Retrace server

1.8, 21 February 2013

Karel Klic (kklic@redhat.com), **Michal Toman** (mtoman@redhat.com)

Table of Contents

1 Overview

1.1 Binary crashes

Analyzing a program crash from a coredump is a difficult task. The GNU Debugger (GDB), that is commonly used to analyze coredumps on free operating systems, expects that the system analyzing the coredump is identical to the system where the program crashed. Software updates often break this assumption even on the system where the crash occured, making the coredump analyzable only with significant effort. Furthermore, older versions of software packages are often removed from software repositories, including the packages with debugging symbols, so the package with debugging symbols is often not available when user needs to install it for coredump analysis. Packages with the debugging symbols are large, requiring a lot of free space and causing problems with downloading them via unreliable internet connection.

Retrace server solves these problems for Fedora 15+ and RHEL 6+ operating systems, and allows developers to analyze coredumps without having access to the machine where the crash occurred.

Retrace server is usually run as a service on a local network, or on Internet. A user sends a coredump together with some additional information to a retrace server. The server reads the coredump and depending on its contents it installs necessary software dependencies to create a software environment which is, from the GDB point of view, identical to the environment where the crash happened. Then the server runs GDB to generate a backtrace from the coredump and provides it back to the user.

Coredumps generated on i386 and x86_64 architectures are supported within a single x86_64 retrace server instance.

1.2 Kernel crashes

Analyzing kernel crashes is a little easier. If `kdump` service is configured properly, a vmcore (kernel coredump) is generated on kernel panic. This can be debugged with the `crash` tool, which uses an embedded GDB and only requires the vmcore itself and the appropriate '`vmlinux`' (found in the kernel-debuginfo package).

Retracing a vmcore only means extracting the kernel log from the vmcore. The oops message (backtrace) is generated automatically by the kernel and is included in the kernel log saved by `kdump` service.

There is one difference between retracing binaries and kernel: the kernel can only be retraced on a machine of exactly the same architecture as the vmcore was generated on. This makes difficult to retrace i386 vmcores on x86_64 host. However using a chroot with i386 packages installed fixes the problem.

1.3 Design

The retrace server consists of the following major parts:

1. an HTTP interface, consisting of a set of scripts handling communication with clients
2. a retrace worker, doing the coredump/vmcore processing, environment preparation, and running the debugger to generate a backtrace

3. a cleanup script, handling stalled retracing tasks and removing old data
4. a package repository, providing the application binaries, libraries, and debuginfo necessary for generating backtraces from coredumps

2 HTTP interface

The client-server communication proceeds as follows:

1. Client uploads a coredump/vmcore to a retrace server. Retrace server creates a task and sends the task ID and task password in response to the client.
2. Client asks server for the task status using the task ID and password. Server responds with the status information (task finished successfully, task failed, task is still running).
3. Client asks server for the backtrace from a successfully finished task using the task ID and password. Server sends the backtrace in response.
4. Client asks server for a log from the finished task using the task ID and password, and server sends the log in response.

The HTTP interface application is a set of scripts written in Python, using the Python Web Server Gateway Interface (WSGI) to interact with a web server. The only supported and tested configuration is the Apache HTTPD Server with mod_wsgi.

Only secure (HTTPS) communication is allowed for communicating with a public instance of retrace server, because coredumps and backtraces are private data. Users may decide to publish their backtraces in a bug tracker after reviewing them, but the retrace server doesn't do that. The server is supposed to use HTTP persistent connections to to avoid frequent SSL renegotiations.

HTTPS can be disabled on a trusted local network to improve performance, however it is highly recommended not to do this anyway.

2.1 Creating a new task

A client might create a new task by sending a HTTP request to the `https://server/create` URL, and providing an archive as the request content. The archive contains crash data files. The crash data files are a subset of some local '`/var/spool/abrt/ccpp-time-pid`' directory contents, so the client must only pack and upload them.

The server supports uncompressed tar archives, and tar archives compressed with gzip and xz. Uncompressed archives are the most efficient way for local network delivery, and gzip can be used there as well because of its good compression speed.

The xz compression file format is well suited for public server setup (slow network), as it provides good compression ratio, which is important for compressing large coredumps, and it provides reasonable compress/decompress speed and memory consumption. See Chapter 6 [Traffic and load estimation], page 11 for the measurements. The XZ Utils implementation with the compression level 2 is used to compress the data.

The HTTP request for a new task must use the POST method. It must contain a proper *Content-Length* and *Content-Type* fields. If the method is not POST, the server returns the `405 Method Not Allowed` HTTP error code. If the *Content-Length* field is missing, the server returns the `411 Length Required` HTTP error code. If an *Content-Type* other than '`application/x-tar`', '`application/x-gzip`', '`application/x-xz-compressed-tar`' is used, the server returns the `415 unsupported Media Type` HTTP error code. If the *Content-Length* value is greater than a limit set by *MaxPackedSize* option in the server configuration file (50 MB by default), then the server returns the `413 Request Entity Too Large` HTTP error code. The limit can be checked by calling `https://server/settings`.

The limit is changeable from the server configuration file. Client is allowed to specify an optional *X-Task-Type* header, identifying the task type. At the moment `TASK_RETRACE (0)`, `TASK_DEBUG (1)` and `TASK_RETRACE_INTERACTIVE (3)` can be used for binary crashes and `TASK_VMCORE (2)` and `TASK_VMCORE_INTERACTIVE (4)` for kernel crashes. If no *X-Task-Type* is specified, `TASK_RETRACE` is used.

If unpacking the archive would result in having the free disk space under certain limit in the '`/var/spool/retrace-server`' directory, the server returns the `507 Insufficient Storage` HTTP error code. The limit is specified by the *MinStorageLeft* option in the server configuration file, and it is set to 1024 MB by default.

If the data from the received archive would take more than 1024 MB of disk space when uncompressed, the server returns the `413 Request Entity Too Large` HTTP error code. The limit can be checked by calling `https://server/settings`. The limit is changeable by the *MaxUnpackedSize* option in the server configuration file. It can be set pretty high because coredumps, that take most disk space, are stored on the server only temporarily until the backtrace is generated. When the backtrace is generated the coredump is deleted by the `retrace-server-worker`, so most disk space is released.

The uncompressed data size for xz archives is obtained by calling '`xz --list file.tar.xz`'. The '`--list`' option has been implemented only recently, so updating `xz` on your server might be necessary. Likewise, the uncompressed data size for gzip archives is obtained by calling '`gzip --list file.tar.gz`'.

If an upload from a client succeeds, the server creates a new directory '`/var/spool/retrace-server/`*`id`*' and extracts the received archive into it. Then it checks that the directory contains all the required files, checks their sizes, and then sends a HTTP response. After that it spawns a subprocess with `retrace-server-worker` on that directory.

The following files from the local crash directory are required to be present in the archive for binary crashes: '`coredump`', '`executable`', '`package`'. The '`os_release`' file is not required, but may be really helpful. Without this one the server tries to guess release, but the client can not rely on a correct identification of its system.

For kernel crashes, only the '`vmcore`' needs to be present in the archive. Although it's just one file, it needs to be `tar`-compressed as well. As the vmcore may be a large file, the server administrator may specify the *VmcoreDumpLevel* config option to run `makedumpfile` with the appropriate dump level on each received vmcore. This will drop unnecessary pages from the vmcore and save a significant amount of disk space.

If one or more files are not present in the archive, the server returns the `403 Forbidden` HTTP error code.

If the file check succeeds, the server HTTP response has the `201 Created` HTTP code. The response includes the following HTTP header fields:

- *X-Task-Id* containing a new server-unique numerical task id
- *X-Task-Password* containing a newly generated password, required to access the result

The *X-Task-Password* is a random alphanumeric ('`[a-zA-Z0-9]`') sequence 32 characters long. The password is stored in the '`/var/spool/retrace-server/`*`id`*`/password`' file, and passwords sent by a client in subsequent requests are verified by comparing with this file.

The task id is intentionally not used as a password, because it is desirable to keep the id readable and memorable for humans. Password-like ids would be a loss when an user authentication mechanism is added, and server-generated password will no longer be necessary.

2.2 Task status

A client might request a task status by sending a HTTP GET request to the `https://someserver/id` URL, where *id* is the numerical task id returned in the *X-Task-Id* field by `https://someserver/create`. If the *id* is not in the valid format, or the task *id* does not exist, the server returns the `404 Not Found` HTTP error code.

The client request must contain the *X-Task-Password* field, and its content must match the password stored in the '`/var/spool/retrace-server/id/password`' file. If the password is not valid, the server returns the `403 Forbidden` HTTP error code.

If the checks pass, the server returns the `200 OK` HTTP code, and includes a *X-Task-Status* header containing one of the following values: '`FINISHED_SUCCESS`', '`FINISHED_FAILURE`', '`PENDING`'. The response body contains a more detailed status message.

The field contains '`FINISHED_SUCCESS`' if the file '`/var/spool/retrace-server/id/retrace_backtrace`' exists. The client might get the backtrace on the `https://someserver/id/backtrace` URL. The log might be obtained on the `https://someserver/id/log` URL, and it might contain warnings about some missing debuginfos etc.

The field contains '`FINISHED_FAILURE`' if the file '`/var/spool/retrace-server/id/retrace_backtrace`' does not exist, and file '`/var/spool/retrace-server/id/retrace_log`' exists. The retrace_log file containing error messages can be downloaded by the client from the `https://someserver/id/log` URL.

The field contains '`PENDING`' if neither file exists. The client should ask again after 10 seconds or later.

2.3 Requesting a backtrace

A client might request a backtrace by sending a HTTP GET request to the `https://someserver/id/retrace_backtrace` URL, where *id* is the numerical task id returned in the *X-Task-Id* field by `https://someserver/create`. If the *id* is not in the valid format, or the task *id* does not exist, the server returns the `404 Not Found` HTTP error code.

The client request must contain the *X-Task-Password* field, and its content must match the password stored in the '`/var/spool/retrace-server/id/password`' file. If the password is not valid, the server returns the `403 Forbidden` HTTP error code.

If the file '`/var/spool/retrace-server/id/retrace_backtrace`' does not exist, the server returns the `404 Not Found` HTTP error code. Otherwise it returns the file contents, and the *Content-Type* header is set to '`text/plain`'.

2.4 Requesting a log

A client might request a task log by sending a HTTP GET request to the `https://someserver/id/log` URL, where *id* is the numerical task id returned in the

X-Task-Id field by `https://someserver/create`. If the *id* is not in the valid format, or the task *id* does not exist, the server returns the `404 Not Found` HTTP error code.

The client request must contain the *X-Task-Password* field, and its content must match the password stored in the '`/var/spool/retrace-server/id/password`' file. If the password is not valid, the server returns the `403 Forbidden` HTTP error code.

If the file '`/var/spool/retrace-server/id/retrace_log`' does not exist, the server returns the `404 Not Found` HTTP error code. Otherwise it returns the file contents, and the *Content-Type* header is set to '`text/plain`'.

2.5 Limiting traffic

The maximum number of simultaneously running tasks is limited to 5 by the server. The limit is changeableby the *MaxParallelTasks* option in the server configuration file. If a new request comes when the server is fully occupied, the server returns the `503 Service Unavailable` HTTP error code.

The archive extraction, chroot preparation, and gdb analysis is mostly limited by the hard drive size and speed.

3 Retrace worker

Retrace worker is a program (usually residing in `/usr/bin/retrace-server-worker`) takes a task id as a parameter and turns it into the task directory.

Each task may contain a set of remote resources. These are specified in the '`remote`' file within the task directory. Each line means a separate resource - URL parseable by `wget`. The worker goes line by line calling `wget` on each record. The download is best-effort - it does not die on a missing or invalid resource. The list of errors is later available in the '`retrace_log`' file.

Next performed actions depend on the task type.

3.1 Binary crashes

The worker performs following steps for binary crashes:

1. determines which packages need to be installed from the coredump
2. installs the packages in a newly created chroot environment together with `gdb`
3. copies the coredump to the chroot environment
4. runs `gdb` from inside the environment to generate a backtrace from the coredump
5. copies the resulting backtrace from the environment to the directory

The tasks reside in '`/var/spool/retrace-server/`*`taskid`*' directories.

To determine which packages need to be installed, `retrace-server-worker` runs the `coredump2packages` tool. The tool reads build-ids from the coredump, and tries to find the best set of packages (epoch, name, version, release) matching the build-ids. Local yum repositories are used as the source of packages. GDB requirements are strict, and this is the reason why proper backtraces cannot be directly and reliably generated on systems whose software is updated:

- The exact binary which crashed needs to be available to GDB.
- All libraries which are linked to the binary need to be available in the same exact versions from the time of the crash.
- The binary plugins loaded by the binary or libraries via `dlopen` need to be present in proper versions.
- The files containing the debugging symbols for the binary and libraries (build-ids are used to find the pairs) need to be available to GDB.

The chroot environments are created and managed by `mock`, and they reside in '`/var/lib/mock/`*`taskid`*'. The retrace worker generates a mock configuration file and then invokes `mock` to create the chroot, and to run programs from inside the chroot.

The chroot environment is populated by installing packages using `yum`. Package installation cannot be avoided, as GDB expects to operate on an installed system, and on crashes from that system. GDB uses plugins written in Python, that are shipped with packages (for example see `rpm -ql libstdc++`).

Coredumps might be affected by `prelink`, which is used on Fedora to speed up dynamic linking by caching its results directly in binaries. The system installed by `mock` for the purpose of retracing doesn't use `prelink`, so the binaries differ between the system of

origin and the mock environment. It has been tested that this is not an issue, but in the case some issue occurs (GDB fails to work with a binary even if it's the right one), a bug should be filed on `prelink`, as its operation should not affect the area GDB operates on.

No special care is taken to avoid the possibility that GDB will not run with the set of packages (fixed versions) as provided by coredump. It is expected that any combination of packages user might use in a released system satisfies the needs of some version of GDB. Yum selects the newest possible version which has its requirements satisfied.

After finishing the task (by success or failure) `retrace-server-worker` cleans up the `mock` chroot and deletes the crash data uploaded by the user. The only exception is `TASK_DEBUG` task type, which is designed to keep crash (including coredump) and chroot on the server for deeper Retrace server code debugging.

3.2 Kernel crashes

The worker performs following steps for kernel crashes:

1. determines the kernel version from the vmcore
2. prepares a chroot if the architecture is different to host system's and copies the vmcore into the chroot
3. if required, unpacks appropriate kernel-debuginfo and caches vmlinux
4. runs `crash` tool to generate the log
5. copies the resulting log to the task directory

Determining the kernel version from the vmcore is a tricky thing. The `crash` tool has the `--osrelease` option, which can be used to determine the version. However this does not work for cross-arch vmcores (especially i386 on x86_64). Another problem is that el6+ vmcores contain the kernel version in VRA (version.release.architecture) format, while el5- only contain VR (version.release).

If the host system's architecture does not match the vmcore one, a chroot is required to be able to run `crash` tool correctly. The mock configuration is saved to '`/var/spool/retrace-server/kernel-$ARCH`' directory. The repo used to install the chroot must be specified in the *KernelChrootRepo* config option, using *$ARCH* as a placeholder for the architecture. The chroot is only installed on first use and it is kept to be able to process future tasks.

To perform full debugging, the '`vmlinux`' file from kernel debuginfo is required, as well as debuginfo for all loaded modules. The worker first checks, whether all files are cached under '`/var/cache/retrace-server/kernel/$ARCH/`'. If any of them is missing, it is unpacked from the debuginfo and cached. This incremental process only keeps the required files unpacked and thus does not waste hard drive space. (Note: if every module would be loaded in the set of processed vmcores, one would end up with the whole debuginfo unpacked.)

4 Task cleanup

It is neccessary to watch and limit the resource usage of tasks for a retrace server to remain operational. This is performed by the `retrace-server-cleanup` tool. It is supposed that the server administrator sets `cron` to run the tool every hour (as *apache* user). The commented crontab entry is added when installing the RPM.

Tasks that were created more than 120 hours (5 days) ago are deleted. The limit can be changed by the *DeleteTaskAfter* option in the server configuration file. Coredumps are deleted when the retrace process is finished, and only backtraces, logs, and passwords remain available for every task until the cleanup. The `retrace-server-cleanup` checks the creation time and deletes the directories in '`/var/spool/retrace-server/`'.

If the server administrator does not want to completely delete all tasks, he can set the *ArchiveTaskAfter* config option. When the task becomes old enough to be archived, it is `.tar.gz`-ed into drop directory set by the *DropDir* config option. The original task directory is deleted. Archiving tasks is a possible privacy problem and should not be used on public instances. Please note that if e.g. *DeleteTaskAfter = 120* and *ArchiveTaskAfter = 24*, every task will be archived, because it matches the condition earlier. If both options are set to the same value, the task is deleted.

Tasks running for more than 1 hour are terminated and removed from the system. Tasks for which the `retrace-server-worker` crashed for some reason without marking the task as finished are also removed.

5 Package repository

Retrace server is able to support every Fedora release with all packages that ever made it to the updates and updates-testing repositories. In order to provide all that packages, a local repository needs to be maintained for every supported operating system.

A repository with Fedora packages must be maintained locally on the server to provide good performance and to provide data from older packages already removed from the official repositories. Retrace server contains a tool `retrace-server-reposync`, which is a package downloader scanning Fedora servers for new packages, and downloading them so they are immediately available.

Older versions of packages are regularly deleted from the updates and updates-testing repositories. Retrace server supports older versions of packages, as this is one of major pain-points that the retrace server is supposed to solve.

The `retrace-server-reposync` downloads packages from Fedora repositories, and it does not delete older versions of the packages. The retrace server administrator is supposed to call this script using cron approximately every 6 hours. The script uses `rsync` or `reposync` to get the packages (depending on the protocol) and `createrepo` to generate respository metadata.

The packages are downloaded to a local repository in '`/var/cache/retrace-server/`'. The location can be changed via the *RepoDir* option in the server configuration file.

6 Traffic and load estimation

2500 bugs are reported from ABRT every month. Approximately 7.3% from that are Python exceptions, which don't need a retrace server. That means that 2315 bugs need a retrace server. That is 77 bugs per day, or 3.3 bugs every hour on average. Occasional spikes might be much higher (imagine a user that decided to report all his 8 crashes from last month).

We should probably not try to predict if the monthly bug count goes up or down. New, untested versions of software are added to Fedora, but on the other side most software matures and becomes less crashy. So let's assume that the bug count stays approximately the same.

Test crashes (see why we use `‘xz -2‘` to compress coredumps):

- firefox with 7 tabs (random pages opened), coredump size 172 MB
  - xz compression
    - compression level 6 (default): compression took 32.5 sec, compressed size 5.4 MB, decompression took 2.7 sec
    - compression level 3: compression took 23.4 sec, compressed size 5.6 MB, decompression took 1.6 sec
    - compression level 2: compression took 6.8 sec, compressed size 6.1 MB, decompression took 3.7 sec
    - compression level 1: compression took 5.1 sec, compressed size 6.4 MB, decompression took 2.4 sec
  - gzip compression
    - compression level 9 (highest): compression took 7.6 sec, compressed size 7.9 MB, decompression took 1.5 sec
    - compression level 6 (default): compression took 2.6 sec, compressed size 8 MB, decompression took 2.3 sec
    - compression level 3: compression took 1.7 sec, compressed size 8.9 MB, decompression took 1.7 sec
- thunderbird with thousands of emails opened, coredump size 218 MB
  - xz compression
    - compression level 6 (default): compression took 60 sec, compressed size 12 MB, decompression took 3.6 sec
    - compression level 3: compression took 42 sec, compressed size 13 MB, decompression took 3.0 sec
    - compression level 2: compression took 10 sec, compressed size 14 MB, decompression took 3.0 sec
    - compression level 1: compression took 8.3 sec, compressed size 15 MB, decompression took 3.2 sec
  - gzip compression
    - compression level 9 (highest): compression took 14.9 sec, compressed size 18 MB, decompression took 2.4 sec
    - compression level 6 (default): compression took 4.4 sec, compressed size 18 MB, decompression took 2.2 sec

        - compression level 3: compression took 2.7 sec, compressed size 20 MB, decompression took 3 sec
- evince with 2 pdfs (1 and 42 pages) opened, coredump size 73 MB
    - xz compression
        - compression level 2: compression took 2.9 sec, compressed size 3.6 MB, decompression took 0.7 sec
        - compression level 1: compression took 2.5 sec, compressed size 3.9 MB, decompression took 0.7 sec
- OpenOffice.org Impress with 25 pages presentation, coredump size 116 MB
    - xz compression
        - compression level 2: compression took 7.1 sec, compressed size 12 MB, decompression took 2.3 sec

So let's imagine there are some users that want to report their crashes approximately at the same time. Here is what the retrace server must handle:

1. 2 OpenOffice crashes
2. 2 evince crashes
3. 2 thunderbird crashes
4. 2 firefox crashes

We will use the xz archiver with the compression level 2 on the ABRT's side to compress the coredumps. So the users spend 53.6 seconds in total packaging the coredumps.

The packaged coredumps have 71.4 MB, and the retrace server must receive that data.

The server unpacks the coredumps (perhaps in the same time), so they need 1158 MB of disk space on the server. The decompression will take 19.4 seconds.

Several hundred megabytes will be needed to install all the required packages and debuginfos for every chroot (8 chroots 1 GB each = 8 GB, but this seems like an extreme, maximal case). Some space will be saved by using a debuginfofs.

Note that most applications are not as heavyweight as OpenOffice and Firefox.

7 Security

The retrace server communicates with two other entities: it accepts coredumps form users, and it downloads debuginfos and packages from distribution repositories.

General security from GDB flaws and malicious data is provided by chroot. The GDB accesses the debuginfos, packages, and the coredump from within the chroot under a non-root user, unable to access the retrace server's environment.

SELinux policy exists for the retrace worker, but needs to be updated.

7.1 Clients

It is expected that the clients, which are using the Retrace server and sending coredumps to it, trust the retrace server administrator. The server administrator must not try to get sensitive data from client coredumps. This is a major bottleneck of the Retrace server. However, users of an operating system already trust the operating system provider in various important matters. So when the Retrace server is operated by the OS provider, that might be acceptable for users.

Sending client's coredumps to the Retrace server cannot be avoided if we want to generate good backtraces containing the values of variables. Minidumps lower the quality of the resulting backtraces, while not improving user security.

A malicious client can craft a nonstandard coredump, which will be processed by server's GDB. GDB handles malformed coredumps well.

Users can never be allowed to provide custom packages/debuginfo together with a coredump. Packages need to be installed to the environment, and installing untrusted programs is insecure.

As for attacker trying to steal user's backtraces from the retrace server, the passwords protecting the backtraces in the *X-Task-Password* header are random alphanumeric (‘`[a-zA-Z0-9]`’) sequences 32 characters long. 32 alphanumeric characters corresponds to 192 bit password, because ‘`[a-zA-Z0-9]`’ is 62 characters, and $2^{192} < 62^{32}$. The source of randomness is ‘`/dev/urandom`’.

7.2 Packages and debuginfo

Packages and debuginfo are safely downloaded from the distribution repositories, as the packages are signed by the distribution, and the package origin is verified.

When the debuginfo filesystem server is done, the retrace server can safely use it, as the data will also be signed.

8 Interactive Debugging

8.1 Overview

When the task type specified by *X-Task-Type* header is either `TASK_RETRACE_INTERACTIVE` or `TASK_VMCORE_INTERACTIVE`, the standard process is executed, but the chroot is not cleaned up. It can be later accessed by the '`retrace-server-interact`' tool, which is basically a wrapper over '`mock`', '`gdb`' and '`crash`'. Using either `crash` or `gdb` actions will enter the chroot, run the desired tool, load the core/vmcore and appropriate debuginfo automatically so that the debugger can start typing commands directly to the prompt. Using the `shell` command jumps to the chrooted shell. All commands are executed as non-privileged user by default, but this can be overriden by the `--priv` option.

8.2 Security

The user executing '`retrace-server-interact`' needs to be a member of `retrace` and `mock` groups. The only thing required to interactively debug is the task ID (not even task password!). There are no further security mechanisms, assuming the debugger has a root access to the host machine. Using '`retrace-server-interact`' enables the debugger to jump into root shell within the chroot, which may be a possible security risk.

8.3 Notes

- The interactive task needs to be explicitely removed, as the automatic cleanup skips it.
- When retracing vmcores natively (e.g. x86_64 vmcore on x86_64 host), no chroot is used, thus jumping to chrooted shell is meaningless: you can use the current shell.
- See `man retrace-server-interact` for further information on command syntax.

9 Task Manager

The task manager enables Retrace Server not to execute tasks immediately, but let users/debuggers to choose when to start a particular one.

First of all, *AllowTaskManager* needs to be enabled in the configuration file. There is no authentication in the task manager. All results are publicly visible to anybody who can access the correct URL, thus the task manager should not be used on public instances.

9.1 Creating a managed task

The managed task can be created in three ways:

- Using Retrace Server's standard HTTP API. The request is exactly same as when creating a regular task, except that it needs to include the *X-Task-Managed* header.
- By using an external FTP server. Retrace Server is able to connect to an FTP server and download task data from it. See below for more detailed information.
- By manually creating the task directory, respecting Retrace Server's internal directory structure. This approach works, however it is not recommended to use it, as the internals may change.

9.2 Tasks created from remote FTP files

Retrace Server is able to query task data from an FTP server. It will unpack *.tar.gz*, *.tgz*, *.tar.bz2*, *.tar.xz*, *.tarz*, *.tar*, *.gz*, *.bz2*, *.xz*, *.Z* and *.zip* archives. More formats may be added in the future. So far only vmcores are supported and the task is always created as *TASK_VMCORE_INTERACTIVE*, so make sure *AllowInteractive* is enabled in the configuration file. The local task is created by clicking `Start Task` link in the task manager. The task is automatically executed and begins downloading remote resources immediately.

9.3 Miscleanous results

In addition to the standard '`retrace_log`' and '`retrace_backtrace`', a managet task may contain more than a single result. All excessive data may be saved as miscleanous results. These are key-value pairs saved into '`misc`' subdirectory within the task directory. Each entry is saved into '`<task_dir>/misc/key`' file and the contents are the actual value. All miscleanous results are accessible from the task manager.

10 Configuration

The following options can be specified in the configuration file '`/etc/retrace-server.conf`':

- `RequireHTTPS` boolean; whether to deny non-HTTPS. Default 1.
- `RequireGPGCheck` boolean; whether to use GPG check on the packages installed into chroot (does not apply for rawhide or vmcores chroot). GPG key file named '`release-version`' needs to be present in the '`/usr/share/retrace-server/gpg/`' directory. Default 1.
- `AllowApiDelete` boolean; whether to allow task deleting by a HTTP request to '`https://someserver/<task_id>/delete`'. Default 0.
- `AllowInteractive` boolean; whether to allow interactive tasks. This is a security risk and should not be used on public systems. See the Interactive tasks chapter for more information. Default 0.
- `AllowTaskManager` boolean; whether to allow managing tasks by task manager. See the Task Manager chapter for more information. Default 0.
- `MaxParallelTasks` integer; how many tasks may be running at one time. All new tasks are denied while the number of running tasks is not less than the limit. Default 5.
- `MaxPackedSize` integer; maximum size of the uploaded archive (in megabytes). Default 50 (needs to be increased for vmcores).
- `MaxUnpackedSize` integer; maximum size of the archive content (in megabytes). This is a protection against sparse etc. which can unpack a small archive into a huge file. Default 1024.
- `MinStorageLeft` integer; the amount of storage space that needs to be kept free on the '`/var/spool/retrace-server`' filesystem (in megabytes). Default 1024.
- `DeleteTaskAfter` integer; time (in hours) after which the task is considered `"old"` and should be deleted (next run of `retrace-server-cleanup` deletes it). Any value less or equal to zero means disabled. Default 120.
- `ArchiveTaskAfter` integer; similar to `DeleteTaskAfter`, but the task is archived to `DropDir` (see below) before deleting. Default 0.
- `DBFile` string; the name of file used to save statistics. Default '`stats.db`'.
- `LogDir` string; the directory used to save global logs. Per-task logs are saved to task directories. Default '`/var/log/retrace-server`'.
- `RepoDir` string; the directory where local repositories are saved and searched. For vmcores, this directory also contains cached vmlinux files and needs to be writable for retrace user (thus can not be mounted read-only!). Needs to be synchronized manually to the content of '`/etc/yum.repos.d/retarce-*.repo`', which hardcodes the default. Default '`/var/cache/retrace-server`'.
- `SaveDir` string; the directory where task directories are created. Default '`/var/spool/retrace-server`'.
- `UseWorkDir` boolean; whether to use non-default mock directory. Historical, does not work at the moment. May be revived in the future.
- `WorkDir` string; custom mock working directory. Historical, does not work at the moment. May be revived in the future.

- `DropDir` string; directory, where old tasks are archived before deleting. No effect if archiving is disabled. Default '`/srv/retrace/archive`'.
- `UseCreateRepoUpdate` boolean; whether to call `createrepo` with `--update` option when creating the local repository. This is much faster, but consumes a lot of memory (several gigabytes on a standard Fedora repo). Default 0.
- `KeepRawhideLatest` integer; how many latest packages from rawhide distribution to keep. All older packages are deleted. Default 3.
- `KernelChrootRepo` string; repository used to install chroot for cross-arch vmcore analysis. *$ARCH* is a wildcard for the required architecture. The repository must contain '`bash`', '`coreutils`', '`cpio`', '`crash`' and '`shadow-utils`' packages and their dependencies. Default '`http://dl.fedoraproject.org/pub/fedora/linux/releases/16/Everything/$ARCH/os/`'
- `KojiRoot` string; the directory pointing to the root of Koji directory structure. This is used when looking for kernel debuginfo when processing vmcores. Default '`/mnt/koji`'.
- `UseFTPTasks` boolean; whether to download tasks from remote FTP. No effect if task manager is disabled. Default 0.
- `FTPSSL` boolean; whether to use SSL with the FTP connection. Default 0,
- `FTPHost` string; FTP host. Default empty.
- `FTPUser` string; FTP user. Default empty.
- `FTPPass` string; FTP password. Default empty.
- `FTPDir` string; FTP directory containing tasks. Default '`/`'.

11 Future work

11.1 Coredump stripping

Jan Kratochvil: With my test of OpenOffice.org presentation kernel core file has 181MB, xz -2 of it has 65MB. According to 'set target debug 1' GDB reads only 131406 bytes of it (incl. the NOTE segment).

11.2 Supporting other architectures

Three approaches:

- Use GDB builds with various target architectures: gdb-i386, gdb-ppc64, gdb-s390.
- Run QEMU user space emulation on the server
- Run `retrace-server-worker` on a machine with right architecture. Introduce worker machines and tasks, similarly to Koji.

11.3 Use gdbserver instead of uploading whole coredump

GDB's gdbserver cannot process coredumps, but Jan Kratochvil's can:

```
git://git.fedorahosted.org/git/elfutils.git
branch: jankratochvil/gdbserver
  src/gdbserver.c
   * Currently threading is not supported.
   * Currently only x86_64 is supported (the NOTE registers layout).
```

11.4 User management for the HTTP interface

Multiple authentication sources (x509 for RHEL).

11.5 Make all files except coredump optional on the input

Make '`executable`', '`release`' and '`package`' files, which must be included in the package when creating a task, optional. Allow uploading a coredump without involving tar: just coredump, coredump.gz, or coredump.xz.

11.6 Handle non-standard packages (provided by user)

This would make retrace server very vulnerable to attacks, it never can be enabled in a public instance.

11.7 Update SELinux policy

The SELinux policy needs to be updated because of recent changes.

11.8 Do not refuse new tasks on a fully loaded server

Consider using kobo for task management and worker handling (master/slaves arch).

11.9 Support synchronous operation

Client sends a coredump, and keeps receiving the server response message. The server response HTTP body is generated and sent gradually as the task is performed. Client can choose to stop receiving the response body after getting all headers and ask the server for status and backtrace asynchronously.

The server re-sends the output of retrace-server-worker (its stdout and stderr) to the response the body. In addition, a line with the task status is added in the form `X-Task-Status: PENDING` to the body every 5 seconds. When the worker process ends, either '`FINISHED_SUCCESS`' or '`FINISHED_FAILURE`' status line is sent. If it's '`FINISHED_SUCCESS`', the backtrace is attached after this line. Then the response body is closed.

11.10 Provide task estimation time

The response to the `/create` action should contain a header *X-Task-Est-Time*, that contains a number of seconds the server estimates it will take to generate the backtrace

The algorithm for the *X-Task-Est-Time* time estimation should take the previous analyses of coredumps with the same corresponding package name into account. The server should store simple history in a SQLite database to know how long it takes to generate a backtrace for certain package. It could be as simple as this:

- initialization step one: `CREATE TABLE package_time (id INTEGER PRIMARY KEY AUTOINCREMENT, package, release, time)`; we need the *id* for the database cleanup - to know the insertion order of rows, so the `AUTOINCREMENT` is important here; the *package* is the package name without the version and release numbers, the *release* column stores the operating system, and the *time* is the number of seconds it took to generate the backtrace
- initialization step two: `CREATE INDEX package_release ON package_time (package, release)`; we compute the time only for single package on single supported OS release per query, so it makes sense to create an index to speed it up
- when a task is finished: `INSERT INTO package_time (package, release, time) VALUES ('??', '??', '??')`
- to get the average time: `SELECT AVG(time) FROM package_time WHERE package == '??' AND release == '??'`; the arithmetic mean seems to be sufficient here

So the server knows that crashes from an OpenOffice.org package take 5 minutes to process in average, and it can return the value 300 (seconds) in the field. The client does not waste time asking about that task every 20 seconds, but the first status request comes after 300 seconds. And even when the package changes (rebases etc.), the database provides good estimations after some time anyway (Chapter 4 [Task cleanup], page 9 chapter describes how the data are pruned).

11.11 Keep the database with statistics small

The database containing packages and processing times should also be regularly pruned to remain small and provide data quickly. The cleanup script should delete some rows for packages with too many entries:

1. get a list of packages from the database: `SELECT DISTINCT package, release FROM package_time`

2. for every package, get the row count: `SELECT COUNT(*) FROM package_time WHERE package == '??' AND release == '??'`
3. for every package with the row count larger than 100, some rows most be removed so that only the newest 100 rows remain in the database:
   - to get highest row id which should be deleted, execute `SELECT id FROM package_time WHERE package == '??' AND release == '??' ORDER BY id LIMIT 1 OFFSET ??`, where the `OFFSET` is the total number of rows for that single package minus 100
   - then all the old rows can be deleted by executing `DELETE FROM package_time WHERE package == '??' AND release == '??' AND id <= ??`