Reference Manual

ufdbGuard API

version 2.0.3

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++ and any other programming language that is capable of interfacing with 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 the URL database of URLfilterDB, one can use up to three user-defined URL databases, each can contain up to 200 URL categories.

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. The API supports pthreads for multithreading.

On an Intel Gold 6148[1] CPU with 20 cores and 40 threads the URL classifier reaches 80 million classifications per second using all 40 threads and in dual CPU configuration 120 million classifications per second. The high performance library uses a new database format which is optimized to be used on Intel and AMD CPUs with advanced vector (AVX2) SIMD instructions[2]. Performance may vary with CPU frequency, cache architecture and memory bandwidth. Ports to other CPU architectures are feasible.

The URL database is loaded into memory of the application that uses the API. All products of URLfilterDB use an on-site URL database. Updates to the URL database are retrieved on the internet.

The API does not track clients of URLfilterDB nor its end users. See section 3.3 for more information.

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 Compatibility with Previous API Versions

ufdbGuard API v2.x is not compatible with API v1.x. API v2.x uses a new database format and a set of new API functions that combined made it possible to make a performance leap from version 1.x to 2.x. Since the set of API functions is relatively small, porting an existing application from API v1.x to API v2.x should be an effort that usually takes a few days to implement.

Experiences from previous API versions have given valuable insights in how a new API version could improve performance. While on the same test system API v1.x scales well to 8 threads, API v2.x was tested to scale well up to 40 threads and has a performance increase of 104x (compared with v1.35) to 176x (compared with v1.34).

The above also implies that `.ufdb` database files which were created with `ufdbGenTable` of API v1.x must be regenerated using `ufdbapigendb` of API v2.x.

---

[1] The Gold 6148 Skylake CPU is part pf the first generation of Intel Xeon Scalable Processors and was introduced in 2017.

[2] Intel supports AVX2 since 2013 and AMD supports AVX2 since 2015.

1.2 Platforms

The ufdbGuard API can be used on various Linux distros. Currently the supported distros are RHEL 8 and 9 and Ubuntu LTS 20.04, 22.04 and 24.04.

The ufdbGuard API is also available on the Intel DPDK platform and on Marvell's Octeon III CN7xxx CPU.

Other platforms may be supported if a client shows interest.

1.3 Latest Major Changes and Enhancements

The major changes compared with v1.x is a speed improvement. The API supports multithreaded applications. Since the URL database is read-only the API database query internals are lockless.

The category *localnetwork* is new for API v2.x and contains `localhost` and private network subnets. API v2.0 supports IP subnets. For example, the URL database contains an entry '`10.0.0.0/8`' for the category *localnetwork*.

Starting with version 2.0.3 the API library on the x86_64 platform is a shared library.

1.4 Copyright

1.5 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 Linux running on a CPU with Intel/AMD AVX2 SIMD instructions. Intel supports AVX2 since 2013 when it introduced the Haswell micro-architecture and AMD supports AVX2 since 2015.

The ufdbGuard API needs 8 GB disk space and 8 GB memory.

The ufdbGuard API uses a compressed database and requires the compression libraries zlib and zstd.

The ufdbGuard API has helper programs that communicate with the web server `updates.urlfilterdb.com`. The helper programs need libcurl and OpenSSL libraries.

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++ and any programming language that interfaces with a C library.

In addition, the openssl development files, libcurl development files, zlib development files, zstd development files, `tar`, `make` and `install` commands are required which are all included in most UNIX distributions.

2.2 Prerequisites for Production

The helper applications of the API upload and download files and must be able to communicate with the webserver `updates.urlfilterdb.com` using HTTPS on port 443. Firewalls or a proxy must allow access to all IPv4 and IPv6 addresses of this webserver.

Openssl and libcurl shared libraries must be installed for the helper applications of the API. The helper scripts use a shell (`sh`) and common commands like `cat`, `wget`, `curl`, `tar`, `gunzip` etc. The command `logger` is used to send fatal errors to the system log.

Note that filter policies of the application that uses the API can easily be circumvented if on a filtered network DNS over HTTPS or DNS over TLS is allowed. The URL database has a category *dnsoverhttps* to assist with blocking these DNS requests. The URL database also comes with a plaintext file with IP addresses of servers that are used for DNS over HTTPS which can be used to generate firewall rules on filtered networks.

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 4 categories: initialization and termination, classification, management of uncategorised URLs and miscellaneous functions. In the `src` directory is a file called `api2test.c` which serves as an example on how to use all API functions.

The API functions are in a single library `libufdbapi2.so` which is a symbolic link to `libufdbapi-`*`VERSION`*`.so`. The header file for the API is `ufdb-api2.h`.

Almost all API functions return a status. All functions should return `UFDB_API2_OK`. If not, the application must assume something does not work as expected and must act appropriately. See the header file for all possible status values.

3.1 Initialization and Termination

Applications that use the API initialize by calling `UFDBapi2Init`. The application then creates a handle which is used by most API functions. The application continues with specifying which database(s) in which directory to use and by calling `UFDBapi2loadDB` the database(s) are loaded into memory.

Before calling the classification functions, the application calls the function `UFDBapi2getCategoryProperties` to receive a list of all URL categories of all loaded databases.

Other functions may be called during the initialization but these are optional.

3.1.1 UFDBapi2Init

```
int UFDBapi2Init( const char * program, const char * programVersion, const
char * applicationConfigFile )
```

`UFDBapi2Init` initializes internal data structures of the API. It is mandatory to call `UFDBapi2Init` before almost all other API functions are called and before any API parameter is set. `UFDBapi2Init` returns `UFDB_API_OK` or an API error code.

`UFDBapi2Init` also parses the configuration file and extracts the values of the variables `BIN_DIR`, `DATABASE_DIR` and `UPLOAD_DIR`. These variables *must* contain a full path and not use any other variables. The parameter `applicationConfigFile` may be NULL in which case the default file `/etc/ufdbguard-api2.conf` will be parsed.

The application configuration file is usually stored in the directory `/etc/default` or `/etc/sysconfig`. All lines that do not start with `BIN_DIR`, `DATABASE_DIR` or `UPLOAD_DIR` are ignored. The parser removes quotes from the parameter's value.

Applications typically call `UFDBapi2Init`, then optionally set global options and create a handle to use with most API functions. The only function that may be called before `UFDBapi2Init` is `UFDBapi2setDebug`.

On bare metal systems with a custom memory allocator the function `UFDBapi2mallocInit` must be called before `UFDBapi2Init`.

Applications that use DPDK must call `ret_eal_init` before calling `UFDBapi2Init`.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_RANGE`, `UFDB_API2_ERR_FATAL`, `UFDB_API2_ERR_NOFILE`, `UFDB_API2_ERR_NOMEM`.

3.1.2 UFDBapi2createHandle

```
typedef void * UFDBapi2handle;

int UFDBapi2createHandle( int handleOptions, UFDBapi2handle * h )
```

`UFDBapi2createHandle` allocates a private static memory buffer and returns a database handle. The handle is used in calls to many API functions and therefore the handle must be created during initialization of the application.

`handleOptions` is currently not used and must be set to 0.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_FATAL`, `UFDB_API2_ERR_RANGE`.

3.1.3 UFDBapi2destroyHandle

```
int UFDBapi2destroyHandle( UFDBapi2handle h )
```

`UFDBapi2destroyHandle` deallocates the database handle and associated memory that was previously allocated by `UFDBapi2createHandle`.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_INVALID_HANDLE`.

3.1.4 UFDBapi2setDebug

```
int UFDBapi2setDebug( int debugLevel )
```

`UFDBapi2setDebug` sets the debug level of the API. The level should be between 0 and 9. This function may be called at any time, even before `UFDBapi2Init`.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_RANGE`.

3.1.5 UFDBapi2setOptions

```
int UFDBapi2setOptions( long options )
```

`UFDBapi2setOptions` sets a few options that modify the behavior of some functions. The following values must be logically or-ed when used with `UFDBapi2setOptions`:

`UFDB_API2_OPT_USE_URL_PARAMETERS`	parse URL parameters (everything after a `?` in a URL). Default is off.
`UFDB_API2_OPT_AUTO_UPLOAD`	at appropriate times the API will try to upload files with uncategorised URLs by creating a temporary file and executing the `ufdbapiupload` helper program. Default is off.
`UFDB_API2_OPT_AUTO_CREATE_UPLOAD_FILE`	the API will generate files with uncategorised URLs at appropriate times in the upload directory. Default is off.
`UFDB_API2_OPT_USE_HUGEPAGES`	hint Linux to use transparent hugepages for the in-memory database with the `madvise` OS call. Default is on.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_RANGE`.

3.1.6 UFDBapi2setUploadDirectory

```
int UFDBapi2setUploadDirectory( UFDBapi2handle h, char * directory )
```

`UFDBapi2setUploadDirectory` tells the API to store files with uncategorised URL in a non-default directory. The `directory` parameter must be an absolute path.

Note that UFDBapi2Init parses the configuration file and initializes the upload directory with the value of the UPLOAD_DIR variable.

Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE, UFDB_API2_ERR_NOFILE, UFDB_API2_ERR_ERRNO.

3.1.7 UFDBapi2initDBdirectory

```
int UFDBapi2initDBdirectory( UFDBapi2handle h, char * dbDirectory, int flags )
```

UFDBapi2initDBdirectory tells the API to load the URL databases from a specific directory. dbDirectory may be NULL which instructs the API to use the default database directory. If not NULL, the dbDirectory parameter must be an absolute path.

Note that UFDBapi2Init parses the configuration file and sets the default database directory with the value of the DATABASE_DIR variable, so in most cases one can simply use NULL for the dbDirectory parameter.

This function must be called before UFDBapi2loadDB. This function must be called between 1 and 4 times.

flags is currently unused and must be set to 0 for compatibility with future version of the API.

Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE, UFDB_API2_ERR_NOFILE, UFDB_API2_ERR_ERRNO, UFDB_API2_ERR_RANGE.

3.1.8 UFDBapi2initDBaddUserDB

```
int UFDBapi2initDBaddUserDB( UFDBapi2handle h, char * filename, int flags )
```

UFDBapi2initDBaddUserDB tells the API where to find a user-defined URL database. The user-defined URL database can be generated with ufdbapigendb program.

The API supports a maximum of 4 databases; 1 database of URLfilterDB and 3 user-defined databases. Each database can hold up to 200 categories and for highest performance an application should use as few databases as possible. Performance is not reduced when many categories are used in a single database.

flags is currently unused and must be set to 0 for compatibility with future version of the API.

Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE, UFDB_API2_ERR_NOFILE, UFDB_API2_ERR_RANGE.

3.1.9 UFDBapi2loadDB

```
int UFDBapi2loadDB( UFDBapi2handle h )
```

UFDBapi2loadDB instructs the API to load the URL databases into memory. UFDBapi2loadDB always tries to load the URL database provided by URLfilterDB and optionally load up to 3 user-defined URL databases. The functions UFDBapi2initDBdirectory and UFDBapi2initDBaddUserDB must be called before UFDBapi2loadDB.

Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE, UFDB_API2_ERR_NOFILE, UFDB_API2_ERR_RANGE, UFDB_API2_STATUS_DATABASE_OLD, UFDB_API2_STATUS_DATABASE_EXPIRED, UFDB_API2_ERR_NOMEM.

3.1.10 UFDBapi2unloadDB

```
int UFDBapi2unloadDB( UFDBapi2handle h )
```

UFDBapi2unloadDB instructs the API to unload all previously URL databases associated with handle h. At this moment the API will generate a file with uncategorised URLs or upload uncategorised URLs if the options UFDB_API2_AUTO_CREATE_UPLOAD_FILE or UFDB_API2_AUTO_UPLOAD have been set with UFDBapi2setOptions. All memory resources associated with the URL databases are released

and handle h cannot be used any more to classify URLs. The handle h remains a valid handle that can be reused or destroyed.
Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE.

3.1.11 UFDBapi2getCategoryProperties

```
typedef struct UFDBapi2categoryProperties
{
   uint16_t     id;
   uint8_t      filtered;
   const char * categoryName;
   void *       userData;     // optionally bind application data to category
} UFDBapi2categoryData;
int UFDBapi2getCategoryProperties( UFDBapi2handle h,
UFDBapi2categoryProperties ** data, int * numCategories )
```

UFDBapi2getCategoryProperties queries all URL categories of all URL databases associated with handle h. The results are stored in UFDBapi2categoryProperties which is a read-only static data structure. So each category of all (1-4) URL databases gets a unique integer ID, a category name (string) and user-settable data userData.

The userData enables application to retrieve category-specific data with each URL classification query and eliminates the need to map a category of the database(s) to an internal data structure of an application. Applications that wish to receive category-specific userData first need to call UFDBapi2getCategoryProperties (usually just after UFDBapi2loadDB) and then set userData with UFDBapi2setCategoryProperties. After this, all successive calls to UFDBapi2lookupDomain and UFDBapi2lookupURL return the category-specific userData as part of the classification results.
Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE.

3.1.12 UFDBapi2setCategoryProperties

```
int UFDBapi2setCategoryProperties( UFDBapi2handle h,
UFDBapi2categoryProperties * data, int numCategories )
```

If the application likes to bind its own data with the categories of the URL database, it can call UFDBapi2setCategoryProperties to set userData for all or a selected set of categories. The userData that is set by UFDBapi2setCategoryProperties is returned to the application in UFDBapi2lookupResult by the lookup functions UFDBapi2lookupURL and UFDBapi2lookupDomain. The application is responsible for freeing memory that userData points to when it destroys a handle or unloads a database.

The filtered field can be set to any non-zero value to remove this category from all query results. The most common usage for this feature is to filter out subcategories.

The categoryName field is ignored by UFDBapi2setCategoryProperties.

UFDBapi2setCategoryProperties must be used to effectively set the properties userData and filtered since just changing userData or filtered in the result of UFDBapi2getCategoryProperties has no effect.
Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE, UFDB_API2_ERR_RANGE.

3.1.13 UFDBapi2createThreadData

```
typedef void * UFDBapi2ThreadData;

int UFDBapi2createThreadData( UFDBapi2ThreadData * tData )
```

Each program thread needs a memory area where various thread-specific data structures are stored. UFDBapi2createThreadData creates such thread data and returns it in tData. tData is required for functions like UFDBapi2lookupURL and UFDBapi2lookupDomain.
Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_ERR_RANGE.

3.1.14 UFDBapi2destroyThreadData

```
int UFDBapi2createThreadData( UFDBapi2ThreadData tData )
```

When a thread is terminated or will not use the API any more, it must call UFDBapi2createThreadData to release the memory occupied by tData.

Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE.

3.2 Classification

A URL classification is done with either one of two functions. These functions return the URL classification in UFDBapi2lookupResult.

```
#define UFDB_API2_MAX_RESULT_CATS 16
typedef struct UFDBapi2lookupResult
{
   int       numCategories;
   uint16_t  categories[UFDB_API2_MAX_RESULT_CATS];
   void *    userData[UFDB_API2_MAX_RESULT_CATS];
} UFDBapi2lookupResult;
```

The categories field holds small integers that are identifiers which are the same as the categoryId in UFDBapi2categoryData that was produced by UFDBapi2getCategoryProperties.

The userData field holds a pointer to application-specific data that was set with a call to the function UFDBapi2setCategoryProperties.

3.2.1 UFDBapi2lookupDomain

```
int UFDBapi2lookupDomain( UFDBapi2handle h, UFDBapi2ThreadData tData, const
char * domain, int port, UFDBapi2lookupResult * result )
```

UFDBapi2lookupDomain queries the URL database(s) associated with handle h. The parameter domain must point to a valid domainname with maximum 256 bytes (including terminating \0). If the used port number is not known, the application must use 80. The result is stored in UFDBapi2lookupResult where each userData is either NULL or the value that the application set earlier with UFDBapi2setCategoryUserData.

Return values: UFDB_API2_OK, UFDB_API2_ERR_BAD_URL, UFDB_API2_ERR_NULL, UFDB_API2_ERR_INVALID_HANDLE.

3.2.2 UFDBapi2lookupURL

```
UFDBapi2lookupURL( UFDBapi2handle h, UFDBapi2ThreadData tData, const
char * URL, int port, UFDBapi2lookupResult * result );
```

UFDBapi2lookupURL queries the URL database(s) associated with handle h. The parameter URL must point to a valid URL with maximum 8192 bytes (including terminating \0). If the used port number is not known, the application must use 80. The result is stored in UFDBapi2lookupResult where each userData is either NULL or the value that the application set earlier with UFDBapi2setCategoryUserData.

The parameter URL is parsed and the domainname, port, URL path and parameters are extracted from it. Parsing also includes conversion of %-encoded characters and parsing parameters is relatively expensive. Parsing a URL takes twice as much CPU time as the actual lookup of (domain,path,parameters) in the

URL database. If possible, it is suggested not to use the option `UFDB_API2_OPT_USE_URL_-PARAMETERS`.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_BAD_URL`, `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_INVALID_HANDLE`.

3.3 Management of Uncategorised URLs

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

➡ Unless agreed otherwise with a signed contract, *each application that uses the API must ensure prompt and regular upload of the uncategorised URLs*. There are 3 methods to do this: application controlled upload, a direct upload and an indirect upload. With the application controlled upload the application makes an API call to generate the file and the application takes care of executing the `ufdbapiupload` helper program to perform the actual upload. With the direct upload the API is instructed to automatically generate files with uncategorised URLs and to execute the program `ufdbapiupload` to perform the actual upload of the file. With the indirect upload the API is instructed to automatically generate files with uncategorised URLs in the upload directory and then it is the responsibility of the application or a job scheduler to upload these files with the help of the `ufdbapiupload` program.

The direct and indirect methods to upload uncategorised URLs is implemented by using option `UFDB_API2_OPT_AUTO_UPLOAD` or `UFDB_API2_OPT_AUTO_CREATE_UPLOAD_FILE` set by `UFDBapi2setOptions`. Note that if the flag `UFDB_API2_OPT_AUTO_CREATE_UPLOAD_FILE` is used, the uncategorised URLs are not (yet) uploaded and a scheduled job must upload the generated files at regular intervals and appropriate cron jobs or alternative scheduler jobs must exist.

The API does not track clients of URLfilterDB nor end users. Because of the transparency policy of URLfilterDB, the uploaded files with uncategorised URLs are human-readable to prove that users and clients are not tracked. The uploaded files only contain uncategorised URLs, statistical counters and some system details. Applications control the generation and upload of the files that are uploaded to webservers of URLfilterDB.

Applications may have application-specific counters or application-specific message that needs to be included in the uploaded files. For this purpose the API functions x and y can be used.

3.3.1 UFDBapi2uploadUncategorisedURLs

```
int UFDBapi2uploadUncategorisedURLs( UFDBapi2handle h, const char * agent )
```

`UFDBapi2uploadUncategorisedURLs` instruct the API to perform the direct upload, i.e. produce a temporary file and execute the `ufdbapiupload` helper program to perform the actual upload.

This function uses OS calls `fork` and `execv` followed by `waitpid` to execute the helper program.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_ERRNO`, `UFDB_API2_ERR_FATAL`, `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_INVALID_HANDLE`.

3.3.2 UFDBapi2storeUncategorisedURLs

```
int UFDBapi2storeUncategorisedURLs( UFDBapi2handle h, const char * agent,
const char * filename )
```

`UFDBapi2storeUncategorisedURLs` instruct the API to initiate the indirect upload and to generate a file with uncategorised URLs. The filename may be an absolute path, or any other filename that will be created in the upload directory, or may be `NULL` to instruct the API to generate a filename in the upload directory by itself.

It is the responsibility of the application or a job scheduler to upload these files with the help of the `ufdbapiupload` program.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_ERRNO`, `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_INVALID_HANDLE`.

3.3.3 UFDBapi2setApplicationCounters

```
int UFDBapi2setApplicationCounters(
   UFDBapi2handle h,
   unsigned long  num_errors,
   unsigned long  num_fatal_errors,
   unsigned long  num_blocked,
   unsigned long  num_nxdomain,
   unsigned long  num_cname,
   unsigned long  num_views   )
```

`UFDBapi2setApplicationCounters` may be used to include application-specific counters in future upload files. Applications should use this function at most one time just before an upload file is created.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_INVALID_HANDLE`.

3.3.4 UFDBapi2setUploadMessage

```
int UFDBapi2setUploadMessage( UFDBapi2handle h, const char message[256] )
```

`UFDBapi2setUploadMessage` may be used to include application-specific data to be included in the upload files. The message data must be maximum 256 bytes (including terminating \0) representing ASCII or UTF-8 characters and will be stripped from newlines before it is included in future upload files as an HTTP header field `X-ApplicationMessage`.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_INVALID_HANDLE`.

3.4 Miscellaneous Functions

3.4.1 UFDBapi2dbTimestamp

```
int UFDBapi2dbTimestamp( UFDBapi2handle h, char timestamp[32] )
```

`UFDBapi2dbTimestamp` returns the date and time of the creation of the URL database.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_NOFILE` `UFDB_API2_ERR_NULL`, `UFDB_API2_ERR_INVALID_HANDLE`.

3.4.2 UFDBapi2errorString

```
const char * UFDBapi2errorString( int status )
```

`UFDBapi2errorString` returns a short explanatory string for a status that other API functions return.

Currently the API uses the following status codes.

`UFDB_API2_OK`
`UFDB_API2_ERR_BAD_URL`
`UFDB_API2_ERR_NULL`
`UFDB_API2_ERR_NOFILE`
`UFDB_API2_ERR_READ`
`UFDB_API2_ERR_EXPR`
`UFDB_API2_ERR_RANGE`
`UFDB_API2_ERR_ERRNO`

```
UFDB_API2_ERR_NOMEM
UFDB_API2_ERR_IP_ADDRESS
UFDB_API2_ERR_OLD_TABLE
UFDB_API2_ERR_OUTDATED
UFDB_API2_ERR_INVALID_TABLE
UFDB_API2_ERR_INVALID_KEY
UFDB_API2_ERR_FULL
UFDB_API2_ERR_CKSUM_NOT_VALID
UFDB_API2_ERR_FATAL
UFDB_API2_ERR_DOES_NOT_RESOLVE
UFDB_API2_ERR_BAD_URL
```

3.4.3 UFDBapi2licenseStatus

```
int UFDBapi2licenseStatus( UFDBapi2handle h, const char ** licenseStatus )
```

`UFDBapi2licenseStatus` returns the license status as the return value and a copy of the content of the file <databasedir>/license-status in `licenseStatus`. The license-status file is updated each time that the URL database is downloaded with `ufdbapiupdatedb`. The license status inside the API is updated each time `UFDBapi2loadDB` is called. A license warning is returned when the license will expire in less than 60 days.

Return values: `UFDB_API2_OK`, `UFDB_API2_ERR_NOFILE`, `UFDB_API2_LICENSE_WARNING`, `UFDB_API2_LICENSE_EXPIRED`, `UFDB_API2_ERR_INVALID_HANDLE`, `UFDB_API2_ERR_NULL`.

3.4.4 UFDBapi2getCounters

```
int UFDBapi2getCounters(
   UFDBapi2handle   h,
   unsigned long *  lookups,
   unsigned long *  matches,
   unsigned long *  https,
   unsigned long *  uncategorised,
   unsigned long *  bad_url  )
```

The API has various counters which can be retrieved by calling `UFDBapi2getCounters`. All parameters contain metrics and must be allocated by the caller. `TPS15seconds` contains the peak number of queries per second (averaged over 15 seconds).

`lookups` has the number of calls to `UFDBapi2lookupURL` and `UFDBapi2lookupDomain` with non-NULL parameters. `matches` has the total number of results categories for all queries URLs. `https` has the total number of queries that had port 443 or had protocol `https` in the URL. `uncategorised` has the number of URLs and domains that were not found in any category. `bad_url` has the number of URLs and domains that were found too long or the API could not parse.

Counters are associated to an API handle `h` and are reset to zero when a file with uncategorised URLs – that also includes the counters – is created.

A file with uncategorised URLs is created

- at the request of the application when it calls `UFDBapi2uploadUncategorisedURLs` or `UFDBapi2storeUncategorisedURLs`, or
- when a handle `h` is destroyed by `UFDBapi2destroyHandle` and there are one or more uncategorised URLs, or

- when the application set one of the options UFDB_API2_OPT_AUTO_UPLOAD or UFDB_API2_OPT_AUTO_CREATE_UPLOAD_FILE and the API encountered an event where it needed to write the file.

To obtain meaningful counter values, an application should use UFDBapi2getCounters just before it destroys a handle and just before it calls UFDBapi2uploadUncategorisedURLs or UFDBapi2storeUncategorisedURLs. Applications can also call UFDBapi2getCounters in between, for example every hour.

Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL, UFDB_API2_INVALID_HANDLE.

3.4.5 UFDBapi2version

```
int UFDBapi2version( const char ** version )
```

UFDBapi2version returns the version of the API in version.

Return values: UFDB_API2_OK, UFDB_API2_ERR_NULL.

3.4.6 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, ... )
```

In case that the API needs debugging, one can set the global debug level to a value between 1 and 9 which results in the API calling ufdbLogMessage with debug information.

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.

4 Implementation

4.1 Database Refresh

The simple naive implementation to refresh the URL database is to unload the current database(s), and then load the new database(s). This approach has a short time interval where no database is in memory and hence no URL classifications can occur.

The way that in-memory database(s) are associated with a handle makes it possible to implement a lockless database refresh by loading the new URL database(s) and then start using the new handle for the new database(s). Wait a generous amount of time that guarantees that no single thread is querying the old database(s) associated with the old handle, and then destroy the old handle and unload the old database(s). This way applications can query the database(s) without interruption using a single global handle.

The lockless refresh has the following simplified pseudo code.

```
result = UFDBapi2createHandle( flags, &newHandle )
// set options, directories etc. for newHandle
result = UFDBapi2setCategoryUserData( newHandle, ... )
result = UFDBapi2loadDB( newHandle )
oldHandle = globalHandle
globalHandle = newHandle
```

```
// auto-upload flag is set earlier or call UFDBapi2storeUncategorisedURLs( oldHandle, ...)
sleep( 20 )
result = UFDBapi2unloadDB( oldHandle )
```

Considering that a single thread can do millions of URL classifications per second, the `sleep(20)` is a very generous time to wait.

Warning: there must be more available hardware threads than software threads that use `UFDBapi2lookupURL` to be absolutely sure that after 20 seconds not a single thread will use the old global handle while inside `UFDBapi2lookupURL`. This only happens if the OS schedules other threads while one or more threads are in `UFDBapi2lookupURL` and does not get rescheduled for 20 seconds, i.e. the system is severely overloaded.

An alternative hybrid approach is to load the new database(s) using a new handle and the application organizes a synchronization between the database refresh thread and all classification threads, then does the swap of `globalHandle`. and then unloads the old database(s).

4.2 Examples

See the source file `api2test.c` for a fully functional test program to test predefined URLs or URLs from a specified file.

`api2test.c` can be downloaded with this link:
https://files.urlfilterdb.com/examples/api2test.c.

4.3 Using the ufdbGuard API Libraries

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

C and C++ source code must include `ufdb-api2.h`. and use compiler flags to search the directory `/opt/urlfilterdb/ufdbguard-api2/include` for include files.

3rd party executables that include the ufdbGuard API must be linked against the API library, so the correct linker flags to search for libraries in `/opt/urlfilterdb/ufdbguard-api2/lib` and include the library `ufdbapi2` must be used.

5 Software Installation

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

5.1 Upgrading from a Previous Version

ufdbGuard API v2.x is incompatible with API v1.x. The files of API v2.0 are installed in different directories that the files of API v1.x so API v2.0 and API v1.x can be installed side by side.

The file `/opt/urlfilterdb/ufdbguard-api2/etc/CHANGELOG` contains a short description of all changes between releases.

Starting with release v2.0.3 the API is shipped as a shared library. The path to shared libraries is managed with `ldconfig`. The system administrator must include `/opt/urlfilterdb/ufdbguard-api2/lib` in one of the configuration files of `ldconfig`. It is suggested to add the directory of the shared library to a new file `/etc/ld.so.conf.d/ufdbguard-api2.conf`.

5.2 Installation Directory and Ownership

The files of the ufdbGuard API are installed in `/opt/urlfilterdb/ufdbguard-api2`. 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 in the directory `/opt/urlfilterdb/ufdbguard-api2/urldatabase`.

5.3 Unpack the Software Tarball

The tar file that contains the ufdbGuard API software installs files in the directory `/opt/urlfilterdb/ufdbguard-api2` and subdirectories.

Unpack the ufdbGuard tar file in the directory `/`. Note that when a new version is installed, all previous files all overwritten and it is recommended to save or copy files in `/opt/urlfilterdb/ufdbguard-api2/etc`. Untar the tarball:

```
# cd /
# tar xzf .../ufdbGuard-API-2.0.3.tar.gz
# chown -R bin:bin /opt/urlfilterdb/ufdbguard-api2
```

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

The API is shipped as a shared library. The path to shared libraries is managed with `ldconfig`. The system administrator must include `/opt/urlfilterdb/ufdbguard-api2/lib` in one of the configuration files of `ldconfig`. It is suggested to add the directory of the shared library to a new file `/etc/ld.so.conf.d/ufdbguard-api2.conf`.

Special builds, for example the library for the MIPS OCTEON III platform, appear in subdirectories, e.g. `/opt/urlfilterdb/ufdbguard-api2/octeon`.

5.4 Compiler and Library Test

To verify that a compatible compiler and all required libraries are present, compile the test application `api2test`:

```
$ cd /opt/urlfilterdb/ufdbguard-api2/src
```

```
$ make
```

The produced file is `api2test`.

The most common error is that the *development* packages for openssl, zlib and zstd are not installed. For most operating systems, one can find the packages *openssl-devel* and zstd-*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.

5.5 API Configuration File

The configuration file for the API and the helper programs is `/etc/ufdbguard-api2.conf`. A template for the configuration file is in `/opt/urlfilterdb/ufdbguard-api2/etc`. The configuration file must be installed on each system that uses the ufdbGuard API.

The template assumes that all used files are in `/opt/urlfilterdb/ufdbguard-api2` and its subdirectories and this may be changed to any path by changing the variables in `/etc/ufdbguard-api2.conf`.

5.6 API Helpers

`ufdbapi2download` is a program that opens an HTTPS socket with `updates.urlfilterdb.com` and downloads URL database updates and the `license-status` file.

`ufdbapiupdatedb` is a shell script that executes the `ufdbapi2download` helper program.

`ufdbapi2uploadurls` is a helper program that opens an HTTPS socket with `updates.urlfilterdb.com` and uploads files with uncategorised URLs and statistics.

`ufdbapiupload` is a shell script that uses `curl` to perform the actual upload.

See sections 5.9 and 5.10 for more details about the helper programs.

5.7 Username and Password

The helper programs use a username and password when they connect to `updates.urlfilterdb.com`. The username and password are provided by URLfilterDB. The password is encrypted and converted to a base64-coded string and looks like “e:OGzyF8zNbryTKg==”.

The username and password must be assigned to the variables `DOWNLOAD_USER` and `DOWNLOAD_PASSWORD` in the configuration file. Alternatively, the password may be stored in a separate file and the file name assigned to the variable `PASSWORD_FILE`.

5.8 Optional Client ID

Resellers that use a single generic username for the license on its client systems must include a unique client identifier in the file `/etc/ufdbguard.clientid`. The first line of the file must contain the unique identifier without quotes or trailing spaces.

5.9 Get Daily Updates

The script `ufdbapiupdatedb` takes care of downloading a new version of the URL database. The script is in `/opt/urlfilterdb/ufdbguard-api2/bin`. It is the responsibility of the system

integrator to integrate `ufdbapiupdatedb` in its processes such that after `ufdbapiupdatedb` has downloaded a fresh URL database, the application loads the new URL database.

The `ufdbapiupdatedb` 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 /opt/urlfilterdb/ufdbguard-api2/etc/ufdbguard
...
DOWNLOAD_USER="lic99999"
DOWNLOAD_PASSWORD="e:OGzyF8zNbryTKg=="
```

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

Test the `ufdbapiupdatedb` script with the verbose option:

```
$ ufdbapiupdatedb -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 .../urldatabase-latest.tar.gz
Unpacking the database...
The downloaded database is installed in directory
/opt/urlfilterdb/ufdbguard-api2/urlfilterdb and its subdirectories
Sending HUP signal to the ufdbguardd daemon to load new configuration...
URL database creation date:  Fri Oct 18 13:54:47 CEST 2024
<retrieving license status>
URL database license status: OK
done.
```

5.9.1 Exit Codes of ufdbapiupdatedb

To monitor URL database updates, `ufdbapiupdatedb` 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 `ufdbapiupdatedb` is exit code of `ufdbapi2download` + 20. `ufdbapi2download` is the helper program that downloads the new URL database from the servers of URLfilterDB.
41-60	exit code of `ufdbapiupdatedb` 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 `ufdbapiupdatedb` 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 `ufdbapiupdatedb -v` from the command line to have more feedback about what is going wrong. License expiration warnings are also issued bu ufdbguardd.

5.9.2 Firewall and Proxy for ufdbapiupdatedb

`ufdbapiupdatedb` 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 5.9.3 for the URL that is used to download the URL database.

A proxy can be used to download the URL database: edit `/opt/urlfilterdb/ufdbguard-api2/etc/ufdbguard-api2.conf` and assign the appropriate values to the variables `https_proxy`, `PROXY_USER` and `PROXY_PASSWORD`.

5.9.3 Download URL Database

`ufdbapiupdatedb` is a script that does some basic checks and then executes the helper program `ufdbapi2download` that does the actual download from the website `updates.urlfilterdb.com` using HTTPS on port 443.

5.10 Upload Uncategorised URLs

`ufdbapiupload` is a shell script that does some basic checks and executes `curl` for the actual upload.

`curl` is a program that creates an HTTPS connection to `updates.urlfilterdb.com` and uploads all files with uncategorised URLs that it can find in the upload directory. After the upload, the files with uncategorised URLs are either removed or renamed. By default files are removed but if it is desired to keep the renamed and uploaded files a configuration `/etc/ufdbguard-api2.conf` can be used to rename the files. To rename the files one can set the variable `RENAME_UPLOADED_FILES` to `yes`. If set to `yes`, the renamed files have the prefix '`uploaded.`'.

Uncategorised URLs are uploaded to servers of URLfilterDB to be analyzed. Files with uncategorised URLs are made at the request of the application that uses the ufdbGuard API. So it is the responsibility of the programmer and application manager to make sure that the files can be created and will be uploaded. See section 3.3 for more information.

Unless agreed otherwise with a signed contract, *each application that uses the API must ensure prompt and regular upload of the uncategorised URLs*. See section 3.3 for more information.

5.11 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 /opt/urlfilterdb/ufdbguard-api2/src
$ make
$ ./api2test
```

6 Installation on a Production System

On a production system only a few files are required. Besides the application that is linked with the static ufdbGuard API library, one configuration file and a handful of helper programs are installed and directories for database files and upload files need to be created.

Scheduling a database refresh and upload of uncategorised URLs is required but not further explained here.

6.1 Production Configuration File

The configuration file is `/etc/ufdbguard-api2.conf`. Helper scripts and programs use this file and it must be installed. A template for this file is provided; see section 5.5 for more information.

At least the variables `DOWNLOAD_USER` and `DOWNLOAD_PASSWORD` must be assigned in the configuration file.

The template file has a comment for each variable that can be set and one can change the database directory, directory for helper programs, web proxy parameters and more. The template file is in `/opt/urlfilterdb/ufdbguard-api2/etc/ufdbguard2-api2.conf` and contains the variables `DOWNLOAD_USER`. `DOWNLOAD_PASSWORD`, `PASSWORD_FILE`, `DATABASE_DIR`, `UPLOAD_DIR`, `BIN_DIR` and a few other variables that are optional. The variables ending with `_DIR` have default values that start with `/opt/urlfilterdb/ufdbguard-api2/` and may be changed if desired – or course the contents of these directories must move to the appropriate directories.

6.2 Helper Programs

The API helper programs are scripts and executables that download the URL database and upload uncategorised URLs. The directory for the helper programs is set in the variable `BIN_DIR` in the configuration file. All helper programs must be installed in the same directory. The helper programs are: `ufdbapigendb`, `ufdbapiupdatedb`, `ufdbapiupload`.

6.3 Directories in Production

The following directories are required on production systems. The directories are referred to with the corresponding variable in the configuration file.

`BIN_DIR`	helper programs
`DATABASE_DIR`	URL database(s) and license status file
`UPLOAD_DIR`	files with uncategorised URLs

7 User-defined URL Databases

The URL database of URLfilterDB is a read-only database and it is not possible to modify it in any way. However, in cases where additions or exceptions to the categories of URLfilterDB are desired, an administrator can create one user-defined URL database with user-defined URL categories. A user-defined databases can have a maximum of 200 categories.

7.1 Creating a URL Database

A databases consists of one or more categories where all data of all categories is merged into a single database file with a `.ufdb` suffix. To create a database one needs to create a directory hierarchy where under the top directory subdirectories exist for each user-defined category. Inside each category directory it is mandatory to have a `domains` file with at least one domain, and optional `urls` file where each line has a full URL, and an optional `ips` file where each line has an IPv4 or IPv6 address or subnet.

A common case of a user-defined URL category 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 `company.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. For example:

```
$ cd /opt/urlfilterdb/clientdb1
$ vi alwaysallow/domains
```

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

```
company.com
api.partner.net
payments.bank.com
```

Note that `ufdbapigendb` will always remove any `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 files with `ufdbapigendb`. The `ufdbapigendb` command accepts various command line flags:

-T topdir	topdir is the top of the directory tree that contains the user-defined database
-c category-dir	category-dir is the name of the subdirectory that contains a URL category. The `-c` option may be repeated 200 times.
-o filename	specify the output filename
-q	be quiet – suppress many warnings about URLs
-z	use zstd compression. May be repeated 3 times to increase compression level.
-Z	use zlib compression
-D	debug mode

```
$ cd /opt/urlfilterdb/ufdbguard-api2/urldatabase
$ ufdbapigendb -o clientdb1 -T /opt/urlfilterdb/clientdb1 -c alwaysallow
```

The above command generates a database file with one category in the file `/opt/urlfilterdb/ufdbguard-api2/urldatabase/clientdb1.ufdb` and this command must be repeated each time that the domains file is changed.

Do not forget to manage the categories of the user-defined database(s) in the application that uses the API, i.e. the application should load the user-defined database(s).

7.2 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.

8 SafeSearch

The API itself does not enforce 'safe' searching on search engines. This section describes how safe searches can be enforced on popular search engines and Youtube.

In the following subsections a CNAME DNS record can be configured to enforce safesearch for some search engines. This is usually done with the Response Policy Zones (RPZ) feature of the DNS server.

Note that DNS RPZ policies and filter policies of the application that uses the API can be circumvented if on the filtered networks DNS over HTTPS or DNS over TLS is allowed.

8.1 SafeSearch of Google

Google offers an option 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, www.google.com.br etc.

8.2 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.

8.3 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 and one can configure the DNS server to have a CNAME record entry for www.bing.com pointing to strict.bing.com.

8.4 SafeSearch on Duckduckgo

Duckduckgo also offers safesearch enforcement which is explained on their website: https://duckduckgo.com/duckduckgo-help-pages/features/safe-search/ and supports a DNS-based filter with a CNAME record entry for duckduckgo.com pointing to safe.duckduckgo.com.

8.5 SafeSearch on Ecosia

Ecosia also has started to offer safesearch enforcement via DNS which is explained on their website: https://ecosia.helpscoutdocs.com/article/562-how-to-enforce-safe-search-at-your-organization and supports a CNAME record entry for www.ecosia.org to strict-safe-search.ecosia.org.

8.6 SafeSearch on Qwant

Qwant has two options to enforce safesearch: redirecting www.qwant.com with a CNAME record to www.qwantjunior.com and redirecting api.qwant.com to safeapi.qwant.com.

8.7 Safe Search Engines

There is a number of search engines that always filter adult content and lack an option to disable the filter. The probably incomplete list of search engines that always filter content is:
blinde-kuh.de
favoes.com
fragfinn.de
helles-koepfchen.de
kiddle.co
kidzsearch.com
searchmixer.com
vinden.nl
qwantjunior.com
yougl.de

There are other search engines[3] that try to filter all adult content but fail, e.g. they show adult content for the search term “cougar stepson” or “hot cougar”.

[3] findarios.com nakoona.com such.de woooom.com xuve.com

9 Performance Tuning

9.1 Bind threads to cores

Binding programming threads to CPU threads/cores increases CPU L1/L2 cache efficiency and total performance is increased by a few percent.

9.2 Use Hugepages

Total performance is increased by a few percent if the application can use hugepages. The number of *data TLB load misses* can be reduced significantly by using hugepages (2MB) instead of regular pages (4K). CPUs have an average of 1500+ TLB entries for virtual memory mapping and can address 1500 * 2 MB = 3 GB without TLB misses. For each TLB miss the CPU causes a fault where the Linux kernel has to fill the TLB with a new entry which is a relatively time consuming operation that should be avoided if possible. Considering that the URL database of API v2.x currently fits entirely in 3 GB and that the API can hint the Linux kernel to use transparent hugepages (THP) with the `madvise` OS call, optimal performance is possible.

The Linux kernel THP policy must be set for `madvise` to have any effect. The special file /sys/kernel/mm/transparent_hugepage/defrag is used to set the Linux kernel THP policy and it is suggested to set it to madvise or defer+madvise.

10 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.

AI Chat

Websites where people can chat with an AI bot. Chat bots for education, business and customer support etc. are not included. Websites with *AI friends* and *unrestricted chat* are in the adult category.

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, fintech, crypto and insurance companies.

Trading

Websites about stock markets and trading systems as well as websites related to investments.

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.
Note that the use of DNS over HTTPS/TLS is an effective way of circumventing any internet filter.

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.

Localnetwork

The category localnetwork is new for API v2.x and contains `localhost` and private network subnets.

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*.

11 Privacy Policy

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

12 More Information

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