



(S) Engineering Development Group

(S) TheImp
V1.1

(U) Tool Documentation

Rev. 1.2
5 February 2015

Classified By: 2417940
Reason: 1.4(c)
Declassify On: 25X1, 20641125
Derived From: CIA NSCG COL S-06

Table of Contents

1 (U) Tool Summary.....	3
2 (U) Release Notes.....	3
2.1 (U) Code Reuse.....	3
3 (U) User's Guide.....	4
3.1 (U) Change Log.....	4
3.2 (U) File Information.....	4
3.2.1 (U) File/Registry Access.....	4
3.3 (U) Configuration.....	4
3.3.1 (U) IMPORTANT: Warnings.....	4
3.3.2 (U) Log file.....	5
3.3.3 (U) Prompted configuration.....	6
3.3.4 (U) Manual Configuration/Re-Configuration.....	6
3.4 (U) Operation.....	7
3.5 (U) Unpacking.....	7
4 (U) Troubleshooting/FAQ.....	8
5 (S//NF) PSP Information.....	9

Classified By: 2417940
Reason: 1.4(c)
Declassify On: 25X1, 20641125
Derived From: CIA NSCG COL S-06

(S) TheImp v1.1

1 (U) Tool Summary

(S//NF) TheImp is a folder monitoring tool which collects files that are added/changed in the folders being monitored. TheImp is designed to be persisted and run over long periods of time, to ensure collection of valuable data when operators are not actively on the target machine.

(S//NF) TheImp is designed solely for the FOX FLASH operation. A similar tool, TheHound, is available for non-FOX FLASH operations.

2 (U) Release Notes

(S//NF) TheImp is a ReflectLoad DLL (runs from DLL main). TheImp does not perform exfiltration of the collection data it produces.

2.1 (U) Code Reuse

(S//NF) TheImp borrowed the following code components from other projects:

- Urchin 3.0: List parsing function which parses a buffer into a WCHAR** array using a custom separator. TheImp uses a different custom separator than Urchin
- Urchin/TheHound/AllYourBase: Simple resource extraction function (verbatim copy). This function is extremely simplistic; It calls FindResource, LoadResource, and LockResource back-to-back. It also calls SizeOfResource and returns that value in an OUT parameter. The function is extremely small and is unlikely to be signed in a way that would tell an adversary much of anything
- TheHound: Folder Watching function is similar, but not an exact copy. TheHound uses different structures in its own Folder Watching function compared to TheImp. The similarities between the two implementations revolve around proper use of the Windows API function which performs folder watching, and is modeled after example code found online (but not copied).
- TheHound: File 'cooldown' mechanism is similar to the one used by TheImp. TheImp's file cooldown function is structured very differently, and uses a different event name string.
- TheHound: Thread death recovery feature in TheHound was borrowed for TheImp. While the code was re-written from scratch, the simple concept was re-used.

3 (U) User's Guide

3.1 (U) Change Log

Table 1: (S) Change log (contents SECRET)

Revision	Date	Author	Notes
1.0	25, Nov 2014	AED/RDB	Initial version.
1.1	8, Dec 2014	AED/RDB	Updated for new Flush File feature

3.2 (U) File Information

Table 2: (S) File information (contents SECRET//NOFORN)

3.2.1 (U) File/Registry Access

(S//NF) TheImp's nature is file collection in watched directories. TheImp does not do any registry modifications, nor does it read the registry directly for any reason.

3.3 (U) Configuration

(S//NF) TheImp requires prior configuration before deployment. Failure to configure prior to deployment will result in TheImp exiting immediately, and a stern shake of the head from the developer. Using ImpConfig.py, TheImp can be configured through two different means. Both means require, as an argument, the target binary to configure.

3.3.1 (U) IMPORTANT: Warnings

(S//NF) There are several important edge cases and conditions with regards to TheImp and the Windows API functions it uses. PLEASE read this section thoroughly and keep note of all the points it lays out.

(S//NF) When choosing a path for the kill file, flush file, or output files, **do not have the parent folder of either file be a watch folder top level directory**. Due to the buggy Windows API FindFirstDirectoryChange not working at SYSTEM, TheImp must use ReadDirectoryChangesW for monitoring the output file directory, kill file directory, and flush file directory. If these files are placed in the same top level folder as a watch directory, TheImp will get inconsistent notification from Windows, which usually means it will not get notified promptly if a kill/flush file is laid down. It is OK for the kill/flush/output file(s) to be placed in sub directories under a watched directory. Below is an example of what not do:

(S//NF) Don't do:

- Kill file at C:\kill.txt
- Flush file at C:\flush.txt
- Output files go to C:\imp.bin
- Watching directories C:\ and Z:\

(S//NF) What you can do:

- Kill file at C:\temp\kill.txt
- Flush file at C:\temp\flush.txt
- Output files go to C:\output\imp.bin
- Watching directories C:\ and Z:\

(S//NF) Network drives are mapped for the user that performed the mapping. If user Bob on a machine maps a network drive through the Explorer GUI to Z:, that drive will be mapped for Bob only, and visible to no one else. That means that a process running as SYSTEM cannot see Z:. If a path is suspected to be a network drive or resource, you must run TheImp at the same user level that was used to map the target network resource (Ex. Inject into explorer.exe). TheImp cannot do anything to avoid this situation; TheImp would have to know the remote IP of the network resource and the user credentials to map the resource at whatever user level TheImp is running at. Since this is not realistic, it is up to the user to inject TheImp at the appropriate user level. It may become necessary to have 2 instances of TheImp running: One injected into Explorer.exe to see network drives, and another injected into a SYSTEM process to watch the local drives with SYSTEM level privileges.

(S//NF) For paths suspected of being network drives/resources, do **not** put a trailing '\' at the end of the drive letter. You should not put a trailing '\' at the end of any path configured into TheImp, but it only causes a problem if the target path is a network drive/resource (Windows becomes confused). Example below:

(S//NF) Do not do:

- Y:\
- Z:\somefolder\

(S//NF) Do do:

- Y:
- Z:\somefolder

3.3.2 (U) Log file

(S//NF) The new log file is always required in 1.1. However, two binaries are provided in the 1.1 deliver: One binary (TheImp.dll) does not perform logging. The other binary (TheImp_diag.dll) does perform logging. No matter which DLL you are configuring, a path is required for the log file option. The non-logging binary will simply ignore the argument. This was done to save time when developing this project against a tight deadline.

The log file is not encrypted but also contains very little information. Its primary use is to determine if a watcher thread (thread responsible for watching a configured directory) has died in a controlled manner. Usually this is when there is an issue with the path that was configured (Access denied, folder does not exist, etc.). The log file will contain one logged issue per line. A logged issue will look like: [<path that the thread attempted to watch>] : [<error code>]. Example: [\\?\Z:] : [5]

3.3.3 (U) Prompted configuration

(S//NF) Running ImpConfig.py with a single argument, the binary to configure, will allow the user to configure TheImp by answering a series of questions. Before running ImpConfig, the user first must create a simple text file containing a list of all paths TheImp should watch. The file should have a single path on each line (no quotation marks around paths).

(S//NF) After creating the paths file, run: ImpConfig.py <path to Imp binary to configure>. The python script will prompt the user for answers to configuration questions.

(S//NF) With regards to chunk size, max file size, and max output size, the values should be in KB. Putting 0 for chunk size will cause TheImp to use a default chunk size of 10MB. Putting 0 for max output size will result in TheImp using a default value of 100MB. Using 0 for max file size will disable file size checking on files (no limit enforced)

(S//NF) Output path should be the absolute path, on target, and a file name base. Example: c:\windows\temp\impcollect.tmp. TheImp would create collection chunks c:\windows\temp\impcollect.tmp.0, c:\windows\temp\impcollect.tmp.1.

(S//NF) ImpConfig will produce a copy of the settings in a file named 'itmp.tmp'. The user can save these settings for re-configuration of more binaries later. The target binary will be copied, configured, and given a '.final' extension.

3.3.4 (U) Manual Configuration/Re-Configuration

(S//NF) TheImp can be configured with a settings file produced by the Prompted Configuration method, or with a manually-written configuration file. Please see the warnings below on this method!

(S//NF) Running ImpConfig in this mode requires two arguments: ImpConfig.py --config <config file> <target binary file>. ImpConfig will again copy the target binary file to a new file, configure it, and add the '.final' extension to the configured copy.

(S//NF) **WARNING/NOTE:** Manually writing the configuration file is easy to do, but requires strict adherence to the order of the configuration file. In the interest of time, TheImp contains a simple argument parser that relies on all arguments being in a certain order. The order of the configuration file is:

1. Kill file path
2. Flush file path
3. Passphrase
4. Output file path
5. Max output disk size (KB)
6. Chunk size (KB)
7. Max File Size (KB)

8. Log file path
9. Path(s)

(S//NF) Each entry is newline delimited. No quoting file paths. Each argument is required, and the final argument (Paths) can have 1 or more lines (up to 63 total paths). To use the default values for arguments 4 or 5 (100MB and 10MB respectively) simply put 0 for those values in the configuration file. If no max file size is desired, put 0 for argument 6.

3.4 (U) Operation

(S//NF) Once TheImp is configured, it can be run out of DLL main. TheImp will return 'true' from DLL main if it is able to successfully spawn the main startup thread. TheImp will continue to run until it is killed, or it detects the presence of the kill file.

(S//NF) TheImp scans for the kill file on a 30sec timeout cycle. It then notifies all worker threads to exit, waits for all threads to complete and exit, before finally exiting the main startup thread. During the cleanup phase, TheImp will flush any memory-resident collection data to disk. Upon successfully shutting down and cleaning up, TheImp will attempt to delete the kill file before exiting the startup thread. This is not guaranteed to be successful; TheImp will try once to delete the file, and exit regardless of success.

(S//NF) If TheImp finds the Flush File, it will flush all data currently in the memory buffer to disk. Once TheImp has successfully flushed the file, TheImp will attempt to delete the Flush File to signal completion of the task. If no flush file is desired, the argument is still required. The path must exist, though the file can be non-existent. There should be no harm in having a flush file trigger, even if one does not foresee using the feature.

3.5 (U) Unpacking

(S//NF) After recovering the collection files produced by TheImp, unpacking is as simple as running ImpUnpack.py with 3 arguments: The folder containing collection files to unpack, the passphrase used when configuring TheImp, and an output directory to unpack files to.

4 (U) Troubleshooting/FAQ

(S//NF) The following section will contain brief notes about common issues the user might see when using TheImp.

(S//NF) Can I use environment strings in my paths?

Yes. So long as the expansion of the environment string does not add more than MAX_PATH characters.

(S//NF) TheImp is acting weird

Make sure the configuration file burned in is correct. If you manually wrote the configuration file, make sure the ordering of all parameters is correct.

(S//NF) Can I collect very large files? What about files larger than the chunk size?

TheImp can collect an individual file up to just shy of 2GB big. TheImp can cleanly collect files that are larger than the chunk size.

(S//NF) If only some of the collection files produced by TheImp are recovered, can I still unpack?

Yes. Each collection file produced by TheImp is independent from a decryption/decompression standpoint. You'll be able to get any data in the recovered collection files. The LOST_AND_FOUND folder in the unpacked directory contains data that was found in collection files, without also finding a corresponding file metadata record for the data.

(S//NF) TheImp isn't collecting anything from network drives/resources. Make sure you've injected TheImp at the same user level that created the mapping for the network resource. Check the log file and let the developer know what error was returned. If 3 was in the log file, that typically means you're in the wrong user process to see the network drive.

5 (S//NF) PSP Information

(S//NF) No PSP testing was done by the developer. Considering the nature of TheImp, there should not be much of anything that a PSP would find alerting, malicious, etc. If TheImp is instructed to watch a directory containing PSP files, that may trigger some PSPs to alter when TheImp attempts to collect certain PSP files. Use cation when configuring directories for TheImp to watch.