Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher
Erschienen im Magdeburger Institut für Sicherheitsforschung
http://www.sicherheitsforschung-magdeburg.de/publikationen/journal.html

This article appears in the special edition »In Depth Security – Proceedings of the DeepSec Conferences«.
Edited by Stefan Schumacher and René Pfeiffer

I Wrote my Own Ransomware; did not make 1 iota of a Bitcoin

Thomas Fischer

2016 saw a substantial rise in ransomware attacks and in some cases the return of some favourites with Cryptowall, CTB-LOCKER and TeslaCrypt being some of the most popular. The volume of attacks was in fact pretty steady for a good part of the year, with regular campaigns coming out on a weekly basis. It was interesting to see the variety in mechanisms used for the ransomware which not only included self-contained binaries but went all the way to the use of scripts. As part of the research I conducted last year, I wanted to understand why there's such a drive and lure for ransomware, outside of the victims payment, as well as have some way of properly testing »anti-ransomware« solutions with an unknown variant. So to do that, I went ahead and built my own ransomware and drew some conclusions on why it became so popular. This talk explore the background and process used to build a live ransomware that I was able to use for controlled testing. To finally draw some of my own personal conclusions.

Keywords: Malware, Malware Analysis, Bitcoin, Encryption

Citation: Fischer, T. (2018). I wrote my own ransomware; did not make 1 iota of a bitcoin. *Magdeburger Journal zur Sicherheitsforschung*, *16*, 879–892. Retrieved November 3, 2018, from http://www.sicherheitsforschung-magdeburg.de/uploads/journal/MJS_059_Fischer_Ransomware.pdf

1 Introduction

2016 saw a substantial rise in ransomware attacks and in some cases the return of some favourites with Cryptowall, CTB-LOCKER and TeslaCrypt being some of the most popular. The volume of attacks was in fact pretty steady for a good part of the year, with regular campaigns coming out on a weekly basis.

Some of the campaigns were interesting and demonstrated a variety of mechanisms used for the ransomware which not only included self-contained binaries but went all the way to the use of scripts.

As part of the research I conducted in 2016 to 2017, I wanted to understand why such a drive and lure by malicious parties to release ransomware outside of the victims will pay. An ulterior motive was to have some way of properly testing "anti-ransomware" solutions with an unknown variant. So, to do that, I went ahead and built my own ransomware and drew some conclusions on why it became so popular.

It is important to note that at no point in time was this malware used or released in the wild.

2 Background or Why Create a Ransomware

2.1 FFS Why!!!

The year 2016 may well be known as the year of the ransomware in the Information Security industry. All eyes and marketing were fixated on what new variant and which product might be best. There was not one week without an article in the media or a vendor blog post mentioning some new variant, or some new end user organisation being affected by the latest ransomware.

Working for an endpoint technology vendor at the time and as a threat researcher, you constantly get hounded by the marketing and sales teams to provide the best way or means to demonstrate the tools capabilities. Why? Honestly, because at the end of the day that's what the IT manager, CxO wants to see, at least that's the reasoning. Access to a malware lab or videos while useful doesn't necessarily provide the whiz-bang feeling of seeing something happening right in front of you.

There was another motive as well. A need to understand the foundations of why so many variants were appearing at such a rapid frequency. One way to understand this is to carry out practical steps and put oneself in the same mindset as the malicious party running the campaign.

The goal was set, break down a ransomware and build my own variant to understand the effort and mechanisms but also to provide a demo solution.

2.2 2016 Ransomware Year in Review

There are many means to deliver ransomware, including drive-by on a malicious website, fake advertisement leading to a willing or unwilling download, but by far the most frequent that appears in reports is via email phishing campaigns.

Various vendors regularly publish the state of affairs on attacks and malware. As one example used to understand the level of ransomware, one might refer to the Proofpoint 2016 Q3 Threat Report (https://www.proofpoint.com/us/threat-insight/threat-reports).

Proofpoint highlighted that an increase of 752% occurred from 2015 to 2016 in the number of ransomware variants. The following graph shows that increase over time: (source: Proofpoint)

An interesting fact that they highlight is that the main entry vector remained email and phishing campaigns. However, the report showed that the main attack types were based on file attachments and not URL clicks. The primary files being either Office documents or JavaScript files.

Proofpoint's 2016 year to date graph above shows that in fact over 2016 the biggest vector was JS based attachments. This being a side effect of better controls in organisations that filter malicious URLs and enhanced security and patches on the Office products and files.

2.3 Wait Need More Stats and Affirmation

One of the things that I noticed from my own email honeypot was a recurring theme on how ransomware campaigns were occurring. I surmised that in fact a weekly trend was occurring from volumes, frequency and delivery time frames. Not set-up to do in-depth frequency analysis, I turned to friend and SANS ISC Handle Xavier Mertens (@xme) who also tracks email campaigns. With his help, it was possible to confirm my theory.

A pattern was definitely visible. Weekly campaigns were being launched with varying volumes with typically the start of the month being the most active. The following graph for 2016 shows this pattern of campaigns.

Zooming into one month to better understand this pattern, a very interesting aspect appears. The bulk of these email campaigns are distributed at the start of the week. The campaigns looked to be coordinated and timed in such a way that was reminiscent to IT service delivery or even DevOps: Deploy at the beginning of the week, analyse results and improve then repeat.

Frequently the variant of ransomware stayed similar during the campaign, the changes occurring from one week to another in large part focused on improving the messaging and delivery mechanism.

With Xavier Mertens' help digging a bit deeper, he and I looked at the attachment types to note that a

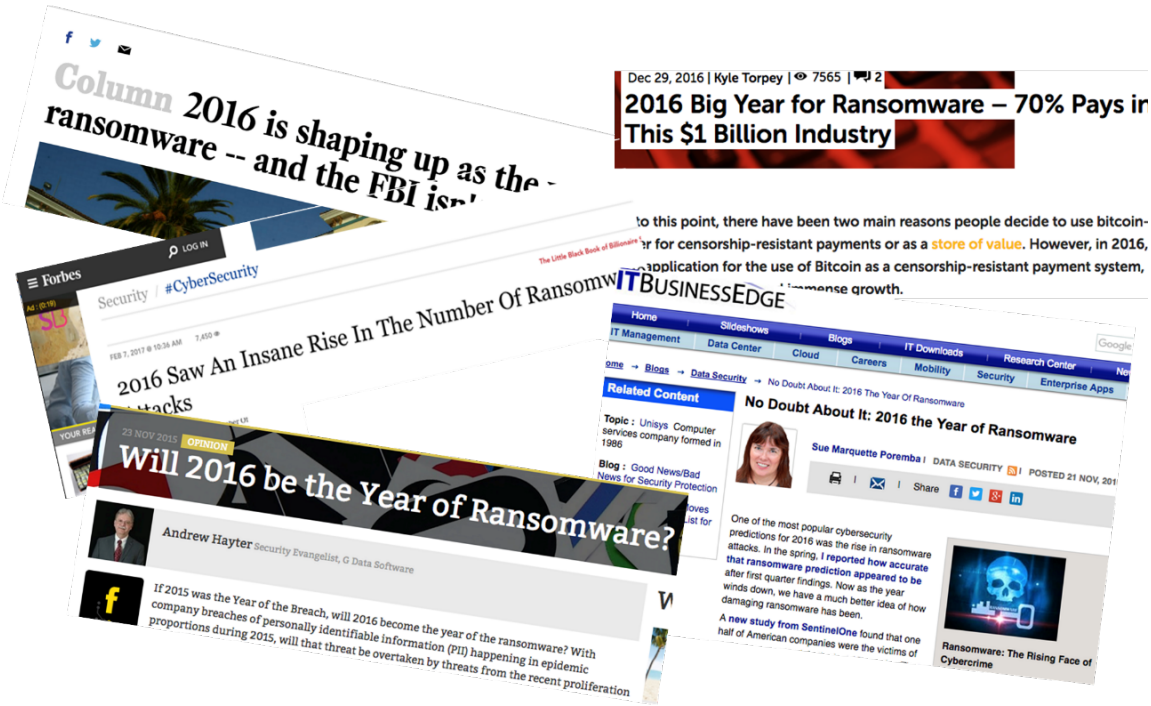


Figure 1: Ransomware in the news

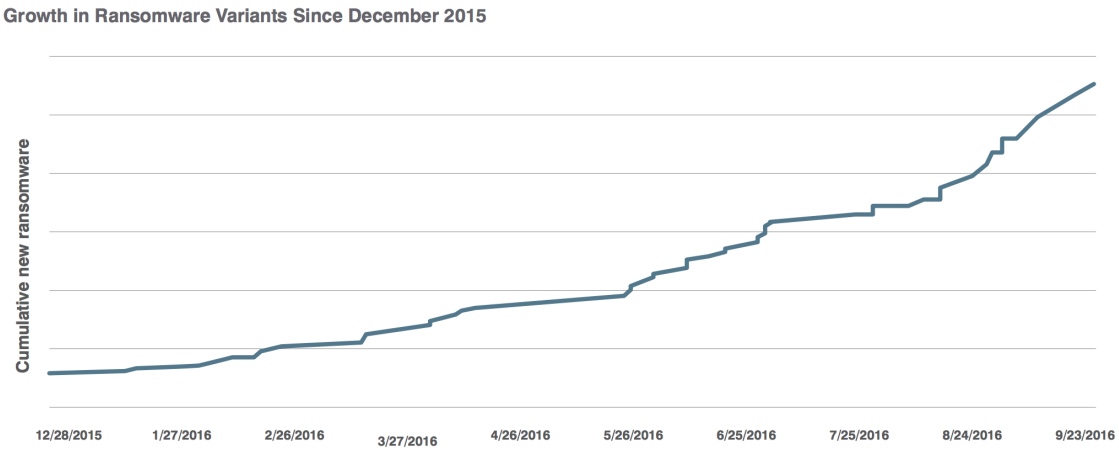


Figure 2: Proofpoint Q3 Threat Report Growth of Ransomware

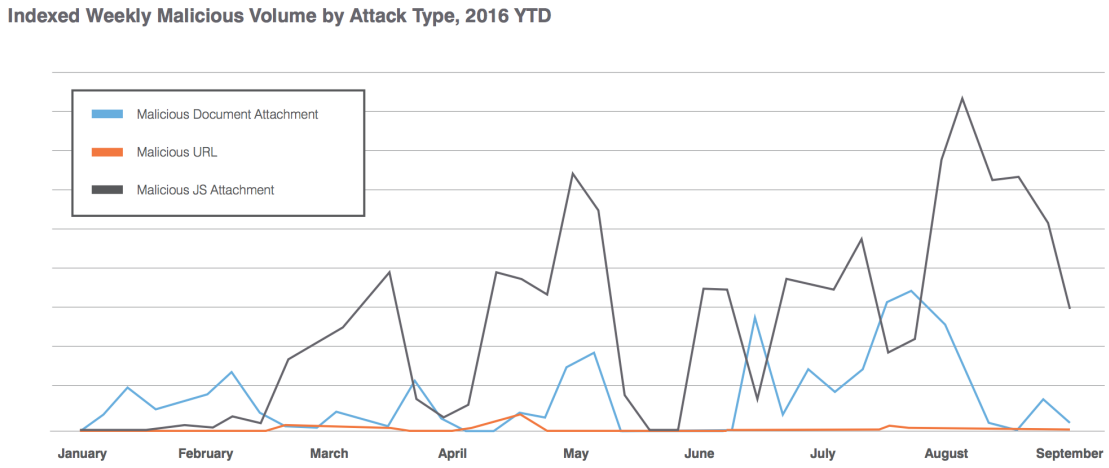


Figure 3: Proofpoint Q3 Threat Report Attack Type Activity

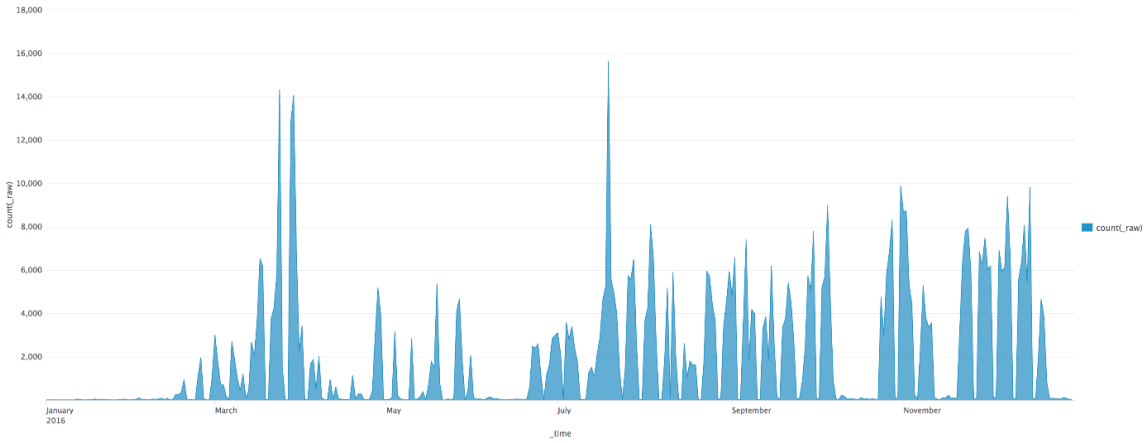


Figure 4: 2016 Phishing Campaigns Frequency (thanks to Xavier Mertens)

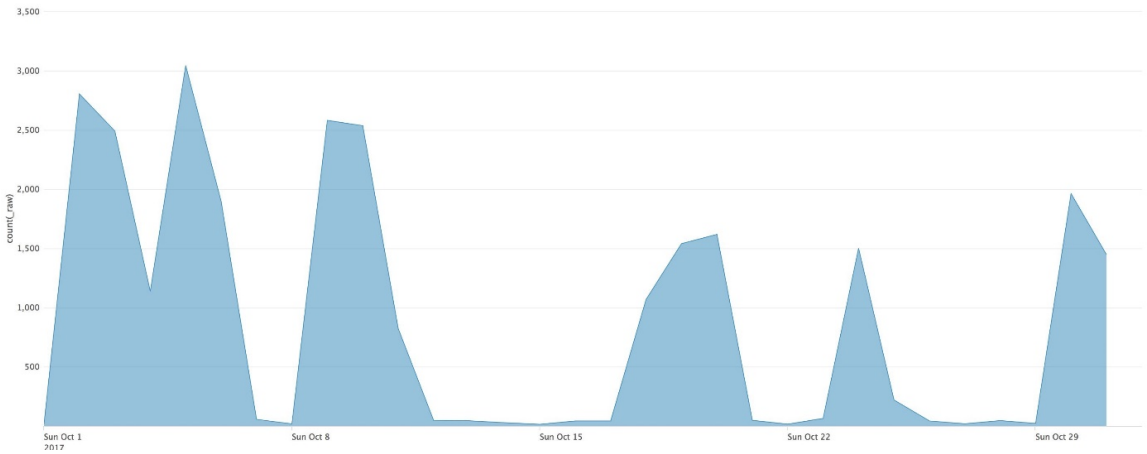


Figure 5: One Month Zoom of 2016 Phishing Campaigns Frequency (thanks to Xavier Mertens)

large majority, close to 75%, of the attachments are being sent out as compressed/zip files.

This is actually understandable considering efforts organisations and Microsoft have deployed to stop office file type attacks from being successful. However, the downside being that these campaigns effectiveness was relying on end users opening the archive and opening the file inside.

Perhaps an area of improvement to be had in security awareness programmes.

3 The Various Faces of Ransomware

A ransomware variant is defined based on the difference in the type of encryption used, the delivery method used, the attack type, the actor or any combination of them. F-Secure has a map of all the different variants that were released over the years. Zooming in on 2016 shows just how many campaigns were being carried out and the number of variants being used.

For our purposes and to be able to understand how a ransomware is built, the methods of encryption become more critical and these can be summed up into 4 simple types. In some cases, the campaign has been seen to evolve from one of the simpler methods using a script to a full-blown binary ransomware.

It becomes important as it does determine the level of complexity and work required which is fundamental to the analysis.

3.0.1 Script Based with Simple Encryption or Hashing

Interestingly, a number of the campaigns used simple tools and a script to encrypt the victim's data. The variant would either use built in tools like PowerShell on Microsoft Windows or go as far as downloading a script engine like python or php.

There are some advantages to using this method in that it is quick and easy to get something up and running. More importantly, it uses »clean« as in non-malicious, according to anti-virus products, binaries so it bypasses most malicious software detection tools. The disadvantage is that in most cases the key to decrypt and the method are plainly available either in the script itself or if you capture the command line. This makes it easier for security teams to unlock any encrypted files.

A simple example of this is a variant of Nemucod which used a batch file and 7zip to create encrypted archives. Similar to Nemucod, you can include variants like HolyCrypt and CryPy as well as Stampado written in AutoIt (an automation language).

3.0.2 Simple Hashing Binaries

While a script may be used to download the dropper, the next step up for a ransomware variant is to code a binary application. The encryption method typically is based on a hashing algorithm such as Base64, RC4 or SHA1 and blowfish. This does require the use of some developer tools such as visual studio and some programming or API knowledge of the target platform operating system.

The prime advantage of a ransomware of this type is the readily available hashing algorithm code and a compact simple binary that can be deployed directly via the dropper script or thru a trojan.

Like the script-based ransomware, these are easy for security teams to reverse. Hashes are easily recognisable and without a proper complex salting method typically easy to reverse. It is also common, unless using a library or taping into the API, to badly code the hashing method in which case it may well be even easier to reverse.

3.0.3 Secret Key Based Ransomware Binaries

The next step up in ransomware variants and most common implementations is to code a binary application based on a proper encryption method such as AES, GOST, DES/Triple-DES, ROT13 or XOR.

This does raise the complexity of the variant as you do need to manage a secret key and ensure that your encryption is not flawed or easily reversible. There are a lot of code sample algorithms out there and it is quite easy to find open source implementations. These variants become harder to reverse and thus more likely to ensure encrypted files stay encrypted.

The downside is the secret key management. There are two options when dealing with this type of ransomware. Either the secret key is hardcoded into the code or a key management system is required in the command-n-control backend and a unique key is generated for each victim. Using a hardcode secret key is easier to implement and manage but will be the same for every victim. Thus, if one victim retrieves that key all other victims can use it. A unique key per victim increases the complexity of the code. Either the key is generated by the code and gets uploaded or registered into a system managed by the campaign author; or the key is generated by a backend system when the ransomware variant connects to fetch it for the encryption process. In both cases, the key needs to be managed in some kind of key management system operated by the malicious party. If it is not, there is no way for the victim to get the unlock code.

This is one of the reasons the ransomware messages may contain an ID or code that must be sent to the malicious party. They use this to identify what unlock key is needed.

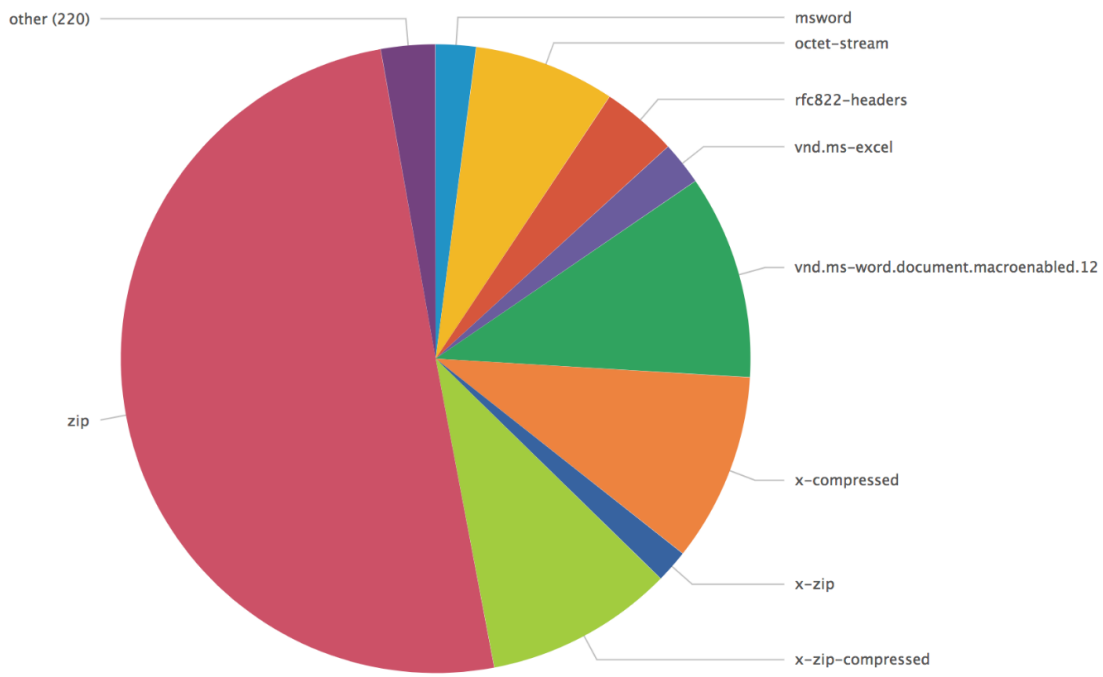


Figure 6: Types of Attachments from 2016 Phishing emails (thanks to Xavier Mertens)

3.0.4 Public Key Based

While very similar in terms of implementation advantages and disadvantages, the most complicated to implement as a ransomware is a public key based version. Most common is naturally an RSA based encryption used in ransomware. Like the latter two types, this does require good programming skills, developer tools and a good knowledge of the target platform.

The complexity to implement is much higher as the malicious party needs to have a back end key management system to store and generate the private key as well as the public version. The code also needs to be able to fetch the public key to be able to encrypt the target files.

3.1 But They All Behave Using the Same Principal

Whatever version is used, a ransomware has a pretty constant workflow and demonstrate the same phases, as shown in Fig. 9.

Delivery is the first step, which is either achieved directly or through a dropper. This initial stage's goal is to deploy the ransomware pure and simple; downloading the binaries and installing them on the local media. In some cases, it may also introduce other malicious software such as a remote access tool.

The second step is for the ransomware to execute. This is the launch of the ransomware either done directly from with-in or from a script initiated by the delivery mechanism. This initial execution phase may also set-up the environment before the encryption occurs: creating or fetching the encryption key, preparing persistence and dropping other non-encrypted artefacts.

The Encrypt step is the third and most important one. This is where the ransomware identifies and carries out the encryption of the host it has been deployed on. Typically, and in order to optimise speed of encryption, the ransomware will look for specific »user data files« across all media and drives, optionally will also look for any mounted shares. In some cases, the ransomware will focus the encryption on the file header and the first few KBs in order to avoid being bogged down in the encryption of very large files.

The fourth step is an optional step and not seen in every variant. The ransomware may look at spreading by deploying droppers on other hosts in the network either thru exploits or through open shares and removable devices.

Finally, the ransomware will display its ransom note and deploy its persistence techniques including the Run registry key and setting file type registry keys to open the note.

Why is knowing about this workflow important? Identifying them is key to understanding what needs to be developed to build or simulate a ransomware. The important steps to simulate a ransomware being execute, encrypt and ransom note, as these are the minimal activities needed to demonstrate a host's data being encrypted for ransom. Optionally implementing a delivery phase provides a more complete demonstration or testing platform.

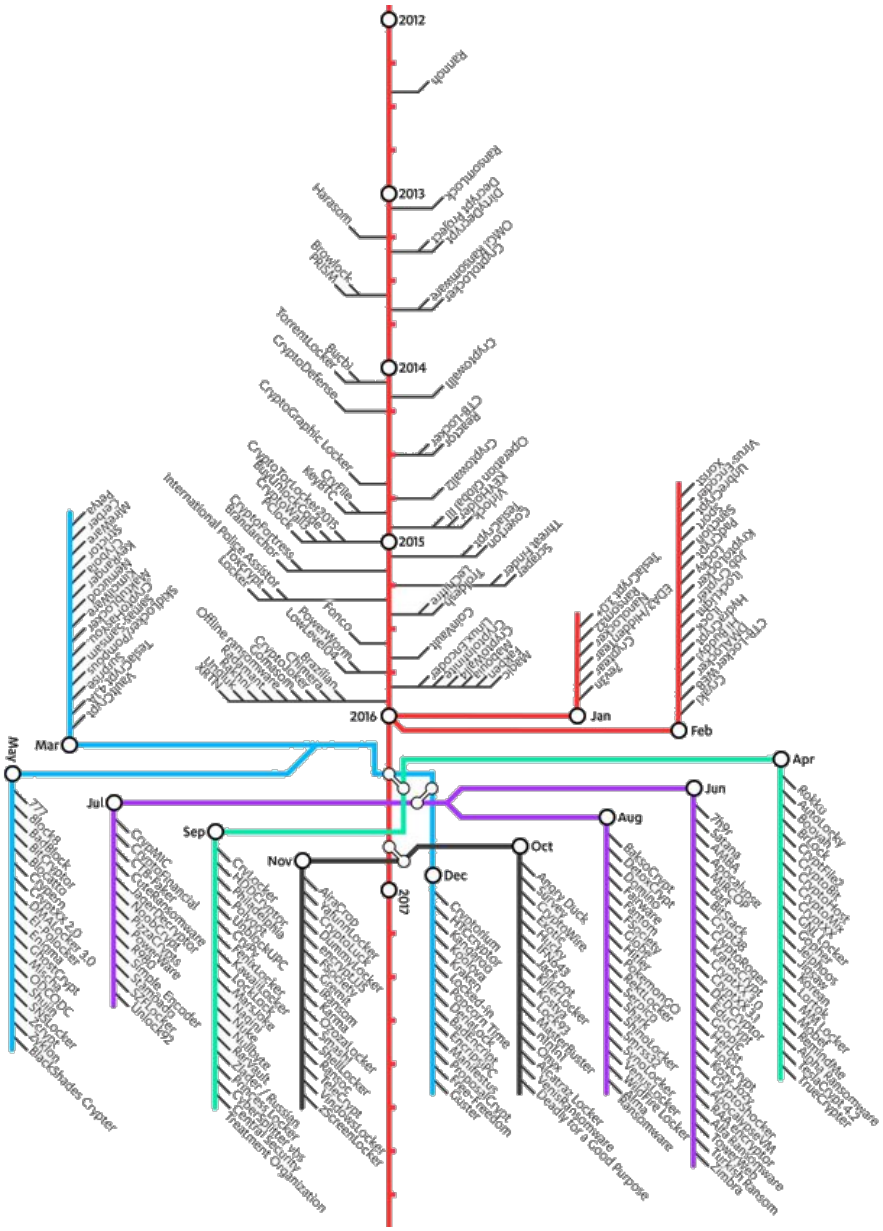


Figure 7: F-Secure Organigramme of Ransomware Variants for 2016

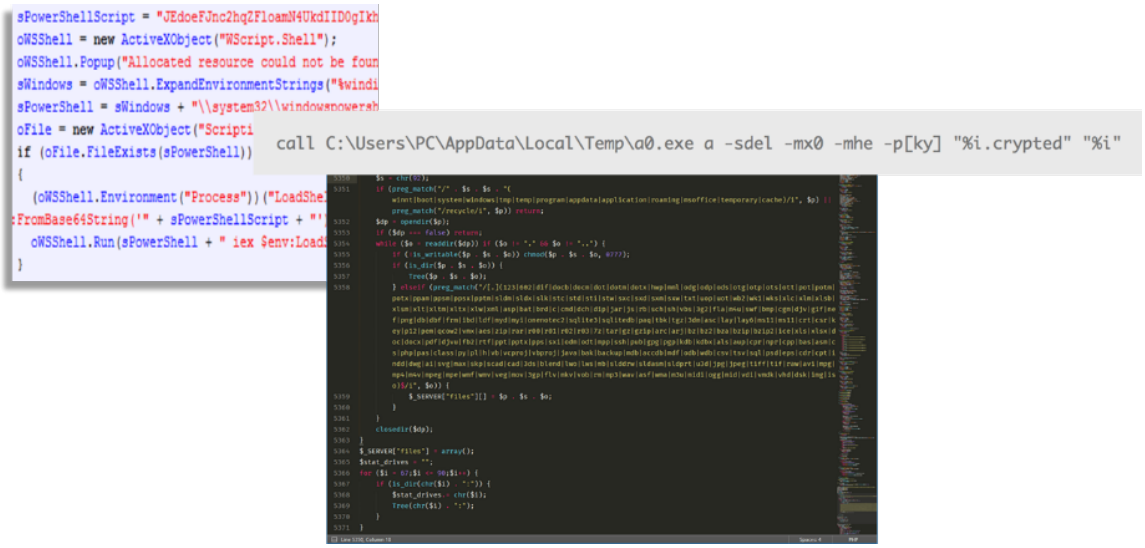


Figure 8: Example Script Based Ransomware

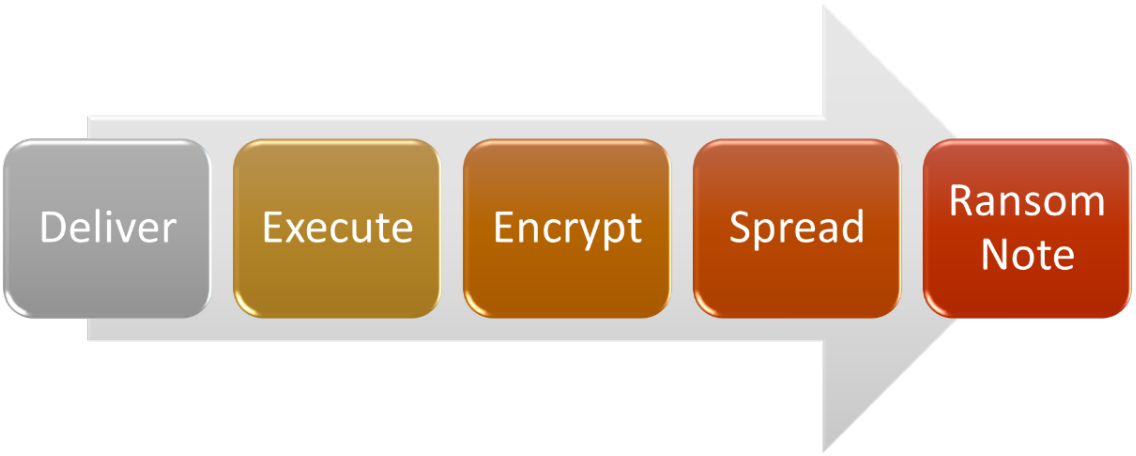


Figure 9: Workflow of a Ransomware

4 Where to Begin

Ultimately if you think about it, this is a software development project. Treating it as one helps give structure and ensures that key aspects are not missing and allows planning of how the simulation is going to be programmed and work.

4.1 Project Management vs. »Project« Management

Having been a developer and spent a fair amount of time working with project management as well product management, my initial thought was to think of this as a project but that lead my mind down the project planning rabbit hole. Trying to keep this simple, it made no sense to go down that hole.

Instead, a simpler method was to build a mind map of what I needed to address to develop this simulation of a ransomware. The full mind map is displayed in the following picture

Staying as close as possible to the typical workflow steps of a ransomware, the initial nodes in the mind map are delivery, encryptor (to represent the encryption phase), run (for the execution phase) and dropper (the delivery phase). I also added a decryption node as it would be needed to restore the encrypted files, c.f. later on about lessons learnt.

4.2 Getting the Right Encryption Method

The bulk of work really is the encryptor node of the mind map. This is the more complex part of the work to be carried out. Breaking it down you need an encryption method, a programming platform and finally the steps to parse for files and apply the encryption method.

Deciding on what programming language, it was necessary to take into account that this needs to be simple and quick. My initial thought was to shy away from more complex development using C++ or C# and to focus my attention on scripting languages.

Here you have a choice as it is easy to get either PowerShell, python or php running on the target machine. There is some limitation when using a scripting subsystem as depending on the version certain features might be available or not, making it harder to build the encryption or access local system files. PowerShell itself posed another problem depending on the version of Microsoft Windows being used as it was not necessarily enabled and, in some cases, posed limited set of commands making it difficult to build the encryption engine.

Ultimately, the decision was to go with php for its simplicity, ease of deployment and some other factors that are covered in the lessons learnt.

4.3 Yeah, but Which Encryption Algorithm

One important factor of the ransomware build is going to be the choice of the encryption method used. There are so many to choose from and plenty of libraries out there to help to get started.

4.3.1 Advanced Encryption Standard (AES)

The most common method used today is a symmetric key based solution known as AES or Advanced Encryption Standard. AES itself is public, readily available and was originally proposed and accepted as a replacement of DES. The algorithm named »Rijandael« developed by the Belgian cryptographists Daemen and Rijmen was adopted because of its security, speed and flexibility of implementation. That flexibility has seen it adopted into Wi-Fi encryption standards.

The algorithm works on the principal of taking data blocks of 16 bytes then applying several substitutions, permutations and linear transformation; commonly known as blockcipher. The strength comes as these operations are repeated several times, »rounds«. A unique key is derived from the encryption key for each round and applied to the calculation. AES block structures can vary, which provides us with an added level of complexity. This is where the terms Electronic

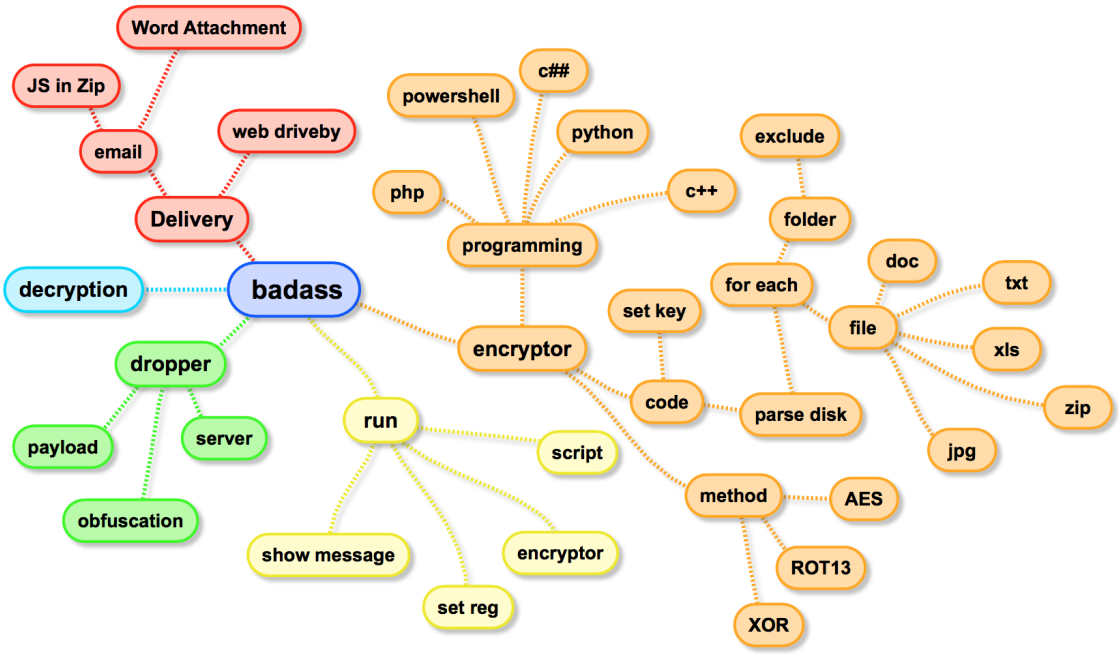


Figure 10: Mind Map for Developing the Ransomware Simulation

Codebook (ECB) and Cipher Blocker Chaining (CBC) appear alongside AES.

4.3.2 RSA Encryption

RSA is probably one of the most complex to implement and requires the use of libraries. The foundation of RSA Encryption is the use of asymmetric keys and is the brain child of cryptologists Rivest, Shamir and Adleman. RSA works by using two different but associated keys: public and private. Although associated, it is not possible to calculate the private key from its associated public key. The public key, as its name implies, is made public and anyone can use it to encrypt or verify messages. The private key should be kept safe by its owner and is used to decrypt the information or sign it.

The security principal of RSA derives from the mathematical problem of integer factorization. Essentially when you encrypt the data, it is raised to the power of the key then divided with the remainder of the product of two primes. A simple explanation is already complex in itself, so you can imagine that implementing this is hard and error prone. Thus, not a good choice for something simple.

4.3.3 The Simple Solution Emanates from the Days of Caesar

While AES might seem like a good choice, it would still be complex and time consuming to develop properly. In the history of encryption, one of the original ciphers was known as the »Caesar Cipher«. It is a simple substitution algorithm in which each letter is replaced by another letter based on a shift key or a map. So, for example, if you have the letter 'E' and use a key of left shift by 3 the encrypted value is 'B'. The advantage of such a cipher is the simple programmatic application as all you need is an array to map the original value to the new. Over the years this transformation has evolved and is now commonly referred to as a technique called ROT13; applying a shift of 13 letters.

While tempting, ROT13 would not offer the best simulation of data encryption and something a little more advanced, hard to brute force but still easy to programme was needed. Introducing XOR (or Exclusive OR). An XOR cipher operates according to the following mathematical principal:

$A \oplus 0 = 0 \quad A \oplus A = 0 \quad (A \oplus B) \oplus C = A \oplus (B \oplus C)$
$(B \oplus A) \oplus A = B \oplus 0 = B$

The $\oplus$ in the formula denotes an exclusive disjunction. In simpler terms, with a given key you apply a bitwise XOR operation to the data. To reverse the encryption, the same key and bitwise XOR is reapplied.

In pseudocode a simple version of this might look something like this:

```
for i = 0 to length of theText
  newstr[i] = theText[i] xor
              key[i mod key_length]
```

Easy enough to code in any programming language.

4.4 Prepare the Development and Testing Environment (Tools)

Part of the requirements was to keep this simple and not have any costly solutions involved in the build

process. It is quite easy to get carried away and be bogged down in complex IDEs (integrated development environments) so Visual Studio would not be a first choice for getting started. In fact, to build a very simple piece of code notepad would be sufficient.

Notepad++ provides an upscale simple text editor with some useful programming additions like code highlighting. It also provides a portable version, making it easier to use in testing environments or across multiple instances.

The programming language selected of php does require the availability of an interpreter or runtime. Again, simplicity is important and the advantage of using an interpreter like php is that older versions are easily obtainable. Older versions are interesting because they typically have a smaller footprint and less requirements.

Like any good programme, a ransomware still needs to be tested and validated. Care needs to be taken when testing the code. The risk when testing is that the host where the code is run ends up encrypted and if there is an error in the code or it is not possible to reverse the encryption would stay encrypted. So a throwaway instance becomes very useful as having to rebuild a test machine can be quite time consuming. Virtual machines with snapshot or image capabilities is the way to go here. The testing environment thus becomes a set of virtual machines with a baseline snapshot, which can easily be restored after encryption to speed up the testing.

5 And thus is Born the Ransomware

Encryption and programming language selected, the next step is to start programming. Having had some formal programming experience and because I like to do things in a structured way, it was natural to layout the main programme workflow using an activity diagram.

5.1 Main Steps in the Programme Flow

The programme itself needs to start with an initialisation phase. This phase will start by setting or generating the secret key. The next step is to set a starting path for the programme to find the files and encrypt them. I choose to do this because it would significantly control where the ransomware would act and also provide the latitude to play around on which directories the programme would act.

The next phase is the main programme loop. This phase is the actual action of finding the files and then encrypting them. The loop starts with a recursive search of all the files in the path. However, this search is limited to certain file types. It is important to limit the file types, because encrypting the wrong files could stop applications from working, or worse, impede the operating system and cause a blue screen of death.

For each file identified, the first step is to encrypt the file using the XOR code and once that is done the file is renamed. Renaming the file afterwards avoids any simple tools that might try to block the ransomware based on the filename extension type.

If there are more files in the directory structure, the process gets repeated. If all the files have been parsed, the loop ends, and the application stops.

5.2 The Code

The full ransomware code is not important and if the readers really wants to research this themselves, it is better to have pointers than the answer. A few portions of the code is highlighted in relation to the previously discussed workflow.

A first step is to establish the encryption key. The easiest would be to just set a string with the key. But thinking like a malicious actor, I realised it would need to be obfuscated or even encrypted itself. This would avoid any prying eyes or sysadmins from immediately figuring out the encryption password. So the key is base64 encoded. Of course this is not a perfect protection but does provide a small level of obfuscation. That's all that is really needed.

Once a key is available and useable in the programme, the main body of the code can begin. Any good programmer knows that to parse a tree, essentially a directory structure is one, the best technique is recursion. Thus the bulk of the search for files to be touched and the encryption itself are placed in a function that will recursively call itself for each directory and then process the files in the directory.

The recursive function is first called using the start directory which can either be the root of the drive, e.g. c:\, or an existing subdirectory on the drive, e.g. c:\Users. To avoid any blue screens of death or system failures, the programme needs to ignore certain file types and avoid processing. Primarily, it needs to avoid directories like \windows\, \system32\, \program files\, etc.

Using the function preg_match, the current directory in the recursion is matched to certain keywords patterns. If the result is true that directory and subdirectories are skipped. If not a match, the directory contents is parsed. If the item in the directory is a directory itself, the new path is passed into the recursive function.

If the item is a file, i.e. the directory bit is not set, the programme checks the file type. As a ransomware, the goal is to encrypt files that are »user content« thus DLL or other binary files are not of interest. The files that are interesting are the ones that typically have as file extensions, things like zip, tar, doc, docx, xls, xlsx, jpg, png, etc.

So if the file matches the criteria, the encryption process kicks in. The code starts by opening the file with fopen function in the 'r+' mode to allow for reading



Figure 11: Initialisation phase of activity diagram



Figure 12: Encryptiong loop of the activity diagram

and writing of the file from its start. The importance of starting at the head of the file is due to the fact that most file structure and type information is located in the first few bytes. A good ransomware needs to encrypt this information so that the original file is no longer useable.

The encryption itself can begin. The first step is to open and read in the file. The code use fread to pull in a buffer of 1024 or 2048 bytes. It is preferable and better to only read the first few bytes of a file for multiple reasons, including reducing the need for the programme to have a large memory usage and providing faster processing. If the programme reads only a portion of the file, it will take less time to access but also less time to run the encryption algorithm on the data loaded in the buffer. Interestingly, a ransomware doesn't need to encrypt the whole file as long as the file header and start of contents is corrupted enough, so the original application or editor won't be able to make sense of it.

This is important as the faster the ransomware can encrypt the contents of the victim's hard drives, the more effective it will be.

A buffer in languages like php can be programmatically accessed as an array which is extremely useful and makes processing the data easier. With the file opened for write and the buffer loaded, the encryption can begin using a for loop to process each array element. The loop changes each array element by XOR'ng its value with the ordinal of the key. The element of the key to use is determined by the current buffer index shifted to the length of the secret key.

Once manipulated the buffer needs to be written back on top of the start of the file and closed. The fseek function repositions the file pointer to the beginning and the fwrite function dumps the buffer back into the file.

The programme closes the file, effectively ensuring the new encrypted file header is written to disk and proceeds to rename the file. To rename the file, the ransomware appends a new file extension such as .crypted, .enc, etc. It leaves the original file extension so that when the decryption occurs removing the new extension makes the file accessible again.

That's it, that's all that is needed to build a basic ransomware.

5.3 One More Thing, Delivery

The demo still needs a means to deliver the now completed ransomware code. The end goal is to have a functional demo, so the ransomware should be delivered through a technique discussed previously. To complement and really finish the demo, a simple zip attachment with a JavaScript or even some obfuscated PowerShell in an office document attachment will suffice. This script simply needs to download the ransomware file(s) from a web server and start it. Actual development of this is left to the initiative of the reader as it can take several forms.

The demo is complete. Just place the fake attachment in an email on the demo platform and start clicking.

5.3.1 Alternate Delivery Methods

There are better methods to deliver this ransomware payload. Most of these techniques require some form of exploitation kit or RAT. These are presented as toolkits that require some digging and introductions in the Dark Web or to services that provide these kits. Most of these tools work on the deployment and delivery via a dropper integrated into an office document or archive attachment. Using them also needs a backend, i.e. a small botnet.

While feasible this would add considerable effort and complexity to the base demo build.

6 Lessons Learnt During the Build Process

No good project would not end with a lessons learnt phase. There were some to be had during the build of the ransomware. Two stood out when reviewing and drawing conclusions on the build of the process.

Firstly, choose the platform and tools wisely. It is easy to get bogged down in the complexity of a tool, for example, using an IDE like Visual Studio or Eclipse, while interesting from the developer's point of view,

```
// REMEMBER TO REMOVE next 4 comments
// OK first we set an encryption code in this case:
//        No, look, there's a blue box. It's bigger on the inside than it is on the outside.
// To hide it from prying eyes we base64_encode
$k=base64_decode('Tm8sIGxvb2ssIHRoZXJlJ3MgYSBibHVlIGJveC4gSXQncyBiaWdnZXIgb24gdGhlIGluc2lkZSB0aGFuIGl0IGlzIG9uIHRoZSBvdXRzaWRlLg==');
$s=chr(92);
```

Figure 13: Hiding the encryption key (code)

```
//Skip system folders and system files or we die a blue death
if(preg_match('/'.$s.$s.'(winnt|boot|system|windows|tmp|temp|program|appdata|application|roaming|msoffice|temporary|cache)/i',$p) || preg_match('/recycle/i',$p)) return;
$dp=@opendir($p);
```

Figure 14: Only search for non-system and important directories

does require a lot of resources. Creating a simple script in these environments is not as simple as just opening a new text file.

The scripting or interpreter that is used to run the ransomware code is extremely important. Newer versions of many of these environments are not standalone and require not only the use of the executable but also require libraries in the form of DLLs. Having additional libraries makes the delivery more complex as more files must be downloaded and deployed to the target host. Going back to older versions helps as these typically are very basic and do not need additional support files. Alternatively, when the interpreter is opensource it is possible to recompile trading off an easy build for something much more complex and needing greater level of testing.

PowerShell comes built into the more modern versions of the Microsoft Operating system, which is great as it simplifies things immensely. The downside is that PowerShell is not installed by default on earlier versions like Windows 7 and requires some form of activation or in the case of non-Windows devices actual installation.

The second main lesson learnt is to snapshot frequently or to ensure a restore image is readily available of your development environment. It is extremely easy to accidentally run the malware while checking the code or building the delivery mechanism. This leads to the encryption of the disk including the ransomware itself and the decryption script, if it exists.

Yes, it happened and of course the snapshot was out of date! Unfortunately, the first iteration of the decryption script failed, so a secondary development environment was needed as well as going back to the drawing board to fix the script. Care is definitely needed!

7 Getting Past the Prophylactics

Part of building this ransomware was to test the various anti-malware solutions and next generation protection solutions. Using the php interpreter as the main executable for the ransomware gave interesting results. Only 5 or 6 anti-malware solutions would flag this as malicious while most online services like VirusTotal showing this as not suspicious. When it was flagged as bad, the main reason was because of the older version of the interpreter executable; updating it would once again flag it as not suspicious.

For next generation products, Cylance was specifically tested late 2016. With the default settings, the ransomware ran effectively. Getting the solution to detect and stop the ransomware required setting an additional option to detect and block scripting. So, there is a benefit to using an interpreter to do the ransomware in avoiding detection.

Effective detection only really comes from products that support behaviour based detection and even then that varied considerably. The biggest let down was the timing of the detection and blocking. Some of these products detect based on the file name change but in many cases this is too late as the encryption occurs beforehand.

8 Conclusions

A study from the security firm TrustLook Inc showed that 38% of victims of ransomware paid it to get their files recovered. A survey carried out by Intermedia showed that 59% of office workers hit by ransomware paid for it themselves without informing their employer. These statistics, although high-level, are interesting as the demonstrate that victims are paying, and it is possible to make money.

The development of the initial ransomware programme was simple enough and by using a language interpreter like php took less than 24 hours to develop and with modern libraries and ransomware as a service offering, the time needed could be far less. The real challenge and complexity comes when an exploit kit or more advanced techniques are introduced, leading to a longer integration cycle.

So, one day's effort for a vanilla ransomware delivered via a simple phishing email. Knowing the results of some of the studies previously mentioned and with an average pay-out per victim of 500 to 1000 USD, it is clear that with a development cycle of less than 24 hours the return on investment is alluring, yet best left to malicious actors who don't abide by the law.

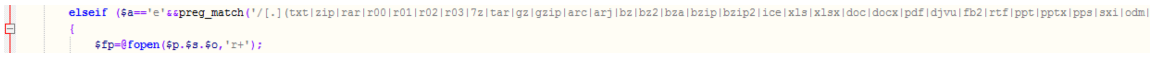


Figure 15: Selecting files of a certain type only

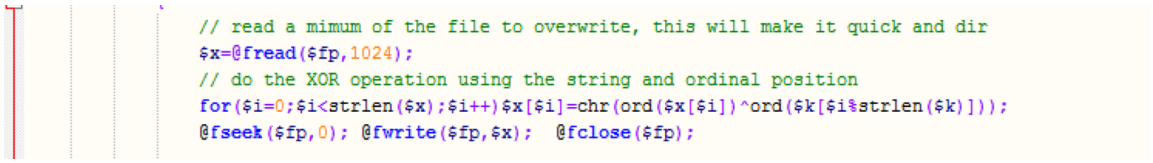


Figure 16: Read a buffer of the file contents and encrypt

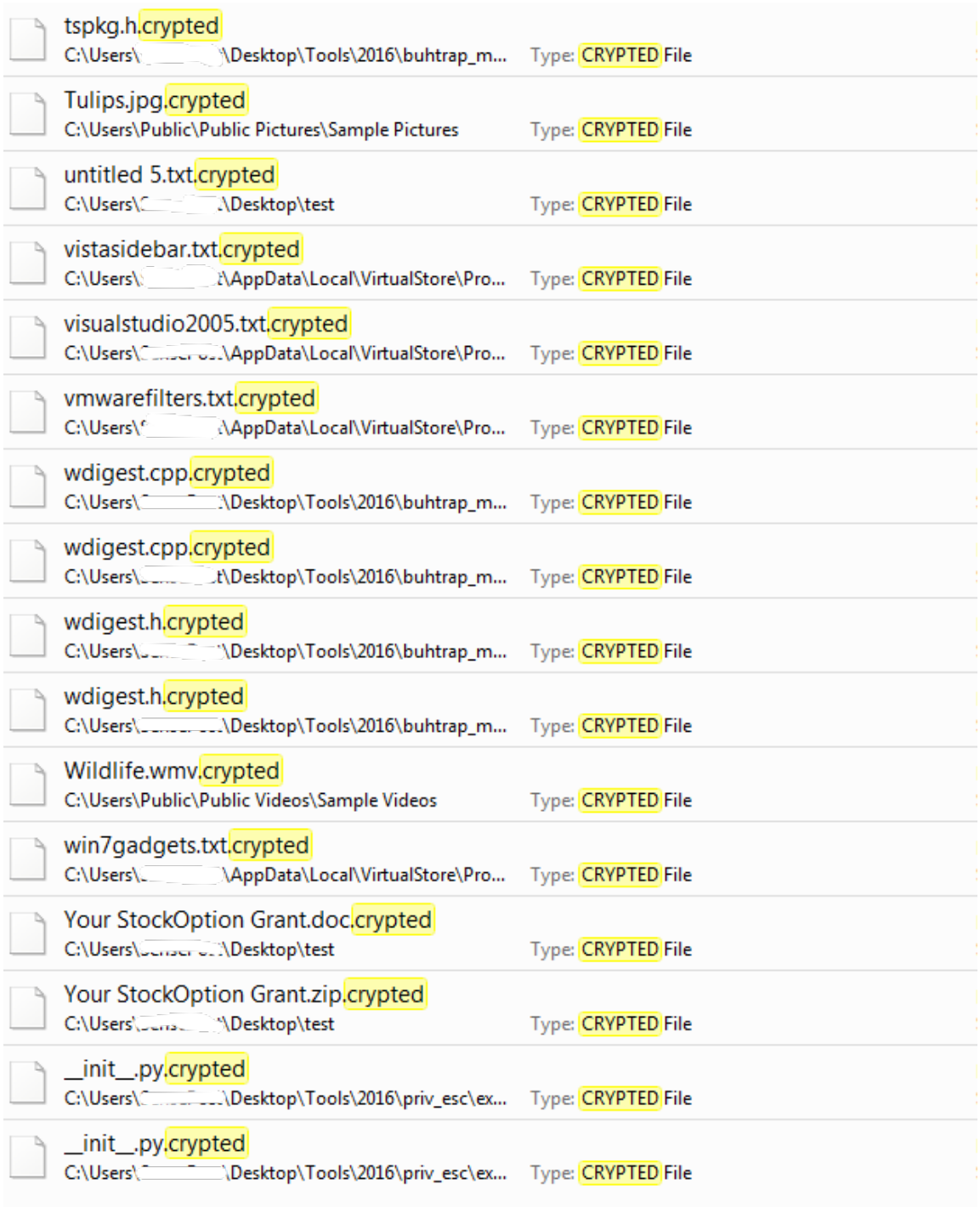


Figure 17: Careful it is easy to end up encrypting the development environment

About the Author

With over 25+ years experience, Thomas has a unique view on security in the enterprise with experience in multi domains from risk management, secure development to incident response and forensics. In his career, he's held varying roles from incident responder to security architect for fortune 500 companies as well as industry vendors and consulting organizations. Currently he plays a lead role in advising customers while investigating malicious activity and analyzing threats for Digital Guardian. He's also a strong advocate of knowledge sharing and mentoring through being an active participant in the infosec community, not only as a member but also as director of Security BSides London and as an ISSA UK chapter board member.