Trends in design of ransomware viruses

Vlad Constantin Craciun[1], Andrei Mogage[2], and Emil Simion[3]

[1]Department of Computer Science, UAIC IASI: vcraciun@info.uaic.ro, mogage.andrei.catalin@info.uaic.ro
[2]Bitdefender: vcraciun@bitdefender.com, amogage@bitdefender.com
[3]University Politehnica of Bucharest, emil.simion@upb.ro

Keywords: cyber threat · ransomware · cryptography · cyber security

1 Abstract

The ransomware nightmare is taking over the internet impacting common users, small businesses and large ones. The interest and investment which are pushed into this market each month, tells us a few things about the evolution of both technical and social engineering and what to expect in the short-coming future from them. In this paper we analyze how ransomware programs developed in the last few years and how they were released in certain market segments throughout the deep web via RaaS, exploits or SPAM, while learning from their own mistakes to bring profit to the next level. We will also try to highlight some mistakes that were made, which allowed recovering the encrypted data, along with the ransomware authors preference for specific encryption types, how they got to distribute, the silent agreement between ransomwares, coin-miners and bot-nets and some edge cases of encryption, which may prove to be exploitable in the short-coming future.

2 Introduction

The ransomware phenomenon has seen an alarmingly increase in the last couple of years, along with its methods of distribution and infection. However, no matter how exotic such an application might look, they all reduce to some common features: spreading, encryption, key sharing and key building. Based on the large number of ransomware samples collected and analyzed in the past few years at Bitdefender Labs, we can bring to surface some of these features and where possible, additional environmental specs. They change as they evolve in a continuously updating chain, to maximize the creators profit and minimize the effort pushed into this business. We can actually see how these threats get more and more complex, through a silent agreement with malicious services provides on black market (Ex: packers, bot-net delivery, spam delivery, etc.). Starting with the fall of 2016 we assist to some visible changes and agreements which take place:

- Troldesh / Crysis and another few ransomwares are now only distributed manually by hackers after they brute-force accounts of company users

- A trend in automating the monetizing system is to have a Ransomware As a Service for your ransomware, so common people will enroll to make money with a given ransomware
- More ransomwares make use of a well known bot-net (Emotet) packer to better protect their payload. This packer is known for its polymorphic cloud infrastructure
- Recent bot-nets just became a service providers for other threat creators and some of them like CoreBot is able to tell ransomware owners if the systems they were deployed on, belongs to companies or not in order to deploy the ransomware where their authors are guaranteed a high percent for the ransom payment
- By the fall of 2017 we have seen a mix between crypto currency miners and ransomwares and UIWIX is such a ransomware which at that time was deployed in certain conditions by Adylkuzz coinminer which among others like it, used the EternalBlue SMB exploit to spread in local area networks

3 Ransomware choice for spreading

Various means of distribution of infection have been seen throughout the entire ransomware phenomenon. One of the easiest and most common way is via spam. This includes emails, which contain malicious scripts, or documents, which download and run the malicious samples. Another way is to create fake websites (usually which include other phishing techniques) which will trick the user into downloading and running the file. Both of these can be combined with the fileless malware attack, where a malware sample would be loaded directly into memory, leaving no physical trace (i.e. on the disk).

Another frequent method is to exploit various security flaws found at software, operating system or hardware levels. A famous example is WannaCry ransomware, which would be distributed via EternalBlue exploit. This was based on a vulnerability at the Samba protocol, which allowed an attacker to inject malicious code remotely. Another example is the second version of SynAck ransomware, which uses Process Doppelgänging technique.This involves replacing the memory of a legitimate process through exploiting some Windows built-in functions (NTFS transactions – in this case) and how the Windows loader works. Both of them are based on fileless malware attacks.

A method of distribution which occurred especially among the Troldesh ransomware family (which includes Troldesh, XTBL, Dharma, Crysis) is brute-forcing the credentials through Remote Desktop Protocol. Once the attacker obtained access, it would manually disable any security product and then manually run the malicious samples. This also occurs in various cases of botnets and/or coin-miners. What is interesting is that only on various occasions any of these techniques are part of an Advanced Persistent Threat, meaning that the attackers are interested in making as many victims as possible, without restricting their attack to a specific zone or victim. A few exception have been seen on some ransomwares which will not infect victims based on geographic location (i.e. GandCrab excepts Russia IPs).

4 Ransomware choice for encryption

As concluded from our research, most of the ransomwares are using standard algorithms both for encryption (Ex: AES, RSA, Blowfish, RC4) and hashing (SHA256, MD5). What is interesting is that there is a high percent (above 80%) of those who prefer standard libraries, such as OS API or OpenSSL and only a few choosing to have proprietary implementations like Salsa, ChaCha, or slightly changed common algorithms like AES512.

We identify about 4 different key management types among all ransomware families:

- Downloading a secret key/key-stream, to globally encrypt all user files
- Creating a random key to encrypt all user files and uploading it to a server
- Using a constant shipped key easy to recover by reverse-engineering the binary sample, or easy to break encryption algorithms
- Using a shipped RSA public key to encrypt random generated AES keys used for file-encryption
- Using ECDH method (only a few cases)

There is a small number of ransomwares which prefers to upload or download their leys, because it is difficult for the author to ensure that the server will be online at the time of encryption. The bad guys will not risk to get discovered by the authorities by leaving public-available servers which could be sized for investigations. They also cannot risk to encrypt user files without making sure that they can correlate a key with an affected user. However, we have a few such cases described in Table-1 and one of them will be further described in **Corner cases** section.

Ransomware name	Key exchange	Encryption
ACCDFISA	Upload	Password Protected Rar SFX archives
HiddenTear	Upload	AES256
LockCrypt	Download	Custom block-cypher

Table-1 : [Threat-Owner] Key sharing mechanism

There is a higher preference for local key generation instead of using a key obtained by other means, which we will discuss soon, due to its increased security and transparency for ransomware creators. Many of the ransomwares who locally generate the encryption key are either using some secure key generator algorithms available in OS API or OpenSSL, deriving a SECP elliptic curve master key, or using an insecure key generator, which can lead to a breakable key as seen in Table-2.

Ransomware name	Encryption	Key
OpenToYou	RC4	Hardcoded Plain-text key
Annabelle	AES	Hardcoded Plain-text key
Nemucod	Cyclic XOR	Hardcoded Plain-text key
Amnesia	AES128, CBC	Time (C rand() function)
Globe V3	AES256, ECB	Time (C rand() function)
Nemesis	AES256/512, ECB,CBC	Time (C rand() function)
Xorist	TEA / XOR	Time (C rand() function)
Xmas	CUSTOM	Time (C rand() function)
LeChiffre	BlowFish	Hardcoded and User-Info
Petya	Salsa20	secp192k1

Table-2 : Ransomwares using weak key generator or weak encryption

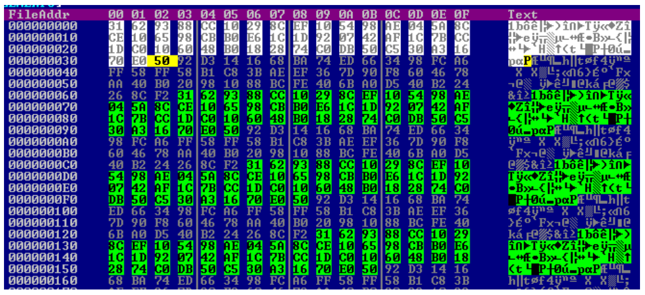
The ransomware families using proprietary encryption algorithms usually prefer easy to revert light XOR operations or RC4 encryption with low-security key generators or even worse, with plain-text keys. Table-2, Fig. 1 and Fig. 2 reveals something about the encryption mechanisms available in Nemucod and OpenToYou ransomwares. The recovery of files affected by these ransomwares is possible not because of the weak-keys involved, but because the encryption algorithms are breakable. For OpenToYou ransomware, it is sufficient to have a pair of encrypted and not encrypted files to recover all the affected files, because we know that RC4 function actually applies a xor operation between a key-stream and the actual data allowing the recovery of the key-stream from a pair of {encrypted, original} pieces of data.

We already know that for RC4 encryption, the following affirmations will lead us to decrypt any file starting from a pair of encrypted and not encrypted files, because the Key Stream can be computed using two different methods:

$$RC4 = \{P, KS, KSA, I, O\}$$
$$P - password$$
$$KS-\text{Key Stream}$$
$$KSA-\text{Key Stream Generator Algorithm}$$
$$I-\text{Input data}$$
$$O-\text{Output data}$$

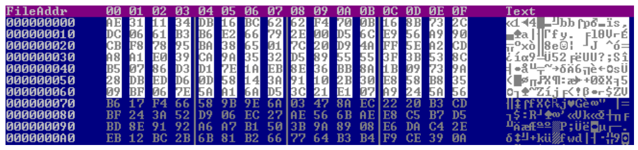
Given the above description and knowing that $KS = KSA(P)$, we can encrypt I with KS, obtaining $O = KS \oplus I$. However having I and O can lead us to a valid KS, without knowing P, because $KS = O \oplus I$

The most of the ransomware families nowadays will choose for RSA / ECDH key exchange mechanisms because they are more secure. They also ease the entire key-management, leaving the hacker with a map of user IDs and RSA keys used for encryption. The ransomware authors only require for the affected users ID, to get their correct decryption keys. There are a lot of sub-cases of using this type of keys, but all of them fall back to the same mechanism of having RSA to encrypt some symmetric encryption keys and not the actual user data. Most of the RSA keys range from 1024 to 4096 bits and there are a few mistakes of



```
FileAddr   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000000  31 62 93 88 CC 10 29 8C EF 10 54 98 AE 04 5A 8C
000000010  CE 10 65 98 CB B0 E6 1C 1D 92 07 42 AF 1C 7B CC
000000020  1D C0 10 60 48 B0 18 28 74 C0 DB 50 C5 30 A3 16
000000030  70 E0 50 92 D3 14 16 68 BA 74 ED 66 34 98 FC A6
000000040  FF 58 FF 58 B1 C8 3B AE EF 36 7D 90 F8 60 46 78
000000050  AA 40 B0 20 98 10 88 BC FE 40 6B A0 D5 40 B2 24
000000060  26 8C F2 31 62 93 88 CC 10 29 8C EF 10 54 98 AE
000000070  04 5A 8C CE 10 65 98 CB B0 E6 1C 1D 92 07 42 AF
000000080  1C 7B CC 1D C0 10 60 48 B0 18 28 74 C0 DB 50 C5
000000090  30 A3 16 70 E0 50 92 D3 14 16 68 BA 74 ED 66 34
0000000A0  98 FC A6 FF 58 FF 58 B1 C8 3B AE EF 36 7D 90 F8
0000000B0  60 46 78 AA 40 B0 20 98 10 88 BC FE 40 6B A0 D5
0000000C0  40 B2 24 26 8C F2 31 62 93 88 CC 10 29 8C EF 10
0000000D0  54 98 AE 04 5A 8C CE 10 65 98 CB B0 E6 1C 1D 92
0000000E0  07 42 AF 1C 7B CC 1D C0 10 60 48 B0 18 28 74 C0
0000000F0  DB 50 C5 30 A3 16 70 E0 50 92 D3 14 16 68 BA 74
000000100  ED 66 34 98 FC A6 FF 58 FF 58 B1 C8 3B AE EF 36
000000110  7D 90 F8 60 46 78 AA 40 B0 20 98 10 88 BC FE 40
000000120  6B A0 D5 40 B2 24 26 8C F2 31 62 93 88 CC 10 29
000000130  8C EF 10 54 98 AE 04 5A 8C CE 10 65 98 CB B0 E6
000000140  1C 1D 92 07 42 AF 1C 7B CC 1D C0 10 60 48 B0 18
000000150  28 74 C0 DB 50 C5 30 A3 16 70 E0 50 92 D3 14 16
000000160  68 BA 74 ED 66 34 98 FC A6 FF 58 FF 58 B1 C8 3B
```

Fig. 1 - Nemucod rolling key



```
FileAddr   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000000  AE 31 11 34 DB 16 BC 62 62 F4 70 0B 16 8B 73 2C
000000010  DC 06 61 B3 B6 E2 66 79 2E 00 D5 6C E9 56 A9 90
000000020  CB F8 78 95 BA 38 65 01 7C 20 D9 4A FF 5E A2 CD
000000030  A8 A1 E0 39 CA 9A 35 32 D5 89 55 55 3F 3B 53 8C
000000040  B5 07 86 D3 D1 7E 1A EB 8E 36 BB 8A 1B 09 73 9A
000000050  28 DB ED D6 0D 58 14 3A 91 10 2B 30 E8 58 B8 35
000000060  09 BF 06 7E 5A A1 6A D5 3C 21 E1 07 A9 24 5A 56
000000070  B6 17 F4 66 58 9B 9E 6A 03 47 8A EC 22 20 B3 CD
000000080  BF 24 3A 52 D9 06 EC 27 AE 56 6B AE E8 C5 B7 D5
000000090  BD 8E 91 92 A6 A7 B1 50 3B 9A 89 08 E6 DA C4 2E
0000000A0  EB 12 BC 2B 6B 81 B2 66 77 64 B3 B4 F9 CE 39 0A
```

Fig. 2 - Block of zero encrypted bytes with RC4 (OpenToYou)

keys less than 1024 bit which have been broken. As far we have seen about three slightly different RSA layering, considered by ransomwares:

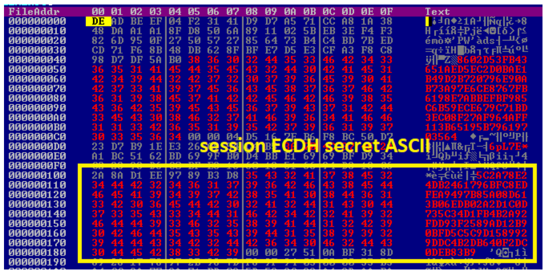
1. shipped RSA public key to encrypt a global AES key. AES key is used to encrypt the user data
2. shipped RSA public key to encrypt random AES keys, AES key is generated for each file to be encrypted
3. shipped RSA public key A to encrypt another random generated RSA private key B. The RSA public key B is used to encrypt random AES keys for each file. The encrypted RSA private key B is stored on user system ready to upload and decrypt on ransomware-owner servers

The third RSA layering is actually used by the WannaCry ransomware and some of its successors. The ransomware owners secure that way their private keys and do not expose them to users which pay the ransom. Obtaining the A private key is not possible. This is a multi layer RSA introduced to deal with cases where an affected user paying the ransom could share the received private keys with other users affected by the same binary. Troldesh/Crysis ransomware creators, for instance will provide a binary file containing a RSA private key after paying the ransom, which will actually decrypt about 15 different sub-versions. Those who run these businesses are aware of the fact that multiple users can get affected by the same ransomware binary, so they need a way to keep secret their keys when users pay the ransom and the same tine to minimize the hacker-user interaction or their servers availability. Table-3 reveals a couple of ransomwares using the RSA key encryption with their preference for one of the above 3 mentioned RSA layering.

Ransomware name	RSA Key count
Troldesh / Crysis	1
GlobeImposter	2
WannaCry	3
Rapid	3

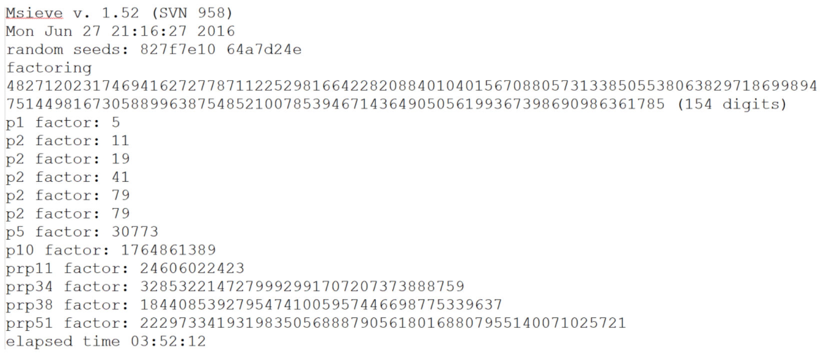
Table-3 : RSA key count / threat

There are a few ransomware families which use ECDH for key exchange. We tend to consider any ransomware using RSA or ECDH to be unbreakable, however about 2 years ago in May 2016, a researcher found that TeslaCrypt ransomware (first two versions) which used ECDH, was breakable because of the key encoding mechanism. We believe that authors wanted to store 1024 bit public keys inside the encrypted files, but they actually stored the 1024 bit key as ASCII hex, reducing the keys to 512 bit. Some of these keys are breakable in about seconds using factoring tools like Yafu and Msieve. The authors updated this weakness in their 3rd and 4th ransomware version. While Fig. 3 reveals an actual 512 bit hex-ASCII key stored in an encrypted file, Fig. 4 is a snapshot taken from an actual factorization of that key.



```
FileAddr    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   Text
000000000   DE AD BE EF 04 F2 31 41 D9 D7 A5 71 CC A8 1A 38
000000010   48 DA A1 A1 8F D8 50 6A 89 11 02 5B EB 3E F4 F3
000000020   82 6D 95 0F 27 50 57 27 85 64 73 B4 C4 BD 7B ED
000000030   CD 71 F6 8B 48 DB 62 8F BF E7 D5 E3 CF A3 F8 C8
000000040   98 D7 DF 5A B0 38 36 30 32 44 35 33 46 42 34 33        8602D53FB43
000000050   36 35 31 41 45 44 35 45 43 32 44 30 42 41 45 31   651AED5EC2D0BAE1
000000060   42 34 39 44 32 42 37 32 30 37 39 36 45 39 30 41   B49D2B720796E90A
000000070   42 37 33 41 39 37 45 36 43 45 38 37 36 37 46 42   B73A97E6CE8767FB
000000080   36 31 39 38 45 37 41 42 42 45 46 42 46 39 38 35   6198E7ABBEFBF985
000000090   43 36 42 35 39 45 43 45 36 37 39 43 37 31 42 44   C6B59ECE679C71BD
0000000A0   33 45 43 30 38 46 32 37 41 46 39 36 34 41 46 46   3EC08F27AF964AFF
0000000B0   31 31 33 42 36 35 31 39 35 42 37 39 36 31 39 37   113B65195B796197
0000000C0   30 33 35 36 34 00 00 04 D5 16 7F B6 F8 BC 50 D7   03564
0000000D0   23 D7 B9 1E E3 26                                           6pL7E*
0000000E0   A1 BC 51 62 BD 69 9F B0 D4 BB E1 69 69 BF D9 34
0000000F0
000000100   2A 8A D1 EE 97 89 B3 D8 35 43 32 41 37 38 45 32           5C2A78E2
000000110   34 44 42 32 34 36 31 37 39 36 42 46 43 38 45 44   4DB2461796BFC8ED
000000120   46 45 41 39 34 39 37 42 38 35 41 30 38 44 36 31   FEA9497B85A08D61
000000130   33 42 30 36 45 44 42 30 32 41 32 44 31 43 30 44   3B06EDB02A2D1C0D
000000140   37 33 35 43 33 34 44 31 46 42 34 42 32 41 39 32   735C34D1FB4B2A92
000000150   46 44 44 39 33 46 32 35 38 39 41 44 31 32 42 39   FDD93F2589AD12B9
000000160   30 42 46 44 35 43 35 43 39 44 31 35 38 39 39 32   0BFD5C5C9D158992
000000170   39 44 44 43 34 42 32 44 42 36 34 30 46 32 44 43   9DDC4B2DB640F2DC
000000180   30 44 45 42 38 33 42 39 00 00 27 51 0A BF 31 8D   0DEB83B9
000000190
```

Fig. 3 - 512 bit public ECDH session key - TeslaCrypt



```
Msieve v. 1.52 (SVN 958)
Mon Jun 27 21:16:27 2016
random seeds: 827f7e10 64a7d24e
factoring
4827120231746941627277871122529816642282088401040156708805731338505538063829718699894
7514498167305889963875485210078539467143649050561993673986909863617785 (154 digits)
p1 factor: 5
p2 factor: 11
p2 factor: 19
p2 factor: 41
p2 factor: 79
p2 factor: 79
p5 factor: 30773
p10 factor: 1764861389
prp11 factor: 24606022423
prp34 factor: 3285322147279992991707207373888759
prp38 factor: 18440853927954741005957446698775339637
prp51 factor: 222973341931983505688879056180168807955140071025721
elapsed time 03:52:12
```

Fig. 4 - Factoring the above key with Yafu on a i7 CPU, 8 threads

5 The environmental advantage

To ensure that file recovery is not possible and the ransomware will execute in most operating system environments, ransomware owners must be aware of the environment where encryption will take place, how it looks like and how can be tuned to help them with the encryption. On Windows operating systems, a ransomware will not start the encryption process until the vssadmin service is not disabled as a preceding step. This service is used for shadow volume copies such that lost data can be recovered if the disk has enough free space. Other ransomwares like GobeImposter which are more recently spread through RDP brute-forced accounts are created to auto-cleanup the author intervention and removes some event-logs and RDP logs after the execution ends. The environment is most of the time vulnerable to certain behaviors, be it an application vulnerability (MS Office Word memory corruption CVE-2018-0802), an administrative weakness (unauthorized installation of WMI scripts/services, unauthorized uninstall of security products) or an operating system kernel vulnerability like MS17-010 first exploited by WannaCry back in 2017. Below there are a few special cases of ransomwares taking advantage of the environment and/or other open-source tools.

- AVCrypt make use of WMI commands to uninstall Emsisoft security product:
  cmd.exe /C wmic product where (Vendor like ”%Emsisoft%”) call uninstall /nointeractive & shutdown /a
- SamSam/Samas ransomware uses Active Directory on Windows servers to query users and then uses them to spread over the Local Area Network. The ransomware is also able to exploit a vulnerability in Boss JMX-Console to bypass authentication.
- NotPetya / Petwrap / BadRabbit make use of Mimikats tools to grab user credentials from operating system internal tables and use them to spread the threat over the network.

6 On ransomware-creators skill-set

One major observation found among the entire ransomware phenomenon is that the authors or attackers rarely have good or even decent cryptography knowledge. Various indicators were found to support this claim:

- Ransomwares such as Xmas, Xorist, Bart, Globe generate an encryption key for AES using the random() function from C. This leads to an easy recoverable key, by having a generation space of 232 bits, which can be brute-forced in a reasonable time.
- Encryption using CBC mode having the Initialization Vector equal to 0. While this does not always support the key recovery, it makes the encryption process less secure and, in combination with the previous point, makes the recovery process even faster.

- Flawed implementation: some authors try manually to implement an encryption scheme and do not take all security measures, resulting in a possible exploitable scheme. Such a case can be seen in some versions of Nemesis ransomware, where the author tried to implement a version of AES based on 512 bits length of both encrypted block and key. A similar case is the first version of Petya (Red Petya) which implemented a custom Salsa20, explained in [2] which use 32 bit words in the proposed design, but only 16 bit words in Red Petya implementation, because the boot-loader, encrypting disk sectors, worked in 16 bit mode. The authors patched this issue in Green Petya and more recent versions like BadRabbit which use 32 bit boot-loader code.
- Using a static key and a basic encryption scheme with no or little cryptographic properties. Such an example can be found in PCLock, Nemucod, OpenToYou, where a static key was used in a basic XOR scheme.
- Using simple, exploitable verifications: Cryp0l0cker, for instance, used to make a request on blockchain.info using a hardcoded BTC address in order to check whether the victim has paid. Once it received the confirmation, it started to decrypt the files. Therefore, manually modifying the address to one which already has the minimum of request bitcoins, will trick the ransomware into decrypting the files.
- Exploitable services: once the ransom payment has been completed, some ransomwares would query a service where they would send the user key and download the decryption. Therefore, anyone who knew a victim's key could make the request and obtain the decryption key.
- Damaging the files: cases have been seen where the ransomware would faulty encrypt or write the encrypted data and, therefore, make the data irrecoverable. GandCrab, for instance, would use the SetFilePointer in order to move the file pointer to overwrite the clear data with the encrypted one. However, the lpDistanceToMoveHigh parameter, which specifies **"A pointer to the high order 32-bits of the signed 64-bit distance to move"** wasn't used, which leads to the function to fail if the purpose is to move the pointer more than 2 GB. Therefore, the file pointer would not be properly moved and data would be lost.
- Memory persistence : Some ransomwares generate a local key which will be used to encrypt the entire system and will be kept in memory for the entire process. However, some of them will not exit once all system's files are encrypted (so the key will be deleted) and will continue to scan for new files. Therefore, anyone with system access can dump the entire memory of the process and obtain the key from there.

7 Corner cases

We believe that the ransomware encryption touches a few corner-cases which can be exploited both by the up-coming technologies and math development the same time. We will refer to 2 special cases of encryption, based on RSA1024 keys which might be breakable in the near future and some custom encryption mechanisms which could be braked using tools like Z3 Theorem Prover / Solver.

1. RSA1024
   RSA1024 keys can already be broken using special hardware like SHARK explained in [1]. The problem is that the implementation requires a lot investment (20-30 M$) and the time is not too short with all this.
   We also presume that quantum computing along with mathematical advancements will make these keys easy to break in the near future.
2. Custom encryption using Theorem Prover / Solver
   A couple of threats (LockCrypt ransomware , ShadowPad botnet) are using block-chain encryption based on 64 bit keys or pairs of 32 bit keys. Our research using Z3 Solver from Microsoft, proves that knowing the first 16-32 bytes of the output can lead to a corresponding equation system which might give us the decryption key without actually attempting a full brute-force.

The below piece of code is an example of encrypting a chunk of data using 2 x 32bit keys, in a block-chain meaner:

```
for (i = 0; i < 0xFB44; i++)
{
  unpacked[i] = key ^ shellcode_encrypted[i + 4];
  key = 0xC9BED351 * ROL(key,16) - 0x57A25E37;
}

```

Unfolding the first few steps of this encryption process, based on a few bytes that we know to be encrypted will result in:

```
iters[0] = 0x340D611E;
iters[1] = key2 + key1 * ROL(0x340D611E, 16);
iters[2] = key2 + key1 * ROL(key2 + key1 * ROL(0x340D611E,
  16), 16);
iters[3] = key2 + key1 * ROL(key2 + key1 * ROL(key2 + key1 *
   ROL(0x340D611E, 16), 16), 16);

```

And the coresponding Z3 bit-vector assertions will be:

```
(declare-const key1 (_ BitVec 32))
(declare-const key2 (_ BitVec 32))
(assert (= (bvand #x340D611E #x000000FF) #x0000001E))
(assert (= (bvand (bvadd key2 (bvmul key1 (bvadd (bvshl #
  x340D611E #x00000010) (bvlshr #x340D611E #x00000010)))) #
  x000000FF) #x000000E6))

```

8 Quantum cryptography

As quantum computing becomes more and more closer to reality, the problem of quantum cryptography arises. Despite the fact that a theoretical theorem has proven that a quantum computer cannot perform general arbitrary computation, it can be used to access specific quantum states and perform operations that are similar. As theoretically proven, a quantum computer can perform calculations much more quickly than a classical one and, thus, can attack most of current encryption schemes.

To prevent this, researchers are working on post-quantum or quantum-resistant cryptography, which provides encryption schemes, most of them based on public-key algorithms, secure against quantum attacks. A way of sharing an encryption key through a quantum network is via quantum key distribution, which allows sharing a key between two parties without having to rely on a third one. Current popular schemes are not resistant to quantum attacks mainly because they rely on difficult mathematical problems, such as discrete logarithm, integer factorization, elliptic-curve discrete logarithm, quadratic residuosity problems.

However, once quantum cryptography will be practically implementable, everything we currently know as security will be shifted. Obviously, the malicious software will be changed as well. Therefore, this bring the possibility of quantum malware or, more closer to the subject, quantum ransomware. One can only assume what such a concept implies at this point in time. Despite the security measures that will be taken, it is highly possible that the quantum computer will be somehow exploitable. Therefore, an attacker might build a quantum malware that performs similarly to a ransomware, but based on quantum cryptography. What is more, other scenarios imply altering the quantum states, altering a system through "illegally" entaglements, perform quantum distributed attacks (similar to DDoS), etc.

9 BadRabbit ransomware – case study

BadRabbit, like its predecessor NotPetya/PetWrap make use of disk encryption and tools like mimikatz to break user credentials and move forward on all possible computers inside a network. It also make use of RDP exploit, but a slightly different version of EternalBlue - EternalRomance. Table-4 reveals a serie of network shares, users and credentials preloaded with the binary:

Network Shares	User names	Passwords
wkssvc	user	god
svcctl	guest	secret
scerpc	work	password
srvsvc	root	test123
samr	Admin	777
spoolss	Adminstrator	qwerty
ntsvcs	Test	123456
netlogon	rdp	123321
lsarpc	support	uiop
eventlog	manager	administrator123
browser	alex	test
atsvc	ftpuser	adminTest

Table-4 : BadRabbit brute-forced network and user names

To simplify the bootloader code for Salsa encryption, this new version switches to Protected Mode 32 bit using 32 bit aligned data, excluding this way special computations to deal with 32 bit from 16 bit mode. The interesting facts about BadRabbit are its true promise that files can be recovered, because we know that its predecessor NotPetya was a wiper, the preference for EternalRomance and the authors preference for characters of Game of Thrones:

```
schtasks /Delete /F /TN rhaegal
schtasks /Create /SC once /TN drogon /RU SYSTEM /TR

```

10 Conclusions

These being said, what it started as a Proof of Concept, ransomware became one of the biggest sources of revenue. What is more, ransomware authors seem to try to keep up the pase with the ongoing security products. The more the security systems improve, the more "smart" ransomware spreading and infecting techniques become. And, as previously mentioned, once quantum computers can be used, more or less, in practical issues, the entire industry might change.

11 References

References

1. Kleinjung Thorsten, Aoki Kazumaro, Franke Jens, Lenstra Arjen K., Thomé Emmanuel, Bos Joppe W., Gaudry Pierrick, Kruppa Alexander, Montgomery Peter L., Osvik Dag Arne, te Riele Herman, Timofeev Andrey, Zimmermann Paul: "Factorization of a 768-Bit RSA Modulus" Advances in Cryptology – CRYPTO 2010, Springer Berlin Heidelberg, P. 333–350, 978-3-642-14623-7
2. Bernstein Daniel J., Robshaw Matthew, Billet Olivier: "The Salsa20 Family of Stream Ciphers" New Stream Cipher Designs: The eSTREAM Finalists, Springer Berlin Heidelberg, P. 84-97, 978-3-540-68351-3