Creating Polyglot Files

Matus Mrekaj

12 Dec 2022

Exploring the creation of polyglot files that are still valid with the latest software, tested on macOS and Linux. Defining several techniques for creating polyglot files, namely Zippers, Concatanation, Parasite, Cavities. A explanation of how to use the file formats MP3, WASM, PDF, ZIP, PNG, JPEG, GIF, ISO and also out of interest NES ROM (Nintento) to create polyglot files that could be used to hide malicious code inside the file and that would still be considered valid by the various programs that interpret them. In addition, we also mention things that no longer work.

1 A Polyglot File

A file is a resource identified by its name, the contents of which are an array of bytes[3]. A file is essentially a sequence of bytes that has no meaning in itself until it is interpreted by a program that recognizes and can interpret that sequence of bytes. A polyglot file is a sequence of bytes that, when given to different programs, each of them interprets it differently. To give an example, consider a sequence of bytes representing PDF and ZIP files. When a PDF is given to a reader, it looks for bytes from which it can assemble the PDF. When given to a ZIP parser it looks for bytes that made up a ZIP. A polyglot file can be crafted such that it can be interpreted by multiple programs as valid. We'll explore creating such files in later sections.

But, how does a program know that it's reading a ZIP or a PDF? There are a number of file format specifications that have been around since the inception of the computer. The program looks at a sequence of bytes and determines the type of file format not based on the file's suffix, but on the byte header, know as a $MagicNumber$ [1] [14]. For example, on MacOS the default app to view file types is Preview. It supports a number of image file formats as well as PDF [15]. Since it supports multiple file formats it looks for the $MagicNumber$ based on the suffix of the file that is requeted to be opened. Given a PDF is will look for the %PDF sequence of bytes that identifies a PDF, for a PNG it will look for 89 50 $4e$ 47 $0d$ $0a$ $1a$ $0a$. It will then read the contents until an EOF marker that's again specific for the given file format specification.

1.1 Why is it possible to create Polyglot Files?

File format specifications are created by individuals or companies; there is no standard committee that

[1] A constant value used to identify file type `https://en.wikipedia.org/wiki/Magic_number_(programming)`

approves the file format [4]. Therefore, the specifications are incomplete and there are ways of using them in ways that have not been considered at all. These will be discussed later when explaining the various file formats.

2 Creating Polyglot Files

By having the possibility of creating polyglot files that act as one format where in reality it's a different format, play a important role for possibly bypassing the security measures of systems, propagating malware, or generally hid malicious code within a file that could be left undetected.

You may be asking how could this be a potential threat, the answer to it is: it depends. A system is secure as its weakest link and many systems aren't build for security from the start, it only becomes a priority much later or worse when a breach is detected. To give a some real examples that actually happened very recently, Rockstar Games and Uber both got hacked within a time span of 1 week of each other by social engineering [16, 17].

Distributing a polyglot file containing malicious code could be just the first step, followed by execution through social engineering or other means that are disposable. Polyglot files just enable another point of attack on the system because the end users do not know the contents of the file or do not care about it, they just want to open it, I am guilty of this myself.

3 Techniques to create Polyglot Files

The techniques that I've used to created the polyglot files were inspired from parts of the presentation given by Ange Albertini where he introduced 4 types of polyglot file creation namely, Cavities, Concatanation (or stacking), Parasites and Zippers[4].

3.1 Concatanation, Stacks

Concatanation is the simplest form of creating a polyglot, it's simply appending the the second file at the end of the first one. Any offsets or pointers or references in the contents of the second file must be adjusted to new positions, this first file is left untouched. The requirment for this is that the second file doesn't enforce *Magic Number* at offset 0 of its contents. As we have talked that file format specifications are not perfect, there exist a few of formats that allow this.

3.2 Cavities

Cavities are a bit harder to create as the idea of it is that in order to create a Polyglot file we use padded, unused zero-filled memory within the first file. Usually this is the case for executables, such as ELF, Mach-O or ROMs or ISOs. It can happen that there exists enough zero-filled static memory that could fit a whole another file and again any offsets pointers or reference within the second file would need to be adjusted [5].

3.3 Zippers

Zippers are the more complicated way of creating Polyglots out of the other options. The idea is that the two files are sliced up into pieces and merged together by making use of the Metadata or Comment blocks of both file formats. One of the files must have the requirment that the header can start later than

at offset 0. In both files the references pointers and offsets would need to be adjusted [5].

3.4 Parasites

Parasites are very similar to Cavities, but instead of using zero-filled unused memory we abuse the spec of the file format by hiding the second file within the first file, by using Metadata or Comments tags that are allowed by the specifications. These block or tags aren't displayed to the user when the file is being read. Usually they are limited in size which varries across the file formats.

4 Explored File Formats

The following file formats have been explored to create Polyglot files.

- WASM - Requires offset at byte 0
- ZIP - Doesn't required offset at byte 0
- PNG - Requires offset at byte 0
- JPEG - Required offset at byte 0
- PDF - Doesn't required offset at byte 0
- GIF - Requires offset at byte 0
- ISO - Doesn't required offset at byte 0
- NES - Requires offset at byte 0

We'll now describe how each of them works and how can we make use of the format specification to hide other files within it. We won't go deeply into the details just enough so that we can create Polyglot Files.

4.1 PDF

The PDF specification is very verbose and long over 700 pages [11]. We don't need to go through the whole specification in order to make use of its features for our benenfit. The following image describes a minimal PDF that has the core components of a PDF.

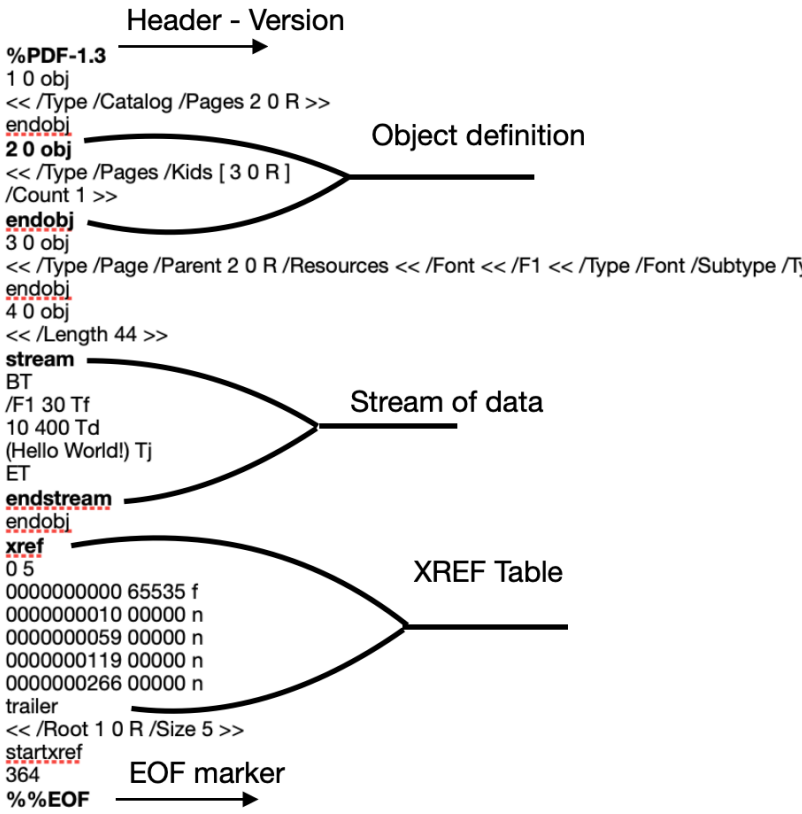

Figure 1: minimal PDF

A PDF is made up of objects that are referenced in the XREF table that is present at the end of the PDF, if it's there, there is not requirment on having the XREF table. The XREF table contains the offset, of each of the defined objects, in bytes from the start

of the file. An object has definition for Fonts, Page layout, contents etc... We don't need to get to deep into this what we only care about the **Stream** part within the object.

A **Stream** is an arbitrary sequence of bytes of any length, that could be encoded/decoded or compressed [12]. The key word for us is that it can contain any data of any length. This is exactly what we need to create Polyglots files. Anything can be hidden within a stream in a PDF.

In the specification of PDF it actually states that the header offset is required at offset 0, however. In practice this is not the rule but rather the exception. Thus PDF files can be mixed with other file formats that require offset at 0.

4.2 ZIP

ZIP is the second format that doesn't require offset at byte 0. This is purely due to historical reasons that was to minimize the number of floppy disks swaps as writing to disks was at that time when the specification was created very time consuming. ZIP are actually parsed bottom-up. To have a valid ZIP file we need to have a **Central Directory** that contains the name of the files with metadata and offsets that are relative to the central directory [19].

Each entry that the central directory refers to starts with a **Local File Header** that contains the size, name, comments etc. Followed by this is the compressed/uncompressed encrypted/unencrypted data.

There are two ways for creating Polyglots with ZIP files, injecting ZIP within other files such as PDF, or inject other files into the ZIP.

For the former option you would expect that the **Central Directory** needs to be the last sequence of

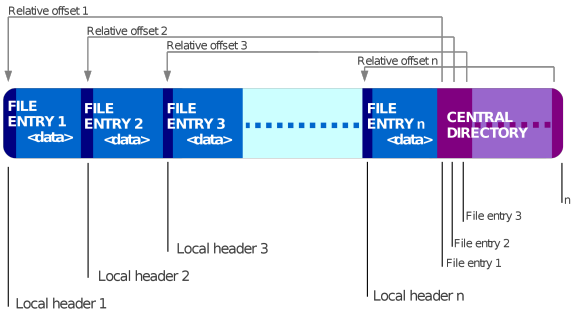

Figure 2: ZIP Layout[19]

bytes in the file. This is not the case, it can be at any position, as long as it's not too far from the end of it as in this case the parsers would give up finding the central directory. There exists a workaround that would allow to have the the ZIP at any location by just making a copy of the central directory and storing it somewhere near the end [1].

The latter makes use of the uncompress/unencrypted data, just storing the raw file within the archive.

4.3 PNG

A PNG starts with the magic number followed by a sequence of chunks and an EOF chunk. The individual chunks describe its contents. There are two types of chunks critical and anciliary. Critical chunks always start with a capital letters, whereas anciliary start with lower-case letters. The layout of a single chunk is constructed by concatinating the following pieces, the length of the chunk, type identifier, the data itself and the crc32 checksum of the whole chunk.

The first chunk following after the magic number must be the IHDR chunk which contains information about the encoded image. After any chunk sequence

can follow. The data for the image is encoded in IDAT chunks.

A minimal PNG is visualized in the following figure.

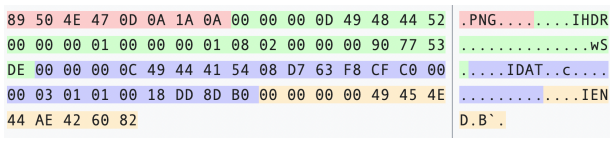

Figure 3: PNG Layout[13]

To create a Polyglot file we make use of the anciliary chunks. When a PNG decoder encounters an anciliary chunk that it doesn't recognize it can safely skip over it and continue reading the image data. We can make use of this to hide other files within a PNG file format. Interestingly we can even make up our own chunk identifier as the decoder with just ignore it, names such as *aaaa* or *bbbb* are valid.

4.4 JPEG

JPEG a bit more complicated compared to PNG but it uses very similar concepts. JPEG start with a magic number followed by segments and ends with an EOF marker segment. The layout of the segment is actually very different from PNGs. A segment is recognized by enclosing the data with marker bytes. The more common segments are listed in the table to the right.

To create a Polyglot File we simply make use of the comment segment, similarly as with PNG, and enclose our data within the **xFF xFE** bytes.

Short name	Bytes	Payload	Name
SOI	0xFF, 0xD8	*none*	Start Of Image
SOF0	0xFF, 0xC0	*variable size*	Start Of Frame (baseline DCT)
SOF2	0xFF, 0xC2	*variable size*	Start Of Frame (progressive DCT)
DHT	0xFF, 0xC4	*variable size*	Define Huffman Table(s)
DQT	0xFF, 0xDB	*variable size*	Define Quantization Table(s)
DRI	0xFF, 0xDD	4 bytes	Define Restart Interval
SOS	0xFF, 0xDA	*variable size*	Start Of Scan
RST*n*	0xFF, 0xD*n* (*n*=0..7)	*none*	Restart
APP*n*	0xFF, 0xE*n*	*variable size*	Application-specific
COM	0xFF, 0xFE	*variable size*	Comment
EOI	0xFF, 0xD9	*none*	End Of Image

Figure 4: JPEG segments[8]

4.5 GIF

GIF is another image format which we can use to hide other files in. The pattern is same, there is a Header, data describing the GIF and an EOF marker. Without going much into the details here there also is a comment section in GIF that allows us to store arbitrary data. The catch here is that it's limited in size. The maximum amount of data a comment can store is 255 bytes [6]

The comment section in the GIF can be inserted between any two segments however the start of it has to be strictly after the Header, logical screen descriptor and the global color table, if there is one present. The comment section is denoted by bytes $0x21$, $0xFE$, followed by the length of the data, the

data itself and terminated with $0x0$

4.6 MP3

The MP3 file format doesn't have something simlar to comments, it however can be used in conjuction with ID3 metadata container. ID3 is used for information such as the artist name, album etc. What's interesting for us with this is that this metadata container has support for comments section. There are two version for ID3, v1 and v2 both can be present for a single mp3 file. Version 1 is roughly 127 bytes long and is always the last 127 bytes of the MP3 file. For version 2 the position is at the start of the MP3 file and there are no limits regarding the size[9].

To create a Polyglot file we need to add a comment section for the ID3v2 metadata. The comment section layout structure is made up out of the encoding, language description and text as depicted by the following snippet which is a high-level description.

```
id3v2.CommentFrame{
     Encoding:    id3v2.EncodingUTF8,
     Language:    "eng",
     Description: "eng",
     Text:        hidden_file_bytes,
}
```

for our use-case the values of the language or description doesn't matter nor the encoding we only care about the data we can hid inside the comment section.

4.7 WASM

WASM is a very recent format that was released in 2017 and actually has some standardization, yet we can still use the specification for our benefit. WASM starts with a Header $0asm$ and 4 bytes for a version number of the binary, after wihch an arbitrary sequence of sections can follow. Each section has its own identification code ranging from 0 to 11. A section layout is made out of the identification code, the length of the section encoded using LEB128 and the data for the section itself. One particular section is intersting and that is the custom section that's identified with 0x0[18].

This section can be used for storing arbitrary information. For us this means that we can create Polyglot files by storing other files inside the custom sections the length of the custom section is variable in size as long as the size doesn't exceeds WASM uint32 which is 4 bytes, giving us a lot more freedom of in terms of the hidden data.

4.8 ISO

ISO format is very different from all of the just mentioned formats. ISO is a disk image that contains all the necessary things that would be needed when written to an optical disk. There are no comments or metadata that we can make use of. There also is no Header that's required at offset 0 nor a EOF marker, however. There is a magic number that is present at byte 32769 [7].

We can use unused zero-fill memory to create polyglot files from ISO formats, as described in Section 3.2 for cavities. Since the magic number is placed at position 32kb there usually is some unused space at the beginning that we can make use of; this is also true for the end of the ISO format. Unfortunutely I couldn't find a pattern that would always work and just tried this with trial and error until it worked.

4.9 NES

For NES ROM files there are no comments/metadata either so the only way to create a Polyglot is using cavities i.e. directly injected the file into padded/unused static memory or we can use a Trainer that is part of the NES ROM. Trainer is 512 bytes of memory that most games do not use and it was sometimes used to inject cheat codes. For our purpose we could use this to store other files that are smaller than 512 bytes [10]. This is a bit harder to do as most emulators that I've tried didn't accept trainers as valid but it's still an option that exists. Trainer follows directly after the Header.

```
Header (16 bytes)
Trainer, if present (0 or 512 bytes)
...
```

To enable the Trainer we need to set the 3rd bit of the 6th byte from the header and then we need to add 512 bytes, if there not present yet, directly following after the header.

5 Polyglot files via Encryption

Using the same techniques, either using comments, metadata, or custom sections, it is also possible to hide files within other files, but instead of storing the raw contents of the hidden file, we can transform the contents of the hidden file using other means, such as encryption.

To demonstrate this we'll create a PDF that when decrypted using CBC mode with the block cipher AES with the right key and IV it will convert to a valid WASM binary.

CBC mode when encrypting XORs the current block with the previous block, in case its the first block it uses the IV, and then proceeds to encrypt the result under the given key. The decryption is the exact opposite, we take the ciphertext block decrypt it under the key and then XOR the previous block, in case of the first block the IV is used, the process is depicted by the following two pictures.

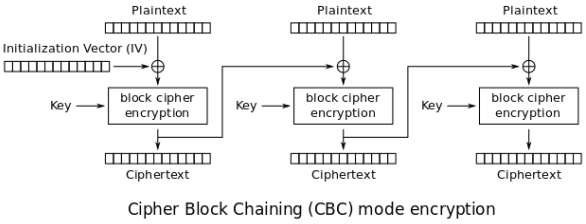

Figure 5: CBC encryption[2]

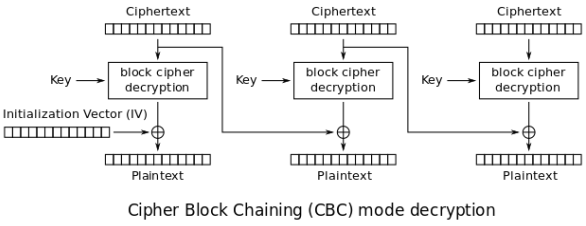

Figure 6: CBC decryption[2]

We can't really do much with controlling the output as changing only a single byte results in a completely different sequence of bytes. What we can do here is control the first block and the IV.

We need to create an IV that when used with a CBC block mode cipher, it decrypts a PDF header into parts of valid WASM binary. We can actually construct such IV with just the first block of the WASM binary, and the first block of the PDF header. We create a minimal valid PDF header %PDF-1obj\nstream, this will get recognized by most PDF reader as valid. We decrypt this header using AES-

CBC under a given key and XOR the result with the first 16 bytes of the WASM binary, this will give us the required IV, that will also will look random.

What we will do then is used this IV and the same key and encrypt the whole WASM binary. The encryption would result into the given PDF header followed by random data that would be hidden inside an unreferenced PDF object. We need to then append the end of the unreferenced object \nendstream\nendobj\n followed by the rest of the valid PDF conents. This will result in a file that is depicted in the image bellow.

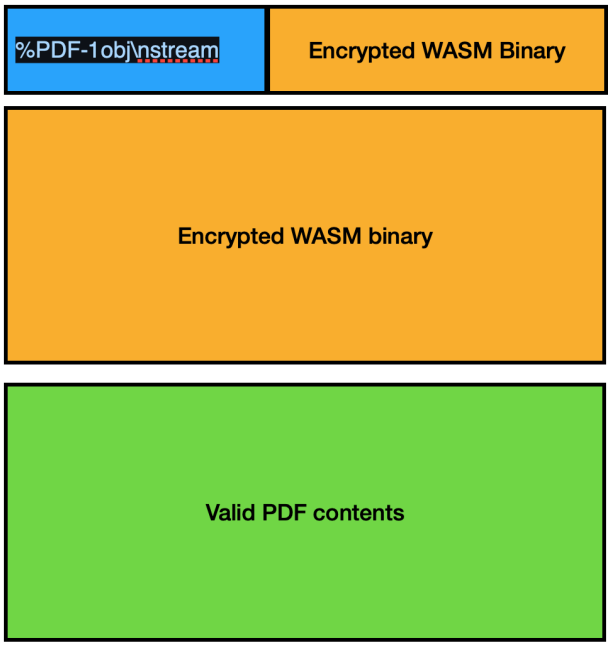

Figure 7: Encryption Polyglot

The WASM binary has to follow certain rules to make this work. The last few bytes of the WASM binary needs to define a custom section with the length of the appended PDF contents encoded as LEB128. This is needed to make it so when we decrypt the file it will be a valid WASM binary as decryption yields random data for the PDF contents and we don't care about them within the WASM binary so we wrap them inside a custom section.

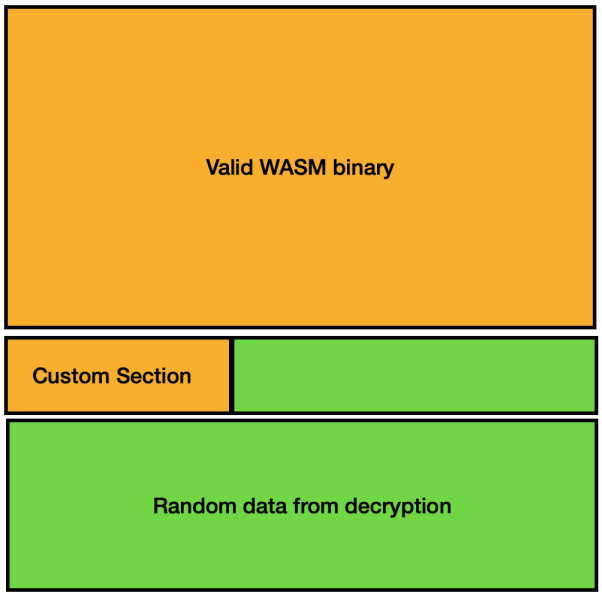

Figure 8: Decrypting Polyglot

This is also possible with other file formats that have metadata or comment sections that we can use to hide data, such as PNG.

6 Conclusion

I've tried creating multiple Polyglot files that still work today with the latest software. There are some catches though. Adobe Acrobat Reader (which directly follows the spec of PDF) doesn't consider the PDF polyglots as valid anymore and has been improved over the last decade to disallow opening polyglot files within it. However other PDF viewers such as Preview on MacOS or Document Viewer on linux have no problem with these polyglots whatsoever in-

cluding browsers such as Safari or Firefox. The same is true for mobile devices such as Iphone or Remarkable Tablet.

Creating Polyglots files that work on MacOs is a bit more harder to do. Especially if you want to create a polyglot with a binary of some sort as MacOs requires the binaries to be signed and its a bit harder to distribute polyglots with self-signed certificates, it's doabble just harder. Also MacOS has some protection for Zip files, it requires that nothing starts before the ZIP file itself, so if you just try to insert a single byte it will just straight reject it, however if you use the command-line to unzip a polyglot is still works.

Further, Google Chrome has also protection agains opening polyglots that contain some sort of binary data. I've tried this and chrome just rejects them.

I haven't encountered any issues with polyglots on Linux. I'm not sure if they'll work on windows as I haven't targeted windows as the primary OS, though I believe some of them if not most would also work.

References

[1] *AngeCryption*. URL: https://www.youtube.com/watch?v=wbHkVZfCNuE (visited on 12/09/2022).

[2] *Cipher block modes*. URL: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation (visited on 12/11/2022).

[3] *Computer File*. URL: https://en.wikipedia.org/wiki/Computer_file (visited on 12/07/2022).

[4] *Funky File Formats*. URL: https://www.youtube.com/watch?v=hdCs6bPM4is (visited on 12/07/2022).

[5] *Generating wierd files*. URL: https://www.youtube.com/watch?v=1VHUwzKAKQI&t=1248s (visited on 12/08/2022).

[6] *GIF*. URL: https://en.wikipedia.org/wiki/GIF#:~:text=The%20format%20supports%20up%20to,256%20colors%20for%20each%20frame. (visited on 12/10/2022).

[7] *ISO*. URL: https://en.wikipedia.org/wiki/Optical_disc_image (visited on 12/11/2022).

[8] *JPEG*. URL: https://en.wikipedia.org/wiki/JPEG (visited on 12/10/2022).

[9] *MP3*. URL: https://en.wikipedia.org/wiki/ID3 (visited on 12/11/2022).

[10] *NES*. URL: https://www.nesdev.org/wiki/INES (visited on 12/11/2022).

[11] *PDF specification*. URL: https://opensource.adobe.com/dc-acrobat-sdk-docs/standards/pdfstandards/pdf/PDF32000_2008.pdf (visited on 12/08/2022).

[12] *PDF Stream*. URL: https://www.oreilly.com/library/view/developing-with-pdf/9781449327903/ch01.html (visited on 12/08/2022).

[13] *Png*. URL: https://en.wikipedia.org/wiki/Portable_Network_Graphics (visited on 12/10/2022).

[14] *Polyglot*. URL: https://en.wikipedia.org/wiki/Polyglot_(computing)#:~:text=In%20computing%2C%20a%20polyglot%20is,two%20or%20more%20different%20formats. (visited on 12/07/2022).

[15] *Preview*. URL: https://en.wikipedia.org/wiki/Preview_(macOS) (visited on 12/07/2022).

[16] *Rockstar Hack*. URL: https://www.theguardian.com/games/2022/sep/19/grand-theft-auto-6-leak-who-hacked-rockstar-and-what-was-stolen (visited on 12/07/2022).

[17] *Uber Hack*. URL: https://www.forbes.com/sites/daveywinder/2022/09/18/has-uber-been-hacked-company-investigates-cybersecurity-incident-as-law-enforcement-alerted/?sh=763cd7bd6056 (visited on 12/07/2022).

[18] *WASM*. URL: https://webassembly.github.io/spec/core/index.html (visited on 12/11/2022).

[19] *ZIP*. URL: https://en.wikipedia.org/wiki/ZIP_%28file_format%29 (visited on 12/09/2022).