

Automation of Binary Analysis: From Open Source Collection to Threat Intelligence

Frederic Grelot¹, Sebastien Larinier² and Marie Salmon¹

¹GLIMPS, Rennes, FRANCE

²ESIEA, FRANCE

Abstract

Many open sources of binaries, including malware, have emerged in the landscape in recent years. Their quality compares very favourably with commercial sources, as emphasised by Thibaud Binetruy (Twitter influencer under a pseudonym, Société Générale CERT, 2020): “Integrating operational threat intel in your defense mechanisms doesn’t mean buying Threat Intel. You can start by using the [mass] of open source indicators available for free.” Some are provided by official sources (Abuse.ch, with data supplied by the Swiss national CERT, among others), while others are made available in more obscure ways, sometimes anonymously (VirusShare, VX-Underground, etc.). Our examination of these sources underlines the wide disparity in quality and quantity between them. We have had to take this diversity into account in our research, designing a dedicated platform that enables us to supply information to our binary analysis products and to conduct daily analyses of correlations between and within malware families on a large scale. This work can then be applied to concrete cases such as Babuk, Ryuk and Conti. We have been able to highlight links for these families by immediately identifying correlations, with additional manual analysis then confirming the genealogy of the samples precisely.

Keywords

Cybersecurity, Machine Learning, Threat Intelligence, CTI, Malware, Detection, Classification.


1. Introduction


When an attack takes place, there is an urgent need to find out who the attacker is, and how they operate. But the heat of the moment is not the time to examine the details. This is when you need the services of a threat intelligence analyst, whose job it is to answer these questions. Analysts work over long time scales, and their constant monitoring gives them an up-to-date understanding of how families of attackers are evolving. But they themselves need to be well-equipped in order to pick up the weak signals providing news about these groups. How can we use public data (files, virus databases, etc.) to derive a better understanding of attackers’ TTPs (tactics, techniques and procedures)? How can a threat intelligence analyst’s tools provide useful information about how to respond to an incident?

To our knowledge, the academic sector has not so far covered the subject of the large-scale automation of end-to-end malware analysis. But the problems we have encountered in our work have led us to the conclusion that the whole chain (from collection to analysis) must be taken into account if we want to achieve a high-quality end result. Here we report on the

C&ESAR 2021: Automation in Cybersecurity, 16 - 17 Novembre 2021, Rennes, FRANCE

 frederic.grelot@glimps.re (F. Grelot)

 © 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

infrastructure and techniques we have put in place to respond to this problem – how can we fuel cyber threat intelligence with precise knowledge about malware families and their relationships, reliably, quickly and fully automatically? We will use a correlation method based on concept code, as this is well suited to correlating large numbers of binaries. This is not an exclusive property of this method, though – other methods can also deliver similar results [1, 2] (SSDEEP, percentage of basic code blocks in common, etc.).

2. Data sources

This section provides a detailed inventory of the open sources of binaries we have identified. The list is not intended to be exhaustive, and focuses primarily on data accessible publicly with no subscription or registration, combined with a validation or peering process (except VirusTotal, which is listed because it is a standard reference).

Table 1

Summary table of used data sources.

Source name	Goodware	Malware	Source code	History	Number of files	Accessibility ¹	Associated tags	Debugging symbols
Linux repository	X		X	+10 years	+10M	++		X
Opensource repository (Github, ...)	X		X ²	+10 years	+10M	+		X
Proprietary binaries	X			1-3 years	+10M	0		
Conan.io	X		X	1-3 years	100k-1M	+		X
Chocolatey	X			1-3 years	100k-1M	+		
Malware-Bazar		X		2 years	10k-100k	++	+ ³	
VirusShare		X		9 years	+10M	-	- ⁴	
VirusTotal	X	X		+10 years	+10G	-- ⁵		
VX-Underground		X	(leaks)	2-3 years	1k-10k	+	++ ⁶	

¹ Accessibility levels from very easy (++) to very difficult (--). 0 represents a neutral level.

² Compilation often requires manual work.

³ Manual tags allocated by the researchers supplying the data.

⁴ Tag quality fairly poor, often containing only the numbers of antivirus detections and names.

⁵ Paid-for access.

⁶ Tags assigned directly by the VX-Underground team.

3. Collection and processing

3.1. Constraints

Our research has led us to develop capacity for the mass processing of open sources, from automatic collection to generating the daily Docker images that constitute our products. In this section, we will present our methods and the collection infrastructure, which are built around three major constraints:

- respect for sources: it is important to prioritise local cache capacity as much as possible, avoid sending multiple requests to servers and respect a reasonable rate of requests. Failing to do this leads to multiple risks, including being blacklisted from a server, but above all it has a negative impact on the ecosystem and endangers the model of open-source data sharing that contributes so effectively to the cyber ecosystem.
- control data volumes: our goal is to collect and store relevant data without exceeding our storage capacity. This makes it important to take account of future needs and availability of sources: what is it essential to conserve? What needs to be duplicated to guard against data loss? What can we allow ourselves to collect again in the future if necessary?
- manage computing time and deadlines: the goal is to be able to ingest data continuously. But we still need to have the possibility (availability of data) and the capacity (volume/computing time) to submit all the files for reanalysis if we update the computation method we use. Otherwise, we run two risks: having to reduce the update frequency, or having to deliberately exclude data that is too old, which could ultimately impair our detection and analysis capacity.

3.2. Process implemented

To collect malware, we use a pipeline (see Figure 1) consisting of three major stages: ingestion, tagging and conversion into concept code. These stages take place in the form of a CI flow on a GitLab CI orchestrator. The orchestrator starts jobs regularly and automates the whole process.

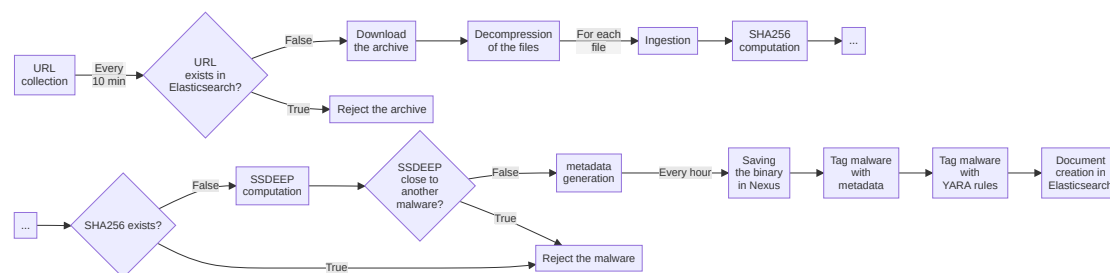


Figure 1: Binary collection and processing steps.

3.2.1. Ingestion

The first phase, which manages ingestion, is itself divided into the following steps: download, extraction, filtering and finally storage of the data.

Download During this first step, we use web collection (Beautiful Soup¹) to find download links in the pages of various data sources. We implement a number of mechanisms to avoid overloading the source servers: minimal parallelisation, resuming interrupted downloads from the point where they stopped, using mirrors (or torrent links) whenever possible, etc. 99% of the time, the links point to archives – we maintain a cache of downloaded URLs to avoid ever having to download the same data twice. These measures enable us to limit our own load, but also and above all the load on the source services.

Extraction The extraction phase is specific to each data source. It allows us to extract additional information depending on the source. For example, for files downloaded from VX-Underground, we can retrieve information on the malware type when it is included in the file name. The way the links are organised on the collection page also allows us to extract the attacker family (APTxx, etc.), which we can use in the tagging phase. For VirusTotal (if we have a subscription): the archives contain the malware and a JSON file summarising the analysis, which can be used to deduce metadata for the family.

Filtering For data filtering, we rely on two types of hash: SHA256 and SSDEEP. The first is a very quick way of making sure we are not downloading an archive or malware that has already been ingested. This obviously avoids duplicates, particularly among different data sources.

SSDEEP, meanwhile, allows us to compute the level of similarity with other malware in the database: as our correlation algorithms are fairly flexible, we prefer to avoid downloading malware that is too similar to what we already have, whereas other chains may need maximum numbers of samples to be collected. The SSDEEP similarity computation is relatively complex, especially as the database grows larger. We thus carry out a preliminary partial search in an Elasticsearch database using an N-gram tokeniser of size 7. This tokeniser selects a very limited subset of the full database, on which a full comparison computation is carried out. With the results of this query, if the score exceeds a set threshold, the malware is rejected on the basis that it is too similar to a strain that has already been ingested.

Storage The storage infrastructure needs to allow us to store data of varying types and to retrieve it on demand. We also need to be able to upgrade it easily as the project progresses. We chose to store metadata in an Elasticsearch cluster and data on a Nexus server (using only a tiny proportion of its features).

¹[https://en.wikipedia.org/wiki/Beautiful_Soup_\(HTML_parser\)](https://en.wikipedia.org/wiki/Beautiful_Soup_(HTML_parser))

3.2.2. Tagging

We used two types of taggers for the tagging phase: metadata and YARA rules. The first type uses the source metadata (typically, MalwareBazaar tags or VX-Underground families, for example) while the second uses a series of YARA rules to identify specific families. Each piece of malware stored passes through both taggers to complete the malware information in Elasticsearch.

3.2.3. Conversion into concept code

Once the binaries have been collected and identified, the final transformation is carried out to convert them into what we call concept code. This process is only relevant for the malware correlation method based on concept code, and is not necessary if other methods are to be used instead. It could be replaced by a disassembly process, for example, which extracts the basic blocks from each binary to enable a similarity calculation based on the basic block hashes. Another option is to extract character strings in order to detect binaries sharing the same strings and thus produce a correlation score based on this metric. We will not examine these methods in detail here, because they are highly dependent on capacity within the organisation wishing to implement a binary analysis automation process (in terms of both computation time and access to the different technologies required).

3.3. Automation

To automate this sequence, we use the following infrastructure:

- A GitLab server for orchestration, triggering, managing and supervising jobs.
- Elasticsearch to store the metadata on the collected files.
- Nexus to store all the binaries.

The collection sequence is asynchronous, making it easier to resume operation if it needs to be interrupted and restarted later. Link searches for each source are carried out every 10 minutes, and tags are updated hourly. So far we have 8,262,249 items of processed malware, of which 4,430,562 are non-rejected and 3,282,091 are tagged. This represents around 3 TB of malware, taking one to two months to ingest. Updates currently take about an hour for a 5 GB archive.

3.4. Problems and areas for improvement

3.4.1. Archive size and format

The archives can vary widely in size, from a few megabytes to several hundred gigabytes – the largest archive we have had to ingest was 800 GB. At this size, it becomes difficult to fully decompress the archive, which would require around twice the disk space available. This makes it essential to plan the process from the beginning for “flow”-based operation, partially decompressing and analysing the archive as we progress. Our collection process has provided us with archives in a variety of formats: zip, rar, 7z and tar. Partial decompression works very

well for the zip, tar and rar formats. For 7z, partial decompression still works, but very slowly. We have thus chosen to decompress these archives in full before analysis, as we have not come across 7z archives that were too large for this mode of operation.

3.4.2. Tagging

With the two taggers we use, we only manage to tag a fairly small proportion of the malware (a few percent), mostly using metadata and in a few cases using YARA rules. Additional work (which we have not yet carried out) will be needed to find better YARA rules, and probably more of them. This proves the significant limitations of this detection method, which ultimately is based on searching for signatures. For our purposes, this is not a major stumbling block (our algorithms are very tolerant of changes, so only a small number of samples are needed for effective detection), but it is important to be aware of it if large-scale use is to be made that requires a higher percentage of correctly identified malware.

3.4.3. Planned enhancements

For tagging, we are planning to use the correlation developed at GLIMPS (based on concept code) to compare malware with specific families. This is possible as long as the pipeline is completely reliable and fairly stable, because the volumes involved are not compatible with the use of this sequence for every update. This will thus be done later and will resolve the tagging problem.

4. Application of the results: generating cross-correlation matrices

Once the data has been collected from the public sources and processed in our infrastructure, we can compute the links between pairs of malware threats, family by family and between families. Computations of similarity between two binaries are carried out using the GLIMPS correlation technology (see Figure 2), based on code conceptualisation. The benefit of code conceptualisation lies in its robust handling of modifications, including changes in compilation chains or compiler versions. However, as described in section 3.2.3, other methods exist for computing similarity, and for this reason we will focus more closely on the use of the results obtained.

In addition, the computation is very fast – a matrix of 1,000 binaries (a million comparisons) can be produced in less than a minute, and the computation time is linear relative to size rather than quadratic, which most other methods are, enabling computation on families of several tens of thousands of binaries. Finally, this method is logarithmic when adding a binary to a collection (adding a row/column depends logarithmically on the size of the matrix, which is very beneficial in terms of computation time for large families). By carrying out the computation on a large scale, we can automatically generate correlation matrices for each family and study the evolution of the samples by reordering these matrices by value (i.e. performing a clustering).

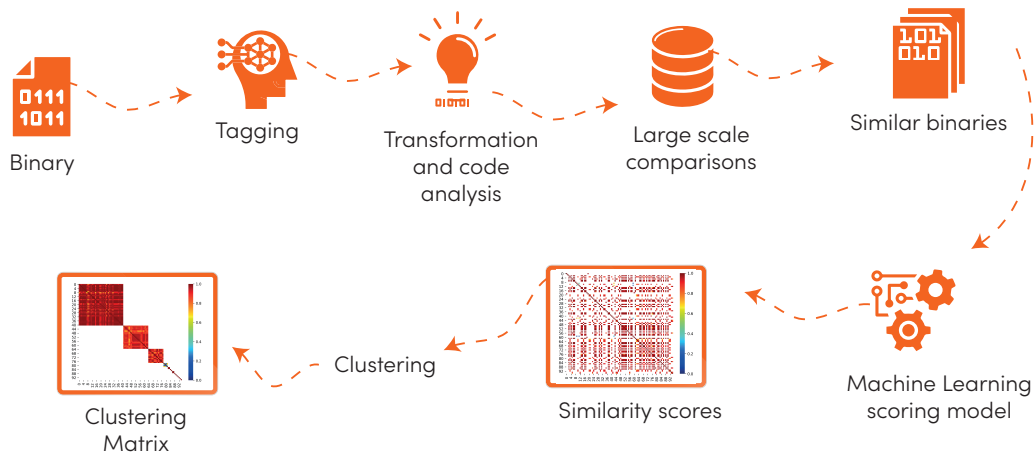


Figure 2: Processing workflow of the GLIMPS correlation technology.

4.1. Correlation matrix and clustering

To highlight the links for and between malware families, we automatically define family clusters that group similar binaries together (see Algorithms 1 and 3). These are constructed from a correlation matrix generated by analysing pairs of binaries to calculate their similarity scores.

The rows and columns of the matrix represent the binaries, each identified by the first five characters of its hash. In Figures 3.b and 4, each identifier is also prefixed by the cluster to which it belongs. The value scale ranges from 0 (blue) to 1 (red), and represents the similarity score between each pair of binaries, where 1 is the maximum value, representing perfect similarity between two binaries (typically the same file or a copy). The absence of any colour indicates that no similarity was detected between two binaries. An example of a correlation matrix (before clustering) is shown in Figure 3.a.

Once the matrix has been constructed, we use it as the starting point for two clustering algorithms, which we use in combination: DBSCAN[3] and a hierarchical clustering algorithm (called AgglomerativeClustering in scikit-learn[4]). The first method finds high-density central samples and develops clusters from them, while the second places each sample in its own cluster and then merges them successively according to a given selection criterion. For matrices larger than 8,000 x 8,000, we select a random subset of binaries, cluster them and then allocate a cluster to each remaining binary using a modified majority voting mechanism applied to the five nearest neighbours of each binary (see Algorithms 2, 3). The clusters are computed iteratively (see Algorithm 3): we determine the initial clusters (using either of the methods depending on the quality of the clustering produced), and then we re-divide them as long as doing so improves the overall clustering. The clustering quality (degree of separation and density) is evaluated by computing the average silhouette coefficient[5] for each binary, which we call the silhouette score in the rest of the article. This is bounded between -1 for incorrect clustering to +1 for very dense, well-separated clustering. Scores around zero indicate many overlaps between clusters.

Algorithm 1: best_clustering

```

input   : Correlation matrix  $M$ 
1  $c_1 \leftarrow$  optimal dbscan clustering on  $M$ ;
2  $c_2 \leftarrow$  optimal agglomerative clustering on  $M$ ;
3 if  $\text{silhouette\_score}(c_1) \geq \text{silhouette\_score}(c_2)$  then
4 |   return  $c_1$ ;
5 else
6 |   return  $c_2$ ;
7 end

```

The optimum clusterings are identified by testing a set of values for both methods.

- Agglomerative clustering: number of clusters to create $\in [2, |M|/2]$.
- DBSCAN: parameter epsilon (maximum distance between two samples for one to be considered a neighbour of the other) $\in [0.02, 0.55]$ (pas de 0.02).

Algorithm 2: improved_majority_vote

```

input   : binary  $x$ , correlation Matrix  $M$ , number of neighbors  $K$ 
output  : Associated cluster
1  $x_{nn}, l_{nn} \leftarrow$  find binaries and clusters of the  $K$ -NN of  $x$  in  $M$ ;
2  $m_{nn} \leftarrow$  mean correlation value on  $x_{nn}$  by cluster label from  $l_{nn}$ ;
3  $v_{max} \leftarrow \max(m_{nn})$ ;
4 if  $\exists! v \in m_{nn} \mid v = v_{max}$  then
5 |   return cluster with mean correlation value =  $v_{max}$ ;
6 else
7 |   return cluster with mean correlation value =  $v_{max}$  that is the most represented in  $l_{nn}$ ;
8 end

```

4.2. Clustering example

Figure 3.b shows a computation for a relatively small family (13 samples). In this case, clustering is fairly fast (around a second) and results in two distinct clusters. The figure was obtained by ordering the samples from the largest cluster to the smallest. Here one sample is placed alone in its own cluster (3dda3), though it has a slight connection with the C1 cluster. The silhouette score is very high, and shows good separation between the clusters. The matrix reveals two samples that will attract the analyst's attention: 704a0 and 58ccb. Both these samples belong to a cluster but also have a high correlation score with the elements of the other cluster, indicating a shared section of code.

Figure 4 shows an example of reclustering carried out on the merged Conti and Ryuk families in order to study the link between them². Again, the silhouette score is high, indicating globally dense clusters with few overlaps. There are a few exceptions, however, and these will serve as a basis for the analyses in the next section.

For comparison, we also computed the SSDEEP[2] distances for these malware families and generated the associated correlation matrices (see Figure 5). These are organised in the same order as the clusterings we defined. For the Babuk family, we can see that the C1 cluster from Figure 3.b is present, but without the 704a0 binary that linked the two clusters. The SSDEEP

²The computation time for these 69 samples was around 2.5 seconds.

Algorithm 3: Iterative clustering

```

input : a set of binaries  $B$ , a list of family labels by binary  $L$ , the associated correlation matrix  $M$ , the
         maximal size for a matrix to recluster  $N$ 
output : The found clustering  $c_t$ 
init:  $modif = \text{True}$ ,  $blacklist = \emptyset$ 
1  $m_{com} \leftarrow \max(|family| \forall family \in \{L\})$ ;
2  $n \leftarrow \max(N/|L|, 0.5 * m_{com})$ ;
3  $B_s \leftarrow \emptyset$ ;
4 for  $family \in \{L\}$  do
5    $B_f \leftarrow$  Select at most  $n$  random binaries from  $family$ ;
6    $B_s \leftarrow B_s \cup B_f$ ;
7 end
8  $c_s \leftarrow \text{best\_clustering}(M[L_s])$ ;
9  $c_t \leftarrow c_s$ ;
10 for  $binary \in B \setminus L_s$  do
11    $c_t[binary] \leftarrow \text{improved\_majoritary\_vote}(binary, M[B \setminus L_s], 5)$ ;
12 end
13  $s \leftarrow \text{silhouette\_score}(c_t)$ ;
14 while  $modif = \text{True}$  do
15    $c_R \leftarrow c_t \setminus blacklist$ ;
16    $modif \leftarrow \text{False}$ ;
17   for  $cluster \in c_R$  do
18      $c_u \leftarrow c_t \setminus cluster$ ;
19      $c_u \leftarrow c_u \cup \text{best\_clustering}(cluster)$ ;
20     if  $\text{silhouette\_score}(c_u) > s$  then
21        $c \leftarrow c_u$ ;
22        $modif \leftarrow \text{True}$ ;
23     else
24        $blacklist \leftarrow blacklist \cup \{cluster\}$ ;
25     end
26   end
27 end

```

method does not identify any other links between the Babuk binaries. For the Ryuk and Conti families, this method highlights only very few similarities, and none between the families. However, there is a strong link between the f0278, 3cd91, c5032 and fa13f binaries in the Conti family, which belong to cluster c5 in Figure 4. We can thus find shared similarities using either of the methods, but we can identify more links between the binaries, including between families, with our method for computing similarity.

The correlation matrices used to generate these clusterings (and other examples) are available from our GitHub repository³. In each case, we provide the input data (i.e. the matrix of correlations between samples) and images of the clusterings obtained with our method.

³https://github.com/glimps-re/automate_malware_CTI

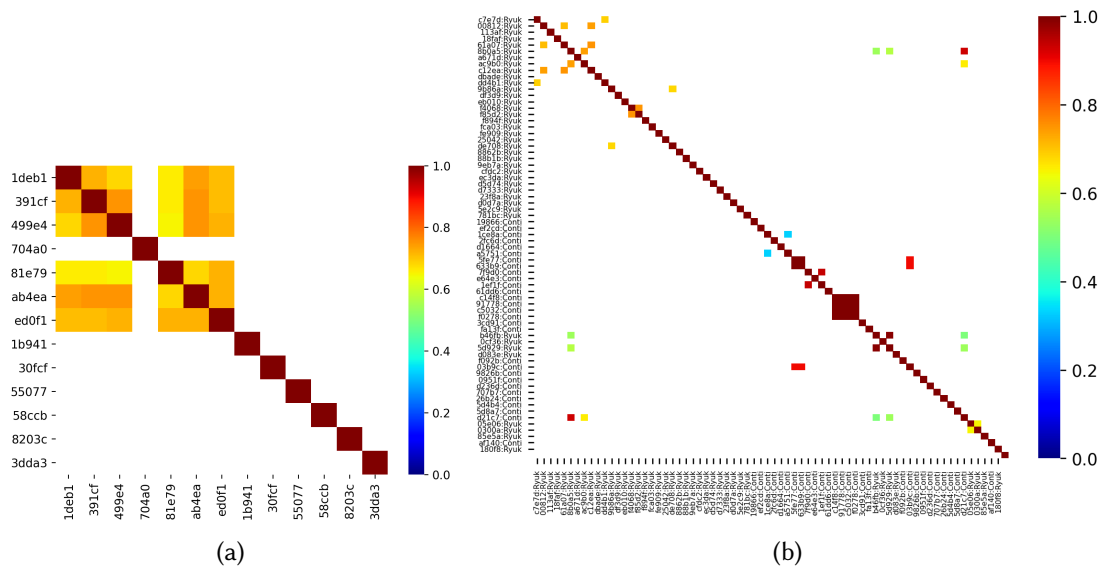


Figure 5: Correlation matrices for the Babuk (a), Ryuk and Conti (b) families, computed using the SSDEEP distance between each pair of binaries.

This kind of analysis is not the only potential application. The clusterings can also be used, for example, to automatically re-tag binaries (based on the clusters found), providing better ground truth for learning (malware detection, malware type classification, etc.). Ultimately, this will make it possible to improve the results of trained algorithms.

It is important to note that a clustering can easily be evolved by automatically injecting a new binary. We just calculate its similarity scores with the binaries already clustered and then allocate it a cluster using majority voting (for example) among the clusters of its N closest neighbours.

5. Use cases: the Babuk, Ryuk and Conti families and their links

To validate the approach presented above, it is important to compare it with a manual analysis of the samples. We thus cross-referenced these results against a reverse engineering analysis conducted with standard tools such as IDA⁴, which enables users to browse through a disassembled binary and gain a detailed understanding, sometimes at the cost of many hours of work.

5.1. Babuk

Babuk is a computer virus in the ransomware family. It neutralises the victim's computer by encrypting all the user's files, or even the whole system if the ransomware is run with the right permissions. The ransomware appeared on VirusTotal for the first time on 22 December

⁴<https://hex-rays.com/ida-pro/>

2020 under the name of Vasa Locker, and the group operating it calls itself Babuk. The group appeared on RaidForums on 2 January 2021[6]. Like many players, the group set up a “leak and sham” site on Tor⁵ to prove the victims had been compromised and blackmailed them in order to negotiate a ransom as high as possible. The first public analysis of Babuk was carried out by the researcher Chuong Dong[6], a student at Georgia Tech and an intern with FireEye’s Flare team. The generic functions of a piece of ransomware are always more or less the same, and Babuk is no exception:

- A function for traversing the file system to encrypt its data is always included.
- A public key algorithm for exchanging the data encryption key: this public key will be sent by the target to the attacker when the ransom is paid, and the private key is then used by the encryption software. Note that in most cases the decryption software is functional – the goal is for the victim to pay the ransom, and an untrustworthy reputation is bad for business. But even if the victim can recover their data, the work is not over. They then have to ensure that the attackers have not left any traces on their computers and rebuild their IT system from scratch. Failing this, the chances are that the incident will recur: according to several studies, over 20% of companies never manage to retrieve their data after paying the ransom, and almost 10% of ransomware victims go on to suffer a second attack [7, 8].
- A private key algorithm for encrypting the data. This type of algorithm is preferred over public key algorithms due to its speed.
- The creation of a file containing information on how to contact the attacker and the terms for paying the ransom.

These core ransomware functions are altered as the malware evolves, corresponding very closely to the correlation matrix presented in the previous section – see Figure 3.b.

In May 2021, the group changed its name to PayloadBin and stopped its ransomware activity[9]. During this period, the ransomware code evolved, and we have analysed it to confirm the clustering proposed automatically. The correlation matrix presented shows the three major evolutions of Babuk, which are summarised in Table 2.

Manual analysis through reverse engineering tells us that the first cluster (named C1) uses:

- a queueing system that traverses the file system recursively,
- the exchange of a shared private key using the public key algorithm curve25519⁶,
- mostly the hc-128 algorithm for data encryption, except for one sample that uses Salsa20.

The second cluster, C2, is older in terms of compilation date (the matrix is ordered based on cluster size, with no reference to date of appearance) and shows the use of less refined algorithms (particularly for traversing the file system and encryption):

- a parallelised mechanism for traversing the file system with threads based solely on the disks present. This results in a very slow encryption process. The process for traversing the file system stops at the 16th level of the folder structure. If files are located below this level, they will not be affected.

⁵[http://gtmx56k4hutn3ikv\[.\]onion/](http://gtmx56k4hutn3ikv[.]onion/)

⁶<https://github.com/agl/curve25519-donna/blob/master/curve25519-donna.c>

- The key exchange algorithm is ECDH, using an open-source library⁷ directly, or RSA.
- The private key algorithm is Salsa20.

Table 2
Functions of the Babuk family.

File hash	Compilation date	Fonctions	Cluster
81e7942a1f	2021-03-15 21:55:04	recursive queue hc-128 curve25519	c1
ed0f154481	2021-04-12 15:40:26	recursive queue hc-128 curve25519	c1
ab4eae618b	2021-02-23 11:12:08	recursive queue hc-128 curve25519	c1
1deb1efad2	2021-03-15 21:50:31	recursive queue hc-128 curve25519	c1
391cfdc153	2021-02-19 16:06:32	recursive queue hc-128 curve25519	c1
499e43933c	2021-02-19 21:19:12	recursive queue hc-128 curve25519	c1
704a0fa7de	2021-01-13 13:12:33	recursive queue salsa20 curve25519	c1
1b9412ca5e	2021-01-02 16:50:38	no queue salsa20 ecdh	c2
8203c2f00e	2020-12-30 11:03:14	no queue salsa20 ecdh	c2
30fcff7add	2021-01-04 11:20:28	no queue salsa20 ecdh	c2
550771bbf8	2021-01-11 17:15:09	no queue salsa20 ecdh	c2
58ccb4fb2	2021-01-27 20:41:07	recursive queue salsa20 rsa	c2
e8ccb4fb2	2021-01-28 20:41:07	recursive queue salsa20 curve25519	c3

However, the samples 704a0 and 58ccb provide a vital information – they make a link between the two clusters, enabling us to establish connections between the two series of ransomware. This link is even more visible when we compare it with the compilation timeline (see Figure 6). Finally, we note the presence of an outlier, the sample 3dda3: it has functions in common with cluster 1, but they have been altered during compilation. It thus appears as separate from the cluster, but with weak links to two samples.

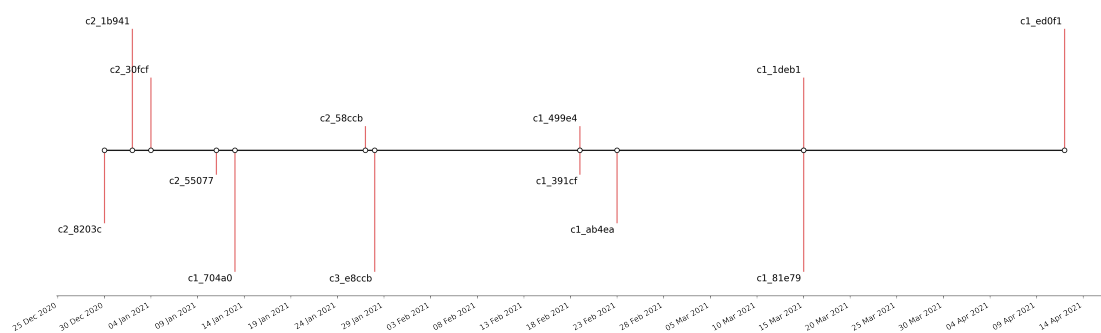


Figure 6: Timeline of the Babuk family.

⁷<https://github.com/kokke/tiny-ECDH-c/blob/master/ecdh.c>

5.2. Ryuk and Conti

This initial analysis of a group with a limited number of samples validated the approach based on creating cross-correlation matrices and exploiting the information this provides. We then successfully applied this method to a larger number of samples from two malware groups (Ryuk and Conti), for which links seem to exist according to the literature.

Like Babuk, Ryuk belongs to the malware family of ransomware. Unlike Babuk, however, Ryuk's developers are not the direct operators of the malware. Ryuk is operated by affiliates, as demonstrated by several articles including an ANSSI publication in March 2019[10]. One of the operators is Evil Corp, the name given to a group of cybercriminals, who compromise the victim's entire information system and then deploy their ransomware before applying encryption. Ryuk has compromised many victims in France, and particularly hospitals[11].

Conti is another member of the same family. It emerged in 2020. In the same way as Babuk, Conti's operators publish a website listing the victims and the data encrypted by the attackers. Again, as with Babuk, this technique aims to put pressure on the victim and guarantee that as high a ransom as possible will be paid. Unlike Babuk, however, these two families use techniques to obstruct reverse engineering analysis. In the case of Conti, for example, no calls are made via the import table – instead, strings of characters are decoded on the fly to reconstruct the table and the addresses of the stored functions so that they can be called later.

The correlation matrix reveals the evolution of these techniques and how they have been implemented in the two families. The techniques are not specific to these malware families, and other viruses have already used them. However, here it is the implementations of these techniques that were compared, revealing the link between the Conti and Ryuk families. A literature review also confirmed that links have been demonstrated showing that the same coders are responsible for the two pieces of ransomware[12, 13]. The cross-correlation matrix shows an identical development persona for certain samples of Ryuk (23f8a, 5e2c9, 781bc, 8862b, 88b1b, 9eb7a, cfdc2, d0d7a, d5d74, d7333, ec3da) and Conti (61dd6, e64e3, 633b9). The author or authors use the same libraries and the same code sequences. If we look at the compilation timeline (see Figure 7), we clearly see the interweaving of Conti and Ryuk. This chronology provides additional context to supplement the similarity found above.

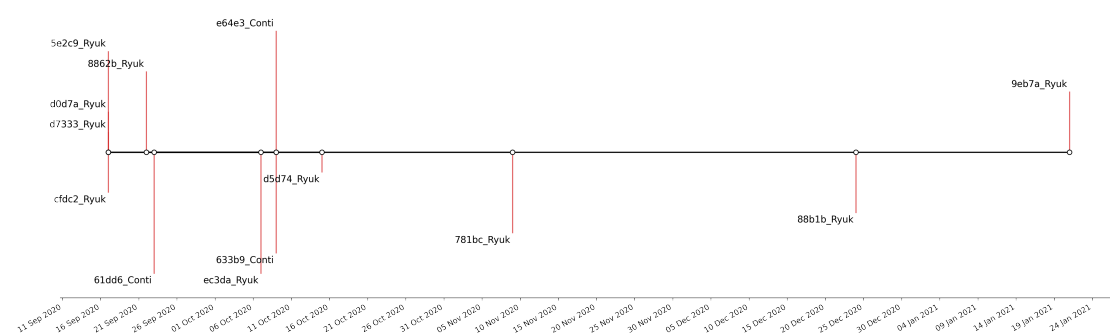


Figure 7: Timeline of the Conti and Ryuk families.

6. Conclusion

A traditional threat intelligence approach can provide global information about the threat associated with a specific group. But as we have seen, a single name (Ryuk, Conti, Babuk, etc.) can hide many variants of the same threat. When an incident occurs and a company or organisation needs to respond effectively, it is vital to know which sample the threat is associated with. The continuous collection of malware and binaries from open sources, together with clustering using a high-performance correlation technology, enables us to keep our exhaustive knowledge of attacker groups and their various malware releases constantly up to date. This means that when threat intelligence analysts are called in to support the teams responding to an incident, they can guide the response differently depending on whether the binary they are dealing with focuses on encryption, lateral movement or leaking data.

In addition, an effective incident response usually requires decision-makers to be involved in the process, enabling them to be kept up to date with the progress of the response and to manage the crisis. As well as the standard CTI indicators, which are purely technical markers (YARA rules, hashes, etc.), the approach presented here offers the prospect of automatically proposing a concrete, explicit view of where a particular sample found in the IT system is positioned within the “galaxy” of malware used by the attacker. This communication tool helps make the threat more tangible and more easily understood by business executives, which means it can be taken into account more thoroughly in decision-making processes.

7. Acknowledgements

We are very grateful to everyone who contributes to the sharing of data and information about cyber threats. The effort to guard against these threats is currently faced with two challenges: the availability of relevant data, and the capacity to process this data on a time scale compatible with an effective response. Our activity is automation, but it relies on data. We would also like to thank everyone who has supported GLIMPS since its creation and enabled this work, including the French armed forces ministry, the DGA, the Cyberdéfense Factory, Pool, Rennes Métropole, the BDI and all our partners.

References

- [1] I. U. Haq, J. Caballero, A survey of binary code similarity, arXiv preprint arXiv:1909.11424 (2019).
- [2] J. Kornblum, Identifying almost identical files using context triggered piecewise hashing, *Digital investigation* 3 (2006) 91–97.
- [3] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise., in: *Kdd*, volume 96, 1996, pp. 226–231.
- [4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, *Scikit-learn: Machine learning in Python*, *Journal of Machine Learning Research* 12 (2011) 2825–2830.

- [5] P. J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *Journal of computational and applied mathematics* 20 (1987) 53–65.
- [6] C. Dong, Babuk ransomware overview, 2021. URL: <https://chuongdong.com/reverse%20engineering/2021/01/03/BabukRansomware/>.
- [7] Sophos, The State of Ransomware 2021, Technical Report, 2021. URL: <https://secure2.sophos.com/en-us/medialibrary/pdfs/whitepaper/sophos-state-of-ransomware-2021-wp.pdf>.
- [8] D. Cecile, Ransomwares : faut-il payer la rançon ?, 2020. URL: <https://business.lesechos.fr/directions-financieres/comptabilite-et-gestion/gestion-des-risques/0602742669879-ransomwares-faut-il-payer-la-rancon-334974.php>.
- [9] Dissent, Babuk re-organizes as payload bin, offers its first leak, 2021. URL: <https://www.databreaches.net/babuk-re-organizes-as-payload-bin-offers-its-first-leak/>.
- [10] Informations concernant les rançongiciels lockergoga et ryuk, Technical Report CERTFR-2019-ACT-005, Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), 2019.
- [11] L. Monde, Après celui de dax, l'hôpital de villefranche paralysé par un rançongiciel, 2021. URL: https://www.lemonde.fr/pixels/article/2021/02/15/rancongiel_6070049_4408996.html.
- [12] S. Kalollu, The ryuk-conti connection: A ransomware blog, 2020. URL: <http://blog.escanav.com/2020/07/the-ryuk-conti-connection-a-ransomware-blog/>.
- [13] Le rançonlogiciel Ryuk, Technical Report CERTFR-2020-CTI-011, Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), 2021.