Avaddon ransomware: an in-depth analysis and decryption of infected systems

Javier Yuste[a,*], Sergio Pastrana[b]

[a] *Universidad Rey Juan Carlos, Madrid*
[b] *Universidad Carlos III, Madrid*

Abstract

Malware is an emerging and popular threat flourishing in the underground economy. The commoditization of Malware-as-a-Service (MaaS) allows criminals to obtain financial benefits at a low risk and with little technical background. One such popular product is ransomware, which is a popular type of malware traded in the underground economy. In ransomware attacks, data from infected systems is held hostage (encrypted) until a ransom is paid to the criminals. In addition, a recent blackmailing strategy adopted by criminals is to leak data online from the infected systems if the ransom is not paid before a given time, producing further economic and reputational damage. In this work, we perform an in-depth analysis of Avaddon, a ransomware offered in the underground economy as an affiliate program business. This threat has been linked to various cyberattacks and has infected and leaked data from at least 62 organizations. Additionally, it also runs Distributed Denial-of-Service (DDoS) attacks against victims that do not pay the ransom. We first provide an analysis of the criminal business model in the underground economy. Then, we identify and describe its technical capabilities, dissecting details of its inner structure. As a result, we provide tools to assist analysis, decrypting and labeling obfuscated strings observed in the ransomware binary. Additionally, we provide empirical evidence of links between this variant and a previous family, suggesting that the same group was behind the development and, possibly, the operation of both campaigns. Finally, we develop a procedure to recover files encrypted by Avaddon. We successfully tested the proposed procedure against different versions of Avaddon. The proposed method is released as an open-source tool so it can be incorporated in existing Antivirus engines and extended to decrypt other ransomware families that implement a similar encryption approach.

Keywords: Avaddon, Ransomware, Malware Analysis, Reverse Engineering, Cybersecurity

1. Introduction

In February, 2018, the USA government estimated that cybercrime costs raised up to between 57 and 109 billions of dollars in 2016 [1]. Cybercrime has been growing for the last decades as it becomes more profitable. The most common goal for cybercriminals is monetary gain and they frequently organize themselves to form online criminal enterprises and businesses [2]. The virtual battlefield where such criminal activities operate allows miscreants to perpetrate crimes in countries which do not have clear extradition laws with the country where the criminals reside. This strategy frequently makes cybercrime hard to prosecute, in addition to other technical characteristics that difficult attribution [3, 4]. In recent times, the underground economy has developed a myriad of approaches that allow cybercriminals to acquire high financial profits. With the cybercrime growth and specialization, many cybercriminals offer their products in an "as-a-service" model, where an attacker can purchase the service through the internet with little technical knowledge. These services reduce the entry level for new criminals and motivate newcomers into the underground [5, 6].

In 2017, Panda security analyzed around 15m potentially malicious binaries [7]. The most noticeable thing was that, upon reviewing the data collected, they realized that 99.1% of the binaries were only seen once. Indeed, a common digital commodity offered in underground markets is malware [8]. One of the most popular variants of malware offered is ransomware [9], where the attacker denies access to the data of its victims until a ransom is paid (hence the name of the threat). When these attacks affect organizations, they might provoke business interruptions, thus causing further economic and social damage [10]. Ransomware operators often partner up with other criminal groups, either in a customer-service relationship (i.e., offering the software for a fixed fee or via a subscription-based access to constant updates) or in a profit sharing scheme (i.e., involved parties are responsible for different tasks in the campaigns and share an arranged percentage of the revenues). Previous works show that criminals can run ransomware campaigns with little technical knowledge, making use of the available services, with an estimated return of investment of between 504% and 12,682% [5].

Due to the profitability and specialization of cybercrime, modern ransomware campaigns have improved their sophistication. First, techniques from well-established cryp-

*Corresponding author
Email address: `javier.yuste@urjc.es` (Javier Yuste)

tography schemes, so-called hybrid cryptosystems, have been recently adopted in ransomware operations, combining symmetric and asymmetric cryptography. Second, modern ransomware perpetrators have incorporated another monetizing technique that further pushes the victims to pay a ransom: data leakage extortion. Apart from encrypting the files, ransomware operators now steal data from the infected systems and threaten victims to leak it online if the ransom is not paid. This extortion scheme was initiated by a threat actor known as *TWISTED SPIDER* in the last quarter of 2019 [11], and was quickly followed by other ransomware groups [12, 13, 14]. In order to face the ransomware threat, and to be able to recover the hijacked files, it is important to understand the criminal ecosystem and also how the malware evolves and operates.

In this work, we study a novel ransomware threat, dubbed Avaddon, whose first campaign was launched on June 2020 in underground forums following a Ransomware-as-a-service (RaaS) model. Since then, Avaddon has been linked to various cyberattacks. This ransomware incorporates a recent trend, which is to publicly *'blame and shame'* victims that do not pay the ransom [15]. At the time of this writing, more than 1,109 GB of data from 62 companies have been leaked and exposed online.[1] In addition, Avaddon operators have recently started to blackmail victims by running DDoS attacks until the ransom is paid. Existing reports described different technical features of Avaddon [17, 18, 19, 20]. However, as far as the authors know, no public decryption procedure is available to recover files from an infection. We aim at filling this gap by providing an in-depth analysis of Avaddon and proposing a decryption procedure to recover encrypted files from infected systems. In particular, we analyze one of the first variants observed in early June, although the proposed decryption method is still functional for the latest samples of Avaddon at the time of this writing. The main contributions of this work are the following:

- We analyze the Avaddon business ecosystem and its similarities with other ransomware families (Section 2). We then provide a detailed analysis of its technical capabilities, using advanced static and dynamic procedures (Section 3). The results of these analyses can be generalized to grasp an overview of how modern ransomware operates, since their *modus operandi* is similar. This knowledge can be used to develop further ransomware defense mechanisms.

- Due to the detailed analysis, we showcase a typographical error in the list of services that Avaddon checks, which is also present in modern variants of another ransomware family, i.e., MedusaLocker. Additionally, we highlight that some similarities on the code of both families hint that they are operated or developed by the same group.

- We propose a method to recover the symmetric keys used for the encryption and decrypt the affected files from infected systems (Section 4). Accordingly, we implement the described method in a tool that we make publicly available to help victims. We provide experimental results in Section 5 by infecting some sandboxed environments and decrypting the file systems with the proposed approach. While this tool was designed using the analysis of the first versions of Avaddon, we have confirmed that it still works with the most up-to-date versions of the ransomware, released in mid-January 2021.

Similar to Avaddon, most modern ransomware strains implement hybrid cryptosystems, mainly using AES and RSA [21] (we analyze these in Section 2.1). Therefore, the proposed decryption method can be adapted to thwart infections by other ransomware families. Our research addresses two important research challenges proposed in the community [22], i.e., the need for deep analysis of ransomware and the need for reactive responses for fighting ransomware attacks. We expect that our contribution will serve for the improvement and development of further ransomware defense mechanisms.

2. Background and related work

In this Section, we first provide background information about ransomware threats. Then, we describe some current directions in ransomware research and existing defense mechanisms. Finally, we present the criminal ecosystem behind Avaddon, including its evolution in the underground economy, and how it has lead to real-world cyberattacks.

2.1. The Ransomware threat

Ransomware is a type of malware that interrupts the business of the victim or denies access to its data until a ransom is paid. This type of malware has direct financial implications for its victims and has promoted the growth of cybercrime, where it is employed as a profitable business model [23]. The first ransomware attack, dubbed *AIDS*, occurred in 1989. It implemented a custom encryption algorithm and only modified the file names [24]. In 1996, Young and Yung warned for the first time about ransomware threats [25]. A decade later, ransomware attacks were observed in the wild [26]. Since then, the methodology of ransomware has evolved, in many cases as a result of the countermeasures developed to thwart them. Some initial variants of ransomware, like *JigSaw*, included a hard-coded secret, used to encrypt all the files. This made decryption trivial, provided that the key could be obtained from the binary [27]. The next move for ransomware was to fetch the keys from a Command and Control server [21], but the reliance on external communications for the actual

[1]For ethical and legal reasons, we have not downloaded nor checked the veracity of the exposed data since otherwise this would cause additional harm to users, and such analysis is not of public interest for the community [16].

encryption was problematic. Thus, modern ransomware mostly uses hybrid approaches, as we detail later. The first strains using hybrid approaches, in 2005, used low-length (56 to 660 bits) RSA keys to encrypt the symmetric keys. However, these symmetric keys could be decrypted by brute force. Therefore, the cryptography used and the key size have evolved [28]. Thus, new variants from 2015 such as *CrytoLocker* or *TeslaCrypt* used 2048-bit RSA keys [26].

Before the popularization of cryptocurrencies, such as Bitcoin, online payment methods were risky for malware authors. SMS text messages, pre-paid cards or premium rate telephone numbers could be traced back easier than Bitcoin [29]. The use of Bitcoin or other cryptocurrencies thwarts tracing the payments sent to criminals. Still, the characteristics of some cryptocurrencies allow for tracking transactions (although not connecting them to the attacker). For instance, Huang *et al.* were able to track over $16 million in likely ransom payments made by 19,750 potential victims during a two-year period [30]. Thus, criminals have adopted privacy-preserving cryptocurrencies, such as Monero, that hinder tracking [31]. These cryptocurrencies, in combination with the cybercrime specialization, have promoted the ransomware threats as a profitable business for cybercriminals [32].

During the lifecycle of a ransomware attack, several actions are conducted [33]. First, the ransomware is distributed. The distribution phase might be carried out using standard methods such as email attachments or website compromises. Then, when the ransomware lands in a system, the infection phase begins. In this phase, the ransomware might perform different actions to ensure its success (e.g., elevating privileges, acquiring persistence, stopping antivirus services, etc.). Once the system has been successfully infected, the ransomware proceeds to encrypt the files. Finally, after encrypting the system, a ransom is demanded to the victim. This ransom demand might be followed by additional extortion techniques. Recently, it has become popular among ransomware authors to publish personal data from their victims (acquired from the infected systems) if they do not pay the demanded ransom.

In Table 1, we report a summary of predominant ransomware strains as of March, 2021. The list includes families that are classified in the top 10 of ransomware threats in 2020 (Top 10 '20), according to PaloAlto Unit42 [34]. It also contains strains classified as Tier 1 (Most Wanted) and Tier 2 (Rising Powers) according to a classification made by Intel471 [15]. Finally, we include other relevant families that have been linked to recent attacks. For each ransomware family, we report the number of victims (Num. attacks) and the number of critical infrastructures (CI) attacked. The number of victims was obtained on March, 2021, from a private investigation source, looking at the different leak sites of the ransomware families on the DarkWeb. The number of attacks against critical infrastructures was obtained from the CIRWA dataset [35]. The number of victims are a lower estimation, i.e., it only accounts for publicly reported attacks. For instance, Ryuk is allegedly linked to thousands of attacks and got payments for the amount of $61m between February 2018 and October 2019 [36]. However, it does not operates a leak site, so the number of victims reported is 0. Still, it is suspected that operators behind Conti (which does have a leak site) are the same as Ryuk [13]. Finally, we report the cryptography schemes used to encrypt files and keys. This information was obtained by consulting different analysis reports from various sources. As it can be observed, most of the modern ransomware families use hybrid approaches in the encryption process: files are encrypted using a symmetric algorithm (mostly AES), and the session keys used to encrypt the files are in turn encrypted using asymmetric cryptography (mostly RSA).

2.2. Key-management in ransomware

Understanding the key management procedures employed by ransomware developers is crucial for the development of effective countermeasures. Some authors have proposed a taxonomy of ransomware families based on their encryption procedures or key management approaches [27, 21]. In previous works, four main key generation strategies have been identified [37]:

1. *Static asymmetric key.* In this strategy, the ransomware file carries a public key, which is used to encrypt the files in the infected system. Decryption can only be performed by using the corresponding private key, which is saved in the infrastructure of the attacker. The main drawback is that the same private key can be used to decrypt the files of all the victims that were affected by that specific version of the ransomware.
2. *Dynamically generated asymmetric key.* In this procedure, an asymmetric key pair is generated on the infected system. Then, the public key is used to encrypt the files and the private key is sent back to the attacker.
3. *Static symmetric key.* In this approach, a symmetric key that comes embedded with the ransomware is used to encrypt the files. This procedure is often avoided by complex ransomware families, since the decryption key could be recovered by reverse engineering the ransomware.
4. *Dynamically generated symmetric key.* In this approach, a symmetric key is generated on the infected system and used to encrypt the files. This symmetric key is then encrypted using an embedded public key and sent back to the attacker or saved in the infected host, such that only the attacker can decrypt it with their private key.

As we show in Table 1, hybrid cryptosystems are the most common approach in modern ransomware, with AES keys of 256 bits and RSA keys larger than 2048 bits. The popularity of hybrid approaches in the encryption process

Ransomware Family	Num. attacks (CI)*	Top 10 '20	Tier (Intel27)	File Encryption	Key Encryption
Conti	291 (14)	✗	②	AES	RSA
MAZE	266 (60)	✓	①	ChaCha	RSA
Egregor	206 (6)	✗	①	ChaCha	RSA
Sodinokibi (REvil)	178 (45)	✓	①	Salsa20	AES + ECDH
DoppelPaymer	174 (21)	✓	①	AES	RSA
NetWalker	144 (26)	✓	①	AES/Salsa20/ChaCha	Curve25519
Pysa/Mespinoza	103 (3)	✗	②	AES	NaN
Avaddon	62 (2)	✗	②	AES	RSA
DarkSide	58 (5)	✗	②	Salsa20	RSA
CL0P	43 (4)	✗	②	RC4	RSA
Ryuk	0 (46)	✓	①	AES	RSA
Suncrypt	22 (2)	✗	②	ChaCha	Curve25519
Ragnar_Locker	22 (3)	✗	②	Salsa20	RSA
AKO	9 (0)	✗	-	AES	RSA
LockBit	9 (2)	✗	-	AES	RSA
RansomEXX	14 (10)	✓	-	AES	RSA
MedusaLocker	0 (0)	✗	-	AES	RSA
Sekhmet	6 (2)	✗	-	ChaCha	RSA
WastedLocker	0 (2)	✓	-	AES	RSA
Dharma/Crysis	0 (2)	✓	-	AES	RSA
Phobos	0 (0)	✓	-	AES	RSA
Zeppelin	0 (0)	✓	-	RC4	RSA

Table 1: Analysis of predominant ransomware strains by March, 2021, together with their number of victims and the encryption algorithms used. The number of victims is obtained from a private investigation source, looking at the different leak sites of the ransomware families on the DarkWeb (data from March, 2021), and CI are the attacks targeting Critical Infrastructures obtained from the CIRWA dataset [35]. The Top 10 in 2020 is based on a report from PaloAlto Unit42 [34]. ① and ② respectively refer to Tier 1 (Most Wanted) and Tier 2 (Rising Powers) strains according to a classification made by Intel471 [15].

difficults the development of decryption procedures without paying the ransom. However, the main drawback of these hybrid procedures is that the symmetric key must be stored in memory at least during the encryption process. Therefore, there exists a time window to recover the symmetric key from memory [38, 39].

2.3. Research in ransomware

As discussed by the community [22], ransomware studies can be categorized into analysis and counteraction. Research in ransomware analysis is devoted to study the techniques and behavior of ransomware threats [40]. These analyses are crucial to support further research in ransomware classification, detection and prevention [41]. On the other hand, counteraction studies aim to prevent or mitigate the threats posed by ransomware. These counteraction studies can be further categorized into prevention, detection and prediction.

Some authors indicate that more features of ransomware threats need to be analyzed in order to increase the effectiveness of ransomware detection [42]. For instance, ransomware analysis is a critical step to develop signature-based detection approaches [43, 44]. In this direction, Hampton et al. [45] analyzed different ransomware strains and their interactions with the infected systems in order to serve as a baseline for the development of further detection strategies. Similarly, Kharraz et al. [46], analyzed 1,359 ransomware samples to help propose new detection strategies. Subedi et al. [47], on the other hand, utilized a set of features extracted from ransomware samples to develop a detection method based on data-mining techniques. Other studies have focused on specific ransomware families, often proposing preventive measures based on the results of the analyses [48, 49, 50, 51].

In counteraction studies, reactive ransomware prevention researches are devoted to mitigate the effect of ransomware attacks by restoring the encrypted files. Most of these studies focus on reverting back to security backups (i.e., older versions of the systems) [52]. However, these precautions (periodically saving backups) are not always taken. In addition, modern ransomware threats might target security copies as part of their attacks, as in the case of Avaddon. In contrast, Le Guernic et al. [53] proposed a technique to decrypt files by exploiting a weakness in some encryption algorithms. Similarly, Kolodenker et al. [38] proposed a key escrow mechanism to capture symmetric keys in ransomware attacks. The captured keys could then be used to decrypt the affected files after analyzing the internals of the encryption mechanism implemented by the ransomware.

Finally, ransomware detection approaches often leverage classical malware detection methods adopted for ransomware-specific behaviors. In this way, ransomware activities can be split in 8 stages [54]: *fingerprint*, *propagate*, *communicate*, *map*, *encrypt*, *lock*, *delete* and *threaten*. For instance, Kharaz et al. focused the detection on common tasks performed by ransomware, such as changing the desktop wallpaper [55]. Following recent trends in malware detection, some Machine Learning-based approaches have also been proposed specifically targeting ransomware detection [56, 57, 58].

2.4. The ecosystem of Avaddon

Avaddon[2] is a ransomware that was offered as an affiliate program on June, 2020 in a Russian underground forum, only accessible by invitation or after the payment of a registration fee. In that program, the operators were looking for partners for their campaign. Additionally, Avaddon was later promoted on other underground forums.[3] Actors that become affiliates are equipped with both the ransomware binary and an administration panel to control their infections. Access to the program is free and constrained only for reputed (and Russian-speaking) actors. In exchange, partners have to share part of the obtained revenues with the ransomware owners. This share depends on the amount of infections, ranging from 35% to 15% for larger volumes. Therefore, affiliates, who are only responsible for distributing and installing the malware on infected systems, obtain a minimum of 65% of the revenues generated by the ransomware, without the need of operating the payment system [17]. Such distribution often relies on botnets hired in a Pay-Per-Install scheme [59]. Additionally, partners can purchase installs on RDP servers, which is another popular product traded in underground economy [60]. Thus, the supply chain needed to enter in this business does not require technical knowledge and it opens the barrier to any criminal entrepreneur [61]. As a restriction in the affiliates program of Avaddon, it is forbidden to target victims in the Commonwealth of Independent States (CIS). We describe the mechanism used to achieve this restriction in Section 3.5.

A few days after their publication on underground forums (on June 4rd, 2020) Avaddon was observed in the wild [19]. In that first campaign, a malicious attachment was distributed in low-quality phishing emails. These emails hinted that a compromising photo of the victim had been leaked, inciting the victim to open the file out of fear. The attached file was a zip-compressed JavaScript file. This file tried to masquerade as a JPG photo, having the extension ".jpg" just before the ".js" extension (e.g., "IMG123456.jpg.js"). Upon execution, the malicious JavaScript file downloaded and executed Avaddon. Allegedly, the first wave of this campaign targeted mostly Canada, although their targets varied later. Indeed, as mentioned before, Avaddon was launched as a RaaS, which means that the targets are not chosen by the ransomware developers (apart from the ban on CIS victims) but by the affiliates. Upon infecting a system, Avaddon leaves a ransom note to the victim with instructions on how to pay the ransom. This note leads the victim to a Tor hidden service, where further instructions are given to make the payment in exchange for the decryptor. At the time of this writing, the payment service is still operative, confirming that the campaign is ongoing. Regarding the decryption process provided by the ransomware operators after paying the ransom, some stories from affected users state that it is unreliable and that recovery is not ensured [62].

Two months after the initial release, in August, 2020, Avaddon was updated to incorporate a new trending technique to their features: extortion to victims [63]. Following the model from other ransomware campaigns, Avaddon operators decided to publish data from their victims to the internet if they do not pay the ransom [64, 65]. By the 22nd of April, 2021, Avaddon has allegedly infected and leaked data from 62 companies (1,109.23 GB of data) and is extorting 30 additional companies. Finally, in January 2021 (concurrent to the writing of this paper), Avaddon included a new technique used for extortion: attacking their victims with DDoS [66]. Therefore, the threat to victims is now three-fold: i) their data is first encrypted in the infected systems; ii) that data is then leaked publicly if the ransom is not paid; and iii) DDoS attacks are performed to disrupt their businesses until the ransom is paid.

At the time of writing, we are not aware of any public decryption tool for Avaddon. Additionally, various reports and recent complaints from Avaddon victims about their decryption support show that the campaign is still operative [67, 68]. In this paper, we fill this gap and release an open-source tool that automatically detects and decrypts files, which could be integrated in existing Antivirus solutions.

3. Ransomware analysis

In this section, we provide an analysis of the Avaddon ransomware. In particular, we analyze a version of Avaddon released as part of their initial campaign in June, 2020. We report a list of Indicators of Compromise (IOCs) of the sample in Appendix A. The analyzed binary (MD5: `c9ec0d9ff44f445ce5614cc87398b38d`) is a Portable Executable (PE) file with a size of 1.1 MB. The PE format describes the structure of executable programs in Windows Operating Systems (OS) [69]. PE files are mainly divided in two important parts: headers and sections. While headers contain information about the program itself and data to be read by the OS in order to correctly load and execute the file, sections contain the actual code and data of the program. Figure 1 shows the methodology used to analyze the sample, partially based on previous works [70, 71]. We depict each step (gray, rounded rectangle) and the

[2]The name of the ransomware, Avaddon, may be derived from the Hebrew term "Abaddon", the name of an angel of the abyss in the Bible, mainly associated with the meaning of "destruction" [18].

[3]Due to ethical reasons, and to avoid promoting the site, we do not provide the name of the forums.

tools used (inside each rectangles). We have divided the steps in two vertical blocks, differentiating those activities performed statically (i.e., without executing the binary) from those performed dynamically (i.e., running the binary in an isolated environment). Additionally, the activities are grouped in three different stages: basic analysis, behavioral analysis and code analysis. For the sake of simplicity, we have made the following assumptions when designing the methodology: i) the sample to be analyzed is a PE file and ii) there are not packing protections in the file.[4] First, we analyze the headers of the PE file (step 1), which contain useful information about the sample, with Pestudio.[5] We observe that the compilation time field in the headers is set to June, 3, 2020, at 11:47:22 (UTC). Although this field is prone to be modified by malware authors, the timestamp is similar to the time of the first appearances of Avaddon samples, which indicates that we are analyzing one of the first versions of Avaddon. Next, we analyze the functions present in the Import Address Table (step 2) and the readable strings in the binary (step 3) to identify possible capabilities. Once we finish the basic analysis stage, we have some initial hypotheses about the capabilities of the sample. Then, we proceed to run the file in an isolated environment and analyze its behavior (behavioral analysis). In order to widen the set of capabilities shown by the sample at runtime (some functionalities might remain hidden if anti-analysis measures are implemented in the ransomware), we execute it in two different sandboxes. First, we execute the sample in an online sandbox (step 4), Any.Run,[6] in a Windows 7 x32 system. Then, in order to corroborate the results and identify further details, we execute the sample in a local virtual machine (steps 5 and 6). The OS installed in the virtual environment, which is built on top of VirtualBox,[7] is Windows 7 x64. The set of capabilities (hypothesized or confirmed) identified in the previous steps are then used as a baseline for the code analysis phase, where we reverse engineer the sample using static (step 7) and dynamic (step 8) techniques. For these tasks, we utilize well-known reverse engineering tools (i.e., Binary Ninja [8] and x64dbg [9]). In this stage, we utilize both static and dynamic techniques interchangeably to analyze the previously identified capabilities and find new ones. Finally, we conclude the process when there are not remaining capabilities that have not been analyzed (step 9).

In Figure 2, we represent a summary of the behavior identified in the sample after applying the aforementioned methodology. Upon executing the binary (step 0), it first checks for debuggers attached to the process (step 1). Then, the sample retrieves information about the victim machine

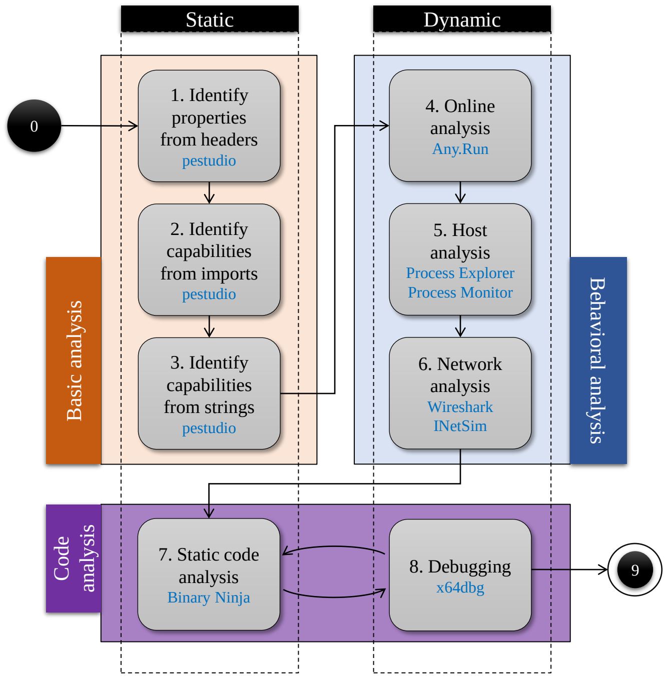

Figure 1: Malware analysis methodology.

(step 2). After performing some checks on the information obtained, the sample elevates privileges in the system (step 3) and acquires persistence to survive reboots (step 4). Finally, it proceeds to execute its main functionality by: manipulating processes and services that might interfere with its execution (step 5); deleting shadow copies and security backups that could be used to restore the system (step 6); and encrypting the file system (step 7).

The rest of this section presents details of the analysis. First, we discuss the packing protections of the analyzed binary in Section 3.1. Next, we show the imported functions in Section 3.2. In Section 3.3, we describe the protections implemented in the binary to hide strings from static analysis. Then, the remaining of the section is organized based on the outline depicted in Figure 2, focusing on specific capabilities or mechanisms. In Section 3.4, we report the anti-analysis techniques (step 1 in the figure) employed by the sample. We show how the ransomware authors implemented a protection to not infect Commonwealth of Independent States (CIS) victims in Section 3.5 (step 2). The privilege escalation techniques are analyzed, step by step, in Section 3.6 (step 3). Then, we showcase the details of the persistence mechanism in Section 3.7 (step 4). Interactions with other processes and services are presented in Section 3.8 (step 5). Finally, we describe the key management procedure in Section 3.9 and the file encryption mechanisms (steps 6 and 7) in Section 3.10.

3.1. Packing protections

In order to ease the analysis of malware samples, it is recommendable to eliminate packing protections before-

[4]This is the case of Avaddon samples, but for packed binaries an initial step to unpack them could be easily included
[5]`https://www.winitor.com/`
[6]https://any.run/
[7]https://www.virtualbox.org/
[8]https://binary.ninja/
[9]https://x64dbg.com/

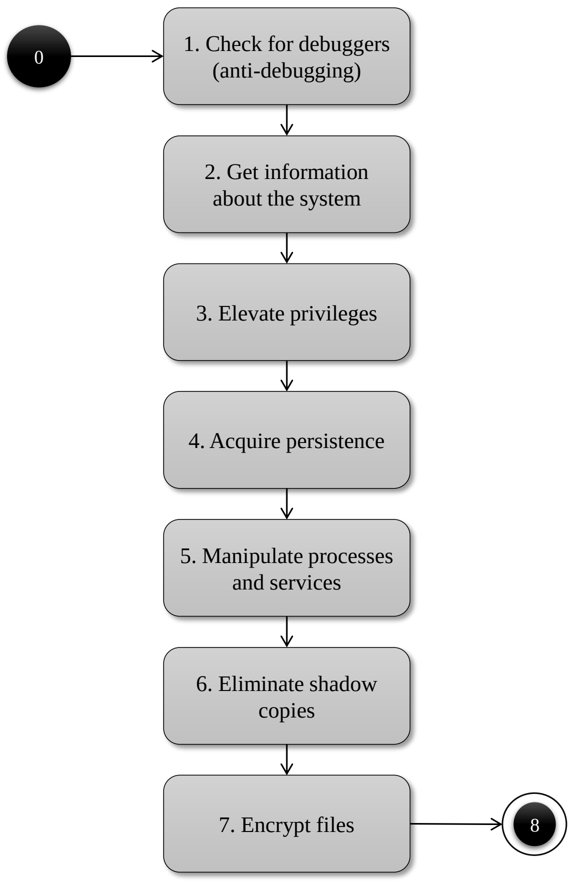
0
1. Check for debuggers (anti-debugging)
2. Get information about the system
3. Elevate privileges
4. Acquire persistence
5. Manipulate processes and services
6. Eliminate shadow copies
7. Encrypt files
8

Figure 2: Outline of the behavior of the ransomware sample analyzed.

hand. In this case, we suspect, due to some properties of the PE file, that there are not packing protections in the binary. First, we find that the PE file contains 4 sections which have almost no differences in size between disk and memory. This in an indicator of the PE file not being packed, since the presence of a virtual section (i.e., a section that requests space in memory but does not occupy bytes in disk) is a common indicator of packing protections. Then, we find over 200 imported functions, which present some useful information about the capabilities of the ransomware. Finally, we are able to identify several meaningful strings, in addition to some encrypted ones. Packing protections often hide (i.e., encrypt or encode) imports and strings in order to avoid detection from automated static analysis tools. Therefore, we conclude that there are not packing protections in this sample, although there might be some obfuscation techniques for a subset of the observed strings.

3.2. Imported functions

The Windows OS offers an Application Programming Interface (API) for applications to interact with many functionalities of the OS, e.g. to interact with files, processes, etc. This API also provides an abstraction layer for the underlying hardware. In order to call functions from the API, programs need to know their location in memory. This need might be fulfilled in different ways, but the most common method consists of importing the required functions prior to execution. This method is performed by the OS loader before transferring control to the program. To do so, the PE file contains an Import Address Table (IAT) in the headers, which includes a list of functions to be imported by the OS loader. When the file is executed, the OS loads the file in memory and fills the IAT with the addresses of each requested function. Then, the program is able to call those functions because it now knows their addresses in memory. Therefore, the IAT provides useful information about the capabilities and intentions of the program, since it offers hints about the interactions with the system that the sample might want to perform. In addition, these imported functions might guide us towards specific addresses within the analyzed binary in the Code Analysis phase. For instance, we might look for the encryption procedure by identifying calls to an imported function like *CryptEncrypt* within the code.

The functions imported by the analyzed sample show capabilities that are frequently implemented in ransomware, such as encryption (e.g., *CryptGenKey* or *CryptEncrypt*), persistence (e.g., *RegCreateKeyW*, *StartServiceW*), anti-analysis (e.g., *IsDebuggerPresent*) or activity control (e.g., *DeleteService* or *TerminateProcess*).

3.3. Strings

Looking for strings through a PE file allows analysts to identify capabilities of the binary, as well as looking at the IAT. Indeed, some imports will appear when searching for strings if they are imported by name (external functions may be imported by name or ordinal [69]). Therefore, we proceed to extract all readable strings that have more than 4 characters in the whole file. Then, we filter the extracted strings and exclude those that are not meaningful (bytes that are part of code might non-intentionally form readable strings that are not meaningful). In this case, as aforementioned, we find enough meaningful strings to think that the PE file is not packed.

Many of the strings that are present in the PE file are paths to folders or files (e.g., "`C:\Temp`"). While we initially can not know the actual purpose of those files, we hypothesize that some of them may be used to drop additional payloads or to move the PE file upon infection to a different location (we confirm this hypothesis in Section 3.7). Then, we observe two strings that refer to cryptography providers (i.e., "`Microsoft Enhanced Cryptographic Provider v1.0`" and "`Microsoft Enhanced RSA and AES Cryptographic Provider`"). These strings are normally used to acquire cryptography contexts using the Windows API, which are later needed to perform some cryptography operations. Additionally, some strings indicate that the ransomware was developed in C++, which is an object-oriented programming language. Although this characteristic does not provide any information about the capabilities of the sample, the particularities of C++ programs must be taken into account in the Code Analysis process. We will highlight some C++ properties that allow us to extract conclusions in the reverse engineering process, but discussing the differences between C++ and other languages at assembly level is out of the scope of this work. For more information, we refer the reader to other works that focus on C++ reverse engineering [72].

Interestingly, we find many strings that are Base64 encoded. However, upon decoding them, no legible string is recovered. Therefore, we suspect that these strings are obfuscated by other means (i.e., encoding or encryption) in order to hide their content. If that is the case, then those strings might be important to identify additional capabilities of the malware. We later confirm, after analyzing the code of the binary, that these strings are indeed encrypted and they are only decrypted at runtime on demand, i.e., when they are required by the program. First, global variables are created to hold the encrypted strings, making them accessible from every function in the binary. In Algorithm 1, we show one of the functions (0x4012a0 in this case) that creates a global variable pointing to an encrypted string. There, the encrypted string and its size are pushed onto the stack (lines 1-2). Then, a global variable is created at 0x4f8a28 with the content of the encrypted string (lines 3-4). Finally, a destruction function is registered (lines 5-6). This function will be called when the process exits. The global variable is then referenced wherever this particular string is needed in the program. For each encrypted string, there is an initialization function like the one we described. We know that these variables were global variables in the source code because:

1. There is a global variable per encrypted string.
2. There is a constructor function for each global variable.
3. Each global variable has a predefined address. These addresses are hard coded in each constructor function.
4. After initializing the global variable, a destructor function is registered to be called upon terminating the program.

Algorithm 1: One of the functions responsible for initializing a global variable with the value of an encrypted string.

```
1 0x4012a0: push 0x30;
  // Size of the encrypted string
2 0x4012a2: push 0x49e180;
  // Encrypted string
3 0x4012a7: mov ecx, 0x4f8a28;
  // Global variable
4 0x4012ac: call 0x40a390;
  // Creates a global variable at ecx
     (0x4f8a28 in this case) with the string
     stored at the value previously pushed
     (0x49e180 in this case)
5 0x4012b1: push 0x4874a0;
  // Destructor
6 0x4012b6: call _atexit;
  // Register the destructor function to be
     called when the process ends
7 0x4012bb: pop ecx;
8 0x4012bc: retn;
```

Once initialized, the strings are decrypted and used as needed by referencing the global variables. In Algorithm 2, we show an example of this procedure. In particular, we show an example of a string containing some command line arguments that is decrypted and used to create a process. In this case, the goal is to delete security backups. First, the decryption function is called passing the global variable as an argument (lines 1-3). This function returns a new string with the decrypted value, which is immediately used to create the aforementioned process (lines 4-5). The sequence of instructions described can be summarized in the following pseudo code:

```
CommandLine = DecryptString(GlobalVariable);
CreateProcess(CommandLine);
```

The decryption function is located at address 0x40c780. First, the received string is decoded from Base64. Then, as shown in Algorithm 3, each character is decrypted by substracting 2 units from its value (line 3) and XOR-ing the result with 67 (line 5). These instructions are executed once for every character in the string.

Since we know the address in which global variables are placed, we have automatically re-labeled them in order to

Algorithm 2: Decryption of the value of a global variable into a temporary register.

```
1 0x40d110: mov edx, 0x4f8a28;
    // Global variable that contains an
       encrypted string
2 0x40d115: lea ecx, [esp+0x8];
    // Local variable that will hold the
       decrypted string
3 0x40d119: call decrypt_string;
    // Decrypts the string at edx (the global
       variable) and stores the result in ecx
       (the local variable)
4 0x40d11e: push eax;
    // eax now contains the decrypted string
       (it is equal to [esp+0x8], the local
       variable) which, in this case, contains
       command line arguments
5 0x40d11f: call create_process;
    // Creates a process with the command line
       received as argument
```

Algorithm 3: Procedure to decrypt obfuscated strings.

```
1 0x40c820: mov al, byte [esi];
    // Move the current character to al (the
       lower 8 bits of eax)
2 0x40c822: mov edx, dword [ebp-0x1c];
3 0x40c825: sub al, 0x2;
    // Substract two units from the character
4 0x40c827: mov edi, dword [ebp-0x18];
5 0x40c82a: xor al, 0x43;
    // XOR the result with 0x43
6 0x40c82c: mov byte [ebp-0x30], al;
7 0x40c82f: cmp edx, edi;
8 0x40c831: jae 0x40c84d;
```

improve the readability of the code for the analysts. To allow for reproducibility and to assist other analyses on this and similar malware samples, we have published a script that automates these tasks using *Binary Ninja* in our public repository.[10]

3.4. Anti-analysis techniques

Successfully infecting a system critically depends on not being detected. Thus, malware authors often implement different techniques to evade antivirus systems or sandboxes. Additionally, mechanisms are frequently put in place in order to delay analysts and, therefore, increment the time needed for building detection tools for the sample (e.g., signatures). In the case of Avaddon, we observe some anti-analysis techniques, which we describe next.

String obfuscation. As mentioned in prior sections, some of the strings are encrypted, which hides important capabilities of the ransomware. This technique is commonly used to: i) evade detection, and ii) delay analysts. In Section 3.3, we analyze this obfuscation technique and the process used to decrypt the strings.

Anti-debugging. We found a call to *IsDebuggerPresent* at offset 0x42e03d. Debuggers are programs designed to analyze other programs at runtime (i.e., processes), and they are frequently used by security analysts to dynamically inspect malware. Hence, malware authors often embed code in their programs that checks for debuggers and terminates the malicious processes (or changes their behavior) if a debugger is detected. In particular, *IsDebuggerPresent* is a function provided by the Windows API that returns a true value if a debugger was attached to the program. If *IsDebuggerPresent* returns true, the Avaddon sample exits prematurely, without encrypting the system. To circumvent this protection, we considered two strategies:

1. Hook the call to *IsDebuggerPresent* so it always returns false. By doing this, we would be changing the code on the fly, and the debugger would not be detected by the sample.
2. Change a binary value in the Process Environment Block (PEB), a data structure that holds information about the process. That structure is built by the OS when the program is executed and it is unique for each process. Among other information, it contains a bit that indicates if a debugger has been attached. When a call to *IsDebuggerPresent* is made, it returns the value of that bit. Therefore, changing that value in the PEB would successfully hide the debugger from that call and from any manual check (the PEB can also be manually walked through by parsing its structure).

In order to avoid further anti-debugging mechanisms that might parse the PEB (i.e., not using *IsDebuggerPresent*), we decided to implement the second strategy. That is, we decided to modify the PEB of the process.

[10] `https://github.com/JavierYuste/AvaddonDecryptor`

3.5. Language checks

To ensure that citizens from some regions are not infected, it is frequently observed that malware binaries implement techniques to check the country where the infected machine is located. It is common to see that CIS victims are dodged in many malware samples, as is the case of Avaddon. The most popular approach is to check for the keyboard layouts and the OS language. In this sample, we found both checks (addresses 0x42e0ec and 0x42e0b6, respectively) for different layouts and languages. In particular, we discovered checks for language locales (Russian and Ukrainian) and keyboard layouts (Russian, Sakha, Tatar and Ukrainian). If any of these keyboard layouts or OS locales is found, the binary exits without harming the landed system. That is, this sample of Avaddon ransomware is designed to avoid infecting Russian and Ukrainian systems. This, together with the fact that the malware was first advertised in a Russian underground forum, provides strong (though not conclusive) evidence that the origin of the malware is Russia.

3.6. Privilege escalation

After gaining initial access to a system, malware activities often require administrator privileges to accomplish some critical tasks (e.g., acquire persistence, infect system files or processes, etc.). However, asking the user to concede those privileges might raise suspicions. In addition, some users might not be able to concede those administrator privileges (e.g., in restricted environments, some users might not be administrators of the system in which they are working). Therefore, reducing the number of clicks needed from the victim to successfully infect the system is critical. In this case, escalating privileges is critical because the ransomware needs to i) acquire persistence through registry keys (Section 3.7), ii) stop processes and services (Section 3.8), and iii) delete backups (Section 3.10).

The process implemented in the analyzed sample to elevate privileges is a well-known User Account Control (UAC) bypass. Indeed, there exist public open-source implementations [73] and it is not uncommon to find this technique in different malware families [74, 75]. Next, we briefly summarize this process and its implementation in Avaddon. First, three registry keys are added or modified (at offset 0x40ed20):

1. `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System EnableLUA=0` (disables the "administrator in Admin Approval Mode" user type [76]).
2. `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System ConsentPromptBehaviorAdmin=0` (this option allows the Consent Admin to perform an operation that requires elevation without consent or credentials [77]).
3. `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System EnableLinkedConnections=1` (makes user mapped drives available to the administrator versions of those users [78]).

The first two registry key values allow the sample to elevate privileges without alerting the user, while the third one enables access to volumes of the current user when administrator privileges are acquired.

Then, the sample checks its privileges (offset 0x41a5c0). If it has administrator privileges, the execution is continued without running the rest of the UAC bypass. Otherwise, administrator privileges are obtained via the following procedure (implemented at 0x40ef90):

1. First, a Class Identifier (CLSID) is decrypted. This CLSID is stored in the binary as an encrypted string, as we described in Section 3.3. The decrypted value is "`{3E5FC7F9-9A51-4367-9063-A120244FBEC7}`", which corresponds to CMSTPLUA. For the rest of this section, we refer to that value as CLSID_CMSTPLUA.
2. Next, an Interface Identifier (IID) is decrypted in the same way, obtaining the value "`{6EDD6D74-C007-4E75-B76A-E5740995E24C}`". For the rest of this section, we refer to it as IID_ICMLuaUtil.
3. Then, a third string is decrypted, which contains the value "`Elevation:Administrator!new:`".
4. Once the three strings have been decrypted, a new string is built by concatenating "`Elevation:Administrator!new:`" and CLSID_CMSTPLUA.
5. Next, the function *CoGetObject* is called in order to obtain a pointer to CMLuaUtil. The parameters of the call are the following:

   ```
   CoGetObject("Elevation:Administrator!new:
   {3E5FC7F9-9A51-4367-9063-A120244FBEC7}", 0x2
   4, &IID_ICMLuaUtil, &CMLuaUtil)
   ```

   At this point, user interaction might be needed to grant administrator privileges for the program in some systems. In some cases, this might be accompanied by social engineering techniques, e.g. instructions accompanying the phishing email where the malware is attached. In this case, we have not observed any particular behavior.
6. If the call is successful, CMLuaUtil now points to a structure (lpVtbl) that contains the address of a function named *ShellExec* (CMLuaUtil→lpVtbl→*ShellExec*).
7. Finally, the binary executes itself with administrator privileges by calling *ShellExec* with the following parameters:

   ```
   ShellExec(CMLuaUtil, "C:\[...]\sample.ex
   e", [...])
   ```

3.7. Persistence and infection tracking

In order to survive across reboots, malware samples must be run automatically on infected systems after the initial foothold has been obtained [79]. Otherwise, they would need to infect the system again if further runs are required. In order to achieve persistence in a system, there exist many approaches. Usually, malware authors acquire persistence by adding registry keys, creating services or registering scheduled tasks. By doing so, the malware sample is periodically executed by the OS (e.g., at scheduled times or at every reboot). Additionally, malware samples often implement mechanisms to prevent re-infection of already-infected systems, in order to minimize the risks of detection or to prevent disruption of previous runs.

In Section 3.2, we hypothesized that the ransomware acquires persistence via registry keys, since we observed that related functions were imported (e.g., *RegCreateKeyW*). Then, we confirmed this behavior via dynamic analysis, noticing that the following registry keys were added by the ransomware at runtime:

- `HKU\S-1-5-21-2724635997-1903860598-4104301868-1000\Software\Microsoft\Windows\CurrentVersion\Run\update: "C:\Users\%UserProfile%\AppData\Roaming\%sample%.exe"`
- `HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run\update: "C:\Users\%UserProfile%\AppData\Roaming\%sample%.exe"`

Upon inspecting the code, we identified that this functionality is implemented at address 0x40cf50. The function located at that address is responsible for acquiring persistence by adding the two aforementioned registry keys. First, two encrypted strings (see Section 3.3), which contain the names of the registry keys to be added, are decrypted. Then, the decrypted strings are used to create the registry keys (address 0x40c45e) and set their values (address 0x40c487). With those registry keys present in the system, the PE file is executed at each system reboot (notice that a copy of the sample is dropped at runtime in `"C:\Users\%UserProfile%\AppData\Roaming\%sample%.exe"`, where "`%sample%`" is the name of the PE file).

In addition, to avoid re-infecting the system, a mutex is created with the value `{2A0E9C7B-6BE8-4306-9F73-1057003F605B}`. If this mutex is already present in the system, the binary exits and does not encrypt files. In addition, the ransomware takes measures to avoid encrypting already encrypted files, as we describe in Section 3.10. These mechanisms are useful to avoid reinfecting victims that have already paid a ransom or that are being currently affected. Nevertheless, the fact that the presence of such mutex is checked allows users to prevent Avaddon infections. By creating such mutex in a healthy system, Avaddon ransomware samples that check for the presence of that mutex will not execute. However, not every sample of Avaddon uses the same mutex, as it might change among versions.

3.8. Processes and services manipulation

In order to avoid being detected or neutralized, some malware samples try to stop anti-malware solutions. To do so, administrator privileges must be acquired. However, it is often easier to acquire administrator privileges without being detected than to encrypt the whole file system without rising awareness. In Section 3.2, we highlighted that the PE file imported some functions that may indicate an attempt to control some anti-malware solutions by interacting with services and processes. Additionally, before encrypting the infected system, it is important to stop processes that might be locking files. For instance, ransomware authors often try to stop database processes that might be locking files.

In this case, we found two functions (located at offsets 0x41a8f0 and 0x40c990) that try to stop a list of services and processes. As expected, we have found anti-malware solutions (e.g., "DefWatch") and database processes (e.g., "sqlservr") in that list.

Additionally, we noticed that the name of one of the targeted services is misspelled. In particular, "vmware-usbarbitator64" is missing an 'r' (and should instead be "vmware-usbarbitrator64"). Interestingly, this typographical error was found in another ransomware family, MedusaLocker. This indicates that developers reuse code from other families [80, 81]. We are unaware of whether this is due to the same actor developing both families, or due to code reuse from one to another (though we have not found evidence of the source code of MedusaLocker being leaked). Indeed, we noticed that the Tactics, Techniques and Procedures (TTPs) of Avaddon are very similar to those of MedusaLocker if we compare our analysis with the report on MedusaLocker from Carbon Black's Threat Analysis Unit [80]. This is an interesting fact regarding the attribution of this campaign which might require further investigation if future families share this peculiarity.

3.9. Key generation

One of the most critical parts of a ransomware campaign is the encryption process. The keys used, how they are imported or generated, how they are exported, the encryption algorithm chosen, etc., are important decisions for malware developers. An error in this process may allow analysts to develop measures to recover encrypted files, completely neutralizing the campaign revenues. In the case of Avaddon, two keys are used in the encryption process in a so-called hybrid scheme. One key (the session key) is randomly generated in each execution and used to encrypt the files in the system. This key is used in a symmetric encryption scheme, AES256. Therefore, the same key must be used to decrypt the affected files. The second key is a public one, part of an asymmetric scheme, RSA. This key is imported (it is present in the PE file) and used only to encrypt the previously generated session key. Therefore, the session key can only be decrypted by the malware authors, since the private key of the asymmetric scheme is only known by them.

The whole process that we described in the previous paragraph is split in three functions in the PE file. These functions, responsible for key management, are located at offsets 0x413600, 0x413a60 and 0x413f50 respectively.

Public key import. The function at 0x413600 is responsible for importing the public key. The import is made by calling the Windows API function *CryptImportKey* with the following parameters:

```
CryptImportKey(hProv:CSP, pbData: Key to be imp
orted, dwDataLen: Length of the key, hPubKey: 0, d
wFlags: 0, phKey: Handle to the imported key after
the call)
```

The key (which is Base64 encoded) is part of a RSA public/private pair. As per the documentation [82], the parameter *hPubKey* must be equal to 0 when the key to be imported is a public key (a *PUBLICKEYBLOB* object). This detail indicates that the imported key is actually the public one of the pair.

Session key generation. After importing the public key, a random key is generated. This randomly generated key (the session key) is then used to encrypt the system. The function responsible of generating the session key is the one located at 0x413f50. To generate the session key, a function from the Windows API is called, *CryptGenKey*, with the following parameters:

```
CryptGenKey(hProv: CSP, Algid: CALG_AES_256, d
wFlags: CRYPT_EXPORTABLE, phKey: Handle to the gen
erated key after the call)
```

The parameter *Algid* indicates that the generated key is to be used in AES256. Additionally, notice that the flags passed to the function indicate that the key must be exportable. Once the key has been generated, it is exported and encrypted using the previously imported RSA key. The result is then included in the ransom note, in order to allow the ransomware operators to recover the encryption key and provide a decryption tool to those victims that decide to pay the ransom fee.

Keys destruction. Finally, the function located at 0x413f50 is the one responsible for securely destroying the keys. This function will destroy the public RSA key and the randomly generated AES256 session key. The purpose of this function is to ensure that they do not remain in memory after being used. However, this function is only called when the process exits, which only occurs when the infected system is shutdown (the ransomware process remains active to encrypt any new file that is created in the system). Therefore, the session key is not destroyed until the system is powered off. This is a mistake from the Avaddon developers since, as long as the computer remains active, the key is kept in memory and it can be retrieved using basic forensics techniques. In Section 4, we will take advantage of this detail to develop a method to recover the symmetric key generated and decrypt all the affected files.

3.10. File encryption

In Section 3.9, we presented the mechanisms implemented in the sample to manage the keys used to encrypt the infected system. We showed that the algorithm used to encrypt files is AES256, a symmetric encryption algorithm. Additionally, we hinted that the session key can be retrieved using basic forensic techniques. However, ransomware authors often apply standard encryption schemes (e.g., AES) in custom procedures (e.g., encrypting only the first part of the files). Therefore, in order to be able to decrypt the affected files, we must first analyze in detail the encryption process implemented in the binary. In this Section, we will describe the process followed to encrypt files in the infected system.

The first step performed by the ransomware is to delete backups so the original files cannot be restored by locally saved security copies. To achieve that goal, the function at 0x41a800 executes the following processes:

- `wmic.exe SHADOWCOPY /nointeractive`
- `wbadmin DELETE SYSTEMSTATEBACKUP`
- `wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest`
- `bcdedit.exe /set {default} recoveryenabled No`
- `bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures`
- `vssadmin.exe Delete Shadows /All /Quiet`

In order to successfully execute those processes, administrator privileges are needed, which are obtained by using the procedure that we described in Section 3.6. Finally, the contents of the recycle bin are deleted by calling the Windows API function *SHEmptyRecycleBinW*.

Next, files are encrypted following a depth-first search approach. Microsoft SQL and Exchange folders are prioritized, being the first ones to be encrypted. Then, the root path is encrypted (i.e., `C:\\*`). Finally, shared folders and mapped volumes are enumerated and encrypted (e.g., `D:\\*`, `Y:\\*`, or `\\VBoxSvr\\shared_folder\\*`). Therefore, the order in which folders are encrypted, following a depth-first approach, is the following:

1. `C:\\Program Files\\Microsoft\\Exchange Server\\*`
2. `C:\\Program Files (x86)\\Microsoft\\Exchange Server\\*`
3. `C:\\ProgramFiles\\Microsoft SQL Server\\*`
4. `C:\\Program Files (x86)\\MicrosoftSQLServer\\*`
5. `C:\\*`
6. Shared folders and mapped volumes

For each encountered file, the process performs three checks before the actual encryption:

1. **Strings from a whitelist**. The path is checked to not contain specific strings (see Appendix B for the list of skipped strings). If the absolute path of the file contains one of those strings, the file is left untouched. This check is not applied to Microsoft SQL and Exchange folders.
2. **File extensions**. Files that have one of the following extensions are not encrypted: `bin`, `ini`, `sys`, `dll`, `lnk`, `dat`, `exe`, `drv`, `rdp`, `prf`, `swp`, `mdf`, `mds` and `sql`.
3. **Already encrypted files**. The third test checks if the file has already been encrypted by Avaddon. To do so, a signature (24 bytes in length) at the end of the file (that is left by the ransomware after encrypting the file) is read. If the file contains the hexadecimal values 0x200 and 0x1030307 at offsets 8 and 16 within the last 24 bytes, it is not encrypteds.

If none of these checks is positive, the file is encrypted. The encryption process is performed by the function located at 0x413bb0. This function receives a copy of the session key (see Section 3.9) and the name of the file to be encrypted. We present a high-level pseudo code (some function signatures have been simplified to improve readability) that summarizes the analyzed function in Algorithm 4. First, the size needed for the buffer to hold the bytes after encryption is calculated (line 1). Then, the contents of the file are read in chunks of 0x100000 bytes (line 5) and encrypted in blocks of 0x2000 bytes (lines 6-11). However, although there exists a loop to read and encrypt the whole file, only the first 0x100000 bytes are encrypted. This is due to the last call to *SetFilePointerEx*, which sets the file pointer to the end of the file (line 18). When there are only 0x2000 or less bytes left to be encrypted (line 13), the last chunk of bytes is encrypted (lines 13-15) and written to the file (line 16). Notice that the parameter *Final* (line 15) in the call to the encryption routine is always set to *False*. This parameter should be *True* if the block to be encrypted is the last block of the file. We will need to take this detail into account in Section 4. Finally, 512 unused bytes and the signature are written at the end of the file to mark it as encrypted (lines 20-22).

Therefore, the process can be summarized as follows:

1. Calculate the size of the buffer needed to hold an encrypted block of 0x2000 (8192) bytes.
2. Obtain the size of the file.
3. Encrypt the first 0x100000 bytes of the file in blocks of 0x2000 (8192) bytes.
4. Write the victim ID (512 bytes) and the signature (24 bytes) at the end of the file.

Here, we show an example of a signature written at the end of an encrypted file:

```
4E 4D 00 00 00 00 00 00 00 02 00 00 01 00 00 00
07 03 03 01 01 01 E2 02
```

Algorithm 4: Procedure for the encryption of a given file.

```
Input: File, file to be encrypted
       Key, a duplicate of the AES256 key
1  buffer_size ← CryptEncrypt(hKey: Key, Final: False, pbData: 0, pdwDataLen: 0x2000);
2  file_size ← GetFileSizeEx(hFile: File);
3  file_pointer ← 0;
4  do
5      bytes_read, number_of_bytes_read ← ReadFile(hFile: File, offset: file_pointer, nNumberOfBytesToRead: 0x100000);
6      i ← 0;
7      do
8          bytes_to_encrypt ← bytes_read[i:i+0x2000];
           // The file is encrypted in blocks of 0x2000 bytes
9          encrypted_bytes ← CryptEncrypt(hKey: Key, Final: False, pbData: bytes_to_encrypt);
10         WriteFile(hFile: File, lpBuffer: encrypted_bytes);
11         i = i + 0x2000;
12     while i ≤ number_of_bytes_read - 0x2000;
13     if number_of_bytes_read - i < 0x2000 then
14         bytes_to_encrypt ← bytes_read[i:];
15         encrypted_bytes ← CryptEncrypt(hKey: Key, Final: False, pbData: bytes_to_encrypt);
16         WriteFile(hFile: File, lpBuffer: encrypted_bytes);
17     end
18     file_pointer ← SetFilePointerEx(hFile: File, liDistanceToMove: 0, dwMoveMethod: FILE_END) ;
       // This call sets the file pointer to the end of the file. This is done to stop processing more bytes from the file
19 while number_of_bytes_read ≥ 0x100000 && file_pointer < file_size;
20 WriteFile(hFile: File, lpBuffer: VictimID);
   // The Victim ID is written to the end of the file
21 signature ← GetSignature();
22 WriteFile(hFile: File, lpBuffer: signature);
   // The signature is also written at the end
```

First, in orange, the original length of the file is written (0x4e4d or 20045 bytes in this case). Then, in blue, a hard-coded magic number (0x01030307) is written at offset 16. This value is checked prior to encrypting a file, as we discussed earlier in this section.

4. Decryption of infected systems

In Section 3.9, we described the functions responsible for importing, generating and destroying the encryption keys needed by the ransomware. As we pointed out, the session key used for encrypting the system was randomly generated. Additionally, it was encrypted using a public, asymmetric key before being exported. Therefore, we are not able to know the session key in advance (before it is generated) or to decrypt it after it has been exported, since we do not have the associated private key. However, we hinted that the function responsible for destroying the keys was in fact never called until the system was powered off, since the ransomware process remains active in the background to encrypt new files or drives as they are created or connected. Since the keys are not destroyed and the ransomware process does not exit, we are able to recover the generated session key. The only requirement is the memory of the ransomware process (i.e., a full dump). If such dump of the process (or the whole system) has been obtained, we are able to recover the key. This is of paramount importance, since users, upon seeing a ransom note, might be tempted to power off or reboot their systems, therefore loosing the opportunity of obtaining the key and decrypting the files.

In order to recover the key, we leverage the knowledge acquired during the analysis of the ransomware sample (see Section 3) to identify the structure that points to the desired key. When a key is generated by using the Windows cryptography API (i.e., cryptsp.dll and rsaenh.dll) the key is an object of type HCRYPTKEY, which has the following structure [83]:

```
struct HCRYPTKEY
{
    void* CPGenKey;
    void* CPDeriveKey;
    void* CPDestroyKey;
    void* CPSetKeyParam;
    void* CPGetKeyParam;
    void* CPExportKey;
    void* CPImportKey;
    void* CPEncrypt;
    void* CPDecrypt;
    void* CPDuplicateKey;
    HCRYPTPROV hCryptProv;
    magic_s *magic;
};
```

The first 10 fields of the structure point to functions of the Windows API. The eleventh field, *hCryptProv*, points to the provider of the key and the aforementioned functions (this provider must be first acquired before the key is generated via *CryptAcquireContext* or a similar function). Finally, the last field points to another structure. This pointer is XOR-ed with a constant value, 0xE35A172C. After XOR-ing that pointer with the aforementioned constant, it points to the following structure:

```
struct magic_s
{
    key_data_s *key_data;
};
```

which contains a pointer to the following structure:

```
struct key_data_s
{
    void *unknown;
    uint32_t alg;
    uint32_t flags;
    uint32_t key_size;
    void* key_bytes;
};
```

The *key_data_s* structure contains three fields whose values are known:

- *alg*: contains the ID of the algorithm for which the key has been generated. In this case, its value is 0x00006610, which corresponds to the ID of AES256 [84].
- *flags*: contains the value of the *flags* parameter passed in the call to *CryptGenKey* at 0x48f024. Therefore, its value is 0x00000001.
- *key_size*: contains the size of the key. In this case, the key has 32 bytes (0x00000020).

Finally, the fifth field contains a pointer to the actual key. Since we know the value of 24 of the last 28 bytes that form the structure (skipping the first field) we can search for this 28-byte pattern in the memory of the process. We thus are able to obtain a pointer to the session key that was used to encrypt the system. We recall that the only requisite is that the system has not been powered off since it was infected, in order to maintain the key in memory.

Now that we have recovered the symmetric key generated by the ransomware, we are able to decrypt the infected files. To do so, we have to reverse the operations performed by the ransomware (which we detailed in Algorithm 4). To decrypt any given file, we first parse the signature at the end of the file. There, we obtain the original size of the encrypted file. Then, we truncate the file to eliminate both the signature and the block of 512 bytes appended at the end of the file by the ransomware (536 bytes in total, since the signature is 24 bytes in length). Once we have truncated the file, we proceed to decrypt the first 0x100000 bytes in blocks of 8192 (0x2000) bytes. Notice that, as we showed in Algorithm 4, the *Final* parameter in the calls to *CryptEncrypt* was never set to *True*. According to the documentation, this parameter should be *True* when the last block is encrypted. Although we do not know if this nonstandard behavior is intentional or not, we are forced to do the same in the decryption routine. Therefore, we always set the Final parameter to be *False* in the calls to *CryptDecrypt*. After decrypting the first 0x100000 bytes, we copy the rest of the file as is. Finally, if the file was smaller than 0x100000 bytes, we truncate it once again, now to the original size recovered earlier from the signature appended at the end, to remove the padding bytes.

Obtaining a memory dump of a process can be done by standard forensic tools. In our public repository, we open source a tool that implements the developed procedure to recover the session key from memory and decrypt the infected files:
`https://github.com/JavierYuste/AvaddonDecryptor`.

5. Experimentation

We tested our proposal in two virtual systems. In the first system, we installed a Windows 7 Professional Service Pack 1 x64 OS. In the second one, we installed a Windows 10 Enterprise Evaluation 10.0.19042 x64 OS. We built those systems on top of VirtualBox, in a computer with a 1.60 GHz Intel Core i5-8250U CPU and 16 GB RAM. From the available hardware, we assigned 2 cores and 4 GB of RAM to the first system (Windows 7) and 4 cores and 8 GB of RAM to the second system (Windows 10). In each machine, we installed different software packages (e.g., Python, Visual Studio Code, Teams, etc.) in order to make them appear as legitimate machines and increase the number of files that would be encrypted by the ransomware. Then, we executed different versions of Avaddon in the virtual machines and let them encrypt the whole system. When Avaddon had not utilized more than 0.5% of the CPU time in the last 60 seconds, we stopped the execution and confirmed the infection due to the presence of ransom notes and encrypted files through the whole file system. For our experiments, we decided to test 5 versions of Avaddon that had been released as different times, spanning from June, 2020 to January, 2021.

After infecting the virtual machines, we proceeded to decrypt all the affected files. First, we suspended the ransomware process using Process Explorer, a tool from the SysInternals suite.[11] Note that we can freely drop executable files in the system before stopping Avaddon, since files with an *exe* extension are not encrypted (see Section 3.10). Once the process has been suspended, we can safely operate in the infected system. Next, we dumped the memory of the ransomware process with ProcDump, which is also part of the SysInternals suite. Finally, we executed the developed decryption tool. This tool i) confirms the infection by extracting the signature appended at the end of encrypted files, ii) obtains the AES256 session key from the dumped memory of the ransomware process and iii) decrypts the whole file system.

We show the results of our experiments in Table 2. For each experiment, we report: the MD5 hash of the tested Avaddon sample (MD5); the month when each sample was first seen in VirusTotal (First seen); whether the decryption was successful (Yes) or not (No) for the whole file system (Decryption); the number of files that were present in the system (Total files); the number of files that were decrypted (Decrypted files); and the CPU time (in seconds) needed by our tool to decrypt all the affected files (Time). As we can observe, the decryption was successful in all the experiments (i.e., all infected files were successfully decrypted). In the Windows 7 system, our tool decrypted the whole system in 390.91 seconds in average. In the Windows 10 machine, decryption of the whole system was achieved in 65.90 seconds in average. Notice that there is a difference in the number of files of both systems, which is partially explained by a difference in the set of software packages installed in each system. Additionally, notice that we tested our tool with the most recent version of Avaddon, which was observed from a wild URL on mid-January 2021, when this paper was written. We confirm that the decryptor still works, since we were able to decrypt all the infected files.

We must note some considerations. First, it is important to not turn off the computer after infection, since the proposed approach needs the encryption key to be present in memory. Otherwise, the session key would be destroyed and could only be recovered by means of the official channel proposed by the criminals, i.e. paying the ransom. Second, the proposed tool needs the original version of at least one encrypted file to find the correct symmetric key. This, however, can be easily achieved, e.g. by obtaining known files present by default in the Windows OS version installed in the affected system. Alternatively, a dummy file might be dropped to the infected system while the ransomware process is running (recall that the ransomware process remains active to encrypt new files that are created in the system).

6. Conclusions

Current approaches of cybercrime specialization, including new malware techniques, increase the threat of modern ransomware campaigns. In this work, we have analyzed a new ransomware, Avaddon, operated as a RaaS in a shared profit scheme, first seen in June, 2020. Avaddon incorporates two extortion techniques, which are growing in popularity, aimed at increasing their financial revenues: i) leaking the personal data of their victims if they do not pay the ransom fee, and ii) conducting DDoS attacks against their victims until the ransom fee is paid. Following this procedure, personal data from 62 infected organizations

[11]`https://docs.microsoft.com/en-us/sysinternals/`

MD5	First seen	OS	Decryption	Total files	Decrypted files	Time (s)
c9ec0d9ff44f445ce5614 cc87398b38d	June 2020	Windows 7	Yes	209768	9131	349.92
6ff1ca648505fe8bea6b4 a26616b9722	July 2020	Windows 7	Yes	207305	9130	542.32
275e4a63fc63c995b3e0d 464919f211b	August 2020	Windows 7	Yes	209756	9130	348.79
a2c57182efe72c6ce43f0 2a8f709e857	November 2020	Windows 7	Yes	209022	9132	317.85
4d6ef550cecc0bd988383 3608dd16a00	January 2021	Windows 7	Yes	209305	9132	395.66
c9ec0d9ff44f445ce5614 cc87398b38d	June 2020	Windows 10	Yes	264972	285	77.10
6ff1ca648505fe8bea6b4 a26616b9722	July 2020	Windows 10	Yes	264895	285	81.10
275e4a63fc63c995b3e0d 464919f211b	August 2020	Windows 10	Yes	264952	285	70.38
a2c57182efe72c6ce43f0 2a8f709e857	November 2020	Windows 10	Yes	346368	287	55.78
4d6ef550cecc0bd988383 3608dd16a00	January 2021	Windows 10	Yes	346658	285	45.13

Table 2: Results of the decryption experiments for different Avaddon samples and Windows OS.

has already been published online. While having proper attribution is difficult, our analysis suggests that the threat actor behind Avaddon is from a CIS country. Indeed, the initial announcement of the ransomware was made in a Russian underground forum, and it implements a policy to prevent infection of CIS-based victims. Moreover, a typographical error found in the name of one of the processes targeted by Avaddon suggests that this family is related to a previous ransomware, i.e. MedusaLocker, where the same error is present. Furthermore, the *modus operandi* of Avaddon, that we detailed in this work, is similar to that of MedusaLocker and the list of services to stop is almost identical in both cases.

By examining a sample obtained from the first campaign of Avaddon, we took a grasp on the general "Cyber Kill Chain" of ransomware threats (land, escalate privileges, deactivate defenses, acquire persistence, delete backups and encrypt files) and obtained a detailed analysis of this ransomware in particular. Using an hybrid key scheme, Avaddon attempts to hide the session key from defenders. However, thanks to the aforementioned analysis of this ransomware family, we have developed a method to recover the session key from the memory of infected systems and decrypt all the affected files. We confirm that, at the time of writing, the decryption tool works with the newest variants of the ransomware. The only requirement for this method to work is that the victim's computer has not been powered off after the infection.

Due to the novelty of this ransomware, the business model following a shared profits scheme and the ability to extort and blackmail victims, it is likely to expect new variants of Avaddon and similar ransomware families improving their mechanisms and growing in popularity in the future. Thus, we believe that the analysis and tools provided in this paper can contribute to guide future analyses of ransomware threats and improve existing mitigation and detection mechanisms.

Acknowledgements

We thank the anonymous reviewers for their valuable comments. This work was supported by the Spanish grants ODIO (PID2019-111429RB-C21 and PID2019-111429RB), the Ministerio de Ciencia, Innovación y Universidades (Ref. PGC2018-095322-B-C22) and by the Region of Madrid grants CYNAMON-CM (P2018/TCS-4566), co-financed by European Structural Funds ESF and FEDER, and Excellence Program EPUC3M17. The opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect those of any of the funders.

References

[1] T. C. of Economic Advisers, The Cost of Malicious Cyber Activity to the U.S. Economy, https://www.whitehouse.gov/wp-content/uploads/2018/02/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf, [Online; accessed 28-September-2020] (2 2018).

[2] B. Collier, R. Clayton, A. Hutchings, D. Thomas, Cybercrime is (often) boring: maintaining the infrastructure of cybercrime economies, 2020, workshop on the Economics of Information

Security, WEIS ; Conference date: 14-12-2020 Through 15-12-2020.

[3] National Intelligence Officer, A Guide to Cyber Attribution, https://www.dni.gov/files/CTIIC/documents/ODNI_A_Guide_to_Cyber_Attribution.pdf, [Online; accessed 09-October-2020] (9 2018).

[4] Infosec, The Attribution Problem in Cyber Attacks, https://resources.infosecinstitute.com/attribution-problem-in-cyber-attacks/, [Online; accessed 09-October-2020] (2 2013).

[5] K. Huang, M. Siegel, S. Madnick, Systematically understanding the cyber attack business: A survey, ACM Computing Surveys 51 (2018) 1–36. doi:10.1145/3199674.

[6] S. Pastrana, A. Hutchings, A. Caines, P. Buttery, Characterizing eve: Analysing cybercrime actors in a large underground forum, in: International symposium on research in attacks, intrusions, and defenses, Springer, 2018, pp. 207–227.

[7] PandaLabs, PandaLabs Reveals its Predictions for Cybersecurity Trends in 2018, https://www.pandasecurity.com/mediacenter/pandalabs/annual-report-cybersecurity-predictions-2018/, [Online; accessed 28-September-2020] (11 2017).

[8] R. Van Wegberg, S. Tajalizadehkhoob, K. Soska, U. Akyazi, C. H. Ganan, B. Klievink, N. Christin, M. Van Eeten, Plug and prey? measuring the commoditization of cybercrime via online anonymous markets, in: 27th {USENIX} security symposium ({USENIX} security 18), 2018, pp. 1009–1026.

[9] Auld, Andy, What's behind the increase in ransomware attacks this year?, https://www.pwc.co.uk/issues/cyber-security-services/insights/what-is-behind-ransomware-attacks-increase.html, [Online; accessed 03-October-2020] (2020).

[10] S. Ghafur, S. Kristensen, K. Honeyford, G. Martin, A. Darzi, P. Aylin, A retrospective impact analysis of the wannacry cyberattack on the nhs, NPJ digital medicine 2 (1) (2019) 1–7.

[11] The CrowdStrike Intel Team, Double Trouble: Ransomware with Data Leak Extortion, Part 1, https://www.crowdstrike.com/blog/double-trouble-ransomware-data-leak-extortion-part-1/, [Online; accessed 28-September-2020] (9 2020).

[12] Panda security, Ransomware has a new trick: pay up or suffer a data breach, https://www.pandasecurity.com/mediacenter/security/ransomware-data-breach-blackmail/, [Online; accessed 28-September-2020] (3 2020).

[13] C. Cimpanu, Conti (Ryuk) joins the ranks of ransomware gangs operating data leak sites, https://www.zdnet.com/article/conti-ryuk-joins-the-ranks-of-ransomware-gangs-operating-data-leak-sites/, [Online; accessed 28-September-2020] (8 2020).

[14] M. J. Schwartz, Ransomware + Exfiltration + Leaks = Data Breach, https://www.bankinfosecurity.com/blogs/ransomware-exfiltration-leaks-data-breach-p-2913, [Online; accessed 28-September-2020] (7 2020).

[15] Intel471, Ransomware-as-a-service: The pandemic within a pandemic, https://intel471.com/blog/ransomware-as-a-service-2020-ryuk-maze-revil-egregor-doppelpaymer/, [Online; accessed 18-December-2020] (2020).

[16] D. R. Thomas, S. Pastrana, A. Hutchings, R. Clayton, A. R. Beresford, Ethical issues in research using datasets of illicit origin, in: Proceedings of the 2017 Internet Measurement Conference, IMC '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 445–462. doi:10.1145/3131365.3131389.
URL https://doi.org/10.1145/3131365.3131389

[17] S. Tripathi, Avaddon Ransomware, https://www.subexsecure.com/pdf/malware-reports/June-2020/Avaddon_Ransomware.pdf, [Online; accessed 22-September-2020] (6 2020).

[18] A. Ivanov, Avaddon Ransomware, https://id-ransomware.blogspot.com/2020/06/avaddon-ransomware.html, [Online; accessed 14-October-2020] (6 2020).

[19] H. Security, Avaddon: From seeking affiliates to in-the-wild in 2 days, https://www.hornetsecurity.com/en/security-information/avaddon-from-seeking-affiliates-to-in-the-wild-in-2-days/, [Online; accessed 23-August-2020] (6 2020).

[20] M. Malubay, Ransom.Win32.AVADDON.YJAF-A, https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/Ransom.Win32.AVADDON.YJAF-A, [Online; accessed 22-September-2020] (6 2020).

[21] A. Zimba, M. Chishimba, Understanding the evolution of ransomware: paradigm shifts in attack structures, International Journal of computer network and information security 11 (1) (2019) 26.

[22] B. A. S. Al-rimy, M. A. Maarof, S. Z. M. Shaid, Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions, Computers & Security 74 (2018) 144–166.

[23] R. Brewer, Ransomware attacks: detection, prevention and cure, Network Security 2016.

[24] J. Bates, Trojan horse: Aids information introductory diskette version 2.0 (January 1990). doi:https://www.virusbulletin.com/uploads/pdf/magazine/1990/199001.pdf.

[25] A. Young, M. Yung, Cryptovirology: Extortion-based security threats and countermeasures, in: Proceedings 1996 IEEE Symposium on Security and Privacy, IEEE, 1996, pp. 129–140.

[26] CORE Security, Understanding the evolution of ransomware, https://www.coresecurity.com/core-labs/articles/understanding-evolution-ransomware, [Online; accessed 02-Jun-2021] (2021).

[27] P. Bajpai, A. K. Sood, R. Enbody, A key-management-based taxonomy for ransomware, in: 2018 APWG Symposium on Electronic Crime Research (eCrime), IEEE, 2018, pp. 1–12.

[28] M. Humayun, N. Jhanjhi, A. Alsayat, V. Ponnusamy, Internet of things and ransomware: Evolution, mitigation and prevention, Egyptian Informatics Journal 22 (1) (2021) 105–117. doi:https://doi.org/10.1016/j.eij.2020.05.003.
URL https://www.sciencedirect.com/science/article/pii/S1110866520301304

[29] K. Zetter, What Is Ransomware? A Guide to the Global Cyberattack's Scary Method, https://www.wired.com/2017/05/hacker-lexicon-guide-ransomware-scary-hack-thats-rise/, [Online; accessed 16-October-2020] (5 2017).

[30] D. Y. Huang, M. M. Aliapoulios, V. G. Li, L. Invernizzi, E. Bursztein, K. McRoberts, J. Levin, K. Levchenko, A. C. Snoeren, D. McCoy, Tracking ransomware end-to-end, in: 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 618–631.

[31] S. Pastrana, G. Suarez-Tangil, A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth, in: Proceedings of the Internet Measurement Conference, IMC '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 73–86.

[32] R. Richardson, M. North, Ransomware: Evolution, mitigation and prevention, International Management Review 13 (2017) 10.

[33] McAfee Labs, Understanding ransomware and strategies to defeat it, https://www.mcafee.com/enterprise/en-us/assets/white-papers/wp-understanding-ransomware-strategies-defeat.pdf, [Online; accessed 24-May-2020] (2016).

[34] Unit 42. Palo Alto Networks, Ransomware Threat Report, https://www.paloaltonetworks.com/content/dam/pan/en_US/assets/pdf/reports/Unit_42/unit42-ransomware-threat-report-2021.pdf, [Online; accessed 15-April-2021] (2020).

[35] A. Rege, Critical infrastructure ransomware incident dataset, https://sites.temple.edu/care/downloads/, version 10.9. Funded by National Science Foundation CAREER Award #1453040 (2021).

[36] S. Ranger, Ransomware victims are paying out millions a month. one particular version has cost them the most, https://www.zdnet.com/article/fbi-ransomware-victims-have-paid-out-140-million-one-version-has-cost-them-the-most/, [Online; accessed 19-April-2021] (3 2020).

[37] P. Bajpai, R. Enbody, An Empirical Study of Key Generation in Cryptographic Ransomware, in: International Conference on Cyber Security and Protection of Digital Services, Cyber

Security 2020, Institute of Electrical and Electronics Engineers Inc., 2020.

[38] E. Kolodenker, W. Koch, G. Stringhini, M. Egele, Paybreak: Defense against cryptographic ransomware, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, 2017, pp. 599–611.

[39] P. Bajpai, R. Enbody, Attacking key management in ransomware, IT Professional 22 (2) (2020) 21–27.

[40] M. Nauman, N. Azam, J. Yao, A three-way decision making approach to malware analysis using probabilistic rough sets, Information Sciences 374 (2016) 193–209.

[41] M. Egele, T. Scholte, E. Kirda, C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, ACM computing surveys (CSUR) 44 (2) (2008) 1–42.

[42] A. M. Maigida, M. Olalere, J. K. Alhassan, H. Chiroma, E. G. Dada, et al., Systematic literature review and metadata analysis of ransomware attacks and detection mechanisms, Journal of Reliable Intelligent Environments 5 (2) (2019) 67–89.

[43] M. M. Ahmadian, H. R. Shahriari, S. M. Ghaffarian, Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares, in: 2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), IEEE, 2015, pp. 79–84.

[44] N. Andronio, S. Zanero, F. Maggi, Heldroid: Dissecting and detecting mobile ransomware, in: international symposium on recent advances in intrusion detection, Springer, 2015, pp. 382–404.

[45] N. Hampton, Z. Baig, S. Zeadally, Ransomware behavioural analysis on windows platforms, Journal of information security and applications 40 (2018) 44–51.

[46] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, E. Kirda, Cutting the gordian knot: A look under the hood of ransomware attacks, in: M. Almgren, V. Gulisano, F. Maggi (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, 2015, pp. 3–24.

[47] K. P. Subedi, D. R. Budhathoki, D. Dasgupta, Forensic analysis of ransomware families using static and dynamic analysis, in: 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 180–185.

[48] K. P. Prakash, T. Nafis, S. Biswas, Preventive measures and incident response for locky ransomware, International Journal of Advanced Research in Computer Science 8 (2017) 392–395.

[49] A. Gazet, Comparative analysis of various ransomware virii, Journal in computer virology 6 (1) (2010) 77–90.

[50] M. Akbanov, V. Vassilakis, Wannacry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms, Journal of Telecommunications and Information Technology 1 (2019) 113–124.

[51] D.-Y. Kao, S.-C. Hsiao, The dynamic analysis of wannacry ransomware, in: 2018 20th International Conference on Advanced Communication Technology (ICACT), IEEE, 2018, pp. 159–166.

[52] P. Pathak, Y. M. Nanded, A dangerous trend of cybercrime: ransomware growing challenge, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) 5 (2) (2016) 371–373.

[53] C. Le Guernic, A. Legay, Ransomware and the legacy crypto api, in: Risks and Security of Internet and Systems: 11th International Conference, CRiSIS 2016, Roscoff, France, September 5-7, 2016, Revised Selected Papers, Vol. 10158, Springer, 2017, p. 11.

[54] G. Hull, H. John, B. Arief, Ransomware deployment methods and analysis: views from a predictive model and human responses, Crime Science 8 (2019) 1–22.

[55] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, E. Kirda, UNVEIL: A large-scale, automated approach to detecting ransomware, in: 25th USENIX Security Symposium (USENIX Security 16), USENIX Association, 2016, pp. 757–772.
URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz

[56] D. Sgandurra, L. Muñoz-González, R. Mohsen, E. C. Lupu, Automated dynamic analysis of ransomware: Benefits, limitations and use for detection (2016). arXiv:1609.03020.

[57] R. Vinayakumar, K. P. Soman, K. K. Senthil Velan, S. Ganorkar, Evaluating shallow and deep networks for ransomware detection and classification, in: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, pp. 259–265.

[58] K. Lee, S. Lee, K. Yim, Machine learning based file entropy analysis for ransomware detection in backup systems, IEEE Access 7 (2019) 110205–110215.

[59] J. Caballero, C. Grier, C. Kreibich, V. Paxson, Measuring pay-per-install: the commoditization of malware distribution., in: Usenix security symposium, Vol. 13, 2011.

[60] Kaspersky, xDedic – the shady world of hacked servers for sale, https://securelist.com/xdedic-the-shady-world-of-hacked-servers-for-sale/75027/, [Online; accessed 04-February-2021] (6 2016).

[61] R. Bhalerao, M. Aliapoulios, I. Shumailov, S. Afroz, D. McCoy, Mapping the underground: supervised discovery of cybercrime supply chains, in: 2019 APWG Symposium on Electronic Crime Research (eCrime), IEEE, 2019, pp. 1–16.

[62] PintSizeNore, AVADDON Ransomware (.avdn; [id]-readme.html) Support Topic, https://www.bleepingcomputer.com/forums/t/724607/avaddon-ransomware-avdn;-id-readmehtml-support-topic/page-2#entry5061940, [Online; accessed 14-October-2020] (09 2020).

[63] M. De Jesus, M. Malubay, A. Christelle Ramos, Ransomware Report: Avaddon and New Techniques Emerge, Industrial Sector Targeted, https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/ransomware-report-avaddon-and-new-techniques-emerge-industrial-sector-targeted, [Online; accessed 22-September-2020] (7 2020).

[64] M. J. Schwartz, Avaddon Ransomware Joins Data-Leaking Club, https://www.bankinfosecurity.com/avaddon-ransomware-joins-data-leaking-club-a-14809, [Online; accessed 22-September-2020] (8 2020).

[65] L. Abrams, Avaddon ransomware launches data leak site to extort victims, https://www.bleepingcomputer.com/news/security/avaddon-ransomware-launches-data-leak-site-to-extort-victims/, [Online; accessed 22-September-2020] (8 2020).

[66] L. Abrams, Avaddon ransomware launches data leak site to extort victims, https://www.bleepingcomputer.com/news/security/another-ransomware-now-uses-ddos-attacks-to-force-victims-to-pay/, [Online; accessed 03-February-2021] (1 2021).

[67] Emsisoft, Urgently Needed! Avaddon ransomware (.avdn), https://support.emsisoft.com/topic/33623-urgently-needed-avaddon-ransomware-avdn/, [Online; accessed 21-October-2020] (2020).

[68] B. Computer, AVADDON Ransomware (.avdn; [id]-readme.html) Support Topic, https://www.bleepingcomputer.com/forums/t/724607/avaddon-ransomware-avdn;-id-readmehtml-support-topic/page-2, [Online; accessed 21-October-2020] (2020).

[69] Microsoft, PE Format, https://docs.microsoft.com/en-us/windows/win32/debug/pe-format, [Online; accessed 01-October-2020] (2020).

[70] C. Q. Nguyen, J. E. Goldman, Malware analysis reverse engineering (mare) methodology & malware defense (md) timeline, in: 2010 Information Security Curriculum Development Conference, 2010, pp. 8–14.

[71] J. Bermejo Higuera, C. Abad Aramburu, J.-R. Bermejo Higuera, M. A. Sicilia Urban, J. A. Sicilia Montalvo, Systematic approach to malware analysis (sama), Applied Sciences 10 (4) (2020) 1360.

[72] P. V. Sabanal, M. V. Yason, Reversing C++, in: Black Hat DC, 2007.

[73] hfiref0x2017, UAC bypass using CMSTPLUA COM interface, https://gist.github.com/api0cradle/d4aaef39db0d845627d819b2b6b30512, [Online; accessed 31-August-2020] (2017).

[74] A. Osipov, Trickbot Trojan leveraging a new Windows 10 UAC bypass, `https://blog.morphisec.com/trickbot-uses-a-new-windows-10-uac-bypass`, [Online; accessed 31-August-2020] (2020).

[75] S. in bits, UAC bypass analysis (Stage 1) Ataware Ransomware – Part 0x2, `https://www.securityinbits.com/malware-analysis/uac-bypass-analysis-stage-1-ataware-ransomware-part-2/`, [Online; accessed 31-August-2020] (2019).

[76] Microsoft, EnableLUA, `https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-gpsb/958053ae-5397-4f96-977f-b7700ee461ec`, [Online; accessed 21-July-2020] (2019).

[77] Microsoft, ConsentPromptBehaviorAdmin, `https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-gpsb/341747f5-6b5d-4d30-85fc-fa1cc04038d4`, [Online; accessed 21-July-2020] (2019).

[78] Microsoft, Mapped drives are not available from an elevated prompt when UAC is configured to "Prompt for credentials" in Windows, `https://support.microsoft.com/en-us/help/3035277/mapped-drives-are-not-available-from-an-elevated-prompt-when-uac-is-co`, [Online; accessed 21-July-2020] (2015).

[79] Lockheed Martin, The Cyber Kill Chain, `https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html`, [Online; accessed 08-October-2020].

[80] B. Baskin, TAU Threat Analysis: Medusa Locker Ransomware, `https://www.carbonblack.com/blog/tau-threat-analysis-medusa-locker-ransomware/`, [Online; accessed 19-October-2020] (June 2020).

[81] A. Zsigovits, Ransomware-LockBit, `https://github.com/sophoslabs/IoCs/blob/master/Ransomware-LockBit`, [Online; accessed 19-October-2020] (2020).

[82] Microsoft, CryptImportKey function, `https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptimportkey`, [Online; accessed 27-August-2020] (2018).

[83] Sasza, Structure of HCRYPTKEY Data, `https://forums.codeguru.com/showthread.php?79163-Structure-of-HCRYPTKEY-Data`, [Online; accessed 26-September-2020] (2020).

[84] Microsoft, ALG_ID, `https://docs.microsoft.com/en-us/windows/win32/seccrypto/alg-id`, [Online; accessed 26-September-2020] (2018).

Appendices

A. List of IOCs

Sample 1 (June, 2020):

- MD5: `c9ec0d9ff44f445ce5614cc87398b38d`
- SHA-1: `591ffe54bac2c50af61737a28749ff8435168182`
- SHA-256: `05af0cf40590aef24b28fa04c6b4998b7ab3b7f26e60c507adb84f3d837778f2`
- Vhash: `016046655d156173z12z92z23z8065z23z21z71z67z`
- Authentihash: `18a501e209bca5ffd0c84763bb167b1524f3db86d5d6d2e926051b135a5fceed`
- Imphash: `1156e59d43883136ef73eee451e94e3d`
- Rich PE header hash: `1f751e2aac4a31991712f656456c5442`
- SSDEEP: `24576:Cs6JmdFn5KLOCgHWcAvcrOcEsKfR9uA7rmFbbbbpccf:Cs6JY5KLOCyWcDUfRAA3mFbbbbpc4`
- TLSH: `T152358D3DB4E1C071C73000F05998B7B2996EA9D2CB7204C77B8C9A9B1BB15D9A9375B3`
- Domains contacted:
  - api.myip.com

Sample 2 (July, 2020):

- MD5: `6ff1ca648505fe8bea6b4a26616b9722`
- SHA-1: `7020b4d9e700b697d507a61bffea12c9475a23d2`
- SHA-256: `7b7c16367746efe7583ae46235b2f062ce44602dda990c9a11a730d619b8d365`
- Vhash: `056096655d15156d1c1f6013z13z11z13z1015z13z11z11z17z`
- Authentihash: `65543745fea1a4726410bc447c2bb0c7d170fbd4dc442339f8112ada01ca7580`
- Imphash: `b1ea5fd53e7480d5e00ebc689ced94b3`
- Rich PE header hash: `548892964409124441836e225a26693b`
- SSDEEP: `98304:zDAjjvoF+Cp+/bbbbp7FO1gTL9M5gmoZHOoOVsHalI:zuvAObbbbp78+VwzV0alI`
- TLSH: `T135365CE5B525A1CFD29E07B4E1DACE42982E43F4C7210843B85C757E6FA2CC219D7E29`
- Domains contacted:
  - api.myip.com

Sample 3 (August, 2020):

- MD5: `275e4a63fc63c995b3e0d464919f211b`
- SHA-1: `51d85210c2f621ca14d92a8375ee24d62f9d7f44`
- SHA-256: `cc95a8d100f70d0fbf4af14e852aa108bdb0e36db4054c3f60b3515818a71f46`
- Vhash: `075056651d15156143z12z921z33z5065z2bz87z`
- Authentihash: `7a5cd88dd9f74d9c66714915008a6857525b290de804949476af863a30209401`
- Imphash: `ebcba21b169b4d31880471f7ee399c34`
- Rich PE header hash: `ee5d076f12788d6b5adc6068e849bd8c`

- SSDEEP: `12288:OR8hjUV679Aa4Auw3gveB17cOT1WHWEQTe0udkuHgCNU7SY/qgjjmJ/:quK679Aa4Auw3gveB1TGWEQSzXY/tjq/`
- TLSH: `T15CF49E317D82C077E46A41304E98A7B594BEF8724B320DDB67C86B1D5E706E26E31A73`
- Domains contacted:
  - api.myip.com

Sample 4 (November, 2020):

- MD5: `a2c57182efe72c6ce43f02a8f709e857`
- SHA-1: `3ee2fc7d9367e6e94d4ce859b8461cbbaf6ec27f`
- SHA-256: `721f36f4c79c813a7bc4410ac6052c5974cba096d84294b5656b6c26d85e095b`
- Vhash: `085066657d1d1d056018z55`
- Authentihash: `ce876506850b3fea3e7e0dce630ce0dda041fbe6020b215c92e55f45fa6fa4ae`
- Imphash: `d99a658d2260a0adef1074cf8db5e8c0`
- Rich PE header hash: `8b803b5e91419edf8059ccf169a08d0d`
- SSDEEP: `12288:5XPbnVSHco8Tv8CjSmcK/R6lmYh2DtRrKHlbYwc:xbnYI8Cj6K/slDi3E6Z`
- TLSH: `T1E30512123792D032C4672971AD60B1B25BFAFEB116BBC05B37442B3D5F616E09B7231A`
- Domains contacted:
  - api.myip.com

Sample 5 (January, 2021):

- MD5: `4d6ef550cecc0bd9883833608dd16a00`
- SHA-1: `85cbe22635f92114032d74a3c7c4b56e1492e0c2`
- SHA-256: `1e8df42c2e51f919886eaf955c8fc9630ba9aca8bba47b1541ead131feb55a11`
- Vhash: `075056651d15156163z12z921z33z5065z2bz87z`
- Authentihash: `ca896eba710b4e10ed14bf9b640134a60f8d6519ab2da0a8e10b7b9f5759547b`
- Imphash: `8634a890637b58f527c95218636740c9`
- Rich PE header hash: `561fead76887381034ebdaa8086436be`
- SSDEEP: `12288:w8fM15LL43eYsxN2VH/h51UtfiA+fEJJrR4wPQAReV3foOBuue9vL+Fmk3:w8fMjE36N2VH/h51UtfiAuyhdPQARsgi`
- TLSH: `T1F9F48C223A83C03FD97201368E98BAB541BEE8754B7709D7A3D82F5D4E305D25E31A67`
- Domains contacted:
  - api.myip.com

B. List of skipped strings in the encryption process

“C:\Program Files\Microsoft\Exchange Server”
“C:\Program Files (x86)\Microsoft\Exchange Server”
“C:\Program Files\Microsoft SQL Server”
“C:\Program Files (x86)\Microsoft SQL Server”
“C:\Windows”
“C:\Program Files”
“C:\Users\All Users”
“C:\Users\Public”
“C:\Users\%User Profile%\AppData\Local\Temp”
“C:\Program Files (x86)”
“C:\Users\%User Profile%\AppData”
“C:\ProgramData”
“Tor Browser”
“AppData”
“ProgramData”
“Program Files”
“Windows”
Name of the ransom note (e.g., “363053-readme.html”)
“bckgrd.bmp”

Table 3: List of whitelisted strings in the encryption process.