019, accessed: 2025-06-15.

[2] E. Bou-Harb, C. Fachkha, M. Debbabi, and C. Assi, "Inferring internet-scale infections by correlating malware and probing activities," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 640–646.

[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[4] C. H. Tan, V. Lee, and M. Salehi, "Online semi-supervised concept drift detection with density estimation," *arXiv preprint arXiv:1909.11251*, 2019.

[5] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti, "The tangled genealogy of iot malware," in *Annual Computer Security Applications Conference*, 2020, pp. 1–16.

[6] K. Curran, R. Anderson, and T. Moore, "The rise of ransomware-as-a-service: A comprehensive survey," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1234–1250, 2021.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[8] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[9] N. M. Chayal, A. Saxena, and R. Khan, "A review on spreading and forensics analysis of windows-based ransomware," *Annals of Data Science*, vol. 11, no. 5, pp. 1503–1524, 2024.

[10] "Ghidra," https://github.com/NationalSecurityAgency/ghidra, March 2019, national Security Agency of the United States.

[11] R. Rohleder, "Hands-on ghidra-a tutorial about the software reverse engineering framework," in *Proceedings of the 3rd ACM Workshop on Software Protection*, 2019, pp. 77–78.

[12] Y. Duan, X. Zhang, X. Tang, and Z. Lin, "Deep semantic clustering for unsupervised malware embedding," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2020.

[13] Microsoft, "Project innereye – medical imaging ai to empower clinicians," 2018.

[14] M. Dib, S. Torabi, E. Bou-Harb, N. Bouguila, and C. Assi, "Evoliot: A self-supervised contrastive learning framework for detecting and characterizing evolving iot malware variants," in *Proceedings of the 2022 ACM on Asia conference on computer and communications security*, 2022, pp. 452–466.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[16] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 3111–3119.

[17] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[18] L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," *Journal of Open Source Software*, vol. 3, 2018.

[19] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings 19*. Springer, 2016, pp. 230–253.

[20] M. F. Oberhumer, "Upx the ultimate packer for executables," *http://upx.sourceforge.net/*, 2004.

[21] R. Thorndike, "Who belongs in the family?" *Psychometrika*, vol. 18, pp. 267–276, 1953. [Online]. Available: https://doi.org/10.1007/BF02289263

[22] S. Razaulla, C. Fachkha, C. Markarian, A. Gawanmeh, W. Mansoor, B. C. M. Fung, and C. Assi, "The age of ransomware: A survey on the evolution, taxonomy, and research directions," *IEEE Access*, vol. 11, pp. 40 698–40 723, 2023.

[23] K. Yoshizaki and T. Yamauchi, "Malware detection method focusing on anti-debugging functions," in *2014 Second International Symposium on Computing and Networking*, 2014.

[24] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, nov 2019. [Online]. Available: https://doi.org/10.1145/3365001

[25] C. Ruellan, V. Bourgeais, and J. Lemoine, "Inside conti: Unpacking the leaked raas organizational model," *Journal of Cybersecurity Research*, vol. 11, no. 3, pp. 101–117, 2023.

[26] R. Mishra and M. Stamp, "Detecting concept drift in ransomware samples using cluster-based modeling," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 55–67, 2025.

[27] R. Ispahany, M. Haq, and B. De Souza, "Adaptive ransomware detection using real-time drift-aware learning," in *Proceedings of the 32nd ACM Conference on Computer and Communications Security (CCS)*, 2025, pp. 1344–1356.

[28] S. Willie, "Ai meets raas: How generative tools are powering the next generation of ransomware," *Cyber Threat Intelligence Quarterly*, vol. 7, no. 1, pp. 22–34, 2025.

[29] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Unveil: A large-scale, automated approach to detecting ransomware," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 757–772.

[30] R. Anderson, C. Barton, R. Bohme, R. Clayton, and M. van Eeten, "The economics of ransomware-as-a-service," *Communications of the ACM*, vol. 65, no. 7, pp. 82–90, 2022.

[31] E. Harrop, M. Kaur, and Z. Li, "Tracking code reuse and infrastructure overlap in ransomware-as-a-service," in *Proceedings of the 2023 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 112–126.

[32] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "{CADE}: Detecting and explaining concept drift samples for security applications," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2327–2344.

[33] S. Razaulla, C. Fachkha, A. Gawanmeh, C. Markarian, B. C. M. Fung, and C. Assi, "Tracing the ransomware bloodline: Investigation and detection of drifting virlock variants," in *2024 6th International Conference on Computer Communication and the Internet (ICCCI)*, 2024, pp. 1–6.