Practical Cryptography Project Report

CRYPTANALYSIS OF RANSOMWARES AND SMART-CONTRACT BASED PAYMENT GATEWAY FOR RANSOMWARE

Made By
Ninad Shetty (nshetty1@jh.edu)
Shivam Negi (snegi3@jh.edu)

Guided By
Prof. Matthew Green and Prof. Alishah Chator
Johns Hopkins University

Contents

1 Introduction

Ransomwares have been a serious problem for Internet users and businesses. Ransomware is malicious software that blocks users from accessing their data (usually by encrypting them) and demands a ransom to restore access.

The use of cryptography to mount extortion-based attacks was first proposed by Young and Yung as early as 1996. In their seminal paper, they actually created a new research area called cryptovirology. In their initial proposal, which remains almost unchanged, the attacker places a public/private pair key in malware. The malware then encrypts the victim's files with a locally generated random symmetric key, which is in turn encrypted with the public key. Finally, it provides a ransom note and instructions for payment After the attacker receives the payment, the previous symmetric key is decrypted with the corresponding private key and sent to the victim, who finally recovers the original files.

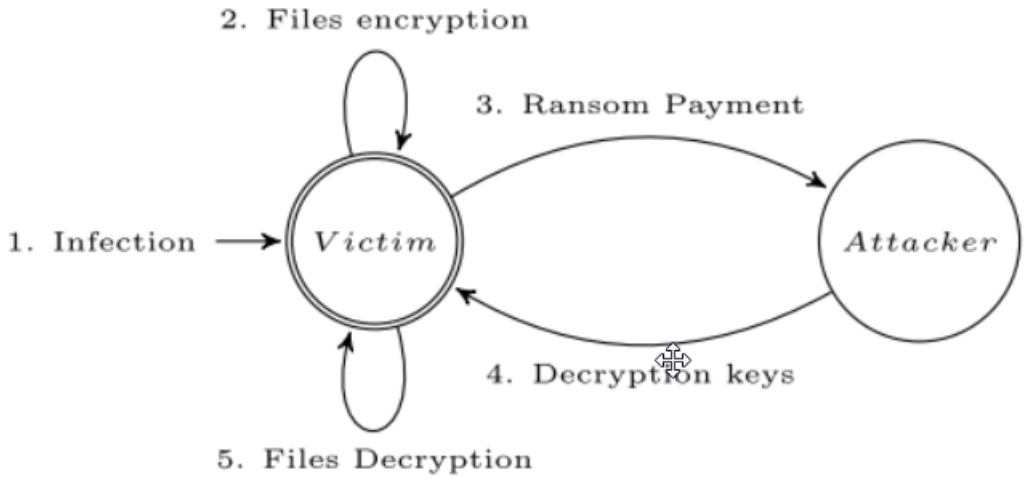

Figure 1: Ransomware Working

Cryptanalysis is the study of ciphertext, ciphers and cryptosystems with the aim of understanding how they work and finding and improving techniques for defeating or weakening them. The objective of cryptanalysis is to find weaknesses in or otherwise defeat cryptographic algorithms, cryptanalysts' research results are used by cryptographers to improve and strengthen or replace flawed algorithms. In this report we will cryptanalyse the three famous ransomware: *Bitcrypt, Dmalocker, Wannacry.*

Further, we will use Ehtereum smart contracts to make ransomware more effective. We will look into different approaches to implement an ideal ransomware. Finally we present an implementation of 'Proof of Life' ransomware scheme.

2 Bitcrypt

2.1 Introduction

BitCrypt encrypts a large range of files, from documents and pictures to archives, application development, and database files. Victims stand to lose access not just to personal files, but also to work projects if they have no external backups. The text file contains information on how to access a specific website hidden on the Tor anonymity network in order to obtain a special decryption program that's unique for every infection

2.2 Cryptanalysis

The sample is packed with a simple packer, Delphi-compiled malware. We then study the Bitcrypt ID generation and files encryption.

When first run on the system, the malware creates a configuration file (bitcrypt.ccw in APPDATA folder). It then choses a random number between 1 and 999, and uses it as an index to extract a base64-looking string from an array. The Bitcrypt ID is then computed as follows, and both are stored in the configuration file:

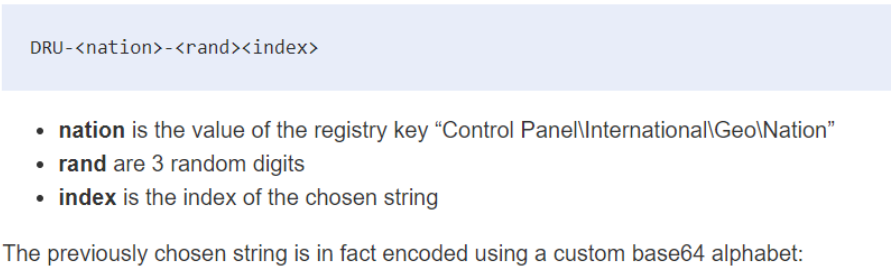

Figure 2: Bitcrypt ID Computation

It says that it locks down files with 1024-bit RSA encryption – but that's not actually the case. The malware then starts looking for interesting files to crypt on the disk, while a watching thread will kill any attempt to run taskmgr.exe or regedit.exe.

The way it works is this: For each interesting file it encounters, a new 16-character random password is generated, and a 192-bit key is derived using PBKDF2 with HMAC-SHA1, with a random salt and 1000 iterations. The resulting key is used to encrypt the file content using AES in CTR mode. All these operations are performed using AESLib.pas. The AES key is then RSA-encrypted using the previously chosen key. This time, the FGIntRSA module is used. The resulting file has the ".bitcrypt" extension appended to its filename.

The whole model seems rather secure, even if one could wonder why a new AES key is generated for each file. However, a quick look at the base64-like strings raises doubts about the key length. For example, here is the configuration file from an infection:

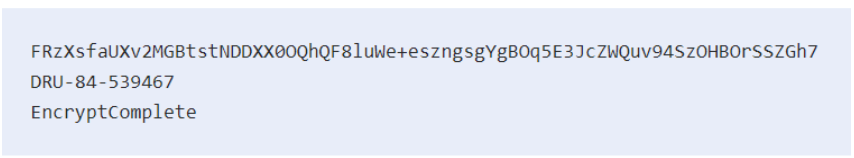

Figure 3: Bitcrypt Config File

Once decoded, the key translates to a 128 digit number. This could indicate a mistake from the malware author, who wanted to generate a 128 bytes key. We simply have to deal with RSA-426 encryption, which can easily be broken on a standard PC in a matter of hours.

3 DMA Locker

3.1 Introduction

The ransomware was first spotted in early 2016 and continued to evolve. The malware adds an identifier into the header of every encrypted file instead of appending a specific extension. Furthermore documents remain openable they are just not readable.With DMA Locker the network is scanned for any available share so if the user has access it will encrypt everything that is available, therefore when restoring the system a full server restore is needed not just the data shares, making backups absolutely critical.

3.2 Cryptanalysis

When deployed, the ransomware moves itself into ProgramData, renamed to fakturax.exe and drops another, modified copy: ntserver.exe. File faktura.exe is removed after execution. Depending on its version, it may also drop some other files in the same location. Symptoms of this ransomware can be recognized by a red window popping up on the screen.

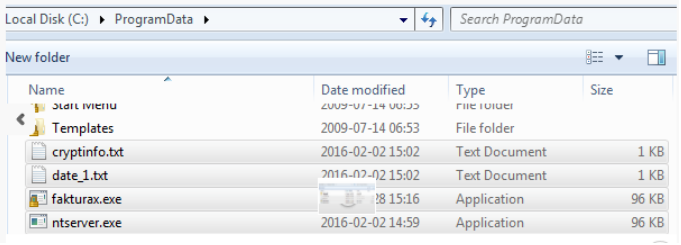

Figure 4: DMA Locker Deployment

In contrast to other ransomware that are offering a separate decrypter, DMA Locker comes with a decrypting feature built-in. It is available from the GUI with ransom note. If the user enters a key (32 characters long) in the text field and clicks the button, the program switches to the decryption mode (using supplied key). In the ransom note, the authors mention that the data is encrypted by AES and RSA.

This ransomware is distributed without any packing and no defense against analysis has been observed. All the used strings and called API functions are in plain text. In fact, the malware provides a lot of debug strings describing all it's activities

3.2.1 DMA Locker 1.0

1. Discovered : January, 2016
2. Version : 1.0
3. Algo : files are encrypted using AES Algorithm in ECB Mode.

4. Notes : Since AES Key is the same for each file, and its stored in the binary and erased after use, decryption is possible. If we're able to retrieve the malware sample (VirusTotal), we can extract the keys and use it.

3.2.2 DMA Locker 2.0

1. Discovered : February, 2016
2. Version : 2.0
3. Algo : files are encrypted using AES Algorithm in ECB Mode.
4. Notes : AES Key is randomly generated for each attacked file. After use, its encrypted by RSA and stored in the file. RSA Public Key comes hard-coded in the binary. However due to the weak RNG, AES Key can be guessed easily.

3.2.3 DMA Locker 3.0

1. Discovered : February, 2016
2. Version : 1.0
3. Algo : files are encrypted using AES Algorithm in ECB Mode.
4. Notes : AES Key is randomly generated for each attacked file. After use, its encrypted by RSA and stored in the file. RSA Public Key comes hard-coded in the binary.However RSA Key is the same full campaign and once private key is bought once, it can be reused for several victims.

3.2.4 DMA Locker 4.0

1. Discovered : May, 2016
2. Version : 1.0
3. Algo : files are encrypted using AES Algorithm in ECB Mode.
4. Notes : AES Key is randomly generated for each attacked file. After use, its encrypted by RSA and stored in the file. RSA Key pair is generated on server per-client. The public key is downloaded. Its not decryptable and it doesn't work offline

After the first look at encrypted content we can see repetitive patterns and entropy is relatively low. We can see patterns of original content reflected in the encrypted content, that suggests that some block cipher has been used. We can suspect, it is AES in ECB mode. The disadvantage of this method is a lack of diffusion. Because ECB encrypts identical plaintext blocks into identical ciphertext blocks, it does not hide data patterns well.

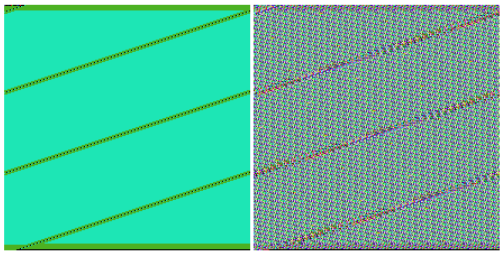

Figure 5: Binary Visualisation of Plaintext and Ciphertext

4 Wannacry

4.1 Introduction

WannaCry is a crypto-ransomware, a type of malicious software (malware) used by cybercriminals to extort money. Ransomware does this by either encrypting valuable files, so you are unable to read them, or by locking you out of your computer, so you are not able to use it. Ransomware that uses encryption is called crypto-ransomware. The type that locks you out of your computer is called locker ransomware. Like other types of crypto-ransomware, WannaCry takes your data hostage, promising to return it if you pay a ransom. WannaCry targets computers using Microsoft Windows as an operating system. It encrypts data and demands payment of a ransom in the cryptocurrency Bitcoin for its return.

4.2 Cryptanalysis

The malware has two hard-coded public keys deployed as part of this ransomware: one is used for the main task of encrypting files, while the other is used to encrypt a small number of files for "demo decryption" — so the ransomware authors can "prove" to victims that they are able to decrypt the files. Once the malware is running on the victim machine it will generate a new unique RSA 2048 bit asymmetric key pair. This means that each victim needs their own decryption key.

Once the new unique key pair is generated, the malware exports the victim's public RSA key to a local file called 00000000.pky using CryptExportKey API. Next, it exports the victim's private RSA key and encrypts it with the hardcoded attacker public key from the malware, and stores it as 00000000.eky on disk. Now that the key has been stored safely, the malware uses CryptDestroyKey API to destroy the private key in memory, which limits the time for recovering private key parameters from memory by any other tool. Unfortunately, the lifetime of private victim RSA keys is so limited that there is no good option to recover it later once the encryption has happened.

WannaCry encrypted data files on infected machines using asymmetric RSA 2048-bit encryption. The attackers held the decryption keys on the C2 and were able to provide them to victims after the ransom was paid.

Wannacry Encryption Steps:

1. Ransomware Executable imports its RSA private key from .data section to decrypt a Payload DLL to file "t.wry."

2. Payload DLL generates a new RSA key pair.
3. New RSA public key is saved to "00000000.pky."
4. New RSA private key is encrypted by CryptEncrypt API by using Payload DLL's RSA public key from its .data section.
5. Encrypted RSA private key is saved to "00000000.eky."
6. Payload DLL finds the target files and generates one AES key per file.
7. Payload DLL uses NEW public RSA key from "00000000.pky" to encrypt AES key of target file and saves encrypted AES key to the target file.
8. Payload DLL AES encrypts the target file and writes it to the new file with .WCRY extension.
9. To decrypt the files, the WANNACRY decryptor looks for "00000000.dky" file, which contains the RSA private key from the malware authors.
10. RSA private key from malware authors can be used to decrypt AES key per file, then decrypt each file using AES keys.

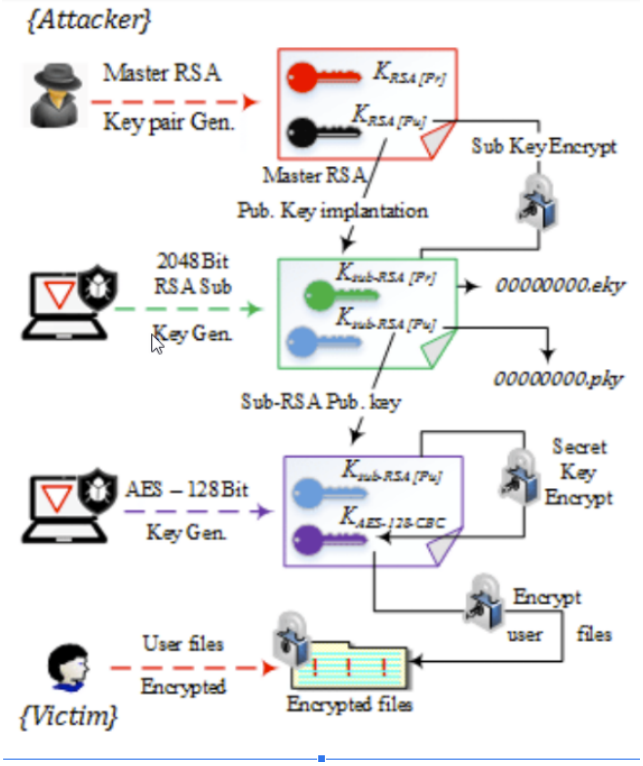

Figure 6: Wannacry Encryption Process

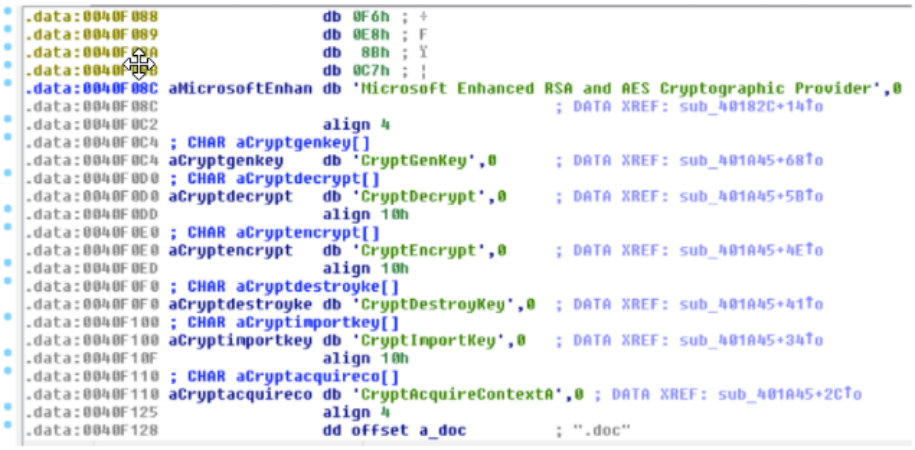

Figure 7: Windows API for Encrypt-Decrypt

4.3 Breaking the Ecryption

The presence of residual RSA primes in the memory address space of the WannaCry process makes it possible to derive a decryption key and subsequent decryption. Nevertheless, this recovery technique is only valid given the associated memory space is not overwritten or flushed. WanaKiwi uses such an approach to derive the decryption key and subsequent decryption of the affected files where the observed exponent in all the samples was 65537 (0x10001). It should be noted however that this method only works if the memory allocated to the WannaCry process is not overwritten or flushed, i.e. no system restart or reallocation of memory.

5 Ransomware with Smart Contracts

5.1 Ideal Smart Contract

From the point of view of an attacker, an ideal ransomware would have the following two properties:

1. **Exchange fairness**: the payment should be delivered to the attacker if and only if the user receives a correct decryption key.

2. **Maximum automation degree**: ideally, the ransomware should not need human operators to process payments and release decryption keys, and to operate autonomously.

5.2 Exchange Fairness

The fairness in the exchange of items or services is a concern that naturally arises in a digital scenario such as the Internet and e-commerce. The fundamental question is how to sell an item in such a way that none of the involved parties can cheat the other.

5.3 Protocols to implement Smart Contracts with Ransomware

We envision the following possible schemes for smart contracts. The schemes are ordered in increasing complexity and capabilities:

1. **Pay-and-pray**: the victim has no special guarantees, and pay for the decryption of all her files as a whole.

2. **Pay-per-decrypt**: the victim can pay for the decryption of individual (chosen) files.

3. **Proof-of-life**: The victim has the guarantee of a limited fair exchange, that is, to be able to recover her money if the attacker finally does not deliver a valid key for the decryption of the files.

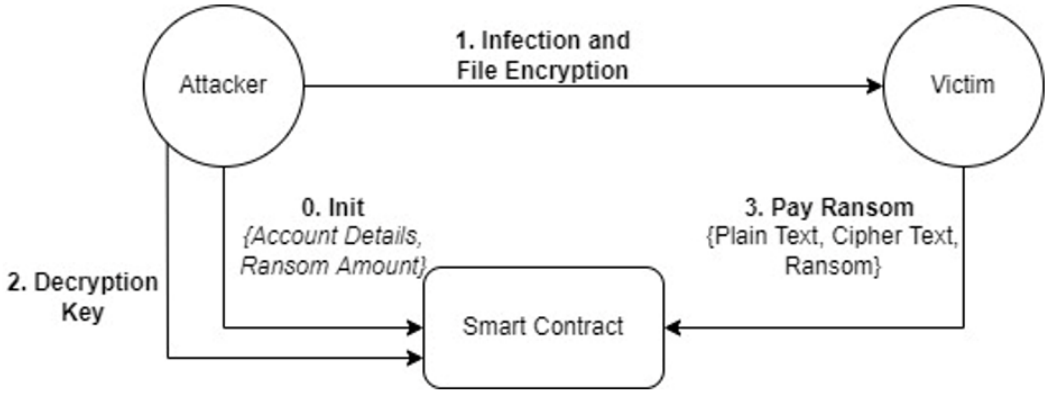

Figure 8: Proof of Life Architecture

5.4 Implementation of a Smart Contract

We implemented the ransomware payment gateway on an Ethereum Virtual Machine. The code was written in Solidity. Solidity is an object-oriented, high-level language for implementing smart contracts. We used Truffle which is a development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine. And the smart-contract was deployed on Ganache.Ganache is a personal blockchain for Ethereum development which can be used to deploy contracts, develop applications, and run tests. Basically, it gives you a virtual blockchain where you can test, develop, and iron out all the glitches before you commit to a public blockchain deployment.

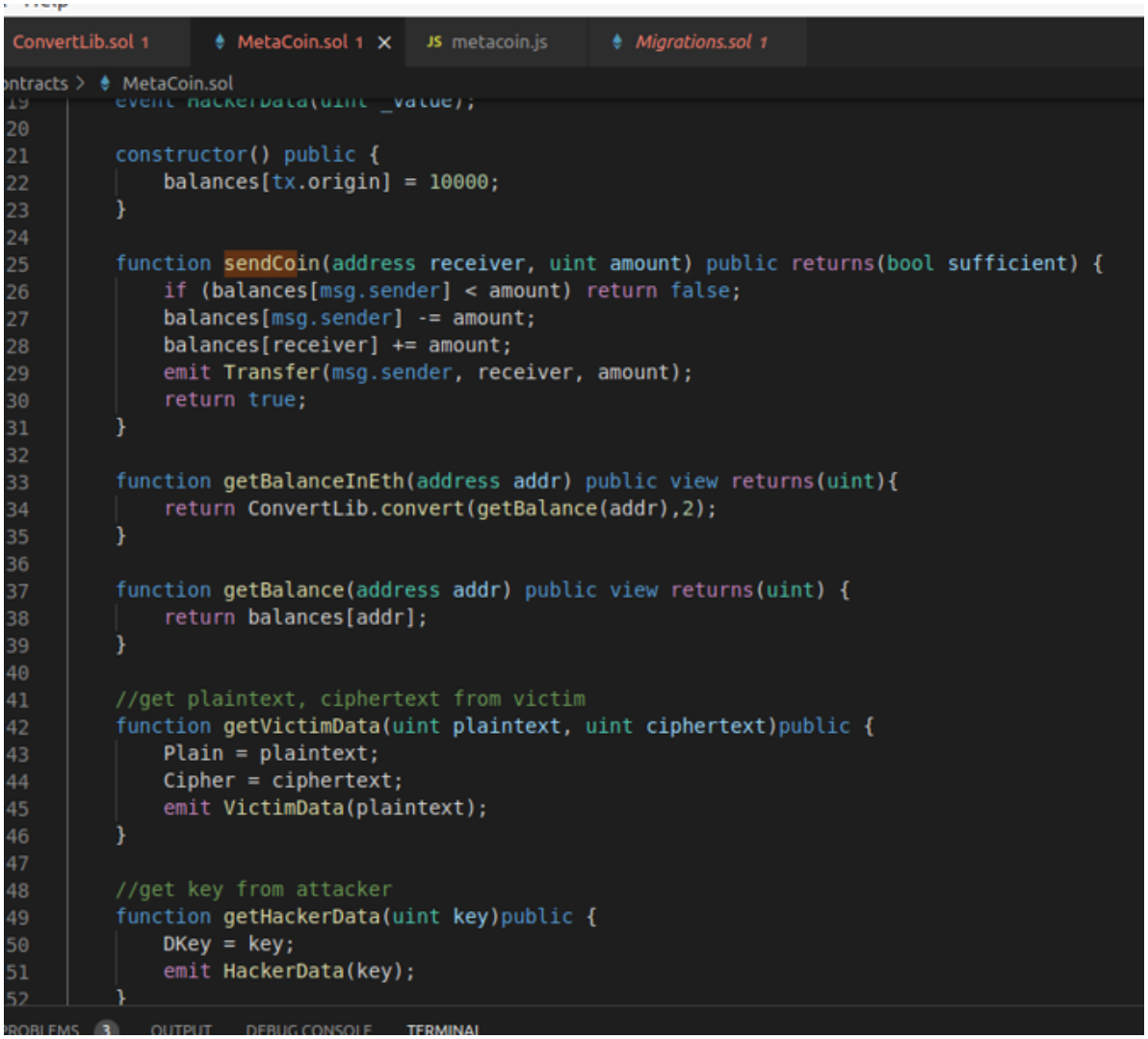

Figure 9: Solidity Code Snippet for Smartcontract

Implementation involves writing functions/api's for :

1. **sendCoin**: transfer the cryptocurrency payment from sender account to receiver account if its a legal transaction
2. **getbalance**: get the available balance in the user account specified
3. **getVictimData**: retrieves input from the victim user. Data required are plaint text and cipher text from a well known file to the user, so that they can be compared.
4. **getHackerData**: retrieves input from the hacker user. Data required is the private key for decryption process.
5. **processRansomRequest**: input for this function are Hacker's account and the ransom amount. It calls the decrypt library and passes the cipher text and private key that was retrieved from user. Only if the output matches the plain text, ransom amount is transferred from the victim's wallet to the hacker's wallet.

We write and execute Unit tests to see if the contract meets the requirements we've specified in the testfiles. The requirements are laid out in .js file.

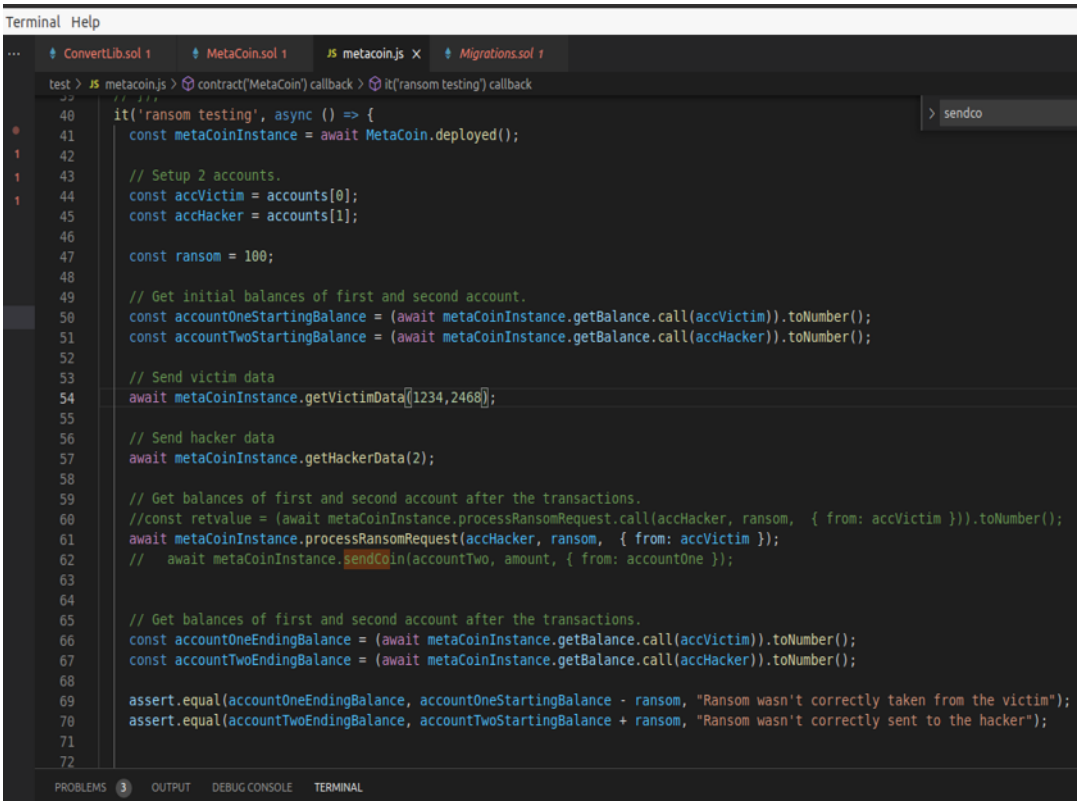

Figure 10: Unit test

5 green check marks to indicate that every test has passed and every function in the contract has worked as expected (see figure below). Since everything looks good, we can now try deploying the contracts on the Ganache test blockchain.

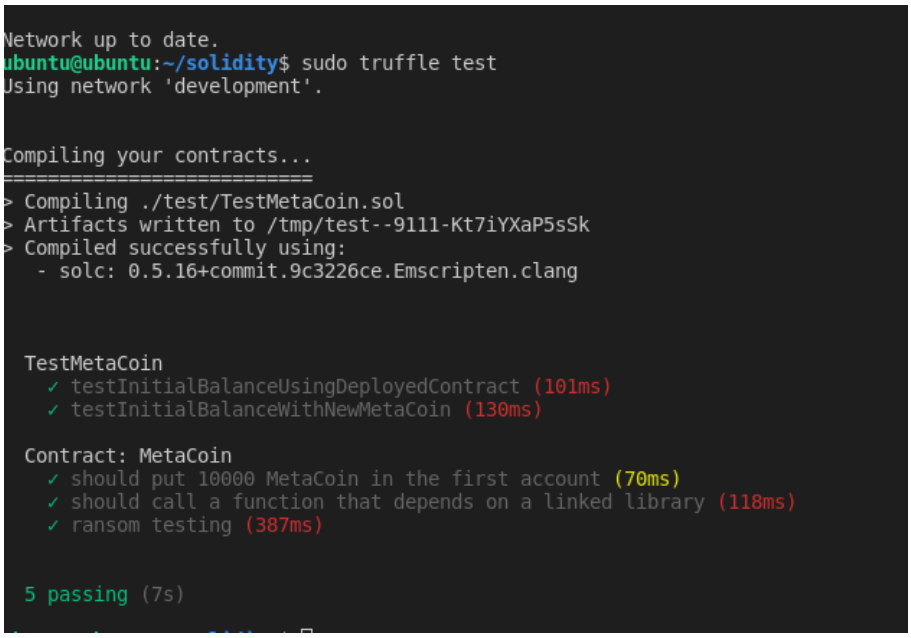

Figure 11: Smartcontract test results

Also 5 blocks are created on the test blockchain: one to deploy Migrations.sol (count 1), one to save the migration to the network (count 2), one each to deploy ConvertLib.sol

and MetaCoin.sol (count 4), and one to save the migration (count 5). (See fig below)

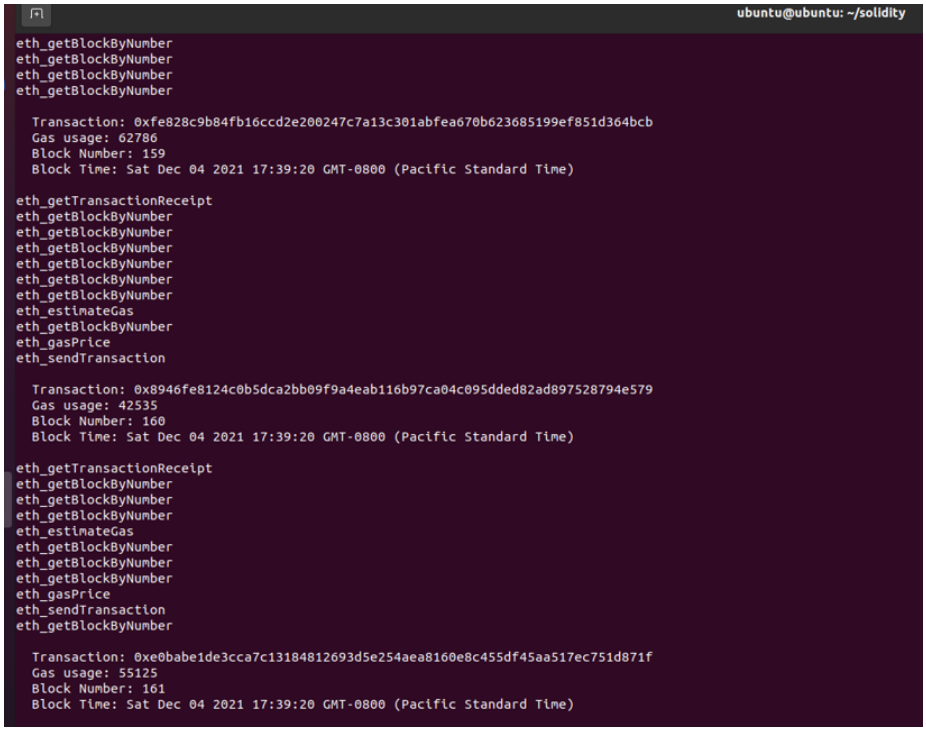

Figure 12: Ganache Blockchain output

6 Future Work

The virtually immutable nature of public blockchains makes finding any kind of countermeasures to this threat extremely difficult. Indeed, smart contracts cannot be blocked, deactivated or removed if the author has not explicitly included appropriate mechanisms designed to do so. Unfortunately there exist very few realistic countermeasures, beyond causing an intentional hard-fork in the blockchain to eliminate the malicious smart contracts. However, in practice this would be extremely difficult, if not impossible.

As it is clear, it is not possible to fully automate the ransomware process, since the "semantic" verification of the decryption of the files must always be carried out, ultimately, manually by the user. A possible solution would be to provide the user with a kind of oracle that could examine a file (with all its metadata, such as complete path, length, timestamps, etc.) and determine if it really belonged to the user initially.Unfortunately, the possibility does not seem realistic, due to that the collection should be created in advance, before the ransomware attack.

7 References

[1] https://blog.malwarebytes.com/threat-analysis/2016/05/dma-locker-4-0-known-ransomware-preparing-for-a-massive-distribution/
[2] https://airbus-cyber-security.com/bitcrypt-broken/
[3] Blockchain-based semi-autonomous ransomware (Oscar Delgado-Mohatar ,José María Sierra-Cámara, Eloy Anguiano)
[4] https://blog.cryptographyengineering.com/2017/02/28/the-future-of-ransomware/ (Matthew Green)