ERNW RESEARCH
pursuing knowledge.

ERNW WHITEPAPER 71

ANALYSIS OF ANTI-VIRUS SOFTWARE QUARANTINE FILES

Version:	1.0
Date:	January 27, 2021
Classification:	Public
Author(s):	Florian Bausch

ERNW RESEARCH
pursuing knowledge.

Table of Content

ERNW RESEARCH
pursuing knowledge.

1 Introduction

Anti-virus software (AV software) is a type of computer software that tries to identify malicious software and to prevent it from running. Since anti-virus software may wrongfully identify harmless files as malicious (*false positives*), AV software makes use of *quarantining* files. If a file is put into quarantine by an AV software, the AV software removes the original suspected malicious file and stores a modified obfuscated version in another location. Obfuscation is achieved by encrypting the original file with more or less strong algorithms. However, the strength of the encryption algorithm is not important, since the encryption is only meant to transform executable files into a representation that cannot be executed by the computer. The encryption is usually symmetric because it is faster and more simple to handle than asymmetric encryption. To be able to restore files from quarantine, the AV software stores additional information besides the encrypted malicious file. This information usually includes the original path, timestamps, checksums (for example MD5, SHA1, SHA256), and the name and type of the suspected malware.

Quarantine files of AV software can be useful sources of information during incident analyses. They preserve malicious files that may have been executed on the system, but also give hints when a computer got infected by malware. They may even include information about processes that created the malicious files. The challenge when trying to use quarantine files during incident analysis is that there is a big variety of AV software, each using a different proprietary undocumented form of storing the information.

Based on these considerations, the quarantine files of different AV software were analyzed. The encryption and obfuscation methods were documented (including encryption keys) and parsers created using Kaitai Struct[1]. Using the parsers and other documented information it is possible to create a unified tool that can automatically analyze quarantine files of different AV software, show the information, transform the information into Sleuth Kit's body file format (Carrier, 2009), look up more information from online services such as VirusTotal[2], and restore malicious files for further static or dynamic malware analysis.

1.1 Contribution

This whitepaper gives a detailed overview over structure, content, and encryption / obfuscation methods of anti-virus quarantine files. The file formats are described so that incident analysists can find useful information more easily. Furthermore, the Kaitai Struct parser definition files can be used to implement tools for automated analysis. The results of this research work are also available on Github: *https://github.com/ernw/quarantine-formats*.[3]

The examined quarantine files belong to the following anti-virus software:

[1] *https://kaitai.io/*
[2] *https://virustotal.com/*
[3] *Licensed under Creative Commons CC BY-SA 4.0, https://creativecommons.org/licenses/by-sa/4.0/*

- Avira
- Windows Defender
- Malwarebytes
- Symantec Endpoint Protection
- G Data
- Sophos Antivirus
- Kaspersky for Windows Server

For G Data, Sophos, and Kaspersky, this whitepaper is the first description of the file formats. For the other file formats, it is – to our knowledge – the first structured, detailed description.

1.2 Related Work

There are several resources on the Internet, e.g. blog posts, about different quarantine file formats that give a more or less detailed view on selected file formats.

A tool that supports the decryption and deobfuscation of quarantine files of various AV software is DeXRAY (`dexray.pl`) by Adam Grant (Grant, 2019a). However, DeXRAY simply restores malicious files and does not analyze metadata. Futhermore, there is no documentation on how the quarantine files are processed and why they are processed this way.

A basic description of the Windows Defender file format was created by Jon Glass. However, it shows how to restore quarantined files without examining metadata and supporting files (Glass, 2015).

The blog post *Recovering Malware from a Quarantine Folder* by M. Gene Shantz (Shantz, 2016) gives hints about analyzing Avira Antivirus files without trying to understand the complete file format. The author shows how to manually extract a quarantined file, but ignores the additional metadata provided by the quarantine file.

For Symantec Endpoint Protection's quarantine files there is a not actively developed program called VBNExtract (conix-security, 2014). It can also only extract files from quarantine without using additional metadata or understanding the whole file structure. A more detailed analysis of this file format was performed and described by Brian Maloney (Maloney, 2018).

Another not actively developed program is a fork of the Cuckoo dynamic analysis framework. It contains a functionality to unquarantine files from different quarantine file formats (Sprengler, 2015). However, there is little documentation about the supported file formats.

1.3 Kaitai Struct

Kaitai Struct[4] is a YAML-based format to define parsers for arbitrary binary files in a generic description language. Parser definitions can be compiled into program code – several programming languages are supported. This simplifies creating and reading a parser definition and to use it in different projects. Furthermore, Kaitai Struct provides the Kaitai Visualizer tool. This tool allows to use the parser definition to interactively browse through a binary file parsed using this definition.

Kaitai Struct comes with a built-in support for xor-processing files or parts of files. This method is often used as a simple encryption method for quarantine files. If another encryption method is used, the decryption routine – for example RC4 – can be implemented in the required programming language and used in the parser definition. (See appendix A – *RC4 Helper Class in Python*, page 35 for an RC4 helper class in Python.)

[4] *https://kaitai.io/*

2 Quarantine Analysis

The samples for the analysis were taken from real-life incident analysis cases and from especially generated quarantine files. These generated files were created by installing an AV software inside of a Windows 7 virtual machine (VM) and adding the EICAR test file (EICAR, 2006), but also real malware samples. After detection by the AV software the quarantine directories of the AV were saved for analysis.

The selection of AV software in this document was influenced by the encountered AV software on analyzed systems during real life incident cases.

For the three anti-virus software solutions by G Data (section 2.1), Sophos (section 2.2), and Kaspersky (section 2.3) no previous, related work could be found. Because of this, these sections contain a more detailed description on the research work. For the other software solutions Avira (section 2.4), Malwarebytes (section 2.5), Windows Defender (section 2.6), and Symantec (section 2.7) it was possible to use findings from other resources. Therefore, these sections contain a less detailed description of the results.

2.1 G Data

The analysis of the G Data quarantine format started with two quarantine files from an incident analysis. The two original malware files were found on the system, too, which simplified the first steps of analysis. G Data quarantine files are located at `/ProgramData/GData/AVK/Quarantine/` and their file names follow this pattern: "`<hostname>DDMMYYHHmmss[0-9A-F]{8}.q`". However, the hostname is not always included in the file names.

The following listing shows as an example a quarantine file containing the EICAR test file. The file consists of three sections; the first two sections are easy to spot at first glance:

1. A section starting with the magic bytes `0xca 0xfe 0xba 0xbe`, followed by its length (`0x54` in this case).
2. A section starting with the magic bytes `0xba 0xad 0xf0 0x0d`, followed by its length (`0x9c` in this case).
3. In this example, the second section ends at `0x100`. The remainder of the file has exactly the length of the quarantined EICAR test file (`0x44` bytes).

```
00000000: cafe babe 5400 0000 e864 787b 7094 5280  ....T....dx{p.R.
00000010: 49cd 626d 7b9f 794c 1178 72cf 1f40 2565  I.bm{.yL.xr..@%e
00000020: d076 b97b 3994 0dd1 e09d 279c c544 09b6  .v.{9.....'..D..
00000030: 5273 9b84 ca8d d276 1595 d60a 9bee 8c7d  Rs.....v.......}
00000040: 35cf e5d1 562a ba46 8beb 5d86 5dcf 3bca  5...V*.F..].].;.
00000050: 902d bb48 8f84 2a40 0886 d6e6 baad f00d  .-.H..*@........
00000060: 9c00 0000 e864 787b 7094 5280 07cd 62ed  .....dx{p.R...b.
00000070: f32d c312 1078 72cf c0be da78 0f46 8896  .-...xr....x.F..
00000080: 1576 99d0 3242 156e fea6 b9b7 bb43 9769  .v..2B.n.....C.i
```

```
00000090: 886f 4177 7c95 ba0a baee ac7d e231 74fe  .oAw|......}.1t.
000000a0: 652a 9246 94eb 6086 3ecf 77ca a52d 9c48  e*.F..`.>.w..-.H
000000b0: 8984 3c40 5386 a5e6 a0ad 4adb 4109 9891  ..<@S.....J.A...
000000c0: b9ec dc76 a6bc 5c0b a676 6885 f615 0643  ...v..\..vh....C
000000d0: 4c36 bed4 0627 6bc3 5511 2ad4 3b28 9ba9  L6...'k.U.*.;(..
000000e0: 0b3a 0cab 221f b076 dcd3 8e7e d220 a9ab  .:.."..v...~. ..
000000f0: b8e7 49bb b4db e817 c156 e7b1 f372 c1ff  ..I......V...r..
00000100: b151 375a 20b1 12c1 1396 56b1 5c89 6427  .Q7Z .....V.\.d'
00000110: 2550 2291 c989 993b bc41 8d5f 3fdd 0f90  %P"....;.A._?...
00000120: e0b0 59c8 d00a 28f7 7337 c2c5 a9d9 dd20  ..Y...(.s7..... 
00000130: 35c7 ef59 d3ba e92e 49e2 cd98 756f ef62  5..Y....I...uo.b
00000140: e3c0 74ac                                ..t.
```

Neither of the three sections of the file shows repetitive patterns, which led to the assumption that a stream cipher instead of a simple xor key was used to encrypt the sections. However, the first two sections show a similar pattern in the beginning (`0xe8 0x64 ...`), which led to the assumption that all three sections might have been encrypted using the same algorithm and key.

To check this hypothesis, the third section of the two quarantine samples was xored with the original malware files found on the system. Both resulting byte sequences were the same (except their length, since the malware files did not have the same length).

The resulting byte sequence was xored with the first two sections of the malware samples. The results showed usable data (see sections 2.1.1 and 2.1.2).

2.1.1 First section

The first section contains the Unix epoch when the file was quarantined (offset `0x0c` in the following listing). The name of the detected malware starts at offset `0x14`. It is encoded as a UTF-16LE string, starting with a BOM (Unicode, 2020), followed by the number of characters in the string (in this example `0x1d`). The string is null-terminated.

```
00000000: 0100 0000 0000 0000 0a00 0080 774c 455e  ............wLE^
00000010: 0000 0000 fffe ff1d 4500 4900 4300 4100  ........E.I.C.A.
00000020: 5200 2d00 5400 6500 7300 7400 2d00 4600  R.-.T.e.s.t.-.F.
00000030: 6900 6c00 6500 2000 2800 6e00 6f00 7400  i.l.e. .(.n.o.t.
00000040: 2000 6100 2000 7600 6900 7200 7500 7300   .a. .v.i.r.u.s.
00000050: 2900 0000                                )...
```

Listing 1: Decrypted first section of G Data quarantine file

2.1.2 Second section

The second section contains a string at offset `0x0c`. In all examined samples this string was empty; therefore, its function remains unknown.

At offsets `0x18`, `0x20`, and `0x28` there are three Windows File Time timestamps (Microsoft, 2018).

The original file path of the detected malware starts at offset `0x38`. Again, it is encoded as a null-terminated UTF-16LE string with BOM and number of characters.

Additionally, the file size of the original malware file is encoded twice, at `0x08` and `0x34`.

```
00000000: 0100 0000 0000 0000 4400 0000 fffe ff00  ........D.......
00000010: 0100 0000 2000 0000 9a30 78ed 6fe2 d501  .... ....0x.o...
00000020: 80df 1ff2 6fe2 d501 9a30 78ed 6fe2 d501  ....o....0x.o...
00000030: 0000 0000 4400 0000 fffe ff2f 5c00 5c00  ....D....../\.\.
00000040: 3f00 5c00 4300 3a00 5c00 5500 7300 6500  ?.\.C.:.\.U.s.e.
00000050: 7200 7300 5c00 4900 4500 5500 7300 6500  r.s.\.I.E.U.s.e.
00000060: 7200 5c00 4400 6500 7300 6b00 7400 6f00  r.\.D.e.s.k.t.o.
00000070: 7000 5c00 6500 6900 6300 6100 7200 5c00  p.\.e.i.c.a.r.\.
00000080: 6500 6900 6300 6100 7200 2e00 6300 6f00  e.i.c.a.r...c.o.
00000090: 6d00 2e00 7400 7800 7400 0000            m...t.x.t...
```

Listing 2: Decrypted second section of G Data quarantine file

2.1.3 Encryption algorithm and key

G Data uses RC4 for the encryption of malware samples and the two metadata sections. The key is located in file `AVKQt.dll` at offset `0x8adb8`, in the `.rdata` section. It has a length of 16 bytes. The value is:

```
0xA7, 0xBF, 0x73, 0xA0, 0x9F, 0x03, 0xD3, 0x11, 0x85, 0x6F, 0x00, 0x80, 0xAD, 0xA9, 0x6E, 0x9B
```

Listing 3: G Data RC4 key

2.1.4 Parser definition

Combining the previous findings, a Kaitai Struct parser definition file can be created:

```
meta:
  id: gdata
  file-extension: q
  endian: le
  title: G Data quarantine file parser
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
```

```
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: magic1
    size: 4
    contents: [0xca, 0xfe, 0xba, 0xbe]
  - id: len_data1
    type: u4
  - id: data1
    size: len_data1
    type: encrypted_data1
    process: util.custom_arc4.custom_arc4(<RC4 key>)
  - id: magic2
    size: 4
    contents: [0xba, 0xad, 0xf0, 0x0d]
  - id: len_data2
    type: u4
  - id: data2
    size: len_data2
    type: encrypted_data2
    process: util.custom_arc4.custom_arc4(<RC4 key>)
  - id: mal_file
    size-eos: true
    process: util.custom_arc4.custom_arc4(<RC4 key>)
types:
  encrypted_data1:
    seq:
      - id: unknown1
        size: 0x0c
      - id: quatime
        type: u4
      - id: unknown2
        size: 0x04
      - id: malwaretype
        type: utf16le
  encrypted_data2:
    seq:
      - id: unknown1
        size: 0x08
      - id: filesize
        type: u4
      - id: unknownstring1
        type: utf16le
      - id: unknown2
```

```
      size: 0x08
    - id: time1
      type: winfiletime
    - id: time2
      type: winfiletime
    - id: time3
      type: winfiletime
    - id: unknown3
      size: 0x04
    - id: filesize2
      type: u4
    - id: path
      type: utf16le
utf16le:
  seq:
    - id: bom
      size: 3
      contents: [0xFF, 0xFE, 0xFF]
    - id: number_of_chars
      type: u1
    - id: string_content
      type: str
      encoding: utf-16le
      size: number_of_chars * 2
winfiletime:
  seq:
    - id: ts
      type: u8
  instances:
    unixts:
      value: (ts * 1e-07) - 11644473600
```

Listing 4: Parser definition for G Data quarantine files

2.2 Sophos Antivirus

The initial Sophos Antivirus samples were collected in an incident. In this case, the original malicious files could not be recovered from the system. Further samples were created in a Windows 7 VM and the EICAR test file.

Sophos uses a database containing the metadata of quarantined files (found in different locations, but most likely under `/ProgramData/Sophos Anti-Virus/Safestore/`). The malicious files are encrypted and stored in separate files next to the database.

2.2.1 SafeStore.db

The metadata is stored in an encrypted SQLite 3 database. The file can be named `SafeStore.db` or `Safestore.db` and there can be more than one database per system. The encryption key is stored in the same directory and named either `safestore_key.txt` or `SafeStore.pw`. The database can be decrypted using sqlcipher.[5]

There are two files per quarantined file. One file is the encrypted malicious file, the other one the encrypted ACLs of that file. The names of the files are stored in `BlobTable.name`. The table `BlobTable` stores other interesting data:

- `originalchecksum`: The SHA256 checksum of the uncompressed, unencrypted file.
- `size`: The file size of the compressed, encrypted file.

More interesting metadata can be found in table `StringPropertyTable`. This table can contain more or less information, depending on the version of Sophos Antivirus. This table stores – depending on the value of `type` – several information about a quarantined file (referenced by `objectid`):

- Type `name`: The original file name of the malicious file.
- Type `location`: The directory of the original file.
- Type `threatname`: The name of the detected malware.
- Type `filecreationtime`: Windows File Time (Microsoft, 2018) of the creation of the original file.
- Type `filelastwritetime`: Windows File Time (Microsoft, 2018) of the last write to the original file.

The table `ThreatObjectTable` contains the information when a file was moved to quarantine:

- `datetime`: The Unix timestamp when the file was quarantined.

2.2.1.1 SafeStore.db SQLite 3 database schema

```
CREATE TABLE "BlobPropertyTable" (
      "blobpropertyid"  integer,
      "type" varchar NOT NULL,
      "objectid"  integer NOT NULL,
      "blobid"    integer NOT NULL,
      FOREIGN KEY("blobid") REFERENCES "BlobTable"("blobid"),
      FOREIGN KEY("objectid") REFERENCES "ThreatObjectTable"("objectid"),
      PRIMARY KEY("blobpropertyid")
);

CREATE TABLE "BlobTable" (
      "blobid"    integer,
      "name" varchar NOT NULL,
```

[5] *https://www.zetetic.net/sqlcipher/*

```
	"size" integer NOT NULL,
	"originalchecksum" varchar NOT NULL,
	"storedchecksum"  varchar NOT NULL,
	"blobversion" varchar NOT NULL,
	PRIMARY KEY("blobid")
);

CREATE TABLE "MetaInfoTable" (
	"schema_vermajor" integer NOT NULL,
	"schema_verminor" integer NOT NULL,
	"schema_verpatch" integer NOT NULL,
	"min_safestore_vermajor" integer NOT NULL,
	"min_safestore_verminor" integer NOT NULL,
	"min_safestore_verpatch" integer NOT NULL,
	"file_encryption_salt" varchar NOT NULL,
	"file_encryption_test" varchar NOT NULL
);

CREATE TABLE "StringPropertyTable" (
	"stringpropertyid" integer,
	"type" varchar NOT NULL,
	"value" varchar NOT NULL COLLATE nocase,
	"objectid"  integer NOT NULL,
	FOREIGN KEY("objectid") REFERENCES "ThreatObjectTable"("objectid"),
	PRIMARY KEY("stringpropertyid")
);

CREATE TABLE "ThreatObjectTable" (
	"objectid"  integer,
	"objectguid" varchar NOT NULL,
	"threatguid" varchar NOT NULL,
	"type" varchar NOT NULL,
	"datetime"  integer NOT NULL,
	"safestoreversion" varchar NOT NULL,
	"status"    varchar NOT NULL,
	PRIMARY KEY("objectid")
);

CREATE INDEX "BlobPropertyTable_Index" ON "BlobPropertyTable" (
	"objectid"  asc,
	"blobid"    asc
);

CREATE INDEX "BlobTable_Index" ON "BlobTable" (
```

```
      "originalchecksum" asc
);

CREATE INDEX "StringPropertyTable_Index" ON "StringPropertyTable" (
      "objectid"  asc
);

CREATE INDEX "ThreatObjectTable_Index" ON "ThreatObjectTable" (
      "datetime"  asc
);
```

Listing 5: Sophos Antivirus SafeStore.db database schema

2.2.2 Encrypted files

The encrypted files have file sizes that are different to the file sizes of the original malicious files. While larger samples showed that the original files were larger than the quarantined files, the (small) EICAR test file was smaller than the matching quarantine file. This led to the assumption that the files were compressed before encryption – for small files the compression overhead actually increased the file size.

In a VM with Windows 7, Sophos Antivirus was installed and the restore functionality analyzed with a debugger. While the encryption algorithm and key could not yet be identified, this analysis showed that the zlib compression library is used to compress the files before encryption. The magic bytes `0x78 0x9C` found in memory show that the standard compression level is used.

2.3 Kaspersky for Windows Server

The samples for Kaspersky for Windows Server came from an incident analysis. The original malicous files could also be found on the system. The quarantine files are located at `/ProgramData/Kaspersky Lab/Kaspersky Security for Windows Server/<version>/Quarantine/`.

Kaspersky for Windows Server uses a central database for metadata of quarantined files (`quarantine.db`). The malicious files are stored separately (the name is a UUID).

The analysis was straightforward since the database was not encrypted and most column names are easily understandable. The xor encryption key for the quarantine files could be found by xoring the original malicious file with the quarantine file.

2.3.1 quarantine.db

The metadata database is a SQLite 3 database called `quarantine.db`.

```
CREATE TABLE objattrs (uuid text primary key,attrs int,rights blob,creation_time int,access_time int,
    ↪ write_time int)
CREATE TABLE objects (uuid text primary key,path text,name text,virname text,virtype int,level int,
    ↪ time int,size int,status int,sent int,user text)
```

Listing 6: Quarantine database schema of Kaspersky for Windows Server

Columns interesting for analyses are explained in the following.

Table `objects`:

- `uuid`: The UUID of the threat. At the same time, it is the file name of the encrypted quarantine file holding the malicious file.
- `path`: Directory of original file.
- `name`: Name of the original file.
- `virname`: The name of the virus.
- `time`: Timestamp of quarantine. This timestamp is in a proprietary format and stores the timestamp in the system's local time without storing timezone information. (See section 2.3.2.)
- `size`: The file size of the original file.
- `user`: Probably the owner of the file.

Table `objattrs`:

- `uuid`: See above.
- `rights`: Binary respresentation of the security descriptor of the original file.
- `creation_time`: crtime (Windows File Time (Microsoft, 2018)).
- `access_time`: atime (Windows File Time).
- `write_time`: mtime (Windows File Time).

2.3.2 Quarantine timestamp

The quarantine timestamp is stored as an integer in the database. The following list shows how to compute the different data out of the timestamp and how many bits each datum takes in the integer:

1. Years: 16 bits: `(time >> 48) & 0xFFFF`
2. Months: 8 bits: `(time >> 40) & 0xFF`
3. Days: 8 bits: `(time >> 32) & 0xFF)`
4. Hours: 8 bits: `(time >> 24) & 0xFF)`
5. Minutes: 8 bits: `(time >> 16) & 0xFF)`
6. Seconds: 8 bits: `(time >> 8) & 0xFF)`

7. (Probably) unused: 8 bits: `qtime & 0xFF`

The date is the local date, which means that during analysis the timezone of the system has to be known to interpret the timestamp correctly.

2.3.3 Encrypted files

The malicious files get encrypted using an eight-byte xor key:

```
0xe2 0x45 0x48 0xec 0x69 0x0e 0x5c 0xac
```

Listing 7: Kaspersky for Windows Server xor key

The encrypted quarantine files are stored in the same directory as the `quarantine.db`. They contain only the quarantined files and no other data.

2.3.3.1 Parser definition

The Kaitai Struct parser definition is very simple since only one datum (the malicious file) is stored in the file:

```
meta:
  id: kasperskyserver
  endian: le
  title: Kaspersky for Windows Server quarantine file parser
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: mal_file
    process: xor([0xe2, 0x45, 0x48, 0xec, 0x69, 0x0e, 0x5c, 0xac])
    size-eos: true
```

Listing 8: Parser definition for Kaspersky for Windows Server quarantine files

2.4 Avira Antivirus

Avira stores quarantined files under `/ProgramData/Avira/Antivirus/INFECTED`. The file extension is `.qua`.

The interesting information are (names refer to section 2.4.1):

- `qua_time`: The unix epoch when the file was quarantined.
- `mal_type`: The name or type of the malware.

- `filename`: The original path of the quarantined file. Sometimes this field seems to hold a process name. Maybe this happens if a process is prevented from writing a malicious file.
- `addl_info`: Additional information, which is not always present.
- `mal_file`: The file content of the quarantined file.

The malicious file is stored obfuscated using a single-byte xor key. The key is `0xAA`.

2.4.1 Parser definition

```
meta:
  id: qua
  file-extension: qua
  endian: le
  title: Avira Antivirus quarantine file parser
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: magic
    size: 16
    contents: ["AntiVir Qua", 0x00, 0x00, 0x00, 0x00, 0x00]
  - id: malicious_offset
    type: u4
  - id: len_filename
    type: u4
  - id: len_addl_info
    type: u4
  - id: unknown1
    size: 0x20
  - id: qua_time
    type: u4
  - id: unknown2
    size: 0x5c
  - id: mal_type
    type: str
    terminator: 0
    encoding: UTF-8
    size: 0x40
  - id: filename
    type: str
    encoding: UTF-16LE
    size: 'len_filename < 2 ? len_filename : len_filename - 2'
```

```
  - id: padding1
    size: 2
    contents: [0x00, 0x00]
    if: 'len_filename >= 2'
  - id: addl_info
    type: str
    encoding: UTF-16LE
    size: 'len_addl_info < 2 ? len_addl_info : len_addl_info - 2'
  - id: padding2
    size: 2
    contents: [0x00, 0x00]
    if: 'len_addl_info >= 2'
  - id: mal_file
    process: xor(0xaa)
    size-eos: true
```

Listing 9: Parser definition for Avira quarantine files

2.5 Malwarebytes

The samples for Malwarebytes were created in a Windows 7 VM with a Malwarebytes installation. The EICAR test file and further malware samples were copied to the VM and quarantined by Malwarebytes.

Malwarebytes creates two files per malicious file. One file contains the encrypted malicious file, the other file contains the metadata. This metadata file is an encrypted JSON file. Both files are encrypted using the following string. This string is hashed using MD5. The resulting hash is used to initialize the RC4 algorithm (Grant, 2019b, pp. l. 1644).

```
Go9r%8hhAl7Ari;vnQ8wwkmeostfETkzLEf5*+6u8MF.CbsYKbTt9w.cVJbJ+pzyvrsT
```

Listing 10: Malwarebytes RC4 encryption key

The file names consist of a UUID and a file extension that indicates whether it contains the malicious file or metadata:

- `.quar`: Malicious file, without any additional data
- `.data`: Metadata file

A metadata file may look like this:

```
{
  "componentsUpdatePackageVersion" : "1.0.613",
  "dbSDKUpdatePackageVersion" : "1.0.12023",
  "ruleID" : 683363,
  "ruleString" : "",
  "schemaVersion" : 8,
  "source" : "scan",
```

```
"sourceID" : "<UUID>",
"sourcePath" : "<PATH>",
"threatID" : 3912,
"threatName" : "<TYPE OF MALWARE>",
"trace" : {
  "cleanAction" : "quarantine",
  "cleanContext" : {
  },
  "cleanResult" : "successful",
  "cleanResultErrorCode" : 0,
  "cleanTime" : "2019-08-15T11:35:10Z",
  "objectID" : "<UUID WHICH IS ALSO PART OF FILE NAME>",
  "objectMD5" : "<MD5 HASH>",
  "objectPath" : "<ORIGINAL FILE PATH>",
  "objectSha256" : "<SHA256 HASH>",
  "objectType" : "file",
  "parentID" : "",
  "relationToParent" : "none",
  "suggestedAction" : {
    "chromeExtensionOther" : false,
    "chromeExtensionPreferences" : false,
    "chromeExtensionSecurePreferences" : false,
    "chromeExtensionSyncData" : false,
    "chromeUrlOther" : false,
    "chromeUrlSecurePreferences" : false,
    "chromeUrlSyncData" : false,
    "chromeUrlWebData" : false,
    "fileDelete" : true,
    "fileReplace" : false,
    "fileTxtReplace" : false,
    "folderDelete" : false,
    "isChromeObject" : false,
    "isDDS" : false,
    "isWMIEventConsumer" : false,
    "minimalWhiteListing" : false,
    "moduleUnload" : false,
    "noLinking" : false,
    "physicalSectorReplace" : false,
    "priorityHigh" : false,
    "priorityNormal" : false,
    "priorityUrgent" : false,
    "processUnload" : false,
    "regKeyDelete" : false,
    "regValueDelete" : false,
```

```
        "regValueReplace" : false,
        "shortcutReplace" : false,
        "treatAsRootkit" : false,
        "useDDA" : false
      }
    }
}
```

Listing 11: Sample JSON metadata file by Malwarebytes

2.6 Windows Defender

The quarantine files of Windows Defender are located at:

`/ProgramData/Microsoft/Windows Defender/Quarantine`

There are three subdirectories holding different kinds of binary files:

1. `ResourceData`
2. `Resources`
3. `Entries`

These files or parts of these files are encrypted using RC4 with the following key (Sprengler, 2015, line 186):

```
0x1E, 0x87, 0x78, 0x1B, 0x8D, 0xBA, 0xA8, 0x44, 0xCE, 0x69, 0x70, 0x2C, 0x0C, 0x78, 0xB7, 0x86,
0xA3, 0xF6, 0x23, 0xB7, 0x38, 0xF5, 0xED, 0xF9, 0xAF, 0x83, 0x53, 0x0F, 0xB3, 0xFC, 0x54, 0xFA,
0xA2, 0x1E, 0xB9, 0xCF, 0x13, 0x31, 0xFD, 0x0F, 0x0D, 0xA9, 0x54, 0xF6, 0x87, 0xCB, 0x9E, 0x18,
0x27, 0x96, 0x97, 0x90, 0x0E, 0x53, 0xFB, 0x31, 0x7C, 0x9C, 0xBC, 0xE4, 0x8E, 0x23, 0xD0, 0x53,
0x71, 0xEC, 0xC1, 0x59, 0x51, 0xB8, 0xF3, 0x64, 0x9D, 0x7C, 0xA3, 0x3E, 0xD6, 0x8D, 0xC9, 0x04,
0x7E, 0x82, 0xC9, 0xBA, 0xAD, 0x97, 0x99, 0xD0, 0xD4, 0x58, 0xCB, 0x84, 0x7C, 0xA9, 0xFF, 0xBE,
0x3C, 0x8A, 0x77, 0x52, 0x33, 0x55, 0x7D, 0xDE, 0x13, 0xA8, 0xB1, 0x40, 0x87, 0xCC, 0x1B, 0xC8,
0xF1, 0x0F, 0x6E, 0xCD, 0xD0, 0x83, 0xA9, 0x59, 0xCF, 0xF8, 0x4A, 0x9D, 0x1D, 0x50, 0x75, 0x5E,
0x3E, 0x19, 0x18, 0x18, 0xAF, 0x23, 0xE2, 0x29, 0x35, 0x58, 0x76, 0x6D, 0x2C, 0x07, 0xE2, 0x57,
0x12, 0xB2, 0xCA, 0x0B, 0x53, 0x5E, 0xD8, 0xF6, 0xC5, 0x6C, 0xE7, 0x3D, 0x24, 0xBD, 0xD0, 0x29,
0x17, 0x71, 0x86, 0x1A, 0x54, 0xB4, 0xC2, 0x85, 0xA9, 0xA3, 0xDB, 0x7A, 0xCA, 0x6D, 0x22, 0x4A,
0xEA, 0xCD, 0x62, 0x1D, 0xB9, 0xF2, 0xA2, 0x2E, 0xD1, 0xE9, 0xE1, 0x1D, 0x75, 0xBE, 0xD7, 0xDC,
0x0E, 0xCB, 0x0A, 0x8E, 0x68, 0xA2, 0xFF, 0x12, 0x63, 0x40, 0x8D, 0xC8, 0x08, 0xDF, 0xFD, 0x16,
0x4B, 0x11, 0x67, 0x74, 0xCD, 0x0B, 0x9B, 0x8D, 0x05, 0x41, 0x1E, 0xD6, 0x26, 0x2E, 0x42, 0x9B,
0xA4, 0x95, 0x67, 0x6B, 0x83, 0x98, 0xDB, 0x2F, 0x35, 0xD3, 0xC1, 0xB9, 0xCE, 0xD5, 0x26, 0x36,
0xF2, 0x76, 0x5E, 0x1A, 0x95, 0xCB, 0x7C, 0xA4, 0xC3, 0xDD, 0xAB, 0xDD, 0xBF, 0xF3, 0x82, 0x53
```

Listing 12: Windows Defender RC4 key

All timestamps are uint8 integers in the Windows File Time format (Microsoft, 2018).

2.6.1 ResourceData files

ResourceData files contain the actual quarantined files and the security descriptor describing the access permissions of the file.

The interesting information contained in this file is (names refer to section 2.6.1.1):

- `binarysd`: The binary security descriptor.
- `mal_file`: The malicious file.

2.6.1.1 Parser definition

```
meta:
  id: windef_resource_data
  endian: le
  title: Windows Defender quarantine resourcedata file parser
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: encryptedfile
    type: rc4encrypted
    process: util.custom_arc4.custom_arc4(<RC4 key>)
    size-eos: true
types:
  rc4encrypted:
    seq:
      - id: fixed
        contents: [0x03, 0, 0, 0, 0x02, 0, 0, 0]
        size: 8
      - id: length
        type: u4
      - id: padding
        size: 0x08
      - id: binarysd
        size: length
      - id: unknown1
        size: 0x08
      - id: len_malfile
        type: u8
      - id: unknown2
        size: 0x04
```

```
    - id: mal_file
      size: len_malfile
```

Listing 13: Parser definition for Windows Defender ResourceData files

2.6.2 Resources files

The Resources files consist of three parts – each part encrypted individually using RC4. The first part is the header containing the length of the two data parts.

The first data part contains (names refer to section 2.6.2.1):

- `file_id`: Probably a SHA1 hashsum that also represents the file name of this Resources file and the matching ResourceData file.

The second data part contains:

- `guid`: A GUID that also represents the name of the Entries file that belongs to this Resources file.

2.6.2.1 Parser definition

```
meta:
  id: windef_resources
  endian: le
  title: Windows Defender quarantine resources file parser
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: header
    type: rc4encrypted_header
    size: 0x3C
    process: util.custom_arc4.custom_arc4(<RC4 key>)
  - id: data1
    size: header.len1
    type: encrypted_data1
    process: util.custom_arc4.custom_arc4(<RC4 key>)
  - id: data2
    size: header.len2
    type: encrypted_data2
    process: util.custom_arc4.custom_arc4(<RC4 key>)
types:
  rc4encrypted_header:
```

```
  seq:
    - id: magic
      size: 0x10
      contents: [0xdb, 0xe8, 0xc5, 0x01, 0x01, 0, 0x01, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    - id: unknown1
      size: 0x18
    - id: len1
      type: u4
    - id: len2
      type: u4
    - id: unknown2
      size: 0x0C
encrypted_data1:
  seq:
    - id: unknown3
      size: 0x08
    - id: file_id
      size: 20
    - id: unknown4
      size: 0x04
encrypted_data2:
  seq:
    - id: guid
      type: guid
guid:
  seq:
    - id: id1
      type: u4
    - id: id2
      type: u2
    - id: id3
      type: u2
    - id: id4
      size: 2
    - id: id5
      size: 6
```

Listing 14: Parser definition for Windows Defender Resources files

2.6.3 Entries files

The Entries files consist of three parts – each part encrypted individually using RC4. The first part is the header containing the length of the two data parts.

The first data part contains (names refer to section 2.6.3.1):

- `guid`: The GUID that is also the name of this Entries file.
- `time`: First detection of this malware.
- `id11`, `id12`, `id13`: The first three parts of the GUID above.
- `mal_type`: The name and type of the detected malware.

The second data part contains a list of entries. Each entry describing an occurrence of this specific malware:

- `path`: Location of the detected malicious file.
- `number_of_elements`: Number of elements in the following list of properties of the malicious file.
- `typestr`: Contained `file` in all samples.
- List of elements: Each element contains the size and type of the encoded data and the actual data. Encoded data can be GUIDs, timestamps, UTF-16LE strings, or integers (uint4, uint8).

2.6.3.1 Parser definition

```
meta:
  id: windef_entries
  endian: le
  title: Windows Defender quarantine entries file parser
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: header
    type: rc4encrypted_header
    size: 0x3C
    process: util.custom_arc4.custom_arc4(<RC4 key>)
  - id: data1
    size: header.len1
    type: encrypted_data1
    process: util.custom_arc4.custom_arc4(<RC4 key>)
  - id: data2
    size: header.len2
    type: encrypted_data2
    process: util.custom_arc4.custom_arc4(<RC4 key>)
types:
  rc4encrypted_header:
    seq:
      - id: magic
```

```
      size: 0x10
      contents: [0xdb, 0xe8, 0xc5, 0x01, 0x01, 0, 0x01, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    - id: unknown1
      size: 0x18
    - id: len1
      type: u4
    - id: len2
      type: u4
    - id: unknown2
      size: 0x0C
encrypted_data1:
  seq:
    - id: guid
      type: guid
    - id: unknown3
      size: 0x10
    - id: time
      type: winfiletime
    - id: id11
      type: u4
    - id: id12
      type: u2
    - id: id13
      type: u2
    - id: number_of_strings
      type: u4
    - id: mal_type
      type: str
      encoding: UTF-8
      terminator: 0
encrypted_data2:
  seq:
    - id: number_of_entries
      type: u4
    - id: offset
      type: u4
      repeat: expr
      repeat-expr: number_of_entries
  instances:
    entries:
      pos: offset[0]
      repeat: expr
      repeat-expr: number_of_entries
      type:
```

```
      switch-on: _index != number_of_entries-1
      cases:
        true: entrys(_index)
        false: entrye
  types:
    entrys:
      params:
        - id: i
          type: u4
      seq:
        - id: entry
          size: _parent.offset[i + 1] - _parent.offset[i]
          type: entry
    entrye:
      seq:
        - id: entry
          type: entry
entry:
  seq:
    - id: path
      type: null_terminated_utf16le
    - id: number_of_elements
      type: u2
    - id: typestr
      type: str
      encoding: UTF8
      terminator: 0x00
    - id: padding
      size: (4 - _io.pos) % 4
    - id: element
      type: listelement
      repeat: expr
      repeat-expr: number_of_elements
null_terminated_utf16le:
  seq:
    - id: character
      type: u2
      repeat: until
      repeat-until: character[_index] == 0x0000
listelement:
  seq:
    - id: length
      type: u2
    - id: elementsubtype
```

```
      type: u1
    - id: elementtype
      type: u1
      enum: elementtypes
    - id: content
      size: length
      type:
        switch-on: elementtype
        cases:
          elementtypes::guid: guid
          elementtypes::winfiletime: winfiletime
          elementtypes::uint4: u4
          elementtypes::utf16: str_utf16le
          elementtypes::uint8: u8
    - id: padding
      size: (4 - _io.pos) % 4
  enums:
    elementtypes:
      0x20: utf16
      0x30: uint4
      0x40: guid
      0x50: uint8
      0x60: winfiletime
str_utf16le:
  seq:
    - id: value
      type: str
      encoding: UTF-16LE
      size-eos: true
guid:
  seq:
    - id: id1
      type: u4
    - id: id2
      type: u2
    - id: id3
      type: u2
    - id: id4
      size: 2
    - id: id5
      size: 6
winfiletime:
  seq:
    - id: ts
```

```
      type: u8
  instances:
    unixts:
      value: (ts * 1e-07) - 11644473600
```

Listing 15: Parser definition for Windows Defender Entries files

2.7 Symantec Endpoint Protection

The samples for Symantec Endpoint Protection were taken from an incident analysis. It was not possible to recover the original malware files from the affected system.

Symantec stores quarantine data in two different files in the `Quarantine` directory. Per quarantined malware file, there is a file that contains only metadata (`<meta>.VBN`, section 2.7.2) and a file that contains the quarantined file and metadata (`<meta>/<quarantine>.VBN`, section 2.7.1). Both file names follow the schema `[0-9A-F]{8}.VBN`. The `Quarantine` directory is located at:
`/ProgramData/Symantec/Symantec Endpoint Protection/CurrentVersion/Data/Quarantine`

2.7.1 Quarantine file

This file stores a lot of metadata – one datum that cannot be found here is the name of the detected malware –, some of it multiple times in different locations of the file. Because of this, only the most relevant information for analyses are highlighted here (names refer to section 2.7.1.1):

- `encrypted_offset`: An offset to an encrypted payload of the file, which was `0x1290` in all samples.
- `filename`: The file name of the original malicious file.
- `meta1`: A CSV-string that contains the most important metadata of the file (Maloney, 2018, section "Log Line").
- `unixts1`: A Unix timestamp which is a date in the future (relative to the quarantine file creation). Maybe this is the date when Symantec Endpoint Protection will permanently remove the quarantined file.
- `timestamp[1-3]`: Three Windows File Time (Microsoft, 2018) timestamps.
- `enc_data`: The encrypted payload. This part is encrypted with a single-byte xor key (`0x5A`).

The encrypted payload is divided into two parts (`meta_entries` and `content_entries`). Both parts contain a list of entries. An entry consists of an identifier, optionally a length field if data is encoded with variable length, and the actual encoded data:

- ID `0x01`: uint1. No length field.
- ID `0x03`: uint4. No length field.
- ID `0x04`: uint8. No length field. Is only used to store the file length of the quarantined file.
- ID `0x06`: uint4. No length field.

- ID `0x08`: Length is encoded as uint4. Contains an UTF-16LE string.
- ID `0x09`: Length is encoded as uint4. Contains a raw data stream, which itself can contain a list of entries, a checksum, or chunks of the quarantined file.
- ID `0x0a`: uint1. No length field.

While the `meta_entries` contain data like the location and the owner of the malware, the `content_entries` contain the SHA1 checksum, the security descriptor and the actual malicious file content. The file content can be found by looking for the entry with ID `0x04`, which contains the file length – all following entries with ID `0x09` will contain chunks of the malware, xored with `0xFF`. The maximum chunk size seems to be `0x1000` bytes.

2.7.1.1 Parser definition

```
meta:
  id: vbn
  file-extension: vbn
  endian: le
  title: Symantec Endpoint Protection quarantine file parser
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: encrypted_offset
    type: u4
  - id: filename
    type: str
    encoding: utf-8
    terminator: 0x0
  - id: padding1
    size: 0x180 - _io.pos
  - id: padding2
    type: u4
  - id: meta1
    type: str
    encoding: utf-8
    terminator: 0x0
  - id: meta2
    type: str
    encoding: utf-8
    terminator: 0x0
  - id: padding3
    size: 0x980 - _io.pos
```

```
  - id: padding4
    type: u4
  - id: unknown1
    type: u4
  - id: unixts1
    type: u4
  - id: timestamp1
    type: winfiletime
  - id: timestamp2
    type: winfiletime
  - id: timestamp3
    type: winfiletime
  - id: unknown2
    type: u4
  - id: padding5
    size: 0xb8c - _io.pos
  - id: threat_location
    type: str
    encoding: utf-8
    terminator: 0x0
  - id: padding6
    size: 0xbbc - _io.pos
  - id: unknown3
    type: u4
  - id: tmp_file_name
    type: str
    encoding: utf-8
    terminator: 0x0
  - id: padding7
    size: 0xd70 - _io.pos
  - id: unixts2
    type: u4
  - id: padding8
    size: encrypted_offset - _io.pos
  - id: enc_data
    type: encrypted_data
    size-eos: true
    process: xor(0x5a)
types:
  encrypted_data:
    seq:
      - id: padding
        size: 8
        contents: [0, 0, 0, 0, 0, 0, 0, 0]
```

```
    - id: meta_offset
      type: u8
    - id: meta_length
      type: u8
    - id: content_offset
      type: u8
    - id: content_length
      type: u8
  instances:
    meta_entries:
      pos: meta_offset
      size: meta_length
      type: list_of_entries
    content_entries:
      pos: content_offset
      size: content_length
      type: list_of_entries
list_of_entries:
  seq:
    - id: entry
      type: entry
      repeat: eos
entry:
  seq:
    - id: type_of_entry
      type: u1
    - id: content
      type:
        switch-on: type_of_entry
        cases:
          0x09: raw_content
          0x08: utf16le
          0x04: u8
          0x03: u4
          0x06: u4
          0x01: u1
          0x0a: u1
raw_content:
  seq:
    - id: length
      type: u4
    - id: raw_content
      size: length
utf16le:
```

```
  seq:
    - id: length
      type: u4
    - id: string_content
      type: str
      encoding: utf-16le
      size: 'length > 2 ? length - 2 : length'
    - id: padding
      size: 2
      contents: [0x00, 0x00]
      if: 'length >= 2'
winfiletime:
  # timestamp: timestamp * (1e-07) --> seconds
  # offset: 11644473600
  seq:
    - id: ts
      type: u8
  instances:
    unixts:
      value: (ts * 1e-07) - 11644473600
```

Listing 16: Parser definition for Symantec Endpoint Protection quarantine files

2.7.2 Metadata file

The metadata file has a structure quite similar to the quarantine file (section 2.7.1). It contains a lot of data, which is already included in the quarantine file. It contains the name of the detected malware, which is not part of the quarantine file. However, the payload is not encrypted. The following names refer to section 2.7.2.1:

- `data_offset`: The offset to the (not encrypted) payload, which was `0x1290` in all samples.
- `filename`: The location of the malware.
- `meta1`: A CSV-string that contains the most important metadata of the file (Maloney, 2018, section "Log Line"). The malware name can be found here.
- `data`: The payload.

The payload is a single list of entries which is similar to the entries in the `meta_entries` of the quarantine file.

2.7.2.1 Parser definition

```
meta:
  id: vbnmeta
  file-extension: vbn
  endian: le
  title: Symantec Endpoint Protection quarantine metadata file parser
```

```
  license: CC-BY-SA-4.0
  ks-version: 0.9
doc: |
  Creator: Florian Bausch, ERNW Research GmbH, https://ernw-research.de
  License: CC-BY-SA-4.0 https://creativecommons.org/licenses/by-sa/4.0/
seq:
  - id: data_offset
    type: u4
  - id: filename
    type: str
    encoding: utf-8
    terminator: 0x0
  - id: padding1
    size: 0x180 - _io.pos
  - id: padding2
    type: u4
  - id: meta1
    type: str
    encoding: utf-8
    terminator: 0x0
  - id: padding6
    size: data_offset - _io.pos
  - id: data
    type: data
    size-eos: true
types:
  data:
    seq:
      - id: meta_entries
        size-eos: true
        type: list_of_entries
  list_of_entries:
    seq:
      - id: entry
        type: entry
        repeat: eos
  entry:
    seq:
      - id: type_of_entry
        type: u1
      - id: content
        type:
          switch-on: type_of_entry
          cases:
```

```
        0x09: raw_content
        0x08: utf16le
        0x04: u8
        0x03: u4
        0x06: u4
        0x01: u1
        0x0a: u1
raw_content:
  seq:
    - id: length
      type: u4
    - id: raw_content
      size: length
utf16le:
  seq:
    - id: length
      type: u4
    - id: string_content
      type: str
      encoding: utf-16le
      size: 'length > 2 ? length - 2 : length'
    - id: padding
      size: 2
      contents: [0x00, 0x00]
      if: 'length >= 2'
winfiletime:
  # timestamp: timestamp * (1e-07) --> seconds
  # offset: 11644473600
  seq:
    - id: ts
      type: u8
  instances:
    unixts:
      value: (ts * 1e-07) - 11644473600
```

Listing 17: Parser definition for Symantec Endpoint Protection additional metadata files

ERNW RESEARCH pursuing knowledge.

3 Summary & Conclusion

This whitepaper shows the structure of quarantine files of seven different anti-virus software solutions. The findings are also documented at *https://github.com/ernw/quarantine-formats/*. Part of the findings are encryption keys and information what data needs to be decrypted. Furthermore, the location of timestamps, malware names, malware location and checksums is documented – this can be helpful during incident analyses to restore samples for malware analysis and identify infection timeframes.

The provided parser definition files can be used to create automated analysis tools for quarantine file analysis. Such a tool is not part of this whitepaper or the provided Github project, but can be subject of future work.

While the creation of the documentation and parser definitions was performed with care, the limited number of samples, however, may have led to an incomplete understanding of the file formats. The purpose of several fields and parts of different formats are still unknown and therefore named as `unknown<X>`.

Future incident analysis cases at ERNW Research GmbH will be used to validate and improve the findings. Additionally, the encryption mechanism of the Sophos Antivirus quarantine files still needs to be understood. Furthermore, future work will be required to analyze files of anti-virus software that was not yet part of the research work.

Participation in this project by providing samples or improving parser definitions and documentation is welcomed.

A RC4 Helper Class in Python

```
#!/usr/bin/python3



class CustomArc4(object):
    def __init__(self, key):
        state = [n for n in range(256)]
        p = q = j = 0
        for i in range(256):
            if len(key) > 0:
                j = (j + state[i] + key[i % len(key)]) % 256
            else:
                j = (j + state[i]) % 256
            state[i], state[j] = state[j], state[i]
        self.p = p
        self.q = q
        self.state = state

    def _next_byte(self):
        self.p = (self.p + 1) % 256
        self.q = (self.q + self.state[self.p]) % 256
        self.state[self.p], self.state[self.q] =\
            self.state[self.q], self.state[self.p]
        return self.state[(self.state[self.p] + self.state[self.q]) % 256]

    def decode(self, plaintext):
        return bytearray([c ^ self._next_byte() for c in plaintext])
```

Listing 18: Helper class for RC4

ERNW RESEARCH
pursuing knowledge.

B References

Carrier, B. (2009). Body file – SleuthKitWiki [https://wiki.sleuthkit.org/index.php?title=Body_file&oldid=400, accessed April 16th, 2020.].

conix-security. (2014). VBNExtract/extractVBN.c [https://github.com/conix-security/VBNExtract/blob/56793a3d9b860a374adb172528841f6dd68325d0/extractVBN.c, accessed April 16th, 2020.]. *Conix Security*.

EICAR. (2006). EICAR – Intended Use [http://2016.eicar.org/86-0-Intended-use.html, accessed April 16th, 2020.]. *EICAR*.

Glass, J. (2015). What happens when Windows Defender Quarantines Stuff... [http://jon.glass/blog/quarantines-junk/, accessed April 16th, 2020.]. *Half Full of Security*.

Grant, A. (2019a). DeXRAY 2.17 update [http://www.hexacorn.com/blog/2019/11/09/dexray-2-17-update/, accessed April 16th, 2020.]. *Hexacorn*.

Grant, A. (2019b). dexray.pl [http://hexacorn.com/d/DeXRAY.pl, accessed April 16th, 2020.]. *Hexacorn*.

Maloney, B. (2018). Symantec Endpoint Protection VBN files [https://malwaremaloney.blogspot.com/2018/03/symantec-endpoint-protection-vbn-files.html, accessed May 17th, 2020.]. *MALoney*.

Microsoft. (2018). File Times [https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times, accessed April 16th, 2020.]. *Microsoft*.

Shantz, MG. (2016). Recovering Malware from a Quarantine Folder [https://forensicdatasolutions.com/apps/blog/show/44156381-recovering-malware-from-a-quarantine-folder, accessed April 16th, 2020.]. *Forensic Data Solutions*.

Sprengler, B. (2015). quarantine.py [https://github.com/brad-sp/cuckoo-modified/blob/master/lib/cuckoo/common/quarantine.py, accessed April 16th, 2020.]. *brad-sp*.

Unicode. (2020). FAQ - UTF-8, UTF-16, UTF-32 & BOM [https://www.unicode.org/faq/utf_bom.html, accessed May 15th, 2020.]. *Unicode*.