The Blockchain Potential in Computer Virology

Joanna Moubarak[1], Eric Filiol[2,3] and Maroun Chamoun[1]
[1]*Faculty of Engineering, USJ, Lebanon*
[2]*Department of Cybersecurity, ENSIBS, Vannes, France*
[3]*High School of Economics, Moscow, Federation of Russia*

Keywords: Armoured Malware, Blokchain, k-ary Codes, DLT, IPFS, Computer Virology.

Abstract: This paper delves into the state of the art of computer virology formalisation then tackles the development of a new malware algorithm. It details how the work leveraged Blockchain to create an undetectable malware depicting two versions of the new malware, starting from a first naive version to achieve an advanced armoured undetectable k-ary malware that leverages decentralized storage namely IPFS. The detection of the new malware algorithm has been proven NP-complete.

1 INTRODUCTION

For decades, researchers have been intrigued by cyber-threats techniques and contributed in defending against those attacks (Moubarak et al., 2017a). According to the theory of computability, some problems are not calculable and the problem of viral detection is one of the undecidable problems.

The prominent formalizations in computer virology instigates from Turing machines (Shannon, 1956) which constitute a theoretical model of any computer program, viral or not. Fred Cohen (Cohen, 1985) has shown that the general problem of viral detection is an undecidable problem. Other works (Filiol, 2006a)(Filiol, 2007d) have shown that the detection of certain classes of malware was NP complete. In this paper, combinatorial techniques of k-ary malware (Filiol, 2007b) were studied to create a k-ary decentralized malware. The goal is to divide the malicious code into several parts (active or not), each of them having no malicious character. It is the interactions between these parts (in series or in parallel) that determine the final maliciousness. A new k-ary model was proposed leveraging the blockchain technology which is mainly a peer to peer network of distributed nodes revolving around cryptographic schemes and a consensus algorithm. The results demonstrate the feasibility of a new type of malware based on IPFS showing that the cyberattack landscape is remodeling novel set of rules. The detection of the malware described is an undecidable problem. The remaining of this paper is organized as follows: Section 2 introduces the preliminary works in computer virology in general and Section 3 presents the k-ary concept. Section 4 overviews Distributed Ledger Technologies and Section 5 highlights the decentralized k-ary malware. Section 6 concludes this paper and presents our future works.

2 PRELIMINARY WORKS

In 1936, Kleene's work (Kleene, 1936) dealt with the formalization of mathematical theorems. However, his work on self-reproduction logic which is an integral characteristic of viruses, made him one of the founder of the computer virology theory. More precisely, the recursion theorems (Kleene, 1938) formalize quine programs that construct a transforming function from a fixed point or environment. Given a function f, there are different programs needed to its computation. If it is the identity function, the codes are identical and consequently, we have the concept of virus and the self-reproduction theorem.

In 1948, John von Neumann's theory on biological evolution specifically the self-reproduction of cellular automata was born. Neumann has developed his automaton around two central elements based on Turing machines: a universal computer and a universal constructor.

In 1980, Kraus defended his thesis on computer virology (Kraus, 1980). His work tackled the relation between computer algorithms and biological self-reproduction behaviors. His model theorized the sim-

Moubarak, J., Filiol, E. and Chamoun, M.
The Blockchain Potential in Computer Virology.
DOI: 10.5220/0009371607960803
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 796-803
ISBN: 978-989-758-399-5; ISSN: 2184-4356

ilarities between the operations' vectors involved in computer and biological viruses.

Fred Cohen defended his PhD. thesis in 1986 defining and formalizing viruses. He is the first to have given a formal definition to the concept of computer virus. Cohen models a computer virus with a set of viral codes propagating on the ribbon of a Turing machine. Cohen formalized Turing Machines focusing deeply on the time feature involved in computer mechanisms (Cohen, 1987b). He studied also the undecidability problem related to viral detection. Fred Cohen was technically the first scientist authorized to conduct experiments to validate his formalization results.

Cohen was interested in systems free of any virus. His work shows the existence of a machine that does not admit any viral set: the machine that does nothing. This observation and his experiments lead to state the adage that only an isolated system is protected from computer infections.

Fred Cohen demonstrated that the general problem of viral detection is an undecidable problem. *There is no Turing machine M able to decide in finite time if a given sequence v is a virus.*

The principle of the demonstration consists in reducing the problem of decidability of the viral set to the halt problem. His advisor Leonard Adleman gave complexity (Papadimitriou, 1994) results on particular instances of the general problem of viral detection.

D. Spinelli (Spinellis, 2003) addressed the detection problem of finite polymorphic viruses with determined mutating length demonstrating that this problem is NP-complete (Filiol, 2006a). For the demonstration, the problem was reduced to a polymorphic virus with fixed scope.

Generalizing Adleman's results, Zuo and Zhou worked on metamorphic viruses and stealth features. In (Zuo and Zhou, 2004), resident viruses, non-resident viruses, polymorphic viruses and stealth viruses were formalized, complementing those of Adleman and adding the notion of furtiveness. Their results show that the complexity problem of detecting a stealth virus surpasses the recognition of a conventional one. Moreover, in (Bonfante et al., 2006), a similar result was demonstrated. Furthermore, Zuo and Zhou presented many results related to the time complexity (Zuo et al., 2005).

E. Filiol reevaluated stealth mechanism using simulability testing (Filiol, 2007d). Filiol studied the use of malicious cryptography and code armouring to complicate code analysis (Filiol, 2012). In (Filiol, 2006b), detection schemes were qualified.

He also formalized and tested k-ary malware that consist in splitting malicious code in different files to evade detection mechanisms. The idea is to distribute the malicious payload into k files with no viral characteristic; it is the combination of all of these files that generates the final payload. The complexity of these codes have been proven to be NP-complete (Filiol, 2007b).

Preliminarily works are summarized in Table 1.

3 k-ary MALWARE

Initially, the viral code was included in a single file that encompasses all its functionalities. Then with the introduction of the k-ary virus, the code is fragmented in a modular approach but still within a predominantly unique environment. The payload will run using a modular malware that downloads and executes malicious codes. For that purpose, many infections stages are needed to achieve the common goal and a coordinated activity distributes each chunk functionality allowing more robust, evasive framework.

k-ary codes presents several advantages:

- The modules are minor in size and therefore easier to obfuscate.
- The modular features would in any case make it difficult to detect.
- Combining several modules will not disclose the final payload.
- Distinct modules can be utilized in specific environments.
- Each module can be modified separately mutating its signature.
- Do not involve actions deemed malicious.
- No direct object use.
- New functionalities can be easily added.

k-ary malware implementations are thoroughly described in Table 2:

Serial/Parallel k-ary Malware. The different subclasses of k-ary codes were validated in serial ($4 \leq k \leq 8$) (Filiol, 2007d) and in parallel ($k = 4$) (Filiol, 2007d). For instance, each part has been able to regenerate the missing codes under different nomenclatures.

A 2-ary Malware using OpenOffice. Multiple proof-of concepts confirmed the complexity of combined viruses for OpenOffice in win32 and Linux environment (de Drézigué et al., 2006).

Table 1: Preliminary works.

Preliminary works			
Author	**Contribution**	**Formalization**	**Complexity**
Alan Turing	(Shannon, 1956)	Turing machines	-
Stephen Kleene	(Kleene, 1936)	Mathematical formalization Quine	- -
John von Neumann	(Von Neumann et al., 1966)	Self-reproduction of cellular automata	-
Veith Risak	(Bilar and Filiol, 2009)	Self-reproducing automata using minimum information exchange	-
Jurgen Kraus	(Kraus, 1980)	Relation with biological self-reproduction behaviors	-
Fred Cohen	(Cohen, 1987b)	Formalization of a computer virus Viral detection Contradictory virus	✓
Leonard Adleman	(Adleman, 1988)	Viral detection Viral protection Computer infections Polymorphism	✓
Diomidis Spinellis	(Spinellis, 2003)	Finite polymorphic viruses	✓
Zhihong Zuo and Mingtian Zhou	(Zuo and Zhou, 2004)(Zuo et al., 2005)	Polymorphic and metamorphic, Stealth features, Resident viruses, Time complexity	✓
Guillaume Bonfante & al.	(Bonfante et al., 2006)	Polymorphism, Stealth virus	✓
Eric Filiol	(Filiol, 2006a)(Filiol, 2006b)(Filiol et al., 2006)(Filiol, 2007c)(Filiol, 2007d) (Filiol, 2012) (Filiol, 2015)	Malicious cryptography and malicious mathematics, Stealth mechanisms, Viral detection, Code armouring, k-ary codes	✓

A k-ary Python Malware. Furthermore, a k-ary virus was implemented in Python (Desnos, 2009) in order to share a secret key utilized to decipher the viral payload. A 3D graphical simulation to represent the interaction and distribution of system calls was presented as well.

A k-ary Goodware. Also, in (Guyot et al., 2012), a k-ary goodware was presented leveraging similar concepts while removing any malicious actor. For that purpose, a Shamir's secret sharing scheme with threshold algorithm has been utilized. The k-ary goodware is composed of k parts that may or may not be executables. The cumulative actions of the constituting chunks define the goodware action. This notion was proposed to better protect the software and sensitive data that each UAV of a swarm will be given. The goodware introduced a method to secure the sensitive information in UAV (Unmanned Aerial Vehicles) prohibiting any leaked information.

A k-ary Experiment for Intrusion Detection System. Furtheremore, the k-ary malware was studied in (Tokhtabayev et al., 2010) while implementing a prototype intrusion detection system.

In the following, we explore the feasibility of an armoured decentralized k-ary malware. This work generalized the modularity concept using the Blockchain network and Distributed Ledgers Technologies: a code fragmented on several environments and which protects itself by harnessing the security of these environments against the analysts and the targets.
This decentralized modularity highlights the major contribution. The new algorithm makes it possible to adapt the activation of the modules according to the context specific to the infected target. With the provision of different codes in decentralized storages,

Table 2: k-ary implementations.

Related work	Implementation	Contribution
(Filiol, 2007d)	Serial and Parallel k-ary malware implementations	k-ary PoCs validation k-ary complexity formalization
(de Drézigué et al., 2006)	2-ary malware using OpenOffice	Increased stealth properties
(Desnos, 2009)	A k-ary python implementation	Leveraging secret keys to decipher viral payload
(Guyot et al., 2012)	A k-ary goodware	Leveraging a k-ary concept to protect sensitive UAV information
(Tokhtabayev et al., 2010)	Interprocess distribution approach	An obfuscation experiment for IDS

the new malware can easily reuse, combine, and adapt pieces of malicious codes that have proven themselves.

4 DISTRIBUTED LEDGER TECHNOLOGIES

This section overviews Distributed Ledger Technologies (DLTs).

4.1 The Blockchain

A blockchain constitutes a peer to peer network that contains the history of all the exchanges made between its users since its creation. The ledger is shared between various nodes, without intermediaries, which allows everyone to check the validity of the chain.

There are public blockchains, open to everyone, and private blockchains, the access and use of which is limited to a certain number of actors.

A public blockchain can therefore be compared to a public ledger, anonymous and forged. Transactions between network users are grouped in blocks. Each block is validated by the nodes in the network, according to the consensus DLT. Once the block is validated, it is timestamped and added to the blockchain. The transaction is then visible to the receiver and the entire network. The process time depends on the blockchain type (about ten minutes for bitcoin, 15 seconds for Ethereum).

Regardless of the technology, DLTs offer numerous security characteristics (Moubarak et al., 2018b):

- Immutability: Once added to a block, a transaction become irremovable.
- Auditability: Each block is characterized by cryptographic schemes and secure Timestamping offering the capacity to audit each transaction.
- Integrity: The SIGHASH function validate the signatures ensuring that any modification will invalidate the transaction.
- Authorization: Elliptical Curve Digital Signature Algorithm (ECDSA) is used to create the links between the blocks. Also, IOTA relies on Winternitz hash-based cryptography signatures.
- Fault Tolerance: Many agreement mechanisms are involved to achieve the consensus in DLTs.
- Transparency: The transactions are appended into blocks and replicated publicly to the peers.
- Availability: Even if peers exit the network, the blockchain network is continually available.
- Consistency: Once the miners agree on the consensus and block arrangement, the distributed ledger is consistent and changes are infeasible.
- Privacy: While the distributed ledger is public, keys relatives to each parties are anonymous.

The decentralized nature of the blockchain, coupled with its security and transparency, promises much broader applications than the monetary field.

4.2 Decentralized Storage

Decentralized data storage is closely linked to blockchains because many concepts overlap. Blockchain itself is a decentralized data storage network, but the operation of the blockchain makes it better suited for storing and managing transactional data.

Blockchains can be used for secure document transfer in several areas. However, storing data in blockchains does not manage large amounts of raw data. This is where decentralized networks can help blockchains.

Many popular decentralized blockchain-driven applications (dApps) uses the concept of decentralized data storage namely the InterPlanetary File Sys-

tem (IPFS). IPFS Merkle DAG is an extremely flexible way to store data. The only requirements are that the object references must be (a) addressed content, and (b) encoded in a specific format. IPFS grants applications complete control over the data field; Applications can use any custom data format they choose, which IPFS may not understand.

IPFS clients require local storage, an external system on which to store and retrieve local raw data for objects managed by IPFS. The type of storage depends on the use case of the node. In most cases, this is simply a portion of the disk space (either managed by the native file system, or by a key-value store such as leveldb, or directly by the IPFS client).

Ultimately, all of the blocks available in IPFS are in the local storage of certain nodes. When users request items, they are found, downloaded, and stored locally, at least temporarily (Mus, 2017). In the next section, we define a new viral algorithmic leveraging the IPFS network and a k-ary concept.

5 DECENTRALIZED k-ary MALWARE

In a previous work (Moubarak et al., 2018a), the development was based on the Bitcoin network to identify the four chunks of the codes, highlighting the features of the Blockchain technnology in the viral domain. However, the main limitation in the aforementioned framework consists in the fact that the validation part requires ten minutes for the Bitcoin network.

This section concentrates on the use of IPFS to create a decentralized k-ary malware (Moubarak et al., 2019) and the capability to remain imperceptible when loaded on the target machine.

5.1 First Naive Malware

The K parts are uploaded independently to IPFS. This first experiment has been implemented under Windows and consists in a simple quaternary malware whose code is injected in a running process.

Here is the general pseudo-code for this malware.

Here we consider an initial malware $\mathcal{M}$ which is split into k different parts in such a way that no part alone enables to guess that it is a component of a malware code.

Proposition 1. *The detection complexity of the malware described in Algorithm 1 is calculable.*

Algorithm 1: Malware - Naive version.

Input: Path of a IPFS directory D that contains the k different malware files.
Output: Execution of the malware.

```
Get the content of the directory D
buffer B ← NULL
For i from 1 to k
Retrieve file F_i
Concatenate file F_i to B
S ← buffer size (in bytes)
Inject the buffer content into a process via its process ID
```

Proof. Even if the malware information is split into k parts, at the end of the `For` loop, buffer B contains the whole code of malware $\mathcal{M}$. From this time instant, the code is detectable. □

It is worth mentioning that aside the detection based on the malware code, code injection into a process is likely to be detected by most modern AV software that use dynamic analysis. If the analysis is based only on signatures and static analysis techniques, the code will not be detected. A basic principle in computer virology is that the environment should be modified as little as possible or not at all.

Additionally, another security drawback consists in that any user who have the hash of the submitted code, can retrieve it from IPFS. Asymetric encryption can be used to secure the IPFS exchange.

To cope these limitations, an armoured advanced malware is created.

5.2 Armoured Advanced Malware

Let us now consider a more sophisticated malware. It is a particular instance of *sequential k-ary code of subclass B* (Filiol, 2007a, Section 4). We will not describe the IPFS/IPNS protocol part which is used by the malware since it is very classical and not malware-specific. The different parts are submitted encrypted to IPFS. We still consider a malware $\mathcal{M}$ whose code is split into k parts and a scheduler program $\mathcal{S}$, in the case of a targeted attack (e.g. Advanced Persistance Threat) against a very limited number of targets. Our PoC is inspired by techniques used in Bradley code (Filiol, 2005) that acquire the keys from a certain website or Pastebin and Dark Paranoid malware (Threats, 2019) where only one instruction is unencrypted in memory. If the key is contained in the body of the code, it can be easily discovered by an analyst. For ethical and responsible disclosure, we will not give all details and code of our

PoC but just the most significant algorithmic aspects.

Here are the main features of this PoC:

- A number of IPFS/IPNS nodes are used which are under the full control of the attacker (possibly coupled with Tor to add anonymity).
- The malware is divided to k parts. Each part P_i (for $i = 1, \ldots k$) is encrypted. The encryption format is raw (there is no header, identification field). Each part is moreover split into IPFS blocks denoted $P_i^j,\ j \in [1, \ldots, n]$.
- Code armouring by encryption has been used in such the way that the malware limits as much as possible its execution with plain text instructions to the least possible. It means that a significant number of operations can be performed with basic operations on the encrypted instructions directly. Each instruction, is deciphered on the fly.
- Each part is contained in a different location in IPFS/IPNS. The location of file F_{i+1} (/ipfs/⟨hash⟩ /f_{i+1}) is contained in the (encrypted) file F_i.
- In order to operate, we use an additional program $\mathcal{S}$ called *scheduler*. Its role is to organize the malware execution according to Algorithm 2. Program $\mathcal{S}$ contains the location of file F_1 (/ipfs/⟨hash⟩/f_1) under an encrypted form. The decryption key is an environmental key (Riordan and Schneier, 1998; Filiol, 2005). The activation environment is considered as likely under the attacker's control.
- Time obfuscation (τ-obfuscation) (Beaucamps and Filiol, 2007) is used randomly during the code execution. Required instructions (in fact IPFS blocks) are decrypted once at a time.
- The malware is able to detect whether it is executed from within a virtual environment or not. If not, the code executes its intended (malicious) tasks otherwise (a virtual environment is present) the code behaves as normal code.
- Other tricks have been implemented in the case of targeted attacks and to activate in very specific environments (for instance (Desnos et al., 2010)).

Let us described the overall simplified pseudo-code.

Algorithm 3 details the initialization of the code and the processing of the first block (P_1^1). The subsequent processing is similar for other $P_i^j,\ j \in [2, \ldots, n]$.

Proposition 2. *The detection of the malware described in Algorithm 2 is an undecidable problem.*

Algorithm 2: Malware - Armoured version.

Input: Operating System Environment $\mathcal{E}$, information I in $\mathcal{E}$ or not
Output: Execution of the malware

while $I \notin \mathcal{E}$ **do**
  Scan randomly and irregularly for I in $\mathcal{E}$

  For i from 1 to 4
Decrypt /ipfs/⟨hash⟩/file_i with I
Retrieve file F_i
Process File F_i

Algorithm 3: Malware - Initialization and processing of the first block.

Input: Operating System Environment $\mathcal{E}$, information I in $\mathcal{E}$ or not
Output: Execution of block P_1^1

$\mathcal{S}$ on initial installation retrieves different information $N_0, N_1, \ldots$ on infected host (IP address, processor ID...) and send them to the attacker as $SHA3(N_0||N_1||...)$
Instructions related to initialization are deleted from memory and $\mathcal{S}$ code.

while $I \notin \mathcal{E}$ **do**
  Scan randomly and irregularly for I in $\mathcal{E}$

$\mathcal{S}$ decrypts IPNS for block P_1^1 with I
$\mathcal{S}$ executes Virtual environment detection
$\mathcal{S}$ retrieves block P_1^1
$\mathcal{S}$ executes P_1^1
Instruction for P_1^1 are deleted
...

Proof. There is two cases to consider. The first one relates to the analysis of files F_i and the second one to the analysis of scheduler program $\mathcal{S}$.

Case 1. Each encrypted part alone does contain neither information on the encryption system nor reference to any other parts. The analyst can just try all possible random sequences to decrypt the code and hence will obtain all possible code instances without being able to determine which are the right ones. We are in the case of Shannon's prefect secrecy (For a perfect encryption scheme, the number of keys is at least the size of the message space - number of messages that have a non-zero probability) (Shannon, 1949). Case 1 consequently refers to unconditional security.

Case 2. Aside the different algorithmic and implementation tricks, we will focus on the fact that scheduler $\mathcal{S}$ falls in the case of Cohen's *contradic-*

tory virus (Cohen, 1985, page 84), (Cohen, 1987a). Only a dynamic analysis (into a virtual environment) can enable the analyst to guess (step by step) program $\mathcal{S}$'s actions. But in this case (detection of virtual environment) program $\mathcal{S}$ performs only benign actions due to the communication with the server under the attacker's control. Hence the result. □

The same antiviral experiments were performed on this armoured version. None of the tests are positives.

6 CONCLUSION

The use of k-ary codes in which a program is no longer a solo binary object but a k-set of files working to yield a final payload constituted the base of this work with the outsourcing of each part to IPFS. In this context, a serial mode operation vector is used, so the execution is done part by part and for each part instruction by instruction (block by block for IPFS). In fact each instruction is deciphered on the fly and then re-encrypted (or emptied of memory depending on the version). For some instructions, they are generated directly from other encrypted instructions before being decrypted and executed according to the previous principle. Therefore, the analyst or the antiviral solution has a compact interpretation on the entire code only because the interaction is performed with a partial subcategory of this k-set. All codes are pulled to repository of P2P source code version control system where each version can be maintained updated and upgraded. Data operations will be recoded to the P2P distributed file system and are tracked. In case of codes' improvement or change, an update can be submitted for fix and later pushed to the target machines.

In conclusion, the potential of the blockchain technology and decentralized storage is increasing. The combination of the blockchain and IPFS is considered to be the future of the distributed internet. Besides, the Interplanetary File System have many potential to host different types of Dapps and is able to connect many blockchains and content directly. Additionally, some of the blockchain security challenges have been exposed revealing the urgent necessity to be sheltered from many assumption reasons of malicious applicability, and which, if remained undeveloped will be exploited. Blockchain has been subject to unique speculation and as we have perceived, it has been advantageous in many domains. However, this backbone that is enabling and protecting several transactions, may drives motives for potential misuses (Moubarak et al., 2017b). Thus, the next step can be to dig further and explore the feasibly of embedding the new malware into Dapps applications.

REFERENCES

Adleman, L. (1988). An abstract theory of computer viruses, in" advances in cryptology—crypto'88", vol. 403. *Lecture Notes in Computer Science*.

Beaucamps, P. and Filiol, E. (2007). On the possibility of practically obfuscating programs - towards a unified perspective of code protection. *Journal in Computer Virology*, 3.

Bilar, D. and Filiol, E. (2009). On self-reproducing computer programs. *Journal in computer virology*, 5(1):9–87.

Bonfante, G., Kaczmarek, M., and Marion, J.-Y. (2006). On abstract computer virology from a recursion theoretic perspective. *Journal in computer virology*, 1(3-4):45–54.

Cohen, F. (1985). *Computer Viruses*. PhD thesis, University of Southern California.

Cohen, F. (1987a). Computer viruses. *Comput. Secur.*, 6(1):22–35.

Cohen, F. (1987b). Computer viruses: theory and experiments. *Computers & security*, 6(1):22–35.

de Drézigué, D., Fizaine, J.-P., and Hansma, N. (2006). In-depth analysis of the viral threats with openoffice. org documents. *Journal in Computer Virology*, 2(3):187–210.

Desnos, A. (2009). Implementation of k-ary viruses in python. *Hack. lu*.

Desnos, A., Erra, R., and Filiol, E. (2010). Processor-dependent malware... and codes. *CoRR*, abs/1011.1638.

Filiol (2015). Malware of the future.

Filiol, E. (2005). Strong cryptography armoured computer viruses forbidding code analysis: the bradley virus. In *EICAR 2005*, pages 216–227.

Filiol, E. (2006a). *Computer viruses: from theory to applications*. Springer Science & Business Media.

Filiol, E. (2006b). Malware pattern scanning schemes secure against black-box analysis. *Journal in Computer Virology*, 2(1):35–50.

Filiol, E. (2007a). Formalisation and implementation aspects of *K* -ary (malicious) codes. *Journal in Computer Virology*, 3(2):75–86.

Filiol, E. (2007b). Formalisation and implementation aspects of k-ary (malicious) codes. *Journal in Computer Virology*, 3(2):75–86.

Filiol, E. (2007c). Metamorphism, formal grammars and undecidable code mutation. *International Journal of Computer Science*, 2(1):70–75.

Filiol, É. (2007d). *Techniques virales avancées*. Springer.

Filiol, E. (2012). Malicious cryptology and mathematics. In *Cryptography and Security in Computing*. IntechOpen.

Filiol, E., Helenius, M., and Zanero, S. (2006). Open problems in computer virology. *Journal in Computer Virology*, 1(3-4):55–66.

Guyot, V., Gademer, A., Avanthey, L., Beaudoin, L., and Erra, R. (2012). Swarm uav attack: how to protect sensitive data. In *Proceedings of European Conference on Information Warfare and Security ECIW 2012*.

Kleene, S. C. (1936). General recursive functions of natural numbers. *Mathematische Annalen*, 112(1):727–742.

Kleene, S. C. (1938). On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155.

Kraus, J. (1980). Selbstreproduktion bei programmen. *University Dortmund (Feb 1980) http://vx. netlux. org/lib/mjk00. html as of 21 oct 2007*.

Moubarak, J., Chamoun, M., and Filiol, E. (2017a). Comparative study of recent mea malware phylogeny. In *Computer and Communication Systems (ICCCS), 2017 2nd International Conference on*, pages 16–20. IEEE.

Moubarak, J., Chamoun, M., and Filiol, E. (2018a). Developing a k-ary malware using blockchain. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–4. IEEE.

Moubarak, J., Chamoun, M., and Filiol, E. (2019). Hiding malware on distributed storage. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pages 720–725. IEEE.

Moubarak, J., Filiol, E., and Chamoun, M. (2017b). Comparative analysis of blockchain technologies and tor network: Two faces of the same reality? In *Cyber Security in Networking Conference (CSNet), 2017 1st*, pages 1–9. IEEE.

Moubarak, J., Filiol, E., and Chamoun, M. (2018b). On blockchain security and relevant attacks. In *Communications Conference (MENACOMM), IEEE Middle East and North Africa*, pages 1–6. IEEE.

Mus, M. (2017). Ipfs, conception and future. http://www.leprogrammeurmarocain.com/ipfs-conception-future/.

Papadimitriou, C. H. (1994). Complexity theory. *Reading: Addison Wesley*.

Riordan, J. and Schneier, B. (1998). Environmental key generation towards clueless agents. In *Mobile Agents and Security*, pages 15–24.

Shannon, C. (1949). Communication theory of secrecy systems. *Bell Systems Techn. Journal*, 28:656–719.

Shannon, C. E. (1956). A universal turing machine with two internal states. *Automata studies*, 34:157–165.

Spinellis, D. (2003). Reliable identification of bounded-length viruses is np-complete. *IEEE Transactions on Information Theory*, 49(1):280–284.

Threats, K. (Retrieved June 5th, 2019). Virus.dos.darkparanoid. https://threats.kaspersky.com/en/threat/Virus.DOS.DarkParanoid/.

Tokhtabayev, A. G., Skormin, V. A., and Dolgikh, A. M. (2010). Expressive, efficient and obfuscation resilient behavior based ids. In *European Symposium on Research in Computer Security*, pages 698–715. Springer.

Von Neumann, J., Burks, A. W., et al. (1966). Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14.

Zuo, Z. and Zhou, M. (2004). Some further theoretical results about computer viruses. *The computer journal*, 47(6):627–633.

Zuo, Z.-h., Zhu, Q.-x., and Zhou, M.-t. (2005). On the time complexity of computer viruses. *IEEE Transactions on information theory*, 51(8):2962–2966.