Forensic Analysis and Remote Evidence Recovery from Syncthing: An Open Source Decentralised File Synchronisation Utility

Conor Quinn, Mark Scanlon(✉), Jason Farina, and M.-Tahar Kechadi

School of Computer Science, University College Dublin, Dublin 4, Ireland
{conor.quinn,jason.farina}@ucdconnect.ie,
{mark.scanlon,tahar.kechadi}@ucd.ie

Abstract. Commercial and home Internet users are becoming increasingly concerned with data protection and privacy. Questions have been raised regarding the privacy afforded by popular cloud-based file synchronisation services such as Dropbox, OneDrive and Google Drive. A number of these services have recently been reported as sharing information with governmental security agencies without the need for warrants to be granted. As a result, many users are opting for decentralised (cloudless) file synchronisation alternatives to the aforementioned cloud solutions. This paper outlines the forensic analysis and applies remote evidence recovery techniques for one such decentralised service, Syncthing.

Keywords: Syncthing · Digital forensics · Remote forensics · Network analysis · Evidence recovery

1 Introduction

In an ever increasing mobile and connected world, the demand for end users to access their data on the go using multiple platforms and devices is higher than ever. While numerous platforms have been developed to respond to this constant information need, these platforms can give rise to data protection and privacy concerns. These concerns primarily lie with cloud-based file synchronisation services such as Dropbox, OneDrive and Google Drive. A number of these services have been leaked as sharing replicated information with government security and spying agencies without first requiring the issue of a warrant [1]. The desire for privacy has led to a rise in cloudless file synchronisation services such as BitTorrent Sync (BTSync), Syncthing and OnionShare.

One of the most popular decentralised file synchronisation services is currently BTSync, which as of August 2014 had over 10 million user installs [2]. However a significant number of these users are not comfortable with the proprietary nature of the application and its handling of their data. This has motivated a transparent alternative being developed, called Syncthing. Syncthing is an open source, cloudless file synchronisation service. Users have the ability to identify how the software finds other active nodes to sync with, transfers data

J.I. James and F. Breitinger (Eds.): ICDF2C 2015, LNICST 157, pp. 85–99, 2015.
DOI: 10.1007/978-3-319-25512-5_7

from node to node, and synchronises information between different devices. With BTSync emerging from beta in March 2015, limitations on how many folders can be synchronised for free have been imposed – with the free tier being limited to syncing ten folders. It is likely that the lack of transparency regarding security and privacy and these new limitations imposed on the free BTSync tier users will push many towards deploying Syncthing for their file replication needs.

Syncthing is a decentralised tool created for the purposes of data backup and synchronisation, teamwork/collaboration, data transfer between systems, etc. From a law enforcement and digital forensic perspective, an area of concern with decentralised services is the possible exploitation of the service to distribute unauthorised/illegal data: industrial espionage, copyright infringement, sharing of child exploitation material, malicious software distribution, etc. [3]. These cloudless services have no regulation by their developers and as a result are at high risk of being used for criminal activity. Syncthing has many desirable features for privacy-concerned users who wish to use file synchronisation but conscious of their data's security. Such features include [4]:

- Private – The synchronised data is never replicated anywhere else other than on devices configured.
- Encrypted Traffic – All communication between devices is secured using TLS.
- Authenticated – Every node is identified by a strong cryptographic certificate; only nodes you have explicitly allowed can connect to your cluster.
- Cost and Limitations – Most main stream cloud-based file synchronisation software give you a small storage allowance at the free tier. Syncthing is limited only by the storage available across your devices.
- Transparency – The software is open source which facilitates analysis to prove that the software is secure.

With increased privacy and security of any tool or service, there is always the contraposition of law enforcement regarding the difficulty (or possibility) of capturing evidence from these systems. At the time of writing, there are no tools available for the recovery of evidence from Syncthing.

1.1 Contribution of This Work

This paper outlines a forensic analysis of the Syncthing client, its communication protocols, its peer discovery methods, its behaviours and its data remnants of synchronised deleted files. The contribution of this work can be summarised as follows:

- An outline of the entry points to a Syncthing investigation, i.e., how to detect whether Syncthing is pertinent to an investigation.
- A description of the services network communication protocol for the purposes of building a remote evidence recovery tool.
- A proof-of-concept tool, Synchronisation Service Evidence Retrieval Tool (SSERT), has been developed for an investigation scenario outlined. The investigation is documented showing how the remote recovery of digital forensic evidence from folders shared using Syncthing might be valuable to forensic investigators.

2 Related Work

The popularity of cloudless file synchronisation services as a viable "install-and-forget" alternative to the more commonplace cloud-based solutions is a recent development. Given the relatively new provisioning of these services, there has been little time for forensic procedures and best practises to catch up. However, there has been some research conducted on the remote recovery of evidence from Syncthing's primary competitor, BTSync, as well as the cloud-based solutions. The below section outlines some of this related work.

2.1 Forensic Analysis of BitTorrent Sync

In a similar vein to the focus of this paper, there has been an investigation methodology developed for BTSync, a cloudless file synchronisation service developed by BitTorrent Inc. [5]. BTSync is a cloudless file sharing tool with the intention of providing one-to-many and many-to-many file transfers as efficiently as possible. The protocol segments a file, which enables each chunk to be managed separately. Once a part of a file is downloaded it can immediately be uploaded to a different peer who has requested that file [5]. In this fashion a file can be shared before the whole file has been downloaded. BitTorrent Sync uses the BitTorrent protocol for data transport, which is analysed in detail in [3]. One of the interesting things about BTSync is the use of keys for managing permissions between peers. Once the creation of a share a master key is constructed, this master key has read/write capabilities which allows the person who has that key to add, modify or remove contents of that share. Other, more restricted keys exist allowing a share participant to give Read Only access or enforce a window before an invitation expires [5].

2.2 Forensic Analysis of Cloud-Based File Synchronisation Services

Towards the remote recovery of evidence from cloud-based sources, a volume of work has been conducted on the recovery of evidence from file synchronisation services. Quick and Choo have analysed the data remnants of deleted files in Dropbox [6], Microsoft SkyDrive (now rebranded as OneDrive) [7], and Google Drive [8]. This volume of work outlines the processes required for the remote recovery of deleted digital evidence from a local machine. The recovery of the data from the cloud-based storage combined with data remnants discovered on the local machine can verify the recovered copy as being a true copy of the original data. The authors also proved that downloading the remote data using a browser or performing a client sync does not interfere with the hash of the recovered evidence or any associated cloud-stored metadata [9]. The work conducted by Quick and Choo on Dropbox forensics has also had similar results confirmed by Federici [10]. In this work, a Cloud Data Imager is outlined. This is a tool developed by law enforcement for the forensically sound remote recovery of evidence from Dropbox.

3 Syncthing Analysis

While Syncthing may have been inspired by BTSync, its purpose is to transparently address features that some users identified as security and privacy issues. The first of these is the fact that BTSync attempts to improve security by keeping its source code secret. This is a common tactic and is known as "security through obscurity", the effectiveness of which is questionable. BTSync also collects usage statistics on its users' activities, which the developers claim only records anonymous bandwidth and usage metrics. Some users raised concerns that were left unanswered by developers and this silence gave rise to fear that there was the possibility of more than just the metrics being stored as was first stated. That fear led to Syncthing, which attempts to assuage user security concerns through transparency in its design and protocol. Users can easily see what the application is doing and how it is doing it. This open-view approach to security risks attackers finding a vulnerability but also allows the multiple interested parties to find and fix any flaw themselves before it is exploited [11].

Syncthing makes use of Block Exchange Protocol (BEP) [12] to minimise the traffic generated by partial file updates. BEP is used between two or more devices to form a cluster. Each device has one or more folders of files described by the `local model`, containing metadata and block hashes. The `local model` containing this data is then sent to the other devices that this device has in its cluster. The combination of all files in the local models and the files selected for highest change version from the `global model`. Each device in the cluster then attempts to align all of its local folders with the global model. The device then requests missing, outdated or corrupted blocks from the other devices it has in its cluster [12]. When file data is described or transferred it is segmented as a series of blocks with each block measuring 128 kB (131072 bytes). The BEP protocol is implemented at the highest level of the stack with the lower levels providing encryption and authentication. The underlying transport protocol must be TCP using this technique [12].

3.1 Data Remnants

To detect if Syncthing is installed on a suspect's machine and to retrieve the required information for identification of remote users sharing the same content, the data remnants left on the hard drive of the machine must be discovered. Syncthing uses a single folder to store all of its configuration files, cryptography certificates and keys [13]. The default install location for this folder on Windows 7 and 8 based systems is located in `%localappdata%\Syncthing`; on Windows XP, it is located in `%AppData%\Syncthing`; on Mac OS X systems, it is located in `~/Library/Application Support/Syncthing` and on *nix systems, it is located in `~/.config/syncthing`. Alternatively, the end user can specify a different "home" directory when launching the application which facilitates a non-default location for these files. The folder contains the following files [13]:

- `cert.pem` – The device's RSA public key.
- `key.pem` – The device's RSA private key.
- `config.xml` – The application's configuration file.
- `https-cert.pem` and `https-key.pem` – The certificate and key for HTTPS GUI connections.
- `index/` – A directory containing the metadata and hashes of the files currently on the disk and available from remote peers.
- `csrftokens.txt` – A list of recently issued CSRF tokens to protect against browser cross site request forgery.
- `index/[IncrementalNumber].log` – The application's log file for all actions taken locally and outlines folders shared with remote hosts.

3.2 Peer Discovery

Each device on the network is identified by its `DeviceID`. The `DeviceID` is made up of a Base32 SHA-256 encoding of the application's public RSA key, which is created during Syncthing's initial installation [14]. When Syncthing is launched, the settings contain the global discovery server address `announce.syncthing.net` [4]. At the time of writing this resolved to the IP address `194.126.249.5`. Once the announce server address has been resolved the application queries it with a valid `DeviceID` via a `query` packet via UDP [15] and, if known and the target has registered itself as being online, a current IP address and port will be returned in an `announce` packet. The IP and port combination returned are used as the destination for the protocol's secure handshake.

As Syncthing is an open source application, users have the option to set up their own announce server to handle internal peer discovery. In this scenario, all clients would require configuration to use this custom server instead of the default one. Another option is to use the built in local peer discovery. This setting is configurable in the application's settings, but is enabled by default. Local discovery can happen in one of two ways depending on the type of network detected:

- **IPv6 Networks** – If Syncthing discovers an IPv6 network it will use Simple Service Discovery Protocol (SSDP) to send a HTTP notify packet to port 1900. Syncthing utilises the FF02::C link local address to limit notification to a network segment only. In testing Go did not support the `setsocket` operation on windows 7 systems so IPv6 beacon packets were not responded to by the clients. This notify packet will contain the same details as the `announce` packet used for global discovery.
- **IPv4 Networks** – If the application detects that IPv6 is not supported inbound, Syncthing will still announce using IPv6 if it is supported outbound. Over IPv4, the announce packet is broadcast to the network on port 21025 with a 56 byte `announce`. This local announce associates an IP:Port combination with a `DeviceID` that is cached for later use.

3.3 Block Exchange Protocol Messages

With initial discovery complete a standard TLS session is established with both parties providing certificate-based authentication. In the case of our emulated client, we present the imported certificate of the suspect system as our proof of identity. Once the secure connection has been negotiated successfully a series of messages are exchanged before any requests involving the transfer of files can be made.

Header. The messaging used by Syncthing involves specific packet types identified by a message header. This header consists of one 32 bit word indicating the message version, type and ID, followed by the length of the message itself [12]. The principal field in the header is the type field. Each message type is denoted by a different hex number as outlined below:

- (Type 0) – `Cluster Config`
- (Type 1) – `Index`
- (Type 2) – `Request`
- (Type 3) – `Response`
- (Type 4) – `Ping`
- (Type 5) – `Pong`
- (Type 6) – `Index Update`
- (Type 7) – `Close`

Also contained in the header is the version, message ID, overall message length and a flag to indicate if compression is used. Figure 1 (a) is a graphical representation of how a header message is constructed.

Cluster Configuration Message. This is the first protocol specific message Syncthing must send after a successful connection to a peer is an informational message containing details about the share topology. This message establishes the local peer's version and ClientID, the number of folders hosted and the `DeviceIDs` of peers the sender is connected to and actively synchronising with. In addition to the standard fields mentioned earlier there is an `Options` section at the end. Once a secure connection is established, a cluster configuration message must be the first packet sent, otherwise the remote peer will forcibly close the session.

Index Message. The next message that must be sent is an `index` message. There must be one `Index` message for each folder reported in the `Cluster Configuration` message and should this index message be sent in an inappropriate order, the secure connection will be dropped. The purpose of the `Index` message is to enumerate the contents of the peer's folders. An Index message with an internal structure, as shown in Fig. 1 (b), represents the current contents of the folder and supersedes any previous index that may have been sent in an earlier transmission [12].

The `Index` message contains a lot of useful forensic information.This includes the name and relative path of each file in the shared folder, the timestamp of the

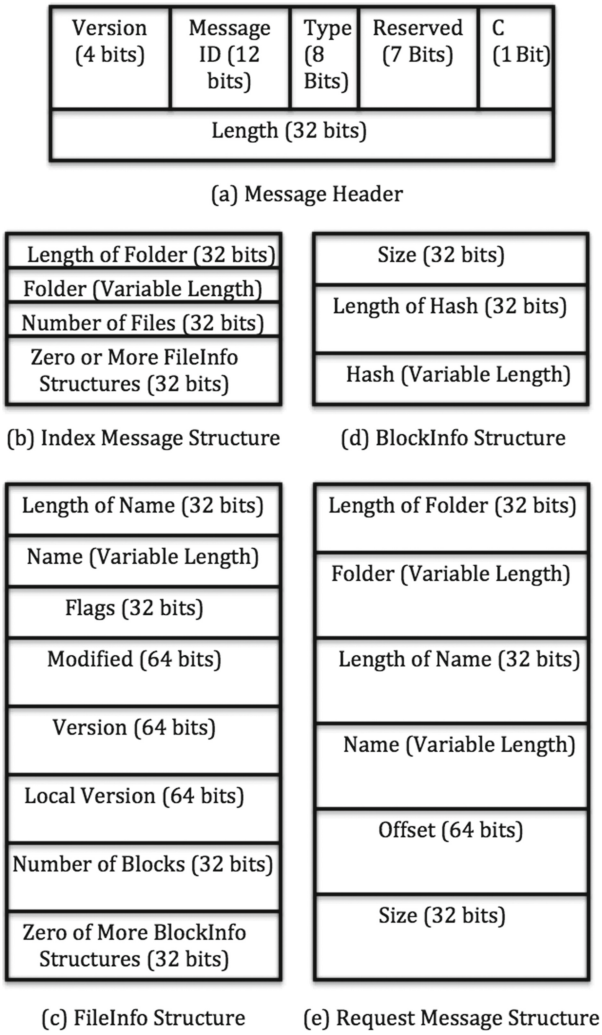

Fig. 1. Network message structures pertinent to evidence recovery

last modification date, and the `BlockInfo`. The `BlockInfo` contains the hash of each 128 kB block that constitutes each file. Syncthing uses these hashes and the modification date to minimise the number of blocks that have to be replicated to an updating peer if it already has an older version of the file. Only modified 128 kB blocks will have to be transferred. This feature also allows investigators to compare the block hash of a file recovered from a remote peer to that recorded in the suspect system's configuration files to determine if the recovered data is a forensic match of the original file [16].

Request Messages. Once the peers have determined which peer holds the most up to date version of a file, the lagging peer needs to update its version. By comparing the file `BlockInfo` the peer with the older version can determine the exact blocks it needs to make the local version of the file equal to that on the remote peer (assuming that it is the local peer that is behind). For each block identified in this manner, a `Request` message is sent to the remote peer containing the length of the folder, the name of the file containing the block being requested and the offset to the start of the block, as depicted in Fig. 1 (e). One `Request` is sent for each block required. Once a valid `Request` has been received, the remote peer responds with a `Response` message containing the requested block.

4 Investigation Methodology

Figure 2 outlines the process developed for the identification of other active nodes involved in sharing the contraband content and for the remote recovery and verification of the gathered evidence.

4.1 Security and Authentication

The entry point to a Syncthing investigation requires the recovery of the public/private RSA key pair from the suspect device. Upon initial execution, the application creates these keys, which are used to self sign certificates that are used in the TLS (Transport Layer Security) handshake. The certs provide identification to other devices when a share is initially established and are subsequently used for ongoing authentication [17]. These keys will be on the suspect device's storage as outlined above.

4.2 Remote Peer Identification

For the purposes of remote peer identification, i.e., to answer the question "What other devices are synchronising with this suspect?", the suspect machine's `*.log` file is required. This file contains a record of the folders shared with each remote peer. As can be seen in Fig. 3, the remote machine's `DeviceID` is displayed alongside some other metadata, such as the remote client version, the remote machine's hostname, and the IP address and port number of that machine at the time. The regular time-stamping used in the log file can be used to identify when that machine was online and what data has been synchronised.

Fig. 2. Process of evidence recovery for syncthing

```
[UYT2U] 17:26:31 INFO: Established secure connection to IX2735N-PPQMNFU-NQ5KLBB-
WG7WT7F-GALJHMH-EP7G2M2-TGD3UNA-F4FZTAJ at 192.168.1.3:1234-192.168.1.10:64715
[UYT2U] 17:26:31 INFO: Device IX2735N-PPQMNFU-NQ5KLBB-WG7WT7F-GALJHMH-EP7G2M2-TG
D3UNA-F4FZTAJ client is "syncthing v0.10.24"
[UYT2U] 17:26:31 INFO: Device IX2735N-PPQMNFU-NQ5KLBB-WG7WT7F-GALJHMH-EP7G2M2-TG
D3UNA-F4FZTAJ name is "Conor"
```

Fig. 3. Sample syncthing application terminal output

While this information alone might be sufficient to focus the investigation on additional devices, the logged information merely records the IP address and port of the remote device at the time. In order to check if the remote device is currently active, a request can be sent to the `announce` server with the persistent `DeviceID`. This should provide an updated IP address and port if the device has been active in the previous 30 min window.

Fig. 4. Sample records included in each `*.log` file

A file of significant interest to a digital investigator are the `*.log` files. The log file contains lists of devices the suspect system has connected to. It also contains a list of folders and corresponding files. A snapshot of a recovered log file from Syncthing can be seen in Fig. 4. In the bottom highlighted area in the figure, the `DeviceIDs` of remote machines can be identified. When a `DeviceID` is recovered, it can be used to query the global discovery server, which return the active IP address and port pair. Once the IP and port number have been identified, a secure connection can be attempted, using the already trusted device it has already been connected to. The topmost highlighted area Fig. 4 shows folder names and file names. The example log file shows the entries for the files `RV.jpg`, `Draft.txt`, `ThesisReport copy.pdf`, which are each contained in a folder called `Test`.

4.3 Remote Evidence Recovery

As part of the investigation it may become necessary to verify that the remotely device has stored a copy of the contraband data and that it is a forensically

sound copy of the original on the local suspect machine. In order to perform such a task, the suspect machine's `cert.pem` and `key.pem` files are also required in order to emulate the suspect's device. As in the previous section, the current IP address and port number of the remote device can be discovered using the `announce` server. To start the remote recovery process the investigator initially requires `DeviceID` of the remote machine. The `announce` server can be queried and if has been active in the last 30 min, its IP and port will be retrieved. Subsequently, a TLS handshake is required to authenticate with the remote device.

4.4 Proof-of-Concept Tool

In order to prove the methodology outlined above, a proof-of-concept tool was created (SSERT). This tool emulates regular Syncthing client communication in order to recover evidence from one or more remote devices. The investigative process using this tool involves:

1. The investigator retrieves the pertinent public/private keypairs and application log files from a suspect device. A list of `DeviceIDs` (which the suspect device has been in communication with) can be retrieved from the application's log files.
2. The investigator provides the retrieved public/private keys and the `DeviceIDs` to the SSERT application. SSERT resolves these `DeviceIDs` to their corresponding IP address and port pairs by querying the `announce` server.
3. Using the suspect machine's credentials, a connection is made to a remote device and the TLS handshake process is completed.
4. Once a connection is established, SSERT requests a list of files available on the remote device and processes the returned `FileInfo` messages, as displayed in Fig. 1 (c).
5. The investigator then selects the file(s) of interest and the emulated synchronisation process begins. After the file is requested, the remote machine responds with N `BlockInfo` messages, as can be seen in Fig. 1 (d). N is the number of 128 KB blocks the requested file is split into for synchronisation. Each block is requested individually, downloaded and verified as a true copy from the remote machine using the supplied SHA256 hash value from the corresponding `BlockInfo` message.
6. Once the synchronisation process completes, the downloaded blocks are recompiled into the complete file. These downloaded files are verifiable as true copies against the suspect machine's local file metadata.
7. The output of SSERT includes the downloaded file(s), an audit log of the actions performed and a record of the network communication back and forth to the remote device.

5 Evaluation and Testing

5.1 Usage Scenarios

The intended usage of SSERT is in the forensic recovery of data from remote peers when the data cannot be recovered from the suspect systems due to encountering a less than ideal forensic environment. The scenarios a digital investigator might encounter whereby the methodology outlined above may prove useful are:

- **Inaccessible files** – This be either be intentional, such as deliberate secure deletion of incriminating evidence, or unintentional, such as data store volume corruption or failure. If the investigator suspects that this inaccessible shared data is pertinent to the investigation, the remote recovery of this data from another device will provide a forensically sound alternative source of evidence to the investigation.
- **Unrecoverable or destroyed external storage** – If the suspect was using a storage medium for sharing data that is not recoverable during the investigation, such as a USB flash drive, external hard drive, network attached storage (NAS), etc., the recovery of this data from a remote storage location may be the only option available to the investigator.
- **Encrypted storage containers** – If the suspect was sharing data from an encrypted container, e.g., using TrueCrypt, BitLocker or FileVault, the local recovery of this data may prove impossible without the decryption keys. In this scenario, the suspect would mount this encrypted container to facilitate synchronisation with a remote device and otherwise leave it encrypted. The forensically significant files outlined above are sufficient to prove the synchronisation of data to the local machine. The remote recovery of these files would be verifiable as true copies of those stored locally through the comparison of the file metadata contained in application log files.
- **Volatility of mobile device storage** – As with most synchronisation tools available on mobile devices, the files accessed through the mobile application are usually not stored permanently. This is typically due to local storage restrictions. Any evidence of a file's existence, e.g., artefacts left behind in slack space, are rare to find as this space is typically quickly re-allocated and re-used by another process. Syncthing's mobile application requires a user to explicitly select the file they want to synchronise to the mobile device in order to avoid accidentally filling all storage capacity by connecting to a larger remote data store. The nature of Syncthing's logging and Block Exchange Protocol means that in order for it to provide the service it offers, it must maintain extensive logs and configuration files. Recovery of these log files from any of the peers may provide enough evidence to show that a copy of the file was synchronised to a mobile device which in turn means that the user of the device had to explicitly select that file. This in turn implies knowledge and intent on the part of the device owner and may also provide evidence of usage of the mobile device based on the timestamp of the Syncthing `Index` and `Cluster Config` messaging.

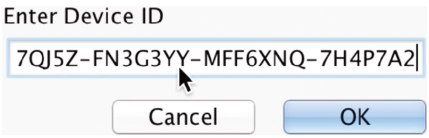

Fig. 5. Starting point: `DeviceID`

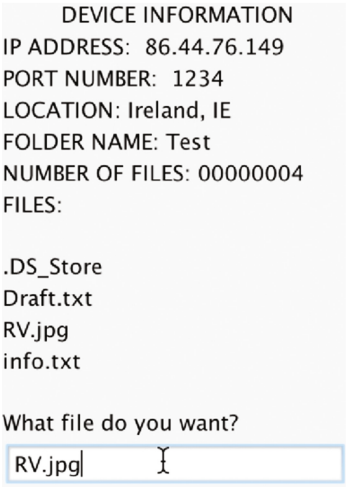

Fig. 6. Retrieved peer information

– **Accomplice identification** – The tracking down of associates of the suspect may be a focus of the investigation, e.g., the sharing of illegal content or sensitive information with an unknown number of parties. The application's log files would show `DeviceIDs` along with the synchronisation timestamps with that device. This information can be used to determine which device was connected when and also whether or not they successfully received a copy of the incriminating data.

5.2 Testing

In order to test the evidence retrieval methodology and the performance of the application a test scenario was devised involving evidence of Syncthing being discovered on a suspect's system (most likely the Syncthing folder stored in the user's AppData\Local\Syncthing folder on a Windows system). Within this folder the forensic investigator can recover the public/private key-pair certificates as well as the log files. However, the folder indicated by the `Index` and logs is encrypted and there is no evidence of the passphrase or decryption key on the system. There is, however, a second 64 character string in the logs, which does not match the `DeviceID` generated for the local system.

Once the certificates have been imported into SSERT, the retrieved `DeviceID` of the remote machine is entered, as can be seen in Fig. 5. A query packet is sent to the `announce.syncthing.net` server over UDP and the server will respond with the required networking information. If the remote peer has not been online in the last 30 min, no network information will be returned and polling of the `announce` server is required.

Fig. 7. Original image

Fig. 8. Partial retrieval

When a response is received, SSERT will contact the remote peer and establish a secure connection using the imported certificates to convince the remote peer that not only is SSERT a valid Syncthing installation but also that it is the suspect system attempting to perform a routine update check. After `Cluster Configuration` and an `Index` messages are received, SSERT displays the folder and file list of items available from the remote system, as can be seen in Fig. 6.

In this test scenario, the investigator was interested in a file named `RV.jpg`, as can be seen in Fig. 7. The investigator then selects the file to recover and SSERT begins the synchronisation process by sending a series of `Request` packets (one for each 128 kb block or part thereof of the file) and listening for the `Response` messages. Once these have been saved, the `RV.jpg` image is reconstituted locally. To verify the evidence, the local copy is hashed using SHA-256 and the result compared to the hash that can be found in the suspect system's `\Index\MANIFEST` file. In one instance during testing, a disconnection occurred and a partial recovery of the remote image was gathered. Due to the JPEG compression algorithm, this partial recovery is sufficient to identify what the original image contains, as can be seen in Fig. 8, but of course the hashes cannot be verified.

6 Conclusion and Future Work

Given Syncthing's open source nature and reliable performance the protocol is likely to be used in other applications in the future. This may be either as a standalone file synchronisation utility or as the foundation for another solution. As with all hash based synchronisation utilities, it is its own activity logging and willingness to verify and check that can be used as a method of enumeration.

The location of the AppData folder provides a strong entry point to an investigation for Windows based suspect systems with a lot of potentially important information for the investigator to recover. Combined with the artefacts retrievable from this source, a strict adherence to the protocol messaging sequencing allows full client emulation including remote peer enumeration through the

`announce` server and full file manifest discovery through the `Config Index` trade. The open source nature of the protocol allowed the creation of SSERT proof of concept build for Syncthing evidence retrieval and in testing the application has proven to be accurate and efficient in the enumeration, recovery and verification of evidence not recoverable directly from the suspect system.

In the future, SSERT can be expanded to perform similar analysis and evidence recovery from additional synchronisation services, such as OnionShare. Combining polling and the analysis of the information available from the remote host, automated evidence downloading and metadata exporting should enable SSERT to function without manual intervention whenever an unavailable node comes online.

References

1. Greenwald, G., MacAskill, E.: NSA prism program taps in to user data of apple, google and others. Guardian **7**(6), 1–43 (2013)
2. Pounds, E.: Introducing BitTorrent Sync 1.4: An Easier Way to Share Large Files (2014). http://blog.bittorrent.com/2014/08/26/introducing-bittorrent-sync-1-4-an-easier-way-to-share-large-files/. Accessed April 2015
3. Scanlon, M., Farina, J., Le Khac, N.-A., Kechadi, M.-T.: Leveraging Decentralisation to Extend the Digital Evidence Acquisition Window: Case Study on BitTorrent Sync, pp. 85–99, September 2014
4. Borg, J.: SyncThing (2015). http://www.syncthing.net. Accessed April 2015
5. Farina, J., Scanlon, M., Kechadi, M.-T.: Bittorrent sync: first impressions and digital forensic implications. Digital Invest. **11**(Suppl. 1), S77–S86 (2014). Proceedings of the First Annual DFRWS Europe
6. Quick, D., Choo, K.-K.R.: Dropbox analysis: data remnants on user machines. Digital Invest. **10**(1), 3–18 (2013)
7. Quick, D., Choo, K.-K.R.: Digital droplets: microsoft skydrive forensic data remnants. Future Gener. Comput. Syst. **29**(6), 1378–1394 (2013). Including Special sections: High Performance Computing in the Cloud and Resource Discovery Mechanisms for P2P Systems
8. Quick, D., Choo, K.-K.R.: Google drive: forensic analysis of data remnants. J. Netw. Comput. Appl **40**, 179–193 (2013)
9. Quick, D., Choo, K.-K.R.: Forensic collection of cloud storage data: does the act of collection result in changes to the data or its metadata? Digital Invest. **10**(3), 266–277 (2013)
10. Federici, C.: Cloud data imager: a unified answer to remote acquisition of cloud storage areas. Digital Invest. **11**(1), 30–42 (2014)
11. Reddit. SyncThing: Open Source BitTorrent Sync Alternative (P2P Sync Tool) (2015). http://www.webupd8.org/2014/06/syncthing-open-source-bittorrent-sync.html. Accessed April 2015
12. Borg, J.: SyncThing: Block Exchange Protocol (2015). https://github.com/syncthing/specs/blob/master/BEPv1.md. Accessed April 2015
13. Borg, J.: SyncThing: Config File and Directory (2015). https://github.com/syncthing/syncthing/wiki/Config-File-and-Directory. Accessed April 2015
14. Borg, J.: SyncThing: Device IDs (2015). https://github.com/syncthing/syncthing/wiki/Device-IDs. Accessed April 2015

15. Borg, J.: SyncThing: Device Discovery Protocol v2 (2015). https://github.com/syncthing/specs/blob/master/DISCOVERYv2.md. Accessed April 2015
16. Garfinkel, S., Nelson, A., White, D., Roussev, V.: Using purpose-built functions and block hashes to enable small block and sub-file forensics. Digital Invest. **7**, S13–S23 (2010)
17. Paul, J.: Java Revisited: Difference Between TrustStore and KeyStore Java SSL (2015). http://javarevisited.blogspot.ie/2012/09/difference-between-truststore-vs-keyStore-Java-SSL.html. Accessed April 2015