NAME

debuginfod-find − request debuginfo-related data

SYNOPSIS

debuginfod-find [*OPTION***]... debuginfo** *BUILDID*
debuginfod-find [*OPTION***]... debuginfo** *PATH*
debuginfod-find [*OPTION***]... executable** *BUILDID*
debuginfod-find [*OPTION***]... executable** *PATH*
debuginfod-find [*OPTION***]... source** *BUILDID /FILENAME*
debuginfod-find [*OPTION***]... source** *PATH /FILENAME*
debuginfod-find [*OPTION***]... metadata** *KEY VALUE*

DESCRIPTION

debuginfod-find queries one or more **debuginfod** servers for debuginfo-related data. In case of a match, it saves the the requested file into a local cache, prints the file name to standard output, and exits with a success status of 0. In case of any error, it exits with a failure status and an error message to standard error.

The debuginfod system uses buildids to identify debuginfo-related data. These are stored as binary notes in ELF/DWARF files, and are represented as lowercase hexadecimal. For example, for a program /bin/ls, look at the ELF note GNU_BUILD_ID:

```
% readelf -n /bin/ls | grep -A4 build.id
Note section [ 4] '.note.gnu.buildid' of 36 bytes at offset 0x340:
Owner        Data size  Type
GNU              20  GNU_BUILD_ID
Build ID: 8713b9c3fb8a720137a4a08b325905c7aaf8429d
```

Then the hexadecimal BUILDID is simply:

```
8713b9c3fb8a720137a4a08b325905c7aaf8429d
```

In place of the hexadecimal *BUILDID*, debuginfod-find also accepts a *PATH* name to to an ELF binary, from which it extracts the buildid. This alternative is triggered if the file name has some character other than **[0-9a-f]**. Files ambiguously named files like "**deadbeef**" can be passed with a **./deadbeef** extra path component.

debuginfo *BUILDID*

If the given buildid is known to a server, this request will result in a binary object that contains the customary **.*debug_*** sections. This may be a split debuginfo file as created by **strip**, or it may be an original unstripped executable.

executable *BUILDID*

If the given buildid is known to the server, this request will result in a binary object that contains the normal executable segments. This may be a executable stripped by **strip**, or it may be an original unstripped executable. **ET_DYN** shared libraries are considered to be a type of executable.

source *BUILDID /SOURCE/FILE*

If the given buildid is known to the server, this request will result in a binary object that contains the source file mentioned. The path should be absolute. Relative path names commonly appear in the DWARF file's source directory, but these paths are relative to individual compilation unit AT_comp_dir paths, and yet an

executable is made up of multiple CUs. Therefore, to disambiguate, debuginfod expects source queries to prefix relative path names with the CU compilation-directory, followed by a mandatory "/".

Note: for software packaged by distributions, the CU compilation-directory may not be obvious. It can be found by inspecting AT_comp_dir values in downloaded debuginfo. For example, the comp_dir of the Fedora 37 version of /bin/ls can be found as follows:

```
% debuginfod-find debuginfo /bin/ls
~/.cache/debuginfod_client/03529d48345409576cd5c82a56ad08555088d353/
% eu-readelf -w ~/.cache/debuginfod_client/03529d48345409576cd5c82a56ad08555088d353/debuginfo | grep comp_dir
        comp_dir        (line_strp) "/usr/src/debug/coreutils-9.1-6.fc37.x86_64/separate"
```

Note: the caller may or may not elide **../** or **/./** or extraneous /// sorts of path components in the directory names. debuginfod accepts both forms. Specifically, debuginfod canonicalizes path names according to RFC3986 section 5.2.4 (Remove Dot Segments), plus reducing any // to / in the path.

For example:

#include <stdio.h>	source BUILDID /usr/include/stdio.h
/path/to/foo.c	source BUILDID /path/to/foo.c
../bar/foo.c AT_comp_dir=/zoo/	source BUILDID /zoo//../bar/foo.c

section *BUILDID SECTION-NAME*

If the given buildid is known to the server, this request will result in a binary file that contains the raw section contents, whether it is found in an executable or a separated debuginfo file.

metadata *KEY VALUE*

All designated debuginfod servers are queried for metadata about all files that match a given key/value query in their index. The results include names and buildids, which may be used in future queries to fetch actual files.

KEY	VALUE	DESCRIPTION
file	*path*	exact match *path*, including in archives
glob	*pattern*	shell-style glob match *pattern*, including in archives, as in fnmatch(FNM_PATHNAME)
buildid	*hex*	all executables and debuginfo files matching the given buildid

The resulting output will look something like the following {

```
 "results":[
   {
    "type":"executable",
    "buildid":"f0aa15b8aba4f3c28cac3c2a73801fefa644a9f2",
    "file":"/usr/local/bin/hello",
    "archive":"/opt/elfutils/tests/test-2290642/R/rhel7/hello2-1.0-2.x86_64.rpm"
   },
   {
    "type":"executable",
    "buildid":"bc1febfd03ca05e030f0d205f7659db29f8a4b30",
    "file":"hello2"
   }
 ],
 "complete":true }'
```

The results of the search are output to **stdout** as a JSON object containing an array of objects, supplying

metadata about each match, as well as a boolean value corresponding to the completeness of the result. The result is considered complete if all of the queries to upstream servers returned complete results and the local query succeeded. This metadata report may be cached. It may be incomplete and may contain duplicates. Additional JSON object fields may be present.

NAME	TYPE	DESCRIPTION
buildid	string	hexadecimal buildid associated with the file
type	string	one of **debuginfo** or **executable**
file	string	matched file name, outside or inside the archive
archive	string	archive containing matched file name, if any

It's worth noting that **type** cannot be **source** since in order to perform such a search fast enough additional indexing would need to be added to the database which would nearly double its size.

The search also always combines both files and archives in the results and at this time further granularity is not availible.

OPTIONS

−v −−verbose

Increase verbosity, including printing frequent download-progress messages and printing the http response headers from the server.

−−help −−usage

Print some help and exit.

SECURITY

If IMA signature(s) are available from the RPMs that contain requested files, then **debuginfod** will extract those signatures into response headers, and **debuginfod-find** will perform verification upon the files. Validation policy is controlled via tags inserted into $DEBUGINFOD_URLS. By default, **debuginfod-find** acts in ignore mode.

If accessed across HTTP rather than HTTPS, the network should be trustworthy. Authentication information through the internal *libcurl* library is not currently enabled, except for the basic plaintext *http[s]://userid:password@hostname/* style. (The debuginfod server does not perform authentication, but a front-end proxy server could.)

The following information is generic. Some debuginfod clients may internally override defaults listed here.

ENVIRONMENT VARIABLES

$TMPDIR

This environment variable points to a file system to be used for temporary files. The default is /tmp.

$DEBUGINFOD_URLS

This environment variable contains a list of URL prefixes for trusted debuginfod instances. Alternate URL prefixes are separated by space. This environment variable may be set by /etc/profile.d scripts reading /etc/debuginfod/*.urls files.

This environment variable can also contain policy defining tags which dictate the response policy for verifying per-file IMA signatures in RPMs. As the space seperated list is read left to right, up-

on encountering a tag, subsequent URLs up to the next tag will be handled using that specified policy. All URLs before the first tag will use the default policy, *ima:ignore*. For example:

```
DEBUGINFOD_URLS="https://foo.com ima:enforcing https://bar.ca http://localh
```

Where foo.com and baz.org use the default *ignore* policy and bar.ca and localhost use an *enforcing* policy. The policy tag may be one of the following:

ima:enforcing Every downloaded file requires a valid signature, fully protecting integrity.

ima:ignore Skips verification altogether, providing no protection.

Alerts of validation failure will be directed as specified in $DEBUGINFOD_VERBOSE.

$DEBUGINFOD_CACHE_PATH

This environment variable governs the location of the cache where downloaded files and cache-control files are kept. The default directory is chosen based on other environment variables, see below.

$DEBUGINFOD_PROGRESS

This environment variable governs the default progress function. If set, and if a progressfn is not explicitly set, then the library will configure a default progressfn. This function will append a simple progress message periodically to stderr. The default is no progress function output.

$DEBUGINFOD_VERBOSE

This environment variable governs the default file descriptor for verbose output. If set, and if a verbose fd is not explicitly set, then the verbose output will be produced on STDERR_FILENO.

$DEBUGINFOD_RETRY_LIMIT

This environment variable governs the default limit of retry attempts. If a query failed with errno other than ENOENT, will initiate several attempts within the limit.

$DEBUGINFOD_TIMEOUT

This environment variable governs the download *commencing* timeout for each debuginfod HTTP connection. A server that fails to provide at least 100K of data within this many seconds is skipped. The default is 90 seconds. (Zero or negative means "no timeout".)

$DEBUGINFOD_MAXTIME

This environment variable dictates how long the client will wait to *complete* the download a file found on a server in seconds. It is best used to ensure that a file is downloaded quickly or be rejected. The default is 0 (infinite time).

$DEBUGINFOD_MAXSIZE

This environment variable dictates the maximum size of a file to download in bytes. This is best used if the user would like to ensure only small files are downloaded. A value of 0 causes no consideration for size, and the client may attempt to download a file of any size. The default is 0 (infinite size).

$DEBUGINFOD_HEADERS_FILE

This environment variable points to a file that supplies headers to outbound HTTP requests, one per line. The header lines shouldn't end with CRLF, unless that's the system newline convention. Whitespace-only lines are skipped.

$DEBUGINFOD_IMA_CERT_PATH
This environment variable contains a list of absolute directory paths holding X.509 certificates for RPM per-file IMA-verification. Alternate paths are separated by colons.

CACHE

Before each query, the debuginfod client library checks for a need to clean the cache. If it's time to clean, the library traverses the cache directory and removes downloaded debuginfo-related artifacts and newly empty directories, if they have not been accessed recently.

Control files are located directly under the cache directory. They contain simple decimal numbers to set cache-related configuration parameters. If the files do not exist, the client library creates the files with the default parameter values as content.

After each query, the debuginfod client library deposits newly received files into a directory & file that is named based on the build-id. A failed query is also cached by a special file. The naming convention used for these artifacts is deliberately **undocumented**.

$XDG_CACHE_HOME/debuginfod_client/
Default cache directory, if $XDG_CACHE_HOME is set.

$HOME/.cache/debuginfod_client/
Default cache directory, if $XDG_CACHE_HOME is not set.

$HOME/.debuginfod_client_cache/
Deprecated cache directory, used only if preexisting.

cache_clean_interval_s
This control file gives the interval between cache cleaning rounds, in seconds. The default is 86400, one day. 0 means "immediately".

max_unused_age_s
This control file sets how long unaccessed debuginfo-related files are retained, in seconds. The default is 604800, one week. 0 means "immediately".

cache_miss_s
This control file sets how long to remember a query failure, in seconds. New queries for the same artifacts within this time window are short-circuited (returning an immediate failure instead of sending a new query to servers). This accelerates queries that probably would still fail. The default is 600, 10 minutes. 0 means "forget immediately".

metadata_retention_s
This control file sets how long to remember the results of a metadata query. New queries for the same artifacts within this time window are short-circuited (repeating the same results). This accelerates queries that probably would probably have the same results. The default is 3600, 1 hour. 0 means "do not retain".

SEE ALSO

debuginfod(8) debuginfod_find_debuginfod(3)