Reference Manual

ufdbGuard API

version 1.35.6

Table of Contents

1 Introduction

This manual is for an audience with a technical background and it is assumed that the reader is familiar with the concepts of compilers and libraries and how to use them.

The ufdbGuard API, or just "API", is a URL classifier. The API is implemented as a C library and header files and can be used by any program written in C or C++.

A URL classifier in a tool that given any URL is able to produce a list of URL categories. The ufdbGuard API is designed to be used together with the URL database of URLfilterDB. In addition to our database, one can use user-defined URL tables.

A program that uses the API uses functions to load URL database categories, and perform URL lookups. The performance depends on the CPU and especially on the performance of its cache. On an Intel E5 1650 v3 CPU with 6 cores and 12 threads the classifier reaches 250,000 classifications per second using 4 threads. The high performance library uses database format 3.0 which is optimized to be used on CPUs with advanced vector instructions and performs 310,000 classifications per second using a single thread of the same CPU. On a more recent AMD Ryzen 9 5950X 16-Core processor a single thread supports 500,000 classifications per second and with 4 threads 1,450,000 classifications per second. Performance may very with CPU frequency, cache architecture and memory bandwidth.

Service providers and system integrators may register as a trial user at www.urlfilterdb.com to receive a 60-day trial license for the API and the URL database.

1.1 Platforms

The ufdbGuard API can be used on various platforms and with various features. In the table below is an overview of current supported platforms. In case that a desired platform is not included one may contact the support desk to find out if the desired platform can be supported.

	OS	CPU	TLS/SSL	regex	pthreads	spinlock
standard	Linux and BSD	x86_64	+ or –	+	+	–
high performance	Linux and BSD	x86_64 AVX[1]	+ or –	–	+	–
high performance	bare metal	MIPS64 Octeon III	–	–	–	+
high performance	Linux with DPDK	x86_64 AVX	–	–	+	+

Up to version 1.34.x the ufdbGuard API was based on database format 2.2 and delivered with source code. The high performance database formats 3.0 and 3.1 are introduced in version 1.35 which is closed source and only released in binary format.

1.2 Copyright

The ufdbGuard API software suite is entirely developed and owned by URLfilterDB B.V. with all rights reserved. URLfilterDB B.V. holds the copyrights on the ufdbGuard API software suite. The ufdbGuard API may only be used in combination with the URL database of URLfilterDB, hence a valid license to use the URL database is required to gain the right to use the ufdbGuard API. Users of the ufdbGuard API are

[1] AVX libraries are compiled with the following flags: `-msse4.2 -mavx -mpopcnt -maes -mpclmul`

not permitted to modify or use the ufdbGuard API with other URL databases. However, users of the API are free to extend the URL database with user-defined categories.

The URL database is a commercial product and has a copyright by URLfilterDB. A license is required to use the URL database which is defined in The Terms of Contract document that can be downloaded at the website: www.urlfilterdb.com.

1.3 Latest Major Changes and Enhancements

Version 1.35 has support for new platforms. Previous versions required pthreads and openssl libraries, but 1.35 can be made independent of pthreads, openssl and regex. An API library for an application on a bare metal system like Marvell's MIPS64 Octeon CPU, or DPDK, see also www.dkdp.org, is also possible. You may discuss all possibilities with the support desk. This version of the ufdbGuard API is only released in binary form.

The manual for version 1.35 has many minor modifications for clarification of the API.

Version 1.34 supports the new OpenSSL 1.1 libraries. The installation directory is `/usr/local/ufdbguard-api`.

A detailed list of all changes can be found in the directory `/usr/local/ufdbguard-api/src` in the file `CHANGES`.

1.4 Support and Feedback

We welcome feedback from those who test our software. Feel free to send your questions and feedback to the support desk: support@urlfilterdb.com.

2 Prerequisites

The ufdbGuard API runs on all flavors of UNIX.

The ufdbGuard API needs 200 MB disk space and 1 GB memory for the 64-bit version.

The ufdbGuard API uses a compressed database and therefore requires the compression libraries zlib and bz2. The bzip2 library is not implemented in the API for bare metal systems and systems with DPDK, and is generally not recommended since it consumes a lot more CPU resources.

The ufdbGuard API optionally uses the OpenSSL libraries to safely communicate with the servers of URLfilterDB. An alternative which does not use OpenSSL libraries, is to produces files that can be uploaded with `curl`.

2.1 Prerequisites for Development

The ufdbGuard API is a library and can be linked with applications that use the C calling convention. Most UNIX distributions come with the free GNU C compiler, `gcc` (see also gcc.gnu.org) or a native C compiler which can be used to link the library with a 3rd party program. The API can also be used by programs written in C++.

In addition, the `wget`, `curl`, `tar`, `gzip`, `make` and `install` commands are required which are all included in most UNIX distributions.

2.2 Prerequisites for Production

The API and the `ufdbUpdate` script must be able to talk to the servers of URLfilterDB. They use ports 80 and 443 to connect to `updates.urlfilterdb.com` over IPv4 and IPv6. A production environment may have a firewall and hence must have a permissive rule to be able to connect to the servers of URLfilterDB.

3 API functions

The ufdbGuard API is multithreaded and in almost all implementations uses the pthreads library.

The functions of the API library are divided in 3 categories: initialization and termination, classification and management of uncategorised URLs and miscellaneous functions. In the `src` directory is a file called `apitest.c` which serves as an example on how to use all API functions. Alternatively, one can also look at the file `urlcats.c` which is the source of a multithreaded file-based URL classifier.

The API functions are divided over two libraries: libufdbapi.a and libufdbhttpsapi.a. The API functions that rely on OpenSSL are in libufdbhttpsapi.a and developers may choose not to use API functions that depend on OpenSSL and therefore do not have to use the httpsapi library. The following subsections document all API functions and includes "(HTTPS)" in the title if the function depends on OpenSSL.

3.1 Initialization and Termination

3.1.1 UFDBapiInit

```
int UFDBapiInit( void )
```

`UFDBapiInit` initializes internal data structures of the API. It is mandatory to call `UFDBapiInit` before are other API function is called and before any API parameter is set. `UFDBapiInit` returns `UFDB_API_OK` or an API error code.

Applications typically call `UFDBapiInit`, then optionally set global variables and finally call `UFDBlocalCategory` for *all* categories.

On bare metal systems with a custom memory allocator the function `UFDBmallocInit` must be called before `UFDBapiInit`.

Applications that use DPDK must call `ret_eal_init` before calling `UFDBapiInit`.

3.1.2 UFDBallocThreadAdmin

```
UFDBthreadAdmin * UFDBallocThreadAdmin( void )
```

`UFDBallocThreadAdmin` allocates a private static memory buffer and returns a pointer which is referred to as the thread handle. The handle is used in calls to some other API functions. `UFDBallocThreadAdmin` must be called once for each thread that calls ufdbGuard API functions.

Note that also single-threaded applications need to use this function.

3.1.3 UFDBfreeThreadAdmin

```
void UFDBfreeThreadAdmin( UFDBthreadAdmin * handle )
```

`UFDBfreeThreadAdmin` deallocates the thread handle and associated memory that was previously allocated by `UFDBallocThreadAdmin`.

3.1.4 UFDBinitHTTPSchecker (HTTPS)

```
int UFDBinitHTTPSchecker( void )
```

`UFDBinitHTTPSchecker` initializes the API to be able to use HTTPS probes. This function initializes OpenSSL and must be called before any thread calls `UFDBcheckForHTTPStunnel`.

`UFDBinitHTTPSchecker` returns a status code which may be `UFDB_API_OK` or `UFDB_API_ERR_ERRNO`.

HTTPS probes are optional and can be used to verify server certificates, detect SSH servers and a number of proxies. Since `UFDBcheckForHTTPStunnel` may take up to 25 seconds to return, for most environments it is not recommended to use HTTPS probes.

3.1.5 UFDBloadCategory

```
int UFDBloadCategory(
   char * dbfile,                // file name of .ufdb file
   char * exprfile,              // file name of expressions, may be NULL
   UFDBcategory * category )     // allocated by caller
```

`UFDBloadCategory` loads a URL category and return a status code which is one of the following:

`UFDB_API_OK`	URL category was loaded
`UFDB_API_STATUS_DATABASE_OLD`	URL category was loaded and the caller is warned that the category is over 4 days old.
`UFDB_API_STATUS_DATABASE_EXPIRED`	URL category is expired (more than 28 days old)
`UFDB_API_ERR_NOFILE`	the file *`dbfile`* does not exist or cannot be opened for reading.
`UFDB_API_ERR_READ`	failed to read the file
`UFDB_API_ERR_INVALID_TABLE`	URL table is corrupt
`UFDB_API_ERR_INVALID_KEY`	URL table could not be decrypted

The caller of `UFDBloadCategory` must allocate `UFDBcategory` and `UFDBloadCategory` fills the struct.

Applications must load all categories since they are all needed to determine whether a URL is uncategorised or not. If the application does not load all main database categories, the management of uncategorised URLs will be harmed.

3.1.6 UFDBunloadCategory

```
void UFDBunloadCategory( UFDBcategory * category )
```

`UFDBunloadCategory` deallocates all memory used by the specified category. After unloading a category it cannot be used any more.

3.2 Database Refresh

Applications that wish to have the shortest time possible to refresh a database without loosing the ability to classify URLs during the database refresh, will load a new URL database while a the current database is still being used. These applications will execute code similar to the following pseudo code.

```
for (i=0; i<numNewCats; i++)
   UFDBloadCategory( fname[i], NULL, &newCat[i] );
SyncAllThreads();                // sync and swap pointers to database tables
for (i=0; i<numOldCats; i++)
   UFDBunloadCategory( &oldCat[i] );
UFDBuploadUncategorisedURLs( “myapp-1.1” ); // or UFDBsaveUncategorisedURLsAndMetaData
```

In `SyncAllThreads` the application uses an application-defined method to signal all categorisation threads that they can no longer use the current database tables, waits for all threads to finish using the current tables, swaps the pointer for the database tables with the new ones, and signals all categorisation threads that the new tables can be used.

3.2.1 Lockless Database Refresh

Since database table lookups are very fast one possible implementation does not use locks but a global boolean variable indicating that the tables are being refreshed and assume that the threads that use the old tables during the refresh will stop using them within 10 ms[2]. Categorisation threads that observe that the global boolean indicates a refresh, have two options: they can pause until the refresh is over, or they can stop filtering and pass URLs unfiltered (and must refrain from using API functions).

The pseudo code for the thread that performs the refresh is:

```
databaseRefresh = 1;             // must be declared volatile
usleep( 10000 );                 // wait 10ms for all classifying threads
for (i=0; i<numOldCats; i++)
   oldCat[i] = currentCat[i];
for (i=0; i<numNewCats; i++)
   currentCat[i] = newCat[i];
numCats = numNewCats;
databaseRefresh = 0;
```

The pseudo code for a classifying thread that waits for the completion of the database refresh, is:

```
while (URL = getNewURL()) {
   while (databaseRefresh)
      usleep( 1000 );            // pause 1ms
   classifyURL( URL, currentCat[], numCats );
}
```

The pseudo code for a classifying thread that passes URLs unfiltered during a database refresh (maximum 11 ms), is:

```
while (URL = getNewURL()) {
   if (databaseRefresh)
      passURL( URL );            // do not use ufdbGuard API functions
   else
      classifyURL( URL, currentCat[], numCats );
}
```

3.3 Classification

A URL classification is done in 3 steps: strip the URL, convert the URL into an internal representation for fast table lookups, and finally the table lookup. For each URL, the first two steps are executed one and the table lookup is executed for each table.

3.3.1 UFDBstripURL

```
void UFDBstripURL( char * URL, char * strippedUrl, char * domain,
                   char * protocol, int  * portnumber )
```

`UFDBstripURL` takes a URL and produces a stripped URL, its domainname, the used protocol and the used portnumber. URLs must be stripped to remove usernames and password, convert character codes, and translate URL paths like `foo/../bar`. The stripped URL must be allocated by the caller and is

[2] On a 3.5 GHz Intel Xeon CPU, a single thread can classify 75,000 URLs per second with database format 2.2 and over 300,000 URLs per second with database format 3.0.

`char[UFDB_MAX_URL_LENGTH]`. The domain must be allocated by the caller and is `char[1024]`. The protocol must be allocated by the caller and is `char[16]`.

3.3.2 UFDBstripDomain

```
void UFDBstripDomain(
const char * domain,        // input: domain
char *       stripped  )    // output: must be char[1024]
```

Applications that do not use full URLs with a protocol, domainname, port, optional username/password, a URL path and parameters, but simply works with domainnames, may use the more efficient `UFDBstripDomain` instead of `UFDBstripURL`.

`UFDBstripDomain` strips a leading www[0-9]*. prefix and converts the domainname to lowercase. `UFDBstripDomain` can only strip plain domainnames.

3.3.3 UFDBgenRevURL

```
UFDBrevURL * UFDBgenRevURL( UFDBthreadAdmin * admin, unsigned char * URL )
```

`UFDBgenRevURL` takes a stripped URL and converts it into an internal representation. This conversion is required since only with the internal representation the table lookup is extremely fast.

This function may return `NULL` if the URL could not be parsed.

3.3.4 UFDBfreeRevURL

```
void UFDBfreeRevURL( UFDBthreadAdmin * admin, UFDBrevURL * revUrl )
```

`UFDBfreeRevURL` frees all resources that were allocated by `UFDBgenRevURL` and must be called after the table lookup of all categories.

3.3.5 UFDBverifyUrlandProtocolCategory

```
int UFDBverifyUrlandProtocolCategory( char * strippedUrl,
       UFDBrevURL * revUrl, char * protocol, UFDBusedCategory * category )
```

`UFDBverifyUrlandProtocolCategory` is used to see if a URL belongs to a certain category and all parameters are used to determine the inclusion or exclusion of a category. The return value may be 0 (excluded) or 1 (included).

3.3.6 UFDBcheckForHTTPStunnel (HTTPS)

```
int UFDBcheckForHTTPStunnel( char * hostname, int portnumber, int flags )
```

`UFDBcheckForHTTPStunnel` performs a probe and opens one or more sockets to port portnumber of the server identified by hostname. This function may take up to 25 seconds to return. The return value is one of the following:

`UFDB_API_OK`	regular https traffic
`UFDB_API_REQ_QUEUED`	request is queued for an other thread
`UFDB_API_ERR_TUNNEL`	https channel is tunneled
`UFDB_API_ERR_UNKNOWN_PROTOCOL`	https channel is tunneled
`UFDB_API_ERR_IS_SKYPE`	https channel is used by Skype
`UFDB_API_ERR_IS_GTALK`	https channel is used by Gtalk
`UFDB_API_ERR_IS_FBCHAT`	https channel is used by Facebook Chat
`UFDB_API_ERR_IS_CITRIXONLINE`	https channel is used by CitrixOnline

`UFDB_API_ERR_IS_ANYDESK`	https channel is used by Anydesk
`UFDB_API_ERR_IS_TEAMVIEWER`	https channel is used by Teamviewer
`UFDB_API_ERR_CERTIFICATE`	TLS/SSL certificate is self-signed or has wrong common name
`UFDB_API_ERR_NULL`	tunnel verification is OFF.

3.4 Management of Uncategorised URLs

The API maintains a list of URLs that are not yet part of the URL database, the *uncategorised URLs*. To maintain the URL database up to date, it is required that the list of uncategorised URLs is regularly uploaded to the servers of URLfilterDB where they are analyzed.

It is required that each program that uses the API maintains and upload the uncategorised URLs. This is done by two API calls:

1) registering an uncategorised URL whenever is it detected that it does not belong to any URL category, and
2) uploading the set of uncategorised URLs.

See also the example implementation in `apitest.c` on how to implement this in an application.

3.4.1 UFDBVerifyURLisUncategorised

```
int UFDBVerifyURLisUncategorised( char * URL, UFDBrevURL * revURL,
        UFDBusedCategory category[], int n_categories )
```

`UFDBVerifyURLisUncategorised` verifies if a URL is uncategorised by verifying the URL in all categories. The return value is one of the following:

`UFDB_API_URL_IS_CATEGORISED`	The URL is in the URL database.
`UFDB_API_URL_IS_LOCALNET`	The URL is part of the local network, e.g. 10.1.1.1/index.html
`UFDB_API_URL_IS_UNCATEGORISED`	The URL is not part of the database.

This function is not required to be used if the application knows that a URL is part of at least one category.

3.4.2 UFDBsaveUncategorisedURLandPort

```
int UFDBsaveUncategorisedURLandPort( char * uncategorisedURL, int portnumber )
```

`UFDBsaveUncategorisedURLandPort` saves the stripped URL and portnumber which will be uploaded later.

`UFDBsaveUncategorisedURLandPort` must be called if a URL is not in any category.

3.4.3 UFDBuploadUncategorisedURLs (HTTPS)

```
int UFDBuploadUncategorisedURLs( const char * agent )
```

`UFDBuploadUncategorisedURLs` uploads all uncategorised URLs that were saved with the function `UFDBsaveUncategorisedURLandPort`. The parameter `agent` must identify the application and include its version number. The value of `agent` is for example “myproxy-1.1”.

`UFDBuploadUncategorisedURLs` or `UFDBsaveUncategorisedURLsAndMetaData` must be called on a regular basis, at least once per two hours on systems with a very high query rate and at most once per 30 minutes. It must be called at least once per 8 hours on systems with a low query rate and may be called more often for medium and high query rates.

`UFDBuploadUncategorisedURLs` depends on the OpenSSL library. Alternatively one can use `UFDBsaveUncategorisedURLsAndMetaData` which does not depend on OpenSSL.

`UFDBuploadUncategorisedURLs` and `UFDBsaveUncategorisedURLsAndMetaData` upload besides the uncategorised URLs also query statistics and host information which is data that URLfilterDB must receive, hence the use of `UFDBuploadUncategorisedURLs` *or* `UFDBsaveUncategorisedURLsAndMetaData` is mandatory.

3.4.4 UFDBsaveUncategorisedURLsAndMetaData

```
int UFDBsaveUncategorisedURLsAndMetaData(
       const char * applicationIdentifier,
       const char * fileName )
```

`UFDBsaveUncategorisedURLsAndMetaData` saves all uncategorised URLs, statistics and meta data to a file. The parameter `fileName` must be a valid filename which `UFDBsaveUncategorisedURLsAndMetaData` will create or overwrite. The parameter `applicationIdentifier` identifies the application and should include its version number. The value of `applicationIdentifier` is for example "myproxy-1.1".

`UFDBuploadUncategorisedURLs` or `UFDBsaveUncategorisedURLsAndMetaData` must be called on a regular basis, at least once per two hours on systems with a very high query rate and at most once per 30 minutes. It must be called at least once per 8 hours on systems with a low query rate and may be called more often for medium and high query rates.

The created file must be uploaded to the servers of URLfilterDB where the upload may be delayed for a maximum of 15 minutes. The command to upload file `foobar` is this:

```
curl --user-agent "UFDB-<Vendor>-<Version>" --request PUT  \
   --upload-file "<filename>"  \
   https://updates.urlfilterdb.com/cgi-bin/uncatmeta.pl
```

The webserver returns a status which should be "OK" so the output of the `curl` command must be checked.

`UFDBsaveUncategorisedURLsAndMetaData` returns `UFDB_API_OK` or `UFDB_API_ERR_ERRNO`.

`UFDBuploadUncategorisedURLs` and `UFDBsaveUncategorisedURLsAndMetaData` upload besides the uncategorised URLs also query statistics and host information which is data that URLfilterDB must receive, hence the use of `UFDBuploadUncategorisedURLs` *or* `UFDBsaveUncategorisedURLsAndMetaData` is mandatory.

3.5 SafeSearch

Many search engines accept enforcement of safesearch by setting a URL parameter. For example, `google.com` accepts URLs with `&safe=active` appended to enforce safesearch. The API function UFDBaddSafeSearch adds appropriate parameters to URLs of many search engines.

3.5.1 UFDBaddSafeSearch

```
int UFDBaddSafeSearch(
  char * domain,
  char * strippedURL,
  char * originalURL  )
```

The function UFDBaddSafeSearch returns UFDB_API_OK if `originalURL` does not have a recognized search engine and returns UFDB_API_MODIFIED_FOR_SAFESEARCH when `originalURL` has been modified to contain a tag to enforce safesearch.

Note that originalURL must have sufficient space to append or or more safesearch tags.

3.5.2 DNS-based SafeSearch of Google

Google offers an alternative to enforce SafeSearch which is explained on their website: https://support.google.com/websearch/answer/186669?hl=en option 3 and configure DNS to have a CNAME record entry for www.google.com pointing to forcesafesearch.google.com. Also make CNAME entries for Google on popular TLDs in your region, e.g. www.google.de, www.google.es or www.google.com.br.

3.5.3 DNS-based Content Restriction on Youtube

Youtube content also can be restricted using DNS. Youtube uses the same mechanism as Google and is explained here: https://support.google.com/youtube/answer/6214622. To implement it one needs to add a CNAME restrict.youtube.com for the following domains: www.youtube.com, m.youtube.com, youtubei.googleapis.com, youtube.googleapis.com and www.youtube-nocookie.com.

3.5.4 DNS-based SafeSearch of Bing

Bing offers an alternative to enforce SafeSearch which is explained on their website: `http://help.bing.microsoft.com/#apex/18/en-US/10003/0` and configure DNS to have a CNAME record entry for www.bing.com pointing to strict.bing.com.

3.6 Miscellaneous

3.6.1 UFDBgetCounters

```
void UFDBgetCounters(
   unsigned long *  lookups,
   unsigned long *  matches,
   unsigned long *  localnet,
   unsigned long *  uncategorised,
   unsigned long *  safesearch,
   double *         TPS1hour,
   struct timeval * TPStime1hour,
   double *         TPS15seconds,
   struct timeval * TPStime15seconds )
```

The API has various counters which can be retrieved by calling `UFDBgetCounters`. All parameters contain metrics and must be allocated by the caller. `TPS15seconds` contains the peak number of queries per second (averaged over 15 seconds).

3.6.2 Messages and Debugging

The API has sometimes something to report and uses the functions `ufdbLogMessage`, `ufdbLogError` and `ufdbLogFatalError`. Every application that uses the API must have these functions and deal appropriately with the messages. The file `ufdblogerror.c` contains examples of these functions. The prototypes of the logging functions are similar to `printf`.

```
void ufdbLogMessage( const char * format, ... )
void ufdbLogError( const char * format, ... )
void ufdbLogFatalError( const char * format, ... )
```

For backward compatibility with older versions of the API, the application must also have a stub function called `ufdbSetGlobalErrorLogFile`.

```
void ufdbSetGlobalErrorLogFile( char * logdir, char * basename, int mutex_is_used )
```

In case that the API needs debugging, one can set the global variable `ufdbGV.debug` to 1, 2 or 3 which results in the API calling `ufdbLogMessage` with debug information. To disable debugging, set `ufdbGV.debug` to `0` (default).

Note that the API may find it necessary to call `ufdbLogError` or `ufdbLogFatalError` at any time, so the application must be aware that this may happen.

3.6.3 API Global Variables

The API has all its global variables in a global C struct `ufdbGV`. Applications should not modify these variables, except for the variables mentioned in the next table.

variable in ufdbGV	explanation	values	previously known as
debug	set debug level of the API	0, 1, 2 or 3	UFDBglobalDebug
debugRegexp	set debug level of regexp processing	0 or 1	UFDBglobalDebugRegexp
CAcertsfile	set filename with CA certificates	file name	UFDBglobalCAcertsfile
OfficialCertificate	check if server uses a correct certificate when probed	0 or 1	UFDBglobalOfficialCertificate
silentLogMessage	suppress all messages generated by ufdbLogMessage	0 or 1	UFDBglobalSilentLogMessage
madviseHugePages	hint Linux to use hugepages (2 MB) for large in-memory database tables (table must be at least 1.6 MB)	0 or 1	-
parseURLparameters	consider parameters and their values when doing URL lookups	0 or 1	-

3.7 Examples

See the source file `apitest.c` for a fully functional test program to test predefined URLs or URLs from a specified file.

See the source file `urlcats.c` for a fully functional multi-threaded URL classifier which uses a file or *stdin* as input.

3.7.1 Simplified Example C program

```
UFDBapiInit();
/* load URL tables */
admin = UFDBallocThreadAdmin();
while ((URL = fetchNextURL()) != NULL) {
   UFDBstripURL( URL, strippedUrl, domain, protocol, &port );
   revURL = UFDBgenRevURL( admin, strippedUrl );
   for (i = 0; i < nCats; i++) {
      if (UFDBverifyUrlandProtocolCategory( strippedUrl, revURL, protocol,
                                            &usedCategory[i] )) {
         /* the URL is in category i */
      }
   }
   UFDBfreeRevURL( admin, revURL );
}
```

3.8 Using the ufdbGuard API Libraries

To use the ufdbGuard API one must include the appropriate header file and link with the libraries.

C or C++ source code must include `ufdbguard-api.h`. and use compiler flags to search the directory `/usr/local/ufdbguard-api/include` for include files.

3rd party executables that include the ufdbGuard API must be linked against the libraries so use the correct linker flags to search for libraries in `/usr/local/ufdbguard-api/lib` and include the libraries `ufdbapi` and `ufdbhttpsapi`.

3.9 File-based URL Classifier

The API libraries are used by an example program called `urlcats` which reads an input file with URLs and produces an output file the URL categories and URLs. The program recognizes the following command line options.

`-i FILE`	input is read from file *FILE*. One URL per line. If omitted, use *stdin* as input.
`-o FILE`	output is written to file *FILE*. If omitted, use *stdout* as output.
`-d DIR`	*DIR* is the URL database directory.
`-e`	also use the *error* category
`-s`	print subcategories
`-1`	print the first matched URL category only, default is to print all matched categories
`-C`	format of input file is '<*number*> <*URL*>'
`-f` *N*	the URL is in field *N* of each input line, space and tab are field separators
`-D`	enable debug output
`-T`	print performance timers
`-V`	print version and exit
`-t` *N*	use *N* threads, *N*=4 by default, maximum is 32
`-u` *URL*	print categories for one URL only
`-a`	use thread/CPU affinity to keep thread *N* on CPU core *N*
`-c` *N*	start with CPU ID *N* as the first CPU for thread/CPU affinity (default is 0)

4 Software Installation

Follow all steps outlined in the following subsections to install the software and URL database, and verify its installation.

4.1 Upgrading from a Previous Version

New versions of the API are backwards compatible with rare exceptions. always read the file `/usr/local/ufdbguard-api/src/CHANGES`.

4.2 Installation Directory and Ownership

The files of the ufdbGuard API are installed in `/usr/local/ufdbguard-api`. In this manual, the word `TOPDIR` refers to the top level installation directory for the ufdbGuard API. Make sure that the installation directory has sufficient space for the URL database.

The URL database is downloaded and installed in `/usr/local/ufdbguard-api/blacklists`. The ownership of the files is user `ufdb` so create this user and run the database update script (see section 4.5) always as this user. Create the user `ufdb`:

```
# groupadd ufdb
# useradd -c "URL database owner" -m -g ufdb ufdb
```

4.3 Unpack the Software Tarball

The tar file that contains the ufdbGuard API software suite must be unpacked in the directory `/usr/local`.

Unpack the ufdbGuard tar file in the directory `/usr/local`. Note that when a new version is installed, all previous files all overwritten and it is recommended to save or copy `/usr/local/ufdbguard-api/etc/ufdbguard`. Create the top directory and untar the tarball:

```
# cd /usr/local
# tar xzf .../ufdbGuard-API-1.35.5.tar.gz
# chown -R ufdb:ufdb .
```

The subdirectories `bin`, `etc`, `lib`, `blacklists`, `src` and `include` have now been created.

Special builds, for example the library for the MIPS OCTEON III platform, appear in subdirectories, e.g. `/usr/local/ufdbguard-api/octeon`.

4.4 Compiler and Library Test

To verify that a compatible compiler and all required libraries are present, compile the test applications `apitest` and `urlcats`:

```
$ cd /usr/local/ufdbguard-api/src
$ make
```

The produced files are `apitest` and `urlcats`. The program `urlcats` also exists in the directory `../bin` and is produced here to verify correct compilation and the presence of all necessary build tools and libraries on the system.

The most common error is that the *development* packages for openssl, zlib and bzlib are not installed. For most operating systems, one can find the packages *openssl-devel* and *bzip2-devel* on the installation media. Note that each Linux distribution uses different package names and you may find that openssl-devel has an other similar name like *libssl-dev*. Likewise, the bzip2-devel package may be called *libbz-dev*. Refer to the Operating System manual on how to install additional packages.

Make sure that this test works and ask for assistance from URLfilterDB in case issues cannot be resolved.

4.5 Get Daily Updates

The script `ufdbUpdate` takes care of downloading a new version of the URL database. The script is in `/usr/local/ufdbguard-api/bin`. It is the responsibility of the system integrator to integrate `ufdbUpdate` in its processes such that after `ufdbUpdate` has downloaded a fresh URL database, the application loads the new URL database.

The `ufdbUpdate` script needs the username and password that you received when the (trial) license was received which can be defined in a system configuration file:

```
$ vi /usr/local/ufdbguard-api/etc/ufdbguard
...
DOWNLOAD_USER=lic99999
DOWNLOAD_PASSWORD=aa22bb
```

Users that evaluate the URL database may use the demoXX username and corresponding password.

Test the `ufdbUpdate` script with the verbose option:

```
$ ufdbUpdate -v
```

The output should be similar to:

```
http_proxy is not set: no proxy is used for downloads
Downloading the current database...
<retrieving URL database>
new database downloaded:
-rw-r--r-- 1 root root  5121312  May 5 14:04 /tmp/urlfilterdb-latest.tar.gz
Unpacking the database...
The downloaded database is installed in directory /local/squid/blacklists and its
subdirectories
Sending HUP signal to the ufdbguardd daemon to load new configuration...
URL database creation date:  Fri May  5 13:54:47 CEST 2023
<retrieving license status>
URL database license status: OK
done.
```

4.5.1 Exit Codes of ufdbUpdate

To monitor URL database updates, `ufdbUpdate` has a defined set of exit codes.

code	*explanation*
0	all OK
1	version warning; most likely there is a new version of the API
2	license expiration warning: less than 2 months to renew license
3	license expired: a license renewal is required immediately
11	configuration error
12	temporary file error

21-40	exit code of ufdbUpdate is exit code of wget + 20. `wget` is the command that downloads the new URL database from the servers of URLfilterDB.
41-60	exit code of ufdbUpdate is exit code of gunzip + 40. `gunzip` uncompresses the downloaded URL database. There may be an issue with file system space.
61-80	exit code of ufdbUpdate is exit code of tar + 60. `tar` unpacks the downloaded URL database. There may be an issue with file system space.

In case of an error, it is advised to run `ufdbUpdate -v` from the command line to have more feedback about what is going wrong. License expiration warnings are also issued bu ufdbguardd.

4.5.2 Firewall and Proxy for ufdbUpdate

`ufdbUpdate` downloads the URL database and obviously needs access to the servers of URLfilterDB. Firewall rules may need to be modified to provide access to `updates.urlfilterdb.com`. See section 4.5.3 for the URL that is used to download the URL database.

A proxy can be used to download the URL database: edit `/usr/local/ufdbguard-api/etc/ufdbguard` and assign the appropriate values to the variables `http_proxy`, `PROXY_USER` and `PROXY_PASSWORD`.

4.5.3 Download URL

ufdbUpdate uses a default URL to download the default URL database. The default URL used the HTTPS protocol and has a hostname and a path. The hostname is usually `updates.urlfilterdb.com` and the default path is `/licensed/databases/ufdbguard/3.0/blacklists-latest.tar.gz`.

Since some resellers and system integrators use a non-standard URL database, the download URL may have to be modified. The ufdbUpdate script has 3 variables that are used to compose the download URL and may be modified to download an alternative URL database. The 3 variables and their default values are:

```
URL_DIR="$UPDATE_HOST/licensed/databases"
GUARD_TYPE="ufdbguard/3.0"
DBFILE="blacklists-latest.tar.gz"
```

Note that the `ufdbUpdate` script is generated by `./configure` from `ufdbUpdate.in` so modifications to `ufdbUpdate` will be overwritten when `./configure` is run.

4.6 Test the API

When the test programs are compiled and linked without errors and the URL database has been downloaded, the correct working of the API can be tested by executing `apitest` and `urlcats`.

```
$ cd /usr/local/ufdbguard-api/src
$ make
$ ./apitest
$ ./urlcats -u www.baidu.com
```

5 User-defined URL tables

In cases where additions or exceptions to the categories of URLfilterDB are desired, an administrator can the URL database with user-defined URL categories.

5.1 Creating a URL Table

A common case of a user-defined URL table is where one wants to ensure access to its own websites and websites of 3rd parties that are used for normal activities. To grant users access to the company websites, the URL `yourcompany.com` needs to be added to a whitelist category, for example *alwaysallow*.

Edit the file that contains the extra sites that should always be allowed:

```
$ cd /usr/local/ufdbguard-api
$ vi blacklists/alwaysallow/domains
```

Add the appropriate URLs and always remove a leading `www.`:

```
yourcompany.com
news.google.com
google.com/news
```

In case that you have a file with many URLs having a leading `www.`, you may use the `-W` option to remove the `www.` prefix automatically.

Additional domains can be added according to the local internet usage policy. For example, if news should be blocked but access to CNN allowed, then `cnn.com` should be added also. Alternatively, when news should be blocked but Google news allowed, `news.google.com` and `google.com/news` should be added.

ufdbGuard only uses proprietary database files, so generate an `.ufdb` database file from the ASCII file with `ufdbGenTable`:

```
$ cd /usr/local/ufdbguard-api
$ bin/ufdbGenTable -W -n -t alwaysallow -d blacklists/alwaysallow/domains
```

The above command generates the file `blacklists/alwaysallow/domains.ufdb` and should be invoked each time that the domains file is changed. The `-W` option removes the initial `www.` from all URLs. The `-n` option specifies to not encrypt the table. The `-t` option specifies the name of the URL category.

Do not forget to add the category to the application that uses the API. Hence make sure that the application calls `UFDBloadCategory` and `UFDBverifyUrlandProtocolCategory` for any new category.

5.2 How URLs are matched against the URL database

The ufdbGuard API uses an algorithm to match a URL against the entries in the tables of the URL database. The algorithm uses the following logic.

1. Port numbers and embedded usernames and passwords are ignored. So a URL like john:secret@example.com:8080/foo is simplified to example.com/foo.
2. If a URL table contains an entry with a domainname example.com it matches all URLs that contain example.com including subdomains, and matches URLs like example.com/foo.html, www.example.com and secure.example.com.
3. If a URL table contains an entry with a domainname with a "pipe tag", e.g. |.example.com, it matches all URLs that contain the domain example.com but not subdomains (*****). This entry matches URLs like example.com/foo and www.example.com.

4. If a URL table contains an entry with a domainname and a path, e.g. "example.com/foobar" it matches all URLs that have the domain example.com (but not subdomains) and have a URL path that *starts* with the given URL path, so its matches www.example.com/foobar.html and does *not* match sub.example.com/foobar.
5. If a URL table contains an entry with a domainname, a path and a pipe tag, e.g. example.com/foobar|, it matches all URLs that have the domain example.com (but not subdomains) and have a URL path *equal* to the given path, so it matches www.example.com/foobar and does *not* match www.example.com/foobar.html.
6. If a URL table has an entry with parameters, the URL is matched if it contains all parameters of the table entry *in any order*. For example, if a table contains example.com/watch?p1=foo, the URLs www.example.com/watch?p1=foo and www.example.com/watch?p0=x&p1=foo&p2=bar are matched.

(*) "www" and "www0"..."www99" are not considered subdomains.

6 Anti-phishing from PhishTank

PhishTank (www.phishtank.com) provides a URL list with URLs used for phishing. Starting with version 1.29, the `ufdbUpdate` script downloads an up-to-date version of the URL list from PhishTank. Users of the ufdbGuard API can choose to include this URL list in the access control list.

URLfilterDB and PhishTank are not associated in any way and the URL list of PhishTank is not part of the URL database of URLfilterDB. The URL list of PhishTank is always downloaded independently on the system where ufdbUpdate is executed. Alternatively, to prevent downloads of the URL list of PhishTank, an empty file with the name `.nodownloads` in the directory `.../phishtank` prevents downloads.

Phishtank allows only one download of their URL table per 3 hours and the script that downloads the Phishtank URL table obeys this rule. To deal with more frequent downloads or downloads from multiple servers using the same WAN IP address, one needs to register with Phishtank for an API key and store the API key in a file called `phishtank_api_key` in the directory `.../blacklists/phishtank`. If this file exists, the API key will be used to download the URL table of Phishtank.

With the use of the API key, Phishtank allows more frequent downloading. To obtain an API key, go to www.phishtank.com and register as a user and then register an application. The registered application will have an API key that can be used for downloads of the URL table. We have tested in September 2019 that one could successfully download the URL table 12 times in a time frame of 2 minutes. Note that Phishtank does not have a published rule for how many downloads per a certain time period is allowed and may change the allowed download frequency. Phishtank states on their website that you may contact them if a more frequent download permission is required.

7 Performance Tuning

7.1 Linux 2.6+ performance

There are various sources for Linux system tuning and this section is not a replacement. This section contains recommendation for tuning Linux systems that runs Squid and ufdbguardd only.

7.1.1 Optimize System Memory Usage

Linux has various settings that control allocation of memory and how the system behaves when real memory gets scarce. Below are given the setting that gave satisfactory system behavior on our test systems.

Add to `/etc/sysctl.conf` the following lines:

```
# swappiness can have a value of between 0 and 100
# swappiness=0 tells the kernel to avoid swapping processes out of physical
# memory for as long as possible
# swappiness=100 tells the kernel to aggressively swap processes out of
# physical memory and move them to swap cache
# default: 60
# For an application server we use a less aggressive setting of 15.
vm.swappiness=15

# VFS cache pressure:
# default: 100
# With values lower than 100, the data cache is reduced more and the
# inode cache preserved
vm.vfs_cache_pressure=50
```

Run the following command to make the new configuration active:

```
# sysctl -p /etc/sysctl.conf
```

7.1.2 Bind threads to cores

Total performance is increased by a few percent if threads are each bound to a different CPU core.

7.1.3 Use Hugepages

Total performance is increased by a few percent if the application can use hugepages. The number of *data TLB load misses* is reduced by a factor of 25 when the next two environment variables are set and at least 1.2 GB in hugepages is reserved:

```
LD_PRELOAD=libhugetlbfs.so HUGETLB_MORECORE=yes
```

Since API version 1.35.1 the API has a global flag `madviseHugePages` to use `madvise(2)` to hint Linux to use transparent hugepages. The API uses `madvise` only for tables which occupy at least 0.8 * hugepagesize (= 1.6 MB).

8 URL Categories

The URL database of URLfilterDB uses the following URL categories. Some categories have subcategories. URLs in a subcategory are also in the parent category.

Ads

Websites with advertisements, traffic trackers, user behavior analysis and web page counters.

Parkeddomain

Websites that are parked. Usually parked domains are expired domains or domains for sale and managed by domain brokers. Many parked sites have ads and some domain brokers use ad brokers that redirect users to adult, gambling or scam sites.

P2P

P2P stands for point-to-point file sharing. The P2P category contains websites that can be used directly or indirectly to upload, download and share files. Most P2P sites have copies of movies, adult content, malware, warez and entertainment, and much of this content violates copyright.

Proxies

Sites that can be used to download content of other sites, URL rewriting sites and VPNs. Proxies are commonly used in an attempt to circumvent a URL filter and it is recommended to always block proxies.

Adult

Websites suitable for adults only (not only sexual content).

Malware

Websites that contain or redirect to viruses or malware.
NOTE: this URL category is *not* a replacement for a antivirus tool.

Warez

Websites with illegal software, illegal software codes, hacker's sites, warez and cracks.

Toolbars

Websites for toolbars of browsers. A toolbar is an extension to a web browser that may violate your privacy or make private files public.

Illegal

Websites explaining how to perform Illegal activities.

Arms

Sites with firearms and toys that look like firearms.

Violence

Websites about violent behavior.

Gambling

Websites offering gambling opportunities.

Drugs

Websites about hard drugs.

Webmail

Email accessible with a web browser. Webmail of business sites is not included while webmail of ISPs is included in this category.

Dating

Websites about love, dating, romantic poetry, and friendship.

Chat

Websites to use IRC and chat. Subcategories exist for AIM, Ebuddy, Facebook Chat, Google Talk, MSN Messenger, Oovoo, Skype and Yahoo Chat.

Forum

Websites where people exchange non-business information in a forum.

Private

Blogs and sites of private persons.

Webtv

sites with a audiovisual streams or television-like streams.

Webradio

sites with a music streams or radio-like streams.

Dailymotion

videos of dailymotion

Vimeo

videos of Vimeo

Youtube

videos of Youtube

Audio-Video

Audio and video streams.

Sports

Websites related to sports including sports sections of news sites, fans of sports, sites about actively doing a sport.

Finance

Websites of banks, insurance companies, stock markets and stock brokers.

Jobs

Websites about and for job applications.

Games

Websites to play games and information about gaming.

Entertainment

Entertainment, lifestyle, hobby, arts, museums, fashion, electronic cards, magazines, horoscopes, desktop wallpapers, clip art, photos, portals, events, fan sites, baby-related, child sites, other sites for interest of private persons that are not related to business.

Food

Websites of restaurants and sites with recipes. Fast food chains, however, are part of the category *shops*.

Religion

Websites related to any religion.

Shops

Websites with shops, price comparisons, and auctions aimed at consumers (b2b is excluded).

Travel

Websites about travel agencies, airliners, tourism sites, hotels, holiday resorts.

News

Websites providing news and opinions.

External Applications

Free web-based document editors, spreadsheet applications, desktops, groupware, etc. where "internal" documents can be stored on external servers.

Social Networks

Sites that focuses on building and reflecting of social networks or social relations among people. Subcategories exist for Badoo, Facebook and Twitter.

DNSoverHTTPS

IP addresses and domainnames of services for DNS lookups over HTTPS. This category has also a text file iplist with all IP addresses which can be used to configure a firewall.

Alternate DNS

There is a collection of alternative DNS systems with alternative TLDs like .coin, .libre, .bazar and .geek. See https://www.opennic.org for more information.
The DNS servers use ports 53, 443 and alternate ports like 8443, 5335 and 5353. This category has also a text file iplist with all IP addresses which can be used to configure a firewall.

Dynaddress

Websites with a dynamic IP address.

Extappl

Websites that deal with off-line documents and data.

Education

Websites of schools, universities and educational institutes.

Health

Websites of doctors, clinics, diseases and other health-related sites.

Qmovies

Websites which contain or link to movies with probable copyright infringement.

Searchengine

URLs used by search engines.

Checked

URLs that are verified by URLfilterDB not to be part of any other category. This category contains business sites, governmental sites and useful sites for the general public. This URL category is also used by the ufdbGuard API to track uncategorized URLs and should always be loaded.

The classification rules for URL database are based on *user intent* and classification is from the point of view of a business. So, a website that has a business use, is by default part of the category "checked". For example, access to a website to sell equipment for building constructors is in category "checked" and hence is not part of the category "shops". Also governmental sites, and all sites for basic human needs like electricity and water are in the URL category "checked".

The nature of the content is more important than the strict definition, so an advertisement with a nude person is classified as adult rather than advertisement (although may be included in both categories), and a forum about games is classified as games.

The general impression is also taken into account when a site is categorized. For example, most buyers at `ebay.com` are consumers rather than business users and therefore `ebay.com` is considered a shop for consumers and part of the *shops* category.

URLs may be part of one or more categories, e.g. `www.usatoday.com` is *news* while `www.usatoday.com/sport` is both *news* and *sports*.

9 Privacy Policy

The privacy policy of URLfilterDB is stated on the website: www.urlfilterdb.com/privacystatement.html.

10 More Information

The support desk can answer all questions. Send an email to support@urlfilterdb.com to ask a question or send your feedback.