Matthew Yaspan

Exploring Algorithmic Methods to Detect Ransomware

Abstract:

Ransomware is a type of software that, when installed on a computer, begins to encrypt files on the system. The software then prompts the user to pay a certain amount of money to regain access to their files, holding potentially valuable information for ransom. This practice is becoming increasingly common, making it desirable for an automated solution to be created, allowing users' sensitive information to be protected. Currently, individuals and firms have managed to create products to identify ransomware programs by providing software to detect programs on a machine that are encrypting files and terminate the threat. Exabeam[1] has developed a product using machine learning algorithms based on user and entity behavior attributes to identify and kill the threat. Ransomwhere?[2] is a product specifically for OS X to alert users to any process that attempts to encrypt personal files. This paper discusses and examines ways in which to detect ransomware programs, and shows that there is some promise in these methods, but none are guaranteed and there is a long way to go until the threat of ransomware is mitigated.

Introduction:

Ransomware is a type of software that, as the name suggests, holds a computer or system's files and functionality hostage in exchange for a ransom. The mechanism by which this generally happens is the following: The user of a computer or system unwitting executes or downloads a program that then takes over the terminal of the user and simultaneously begins encrypting all of the files on the system. In order to cover the files and system functionality, the user must follow instructions given by the program, which generally involves wiring money somewhere. Generally speaking, the sum of money is not small. If this program infects the system of an enterprise, millions of dollars-worth of information can be held hostage, and thus, demand millions of dollars.

The genesis of Ransomware is likely when Dr. Joseph Popp, an evolutionary biologist, created the AIDS Trojan, which was handed out by Popp via floppy disk to tens of thousands of attendees to a World Health Organization AIDS conference. When inserted and the program executed, it took over the machine, encrypted the files via symmetric encryption, and demanded $189 be sent to the "PC Cyborg Corporation"[3] . Popp was eventually caught and a workaround was found due to the fact that the encryption was symmetric, but his actions sowed the seeds for hackers to use more sophisticated iterations of encryption and demand more ludicrous sums of money.

While Popp's actions are the first documented case of Ransomware, it's popularity amongst hackers is likely also based in its relationship to other forms of malware. Indeed, demanding money for protection is also prevalent in "scareware", albeit in a subtler way. Scareware is a class of programs that tend to pop up on a user's screen with an appearance emulating an alert from the user's operating system or anti-virus software telling the user that they need to install software or run a scan to detect or destroy malware. By doing so, they entice the user to download software that is, in fact, malware in and of itself[4] . Generic pop-ups on or from untrusted websites work in similar ways: They entice users to click something which then installs malware on the system. Even if hackers had never heard of Dr. Popp, they certainly could have drawn on pop-ups and scareware and stretched the limits of what malware can do by asking for money.

Trivial Solutions:

Many anti-virus solutions now purport the ability to detect or prevent ransomware. They claim to either disinfect restored computers, use key databases recovered by the police to attempt to decrypt the files (success seems to occur rarely), or some claim to perform behavioral analysis[5]. Some mainstream security suites for a normal PC user provide the ex-post disinfecting, but only a few do the other two. We'll delve a bit into behavioral analysis in a bit more depth below.

Less advanced solutions can potentially work given low sophistication by the hackers. Pop-up blockers can prevent meddling scripts from obfuscating one's ability to use their computer. Sometimes rebooting in safe mode can rid the user of any problems, and if no important files were encrypted, they can get on with their lives. Finally, there are simple steps a user can take, such as ensuring downloaded files don't have any curious extensions[6] to make sure that they have not downloaded any ransomware. It should be noted that in the Windows operating system that the file extension is pivotal in how a given file is opened or executed, so this advice is especially pertinent to windows users.

Finally, there is one trivial solution that is hardly a solution at all. The FBI recommends that the best way to deal with more sophisticated ransomware, such as Cryptolocker or CryptoWall, is to simply pay the ransom[7]. Unlike Dr. Popp's AIDS Trojan, these applications use sophisticated forms of encryption that are impervious to the trivial solutions noted above. They use methods of encryption strong enough so that no algorithm can be reasonably expected to decrypt the files in a reasonable amount of time. Hospitals around the world have been hit by ransomware[8,9], holding people's lives in the balance. The malicious actors were able to squeeze millions out of hospitals in exchange for people's lives. Indeed, in such scenarios this is literally a hostage situation, making the moniker "ransomware" especially apt. The FBI reported losses of $18 million due to ransomware by organizations around the country in fiscal year 2015, and some estimates put total losses due to ransomware in 2015 as high as $350 million[10].

Methods of Detection:

Beyond the trivial methods of detection listed above, softwares such as Ransomwhere?[1] and Exabeam[2] claim that they can use behavioral analytics to detect and neutralize ransomware. The assertion is that certain mathematical methods, that are detailed below, can be used to protect users and stop ransomware before the damage is too great.

The framework can be distilled into three steps:

1) Monitor I/O operations
2) Determine if a file is encrypted or in the process of being encrypted
3) Determine if the process is trusted based on its behavior

The first step is obvious. A programmatic, sustainable solution to ransomware needs to routinely examine the set of processes running on a machine, most importantly those that operate or attempt to operate on the filesystem. Encryption of files is what makes ransomware such a potent tool for attackers, and the ability to access and encrypt files is what indeed makes it ransomware. Using this data, it can proceed to steps two and three to detect unique behaviors or those likely to indicate that a process is ransomware, or at least maliciously making changes to the filesystem

The second step is determining if a file is encrypted or being encrypted. The best public methodologies according to devttys0 include measuring file entropy, the chi-square distribution, and Monte Carlo Pi simulation[11]. File entropy (Specifically Shannon Entropy of the bytes of a file) is a value that estimates the randomness of the data in the file. To calculate it, a histogram is created of the byte values of the file, and using the histogram, entropy (H) is calculated as follows:

$$H = -\sum_{i=0}^{255} p(i)\ log_2\ p(i)$$

Where $p(i)$ refers to the probability of byte value i occurring the file. The maximum value of H by this specification is 8, and it is only that value when the probability of each byte occurring within the file is exactly equal. The minimum value of H is 0, which would indicate no variance at all in the bytes. Generally, encrypted files have H values that are very close to 8, although it can vary by encryption method.

Entropy is a reliable metric to show data randomness, but it suffers in its ability to discriminate encrypted files from compressed ones. As mentioned in devttys0, it can be difficult when using entropy to distinguish between files that are encrypted and files that are compressed. In practice, both generally have entropies that approach 8, and while compression can have chunks of non-random areas in the filestream, generally they are still difficult to detect and depend on the type of encryption or data compression used.

The chi-squared test compares the observed χ^2 value with the null hypothesis value to see if the data is truly randomly distributed. The formula is shown below:

$$\chi^2 \sim \frac{(O-E)^2}{E}$$

Where O refers to the vector of observed instances of each byte value and E refers to the vector of the expected instances of each byte value with the null hypothesis, which will be uniformly distributed.

Monte Carlo Pi approximation is another test of randomness. In Monte Carlo Pi, a randomly drawn set of coordinates is drawn from within a square. If the point would lie in a circle inscribed within the square, it has a value of 1. If not, it has a value 0. The average value of the points should approximate the ratio of the area of the circle to the area of the square. This ratio times 4 should be approximately equal to Pi, because the square in which a unit circle would be inscribed within would have an area of 4. The Monte Carlo Pi Test for randomness in files converts every 3 bytes of the file into coordinates in a square, and these are the points used to estimate Pi. High error in the estimation of Pi indicates little randomness and low error indicates sufficient randomness to adequately estimate Pi. According to devttys0, lower Chi-squared test values (<300) combined with high Pi estimation error (>.03%) is indicative of compression. Higher Chi values and low Pi error are indicative of encryption.

The third step is to determine if the process is trusted. Processes that alter the file system and compress data could be bad actors, or they could simply be performing orders carried out by the user. A detection and/or removal program should have high specificity and not kill processes or alter the user's experience by constantly alerting them whenever routine file operations are taking place. Some steps to test this include checking the checksum of the program to ensure that it is what it claims to be and perhaps corroborating the checksum with a database, but still, a more robust algorithm may be necessarily.

Behavioral analytics, as previously mentioned, is used by current software solutions to ransomware, and is likely to be the last and most important line of defense against it. Behavioral analytics is simply utilizing patterns that ransomware programs have in their interaction with the file system to attempt to single the process out as a bad actor. Ideally, a process that is encrypting files maliciously will have a distinct way of behaving that can allow a system to identify and neutralize it. Methods of behavioral analytics employed by software solutions have included conditional rules that dictate in human-readable rules which programs can be trusted and which cannot. This is a laborious process for the user,

especially when they have little programming or IT experience. The rules can be too general or too specific, and often lack accuracy. Logistic regression can run in the background, but can be computationally expensive and imprecise.

Static Analysis of Ransomware:

In order to understand ransomware a bit better, I downloaded and analyze source code for a Proof of Concept called DUMB[12], a C# Ransomware program that was created to run on Windows in 2004. What is most immediately astonishing about the program is how simple it is. The code, without images, is about 20 KB large, the preponderance of which is whitespace. In short, the code has two basic functions. The first is taking over the host machine's desktop, which it does in a few lines in main:

```
IntPtr hProc = IntPtr.Zero;
try
{
    WinAPI.SwitchDesktop(hNewDesktop);
    bool workdone = false;
    BackgroundWorker bg = new BackgroundWorker();
    bg.DoWork += delegate
    {
        WinAPI.SetThreadDesktop(hNewDesktop);
        try
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            if(!File.Exists(folder + "crypted"))
                Application.Run(new CryptWindow(false)); //Encrypt, you wouldn't actually do this in a ransomware, this is purely for t
            Application.Run(new Main()); //Ransom window
            Application.Run(new CryptWindow(true)); //Decrypt
        }
        finally { workdone = true; }
    };
    bg.RunWorkerAsync();
```

The method of encryption is quite simple, unlike CryptoLocker and CryptoWall, which use asymmetric encryption. DUMB simply generates a key and XORs it with the bytes of files:

```
//XOR works both ways, so we don't need one function for Encrypting and one function for Decrypting, they're both the same thing for XORing
public static void CryptFile(ISAAC csprng, byte[] subkey, string loc)
{
    FileStream s = null;
    int[] oldmem = null;
    try
    {
        s = File.Open(loc, FileMode.Open, FileAccess.ReadWrite, FileShare.None);

        oldmem = new int[ISAAC.SIZE];
        for (int i = 0; i < ISAAC.SIZE; i++) oldmem[i] = csprng.mem[i]; //Fast copy

        for (int i = 0; i < subkey.Length; i++)
            csprng.mem[i] ^= subkey[i];

        byte[] buffer = new byte[ISAAC.SIZE];
        int read = s.Read(buffer, 0, ISAAC.SIZE);
        do
        {
            csprng.Isaac();

            for (int i = 0; i < read; i++)
                buffer[i] = (byte)((buffer[i] ^ csprng.rsl[i]) % 256);
```

The ISAAC class is a random number generator specified elsewhere in the program, and the bytearray passed to subkey is hard-coded. The program is devastatingly simple and yet these programs have shown to be incredibly difficult to detect, which is and should be worrying.

Analysis and Results:

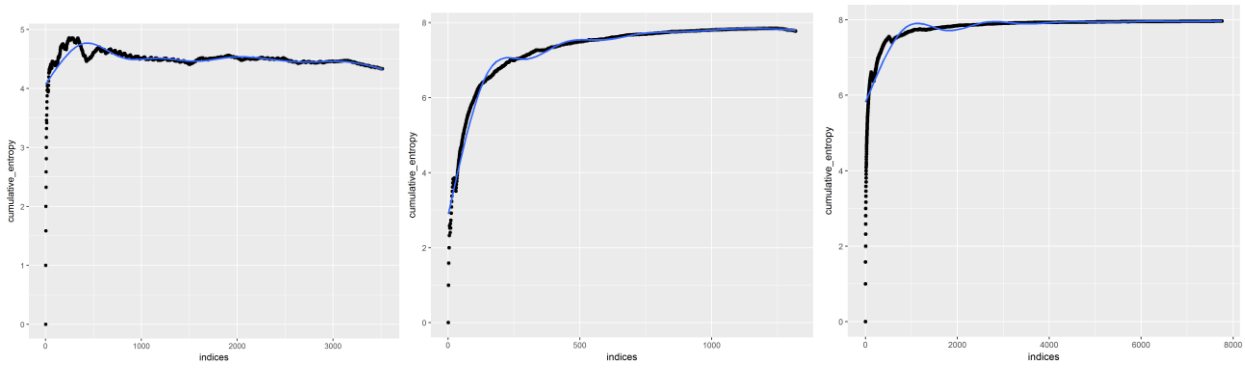
To test the framework detailed above, I experimented by testing the mathematical algorithms detailed above in their ability to discriminate between normal, compressed and encrypted files and also analyzed a dataset of I/O operations on my windows machine while DUMB was holding my data hostage. The results are shown in two sections below.

Encryption Detection:

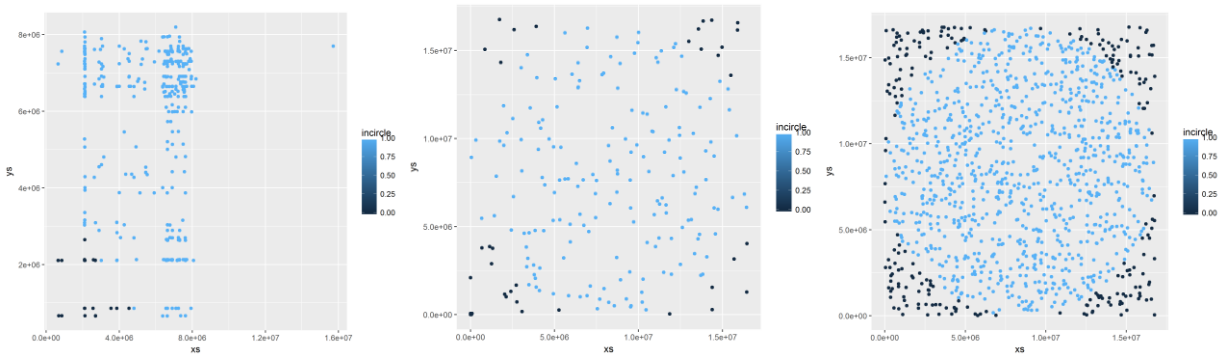
I tested the methods described on a few sets of files, including source files for DUMB, then zipped and encrypted versions of these files, then a word document sized a little over a megabyte and its zipped and encrypted versions. The table below shows the entropy, Monte Carlo Pi error and Chi Squared distribution for these files:

File and Format	*Entropy*	*Chi-Square number*	*Monte Carlo Pi Error*
Deposit.Designer.cs	4.538408	198700	0.0730851
Deposit.Designer.zip	7.815735	598	0.001847518
Deposit.Designer.cs Blowfish	7.81931	2725.9	0.0328813
Crypt.cs	4.334781	127390	0.1250877
Crypt.zip	7.774005	655.44	0.02902479
Crypt.cs AES Encrypted	7.961595	418.94	0.02210489
CTF_writeup.docx	7.992547	20707	0.01579917
CTF_writeup.zip	7.998987	2061.5	0.00013673
CTF_encrypted AES encrypted	7.99985	310.36	0.0008578218

Also included are plots showing the cumulative entropy by byte offset for Crypt.cs, Crypt.zip and Crypt.cs AES Encrypted:



And plots showing the Monte Carlo Pi results of each of these respective files:



The results indicate inklings of the patterns mentioned in the methodology section, the results are not clear enough to delineate cutoff points where we can be sure the file was encrypted. Multiple methods of zipping and encryption make the problem even more difficult: The Blowfish-encrypted file above had near identical entropy than the zipped file and performed worse in other measures of randomness. For each file, entropy calculated the non-altered file as least random. The zipped files were significantly more random than their non-zipped counterparts, and the encrypted versions were the most random. Still, because .docx files, which is the standard Microsoft Word document format, are automatically compressed, the non-encrypted .docx file had an entropy extremely close to the maximum of 8, more so than the encrypted versions of the .cs files. This basic contradiction makes it impossible to use as a generic measure of encryption.

Monte Carlo Pi and Chi-square distribution appeared to be able to differentiate files significantly, but there still exists a similar issue in which no threshold made itself apparent on these basic files, and the patterns were not robust to slight variations in file types and compression algorithms. The Monte Carlo plots above, however, did show that it could be a promising method. The left plot shows an unencrypted file, which has clustered points that barely cover the range of the space. The zip file did so better, while the rightmost compressed file had enough randomly distributed points to very accurately fill the circle and help for a very accurate estimate of Pi.

Overall, these methods seem to dubious to differentiate between encrypted and zipped files in a concrete fashion. It is possible that a mix of conditional rules and these metrics could move the needle a bit, but it doesn't seem likely to work with much degree of specificity.

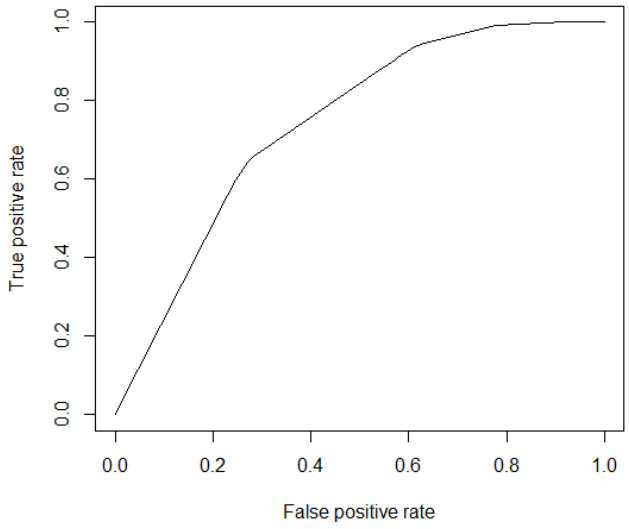
Identifying untrusted processes:

To test out the ability to recognize an untrusted process, I created a sample dataset using my own machine and Windows Process Monitor[13]. I recorded all I/O operations on my computer while I ran the DUMB Proof of Concept program as well as when routinely editing files using a Text Editor and with Microsoft Word so that there were measures of a myriad of activities one could normally do with their computer in addition to the ransomware programs. I then took the log file and marked all operations performed by DUMB and used the dataset to compute a logistic regression. The response variable of the regression was the indicator of whether the process was DUMB, and the predictors were interacted indicators of each I/O Operation and the Result of the operation.

There were too many coefficients to go over in total, but there were some very powerful indicators in terms of I/O operations and results that showed DUMB in action. The most powerful indicators included successful executions of FileSystemControl, NotifyChangeDirectory, and DeviceIoControl, which didn't appear to occur as often in Word or Text editor operations, but occurred in DUMB. Other

positive indicators included INVALID PARAMETER results when the process attempts to run the operators of QueryBasicInformationFile and DeviceIoControl, and more generally whenever the procedure of QueryRemoteProtocolInformation, so perhaps monitoring these operations could be of use.

The ROC curve for the logistic regression on in-sample data is plotted below:



The True Positive Rate indicates observations of File I/O operations the algorithm designates as by DUMB and are correctly identified, and the False Positive Rate indicates the rate of false attribution to DUMB. The way this curve can be interpreted is how well we can discriminate DUMB from the rest of the processes as we increase our algorithm's sensitivity. The Area Under Curve metric is 0.742. The lowest this metric can be is 0.5, indicating zero discriminative ability for our algorithm, and the highest is 1, indicating complete ability to discern non-ransomware from ransomware. 0.742 is underwhelming not good enough for any reasonable level of confidence. This is compounded by the fact that DUMB was written in 2004, and is likely as unsophisticated as it gets.

Discussion and Conclusions:

Ransomware is an ingenious form of malware that is both simple and lucrative. Its simplicity allows it to be both easy to create and extremely difficult to be automatically detected. Indeed, simply identifying encrypted files is a non-trivial task, and any successful method will take a nontrivial amount of time during which a ransomware program could be in the process of encrypting many more files. Combine this with the need to identify unique patterns of operation of a malicious process vs. other non-malicious I/O altering processes, this task seems borderline impossible. In a highly controlled environment, with no dedicated time limit and using advanced mathematical techniques, the algorithms evaluated in this paper performed unreliably in their tasks of identifying encryption or the operation of an untrusted process. This is scary and shows how far we need to come in this field.

To the Community:

Ransomware is a real software threat that will take sophisticated algorithms and advancements in technologies to eradicate, if that is at all possible. It will also take software that allows the user to give

input on whether processes are trusted or not in critical scenarios without making people's machines unusable. Still, the negative effects of ransomware can be mitigated much in the same way any type of malware can be mitigated: By educating people on the proper use of a computer, the internet, and engendering a healthy mistrust of software, we can begin to see diminished losses at the hands of ransomware. If people check the checksums and read the fine print, we can move the needle just a bit. Perhaps this is an unreasonable expectation from most people, but for people who see their machine and information as valuable resources, it is absolutely critical.

Sources:

1. http://www.exabeam.com/wp-content/uploads/2016/06/Exabeam_Ransomware_DS.pdf
2. https://objective-see.com/blog/blog_0x0F.html
3. https://www.knowbe4.com/aids-trojan
4. http://whatis.techtarget.com/definition/scareware
5. http://www.techworld.com/security/7-best-ransomware-removal-tools-how-clean-up-cryptolocker-cryptowall-extortion-malware-3626974/
6. https://www.netfort.com/blog/methods-for-detecting-ransomware-activity
7. https://securityledger.com/2015/10/fbis-advice-on-cryptolocker-just-pay-the-ransom/
8. http://www.csoonline.com/article/3033160/security/ransomware-takes-hollywood-hospital-offline-36m-demanded-by-attackers.html)
9. http://www.itgovernance.co.uk/blog/now-two-german-hospitals-also-being-held-ransom-by-cyber-attackers/
10. http://download.bitdefender.com/resources/files/News/CaseStudies/study/59/Bitdefender-Ransomware-A-Victim-Perspective.pdf
11. http://www.devttys0.com/2013/06/differentiate-encryption-from-compression-using-math/
12. https://github.com/AlphaDelta/DUMB
13. https://technet.microsoft.com/en-us/sysinternals/bb896645.aspx