DEAD AND BURIED IN THEIR CRYPTS: DEFEATING MODERN RANSOMWARE

Samir Mody & Gregory R. Panakkal
K7 Computing, India

Email {samir.mody, gregory.panakkal}@k7computing.com

ABSTRACT

Cryptolocker, Cryptowall, CTB Locker, etc. are well-known families of modern ransomware which use strong encryption algorithms with large asymmetric keys to encrypt target files, rendering them nigh on impossible to decrypt locally since the private keys are controlled by the malware syndicates. Therefore data recovery following a ransomware infection is a huge challenge. It is imperative to arrest the ransomware as early as possible before encryption takes place.

Complex obfuscation and anti-emulation techniques used on the ransomware droppers ensure that static blocking in real time is difficult. However, low-level system-wide interception of designated events by security software allows close monitoring of the behaviour of untrusted executable code – which currently includes ransomware components – thus making contextual dynamic blocking a high-percentage option.

Based on the runtime behaviour of several pieces of modern ransomware, this paper describes in detail the various stages at which ransomware processes can reliably be terminated, mitigating against false positives and performance degradation. We explore in depth the blocking of suspicious events such as data-overwrite attempts at both file system and disk levels, behavioural anomalies of OS processes, incongruous calls to cryptographic functions whether OS crypto APIs or statically linked OpenSSL library code (de-obfuscated in memory), etc. It may even be possible to adopt and adapt certain strategies to arrest ransomware for mobile platforms. We will delve into a novel anti-ransomware solution for *Windows*, optimally combining various strategies to generically detect and prevent attempts to encrypt target file types on disk.

METAMORPHOSES OF A BEAST

The purpose of this paper is to explore strategies for terminating ransomware – the ubiquitous file-encrypting kind – before too much damage has been done. In order to stop ransomware in its tracks it is imperative that we understand the stages encountered in a typical attack scenario, bearing in mind that a firm strike at a particular juncture is likely to stop the ransomware dead.

The *modus operandi* of ransomware involves several stages and binary components. The denouement for our intents and purposes is the point at which all the target files have been encrypted using large keys and industry-standard algorithms, and a demand for ransom has been splashed on the screen to torment the hapless victim. Ransom payment mechanisms are of no relevance to this paper.

The delivery mechanism for primary-stage ransomware components via socially engineered mass-mailed spam, drive-by downloads, botnets, etc., is identical to that of most other malware. It would suffice to mention that standard blocking techniques and good security hygiene at the first line of defence would prevent many such attacks. However, ransomware attacks are generally very successful, implying that the complete delivery of the payload does take place on a

Hash	Wrapper type	Comments
SHA256: f9889210ed894d5da3930689339cc617fb73555d0668542665fd3b0a3a83f319, MD5: d23c1057bfe4f1aaaf5a5a5bc37bd061	MSIL .NET	
SHA256: 114fd64e54c0a3a63327e443bb61e7f8ef3096de681177c834e38125092f5b6b, MD5: 06ea9899946dd36a8a7d71aacd22c19b	Visual Basic	
SHA256: ae4e6153f82c891ce8af249c009dc87ad57bf06cdbdba13fefed45589a245a72, MD5: de25f04dedaffde1be47ef26dc9a8176	NSIS	Contains '06 - Clark Gable.mp3' (649KB) decrypted and loaded by handover.dll (56KB)
SHA256: 55c8378c218443dc23a8b2a1ae30546f2bd65456b67397afaf5dc61ba10fe6e8, MD5: ac174e6f5f0a55f7d281730af01e0316	MS Visual C++ 8	
SHA256: eec15c8c9722feee291df4685dbaa145423b3727b3ebe355c6e39d7944de4bc6, MD5: b5b6aa8ae13ee6a7f0094bd75a25780e	Custom unknown	Linker version suggests underlying VC8 but unknown EP
SHA256: 7b685212ffaa7c11538ac8907f62198e95787e2e570d47cbc7fd3692d638f044, MD5: 1b8011e409cda6f173abe3517557921d	Custom unknown	Downloader component. Linker version suggests underlying VC6 but unknown EP

Table 1: A variety of wrapped CTB Locker components.

regular basis, notwithstanding current static pattern and dynamic behaviour checking, and in-built OS privilege-limitation features such as UAC.

This paper will focus on strategies to contextually block ransomware at runtime, i.e. assuming other security layers have been bypassed. Therefore the ransomware lifecycle described herein will only encompass a microcosm of the ransomware runtime behaviour deemed to represent its core encryption functionality. Note that it is assumed throughout the paper that ransomware components launch solely in user mode, i.e. kernel-mode 'rootkit' drivers are not used to hide ransomware functionality from security software. This assumption is deemed reasonable due to the fact that none of the ransomware families investigated thus far deploy kernel components. Admittedly, any future kernel rootkit functionality in ransomware would jeopardise the efficacy of our generic contextual blocking strategies.

Beauty but a beast

It is worth digressing briefly to investigate the obfuscation techniques used by ransomware which aid the bypass of static pattern-matching, and process injection, which could complicate dynamic behaviour blocking.

The authors of ransomware invest a lot of resources in obfuscating their wares with changing wrappers such that they appear harmless with their nefarious functionality well-hidden, thus rendering blocking based on static methods difficult.

CTB Locker exhibits versatility in its disguises and is therefore considered a good exponent of static-detection-bypass via obfuscation techniques. Table 1 exemplifies CTB Locker's obfuscation prowess.

Given their current delivery mechanisms, ransomware components would be considered untrusted, low-privilege processes by the OS and security software. Even UAC alone would be sufficient to thwart some ransomware functionality, such as encryption of files in certain areas (e.g. Program Files), deletion of system restore points, etc., assuming that an EOP vulnerability has not been exploited in the meantime.

Ransomware families tend to use process injection to unpack their obfuscated code into spawned processes, including OS processes such as explorer and svchost (in the cases of CTB Locker and Cryptowall [1]), in order to complicate the tracking of process activity. Spawning non-OS child processes could lead to a loss of context, and activities which appear to originate from OS processes are less likely to be monitored and tagged as suspicious by security software. On the other hand the process-injection-into-OS-process functionality does provide opportunities for security software, for example:

1. An untrusted process injecting into an OS process is suspicious. Ransomware tends to use WriteProcessMemory to inject its binary content into a running process. A hook on WriteProcessMemory, allowing the context of the process injected into (i.e. OS process) to reliably be determined, enables security software to arrest the injector ransomware process immediately.
2. The incongruity of contextual writes to target file types emanating from a standard OS process, or the presence of artefacts related to cryptographic and hash functions within their process space would be deemed sufficiently suspicious to warrant stern action.

A more elaborate description of the aforementioned context-based writes and presence of certain magic numbers, etc., vis-à-vis ransomware-controlled processes will be given later in the paper.

Casting the pupation spell

Modern ransomware generally deploys advanced encryption algorithms on target files without error. One notable exception is an early version of TorrentLocker using AES in CTR mode with fixed IV and reused key, which allowed extraction of the keystream via simple XOR [2, 3] given an instance of known plaintext, and then application of this key on other encrypted files. Another exception is the inept use of RSA by Bitcrypt, which generated woefully short keys [4]. It is henceforth assumed that decryption of targeted files requires possession of the requisite key.

Figure 1 is an abstract depiction of the stages typically employed by ransomware to encrypt target files.

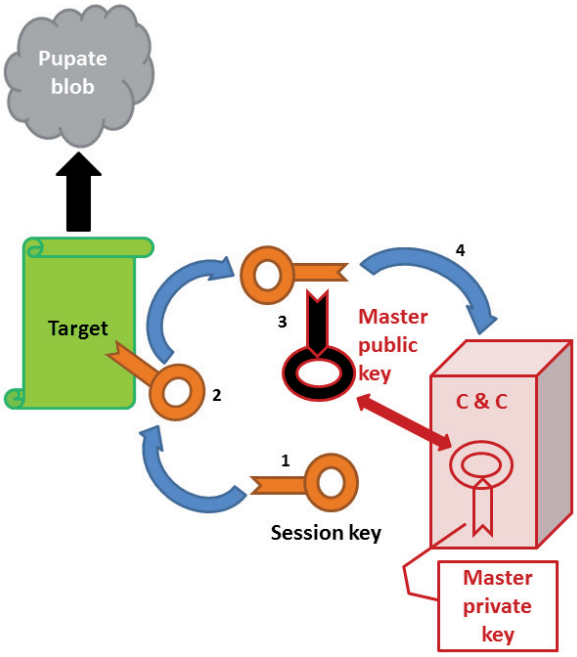

Figure 1: Typical stages involved in ransomware encryption of target files.

1. **Create local session key:** A symmetric cipher is generated using an initialization vector based on random events and identifiers on the local machine. The typical key generated is 256-bit AES, but other symmetric encryption algorithms such as RC4 may be used instead of AES.
2. **Encrypt target using session key:** Files bearing the targeted extensions are encrypted using the session key to form amorphous 'pupate blobs'. Encryption usually covers the entire target file (CTB Locker), but sometimes simply covers a sufficiently large area (TorrentLocker). The ransomware may also append [2, 3] or prepend [5] the encrypted content with metadata required to decrypt the files, e.g. an

encrypted version of the session key along with the public key used to encrypt it. The pupate blobs tend to be saved to filenames with the following format:

```
<exact original filename>.<ransomware extension>
```

The crucial observation is that post-encryption, targeted files are fundamentally altered such that they are no longer of recognized file types, i.e. magic markers such as 'JFIF' (JPG images) are overwritten, and TXT files suddenly contain random binary content. These changes are key (no pun intended) to our anti-ransomware strategy.

3. **Encrypt session key with Master public key:** In order to ensure that encrypted target files can be decrypted only by obtaining the Master private key from the C&C server, the session key is encrypted using the paired Master public key. The asymmetric encryption algorithms used thus far are RSA (e.g. Cryptolocker) and ECDH (e.g. CTB Locker). As mentioned above, the encrypted session key may be embedded into the pupate blobs. It is noteworthy that the actual public key used to encrypt the session key may not be the Master public key *per se* but another key generated locally and linked back deterministically to the Master private key [5]. In addition, in certain cases the Master public key is used to encrypt the target files either as a whole, e. g. CryptoWall (see [1]) or in part, e.g. Dirty Decrypt (see [1]), instead of encrypting using a symmetric session key. Nevertheless, the pupate blobs created would continue to exhibit the fundamental structural changes described in point 2 above.
4. **Despatch encrypted session key to C&C**: Once the session key is encrypted with the Master public key it is sent back to the C&C. Thereafter, the unencrypted session key would be expected to be destroyed so that it cannot be dumped from memory and thus used to recover the targeted file content from the pupate blobs.

Appear Master public key

The Master key pair is generated either on the C&C or another machine controlled by the malware syndicates. Figure 1 suggests that the malware retrieves the Master public key from the C&C, as was the case with Cryptolocker [6] and Cryptowall [7].

If the Master key has to be downloaded before any encryption takes place, an excellent opportunity is provided to security software to neuter the ransomware simply by blocking the network traffic. For example, in the case of Cryptolocker the DGA-based domain names could be calculated offline and blocked *en masse*, and in the case of Cryptowall, a policy block on TOR traffic would do the trick.

However, the Master public key could be pre-embedded in the malware binary, as in the case of CTB Locker. CTB Locker and TeslaCrypt (NB: TeslaCrypt does not currently appear to make use of any asymmetric keys [8]) do not require an Internet connection to cause their damage. The network communication is required simply to deliver the encrypted session key.

SWORDS DRAWN AND READY FOR BATTLE

Security software can and should fight back against ransomware. Let us explore in detail our generic anti-ransomware strategy.

Every family of ransomware adopts the following high-level workflow:

1. **Locate target files**: Ransomware, much like its equivalent legitimate counterparts, performs target selection by enumerating certain drives and directories for files with specific extensions such as TXT, JPG, DOCX, ZIP, etc.
2. **Take data hostage**: The selected target files are encrypted as described earlier.
3. **Demand ransom**: After all target files have been encrypted, a ransom message is displayed.

We can exploit the above workflow to our advantage in detecting ransomware functionality. In order to establish core intent with precision during various file operations, it is important to keep track of their context.

We decided to implement a framework with the help of a kernel-mode driver for this purpose. This hawk-like component helps in tracking the activities of multiple processes/threads in a centralized area and correlating them.

The interception points and technologies used in the kernel driver are listed below.

Remote code injection and execution

As mentioned earlier, many families of ransomware inject code into running processes, including OS processes, to execute their workflow. However, code-injection is a common technique used by many different types of malware. A typical HIPS component could effectively block code-injection attempts made by a potential piece of ransomware without necessarily knowing it was one.

Intercepting a combination of ZwAllocVirtualMemory and ZwWriteVirtualMemory APIs in kernel, and blocking any cross-process attempts at manipulating memory where the target is an OS process (e.g. svchost.exe, explorer.exe, etc.) seals the deal. Placing hooks on ZwCreateThread, or registering for ObjectManager thread callbacks, allows blocking attempts at the point of starting a new control flow in an OS process, such as to enumerate directories.

However, perhaps due to a lack of sophistication, not all ransomware performs suspicious actions such as process injection. TeslaCrypt, which executes the required workflow as part of a freshly spawned copy of itself, could go under the HIPS radar.

Generic ransomware tracking

The kernel driver we developed as the PoC is a file system minifilter, which has interception points on the following IRP requests:

- **IRP_MJ_CREATE**: to track file/directory opening/creation
- **IRP_MJ_DIRECTORY_CONTROL**: to track directory enumeration
- **IRP_MJ_WRITE**: to track file writes
- **IRP_MJ_CLOSE/IRP_MJ_CLEANUP**: to track file/directory close.

Given that directory enumeration is the first critical step performed by the ransomware, and it is not a very common, high-frequency operation across all processes, the driver starts by associating a context when it detects a directory enumeration.

During our study, we have observed ransomware optionally split the workflow over multiple threads. Our TeslaCrypt sample (SHA256: a7ad727abfec279fe5ef781a0714ef781dacd18f6253f443a4677c5a2cf74083, MD5: 41d0cf71eb2c996924feb0a446d943b2) uses the same thread for enumeration and data encryption within its own process space. On the other hand, the CTB Locker sample we studied (SHA256: fa876bc9cd4ff4574f5bfb52e8959764a4703d25accccf4d68723fad05e8c8b, MD5: 8a5fb8b49ed88d093177786c8111736a) uses separate threads spawned from injected code in svchost-process-space for enumeration and encryption. The CTB Locker case implies that tracking context merely at thread level, or at process level would not be sufficient to accurately establish intent. Therefore the kernel driver tracks the context at allocated code-block level within a process.

Once a code-block is flagged for monitoring, all file I/Os emanating from that block that can possibly change a file are tracked, e.g. renames/moves, writes, etc. A small journal of change-activities is stored for the purpose of reverting any operation once a ransomware-like operation is detected. Note that the journal is expected to remain small since the ransomware is expected to be terminated very early in its encryption loop.

At the file-data-write interception point, i.e. before any data is actually overwritten, the existing file type is detected based on its content. If the file types are recognized via magic values at fixed offsets in the headers (e.g. big-endian magic 'JFIF'@0x6 for JPG; 'PK'@0x0 for ZIP/DOCX/XLSX; 0xD0CF@0x0 for DOC, etc.), then an attempt to overwrite the headers with unrecognized content is an instant indicator of a potential encryption attempt. If the target file type is unrecognized, or if it has text-like content, then a significant rise in the file data entropy is enough to flag the encryption operation. It is interesting to note at this stage that the ransomware itself makes no attempt whatsoever to establish target file type but is content to trust the file extension. This fact provides an opportunity to determine the target file type merely by extension, and to present shadow targets to the ransomware, if so required.

After detecting the attempted encryption operation, we are presented with the best opportunity to block ransomware without allowing the malware to do any real damage. However, the driver is required to distinguish between legitimate encryption programs and ransomware. It currently uses various heuristics to distinguish between them:

- The source of the encryption process, if from an OS process, is a dead giveaway of its malicious nature.
- The aggressiveness with which the enumeration and encryption occur is also a strong indicator. A slow encryptor (similar to a slow-infecting virus) is yet to appear and we may not see one due to the short life expectancy of any particular C&C server involved.

In order to reduce the amount of file-data-write tracking, which could have an impact on system performance, we considered limiting monitoring via the detection of magic values used in encryption/hash algorithms such as AES, SHA256, etc. in code-blocks where directory enumeration has been initiated. This is effective when the malware is linked statically to an encryption library such as OpenSSL.

Use of MS Crypto APIs can be detected via hooks on Crypto APIs, or by scanning for Crypto API names within the code-block. However, we resorted to the use of entropy detection and known-data-overwrite to detect an encryption attempt. Of course, entropy is also raised when file content is compressed; however, most compression programs work on newly created files rather than existing files, thus escaping false detections.

Mitigating the risk

The detection framework described above entails the usual risks associated with detecting malware, i.e. performance degradation and false positives, e.g. if a user genuinely wishes to encrypt his/her files. Fortunately, both of these risks can be mitigated against by tightening the process context for interception, monitoring and detection.

Within an end-point environment, the sources of infection can be classified by their probability, allowing us to narrow down the processes that require contextual tracking, even merely for directory enumeration.

Files from external sources such as the Internet, optical media, and removable devices are primary candidates for closer observation. Specific processes that act as network services that may be remotely exploitable are also candidates for closer observation. On the other hand, legitimate encryption software is likely to be installed in the Program Files area, and may well be signed or excluded in some way, thus engendering a trust level which obviates the need for prying eyes.

Sector-level monitoring

If future ransomware were to adopt kernel rootkit-like components it may be possible for them to bypass the interception points discussed above. In order to complicate this type of bypass strategy we considered the use of disk-level interception for monitoring ransomware activities.

However, unlike a file system minifilter driver, a disk filter driver that can intercept disk sector reads and writes lacks contextual information about the caller file or process from which the I/O is initiated. This makes contextual optimal interception at disk level infeasible, leading to potential system performance degradation and increasing the susceptibility to false positives, and therefore this approach was promptly shelved.

IN A FAR AWAY LAND…

Ransomware also infects mobile devices, the most well-known family being Simplocker [9] for *Android*. Although considerably less sophisticated than its *Windows* counterparts, Simplocker does exhibit some advanced functionality such as using AES encryption with changing keys [10], and communicating with a C&C via .onion domains.

The anti-ransomware strategies described above for the *Windows* platform could also work for *Android* and other mobile platforms. However, there is one major stumbling block. Security software on mobile operating systems such as *Android* are accorded the same privilege as any other app

[11], including the ransomware itself. This means that the extent of monitoring and control from a security software perspective is limited, and the ransomware could even execute first, as in the case of the Koler malware family [11]. Security software is unable to intercept API calls, but may only subscribe to broadcast notifications which may well be far too late to minimize the damage.

DEAD AND BURIED IN THEIR CRYPTS

Ignoring the screen-locking species for the present, modern ransomware is indeed complex, utilizing sophisticated techniques to bypass current security measures with ease, and holding data hostage with a vice-like grip.

However, it is possible to fight back very effectively against ransomware, as has been described in this paper. Contextual interception, on *Windows* operating systems, of attempts to encrypt target files can be used to generically detect and terminate ransomware before any major damage is done.

This anti-ransomware strategy, if deployed widely across security software, should sound the death knell for modern ransomware. R.I.P.

REFERENCES

[1] Kotov, V.; Singh Rajpal, M. Understanding Crypto-Ransomware. Bromium, Winter 2014. http://www.bromium.com/sites/default/files/bromium-report-ransomware.pdf.

[2] Crypto blunder makes TorrentLocker easy to crack. Virus Bulletin. http://www.virusbtn.com/blog/2014/09_10.xml.

[3] Léveillé, M.-E. M. TorrentLocker. ESET, December 2014. http://www.welivesecurity.com/wp-content/uploads/2014/12/torrent_locker.pdf.

[4] Researchers crack ransomware encryption. http://www.virusbtn.com/blog/2014/02_21.xml.

[5] CTB-Locker encryption/decryption scheme in details. Zairon. February 2015. http://zairon.wordpress.com/2015/02/17/ctb-locker-encryptiondecryption-scheme-in-details.

[6] Jarvis, K. CryptoLocker Ransomware. Dell SecureWorks CTU(TM), December 2013. http://www.secureworks.com/cyber-threat-intelligence/threats/cryptolocker-ransomware/.

[7] Allievi, A.; Carter, E. Ransomware on Steroids: Cryptowall 2.0. Cisco, January 2015. http://blogs.cisco.com/security/talos/cryptowall-2.

[8] Allievi, A.; Carter, E.; Tacheau, E. Threat Spotlight: TeslaCrypt – Decrypt It Yourself. Cisco, April 2015. http://blogs.cisco.com/security/talos/teslacrypt.

[9] Lipovsky, R. ESET Analyzes First Android File-Encrypting, TOR-enabled Ransomware. ESET, June 2014. http://www.welivesecurity.com/2014/06/04/simplocker.

[10] Chrysaidos, N. Mobile Crypto-Ransomware Simplocker now on Steroids. AVAST, February 2015. http://blog.avast.com/2015/02/10/mobile-crypto-ransomware-simplocker-now-on-steroids.

[11] Mody, S.; Dhanalakshmi, V. Early launch Android malware: your phone is 0wned. Virus Bulletin. October 2014. http://www.virusbtn.com/blog/2014/10_31.xml.