Tinykdump

Improving kernel crash analysis

March 26, 2015

Summary

Error analysis is one of the most crucial parts when resolving issues in a software development project. In the current Linux kernel, kdump[1] is used to provide system logs and memory dumps immediately after the system crashes. Our project's intention is to minimize issues within the kdump infrastructure by making use of features of the modern Linux kernel.

The project

Advanced kernel debugging relies on a mechanism for analyzing the root cause of a kernel crash. Kdump provides a memory dump and kernel buffer of the crashed kernel. The current implementation uses kexec[1] to load a second kernel, the crashkernel, which is executed when the first kernel crashes, to capture the desired information. To store the crashkernel, kexec reserves a fixed amount of physical memory available during the main kernel's start. Currently every distribution ships its own tools to manage loading of the crashkernel image. Fedora's implementation[2] uses dracut to prepare the crashkernel image. The generated crashdump can be stored in several ways (hard-disk, nfs-storage, usb, via ssh).

The goal of the project is to resolve current issues and provide a more generically reliable way of retrieving kernel crash dumps. Furthermore, it will coexist with the current kdump implementation. In order to remove the restriction of reserving the crashkernel memory at startup – which renders the memory useless until a crashkernel is loaded, the kernel-side code of kexec will be improved by using the new refactored contiguous memory allocator[3] (CMA). This renders the reserved memory reusable for movable pages before the crashkernel is loaded.

Current crashkernel debugging is tedious because of the lack of a reliable way to make the kernel mode settings (KMS) cooperate with the kdump crashkernel, as this almost always results in no usable display output. To provide a way of debugging the crashkernel, pstore[4] will be utilized to store the crashkernel's console output.

Fedora's current implementation of the kdump service is powerful and offers several features – all at the price of someone maintaining and testing the implementation. This might work in a static environment where the kernel rarely changes but Fedora is based on fast-paced development and therefore changes happen frequently. Tinykdump will reduce features to gain better maintainability. The current kdump service utilizes dracut[5] to generate the kernel images. To overcome the dependency on dracut, the goal is to write a service which generates kernel images by using the busybox toolchain, usb drivers and makedumpfile. Therefore, instead of supporting several ways of storing the crashdump, tinykdump will only support external usb drives. To protect potentially sensitive data written to usb drives, tinykdump will encrypt data using the gpg framework.

Benefits for Fedora and open-source community

- By implementing a generic service with a minimized feature set and no dependencies to any initramfs generators, maintaining efforts can be reduced. Likewise, porting the service to different platforms is made less troublesome.
- Tinykdump provides a reliable and uncomplicated mechanism for debugging the kernel, in order to simplify debugging kernel issues in Fedora.
- Enabling pstore to store the crashkernel's console ouput makes it easier to discover potential issues in current crashkernel configurations.
- Memory reusability is further maximized by providing patches to enable CMA support to kexec. In addition, low-memory systems can be supported due to the reused memory and small crashkernel, generated by a user space service.
- Patches for CMA support and pstore are useful not only to tinykdump, but also to kdump.
- Improve security by encrypting data written on external devices.

Implementation details

- **kexec dynamic memory reservation**

  Instead of reserving memory at kernel startup, a new CMA area is initialized. By using a wrapper function `crash_init_reserve_memory(...)` which calls `cma_init_reserve_mem(..)` architecture-specific code can be injected. Until now the currently used `struct resource crashk_res` is still initialized to default values, to not break compatibility with kexec-tools. On write to `/sys/kernel/kexec_crash_size` memory allocation is triggered and `crash_contigious_request(..)` is called. `crash_contigious_request(..)` uses `cma_alloc` to move used pages aside. In addition, `crashk_res` is updated to the allocated space from `cma_alloc` and inserted into `iomem_resources`. By using a new configuration flag, `CONFIG_KEXEC_CMA`, this feature can be optionally enabled.

- **Using pstore to log console crashkernel output**

  Pstore uses a generic storage system where new pstore types can easily be added. To add support for kexec, `PSTORE_TYPE_KEXEC_CONSOLE` is added to `enum pstore_type_id`. To capture the actual console output a new console is created and registered to the system via `register_console` which uses `psinfo` to write to available pstore.

- **Implementing a python based tinykdump service**

  The new python service will be based on the current kdump implementation of Fedora. The `configparser` module of the standard python library is used to parse configuration which is located in `/etc/default/tinykdump`. The initramfs which captures the dumps and stores them on the usb drive uses a statically linked busybox (which is available via Fedora packages) and makedumpfile from kexec-tools. Initramfs generation is done manually by generating the directory structure in `/var/run/tinykdump` and compressing it using cpio. The currently installed kernel is used as crashkernel. If no reserved memory is found, the service tries to write the size to `/sys/kernel/kexec_crash_size`, determined by the size of the crashkernel image generated earlier. If memory has already been reserved and its size is greater than required, the service tries to shrink the reserved memory. Functions supported by the python service are:

```
-c | --c <config> use custom config location
-b | --build build/update initramfs image
-m | --module <module, module> additional modules to be loaded by initramfs
-l | --load load crashkernel image. (Implicit build if not done yet)
-s | --script <bash file> script to be executed before the dump is written
-k | --key <gpg public key> to encrypt the captured kernel crash dump
```

Timetable

- **Now — 2 April (pre-selection phase)**

  Looking into the kernel sources to fully grasp the concept of CMA and pstore. Working on proof-of-concept patches for dynamic memory reservation and crashkernel pstore integration. Discussing possible implementations. *(Vacation from 2 until 6 April)*

- **6 April — 27 April (pre-selection phase)**

  Implementing a proof-of-concept python service which allows basic kexec loading (without secure boot). Learning about how secure boot / secure mode works and working on a testing/debugging strategy. Discussing possible implementations of the service.

- **27 April - 6 May (pre-coding phase)**

  Writing the real CMA kernel patch *(Vacation from 6 until 11 May)*

- **11 May — 24 May (pre-coding phase)**

  Testing the CMA patch implementation and debugging/fixing potential issues. Getting the patch community reviewed. Writing a blogpost about my progress, the current status of my work and my new insights.

- **25 May — 31 May (first coding phase)**

  Implementing the real pstore kernel patch

- **31 May — 14 June (first coding phase)**

  Testing the pstore patch and debugging/fixing potential issues. Getting the patch community reviewed. Applying further changes to the dynamic memory reservation patches if requested. Writing a blog post about what I have learned about pstore and what I have done so far.

- **14 June — 26 June (first coding phase)**

  Building kernel images for x86/x86_64 with both patches applied. Running tests on virtual systems and hardware that are available to me. Fixing issues which are found while testing. Writing a blogpost on the proper testing of kernel patches. **Optional**: Building the images on a RHEL base and testing them.

- ## 26 June Midterm evaluation

- **26 June — 14 July (second coding phase)**

  Implementing the python service with features described in "implementation details".

- **14 July — 1 August (second coding phase)**

  Testing the service and fixing potential issues. Testing on virtual systems and hardware that is available to me. Writing about the current status of my work on my blog.

- **1 August - 21 August (second coding phase)**

  More testing and bug fixing. If further changes to the patches have to be done, it should happen at this point. Writing documentation for the service like creating manpages etc.

- **Future**

  After Google Summer of Code I will port tinykdump to other distributions like OpenSUSE and Debian. If not already done: Port the dynamic reservation patches to other platforms if possible. I'd be happy to maintain the tinykdump package in Fedora.

About me

Name: Felix Schnizlein
E-Mail: felix@none.io
IRC: felixsch (in freenode and oftc)
Wiki page: https://fedoraproject.org/wiki/GSOC_2015/Student_Application_felixsch
Source of the proposal: https://github.com/felixsch/gsoc2015-tinykdump

Resources

1. Kernel Documentation https://www.kernel.org/doc/Documentation/kdump/kdump.txt
2. Fedora's kexec-tools git repository http://pkgs.fedoraproject.org/cgit/kexec-tools.git/
3. Article Contigious Memory Allocator http://lwn.net/Articles/486301/
4. Kernel Documentation https://www.kernel.org/doc/Documentation/ABI/testing/pstore
5. Dracut wiki https://dracut.wiki.kernel.org/index.php/Main_Page