OpenDNSSEC PIN store code review

In this code review, I wanted to assure:

- The PIN cannot be accessed by abusive programs
- The PIN cannot be made unavailable by abusive programs
- A failed PIN is not retried so often that an HSM might block
- The PIN mechanism is PKCS #11 compliant
- The old mode of operation works, a nice transition path exists
- Usage of the mechanism is not counter-intuitive

My findings follow, in the hope that they are useful. It should be noted that I traced the code for potential hazards, but that I may have missed other portions that soften these issues; in the latter case I should of course be corrected.

Rick van Rein, `rick@openfortress.nl`

The PIN cannot be accessed by abusive programs

The design paradigm for the PIN storage, as discussed in the past, is to rely on the surrounding system to enforce user/group level privileges. Note that this has a number of implications:

- These assumptions may only be reliably implemented on UNIX and akin systems; in other words, on Windows they may not hold as well as on UNIX and friends. I have not gone into this; we seem to be inclined to think of Windows as a poor choice for such security-related tasks. This may need explicit documentation where PIN storage is concerned. It could also be made to raise warning messages while building OpenDNSSEC.
- The admin (or the packager) should be well aware of the importance of the user/group mechanism, and reserve a group especially for use with OpenDNSSEC. This should definately be part of the documentation of the PIN Storage facility.

The pin.c functions in libhsm are a bit liberal in welcoming a pre-existing PIN store; this is probably needed to deal with a restart of libhsm, which happens on a regular basis when a new daemon and/or utility is started. This opens a window for an attacker to prepare a world-accessible PIN store that he can read out later. This problem has been solved by verifying the group ID (but not the user ID) of the PIN store's creator before writing a PIN into it. Similarly, DoS attacks of this kind are thwarted by a similar check before reading the PIN store.

Not checking the user ID is presumably done to enable the daemons of OpenDNSSEC to run under a different UID, and be able to use `libhsm`. This is somewhat debatable IMHO; it may be prudent for reasons like monitoring or other to welcome external programs to some of the state of OpenDNSSEC, without providing them access to the PIN. If this line is followed, the access privileges for the group can actually be strikken from `SHM_PERM`. I cannot think of a similarly strong reason to assign different User IDs to the Signer and Enforcer. Open to discussion!

Suggestion: Explain why user ID is not checked, or add a check for the user ID to both places in `pin.c` where the group ID is now checked.

A minor programming oversight is the line

:: prompt = malloc(64);

which is used without further checking. In theory, this could lead to all sorts of chaos, possibly even abuse.

Suggestion: Incorporate an assurance check that `prommpt != NULL`, or allocate the space on the stack.

The PIN cannot be made unavailable by abusive programs

A DoS on the PIN is a DoS on the daemons, is a DoS on continued signing, is a DoS on the signed availability of the zones. So, it is important that a DoS on the PIN is not possible.

There does not seem to be a way to overwrite the PIN by any other program than those that can also read it, and are thus considered reliable. In other words, there does not appear to be a problem.

The semaphore used to gain private access to the PIN Storage space is another place where a DoS attack could be mounted. This one could actually work; an attacker could prepare a world-accessible semaphore with the name assumed by OpenDNSSEC; then it could wait until the PIN was entered, and block the future use of the PIN by grabbing hold of the semaphore. It could then quit and leave the admin wondering what had happened.

Suggestion: Just like `shmctl` is used to verify the creator of the PIN Storage space is the same user that included `libhsm`, it would be advisable to use `semctl` to perform similar checks on the semaphore.

It would be helpful to query whether the PIN is loaded. This would be helpful information to monitoring tools, such as Nagios. For instance, this could detect the impaired state of the daemons after a reboot with automatic restart. This is just the sort of situation that you would like to report to an admin who can enter the PIN.

Suggestion: Include a command to verify if the PIN is loaded in a certain, or in all repositories. The result should be easy to process by a script such as a Nagios plugin.

A failed PIN is not retried so often that an HSM might block

When a PIN is tried too often, it usually blocks the HSM. This is a nasty situation, to be avoided at all cost, and certainly not a desirable side-effect of automation. For instance, a number of daemons each trying independently to login with the same, false PIN is asking for trouble.

The `ods-hsmutil login` command has a hard-coded regime of 3 attempts. This scares me somewhat, as thoughtless use might lead to an admin who wants to get out of the tool after a false PIN entry, and does this by a few nonsense entries.

Suggestion: If `ods-hsmutil login` fails, report this to the user, possibly along with a warning about login attempts left, but do not continue to ask for input. After a failed login, the admin could make a deliberate choice to try again, but it should not be automated IMHO.

When calling `hsm_block_pin`, the `mode` parameter could have the value `HSM_PIN_SAVE`, but it is meaningless to the call. The other two values present an opportunity to avoid PIN retries by at least the same daemon. None of these are used.

Suggestion: Disallow `HSM_PIN_SAVE` in `hsm_block_pin`.

Suggestion: Avoid overzealously trying a PIN that already failed, refuse to retry a PIN in this procedure, when `HSM_PIN_RETRY` is specified as a mode.

The last of these suggestions avoids cyclic behaviour, trying to login with a known-failed PIN. This limits the number of attempts of a falsely entered PIN to the number of daemons run, so to two. Ideally, this technical implication of the structure of OpenDNSSEC is not of influence on the number of attempts of PIN entry.

Suggestion: Consider adding one more mode, namely `HSM_PIN_FAILED`, to signal that a PIN was tried but failed. When this is provided, the corresponding part of the PIN Storage is wiped.

Relativation: The PIN is tested by `libhsm` before it is stored in the PIN Storage space. So, any damage due to falsely entered PINs is limited to that one attempt. However, an admin might change the PIN with the daemons running; that would be the only time that this sort of problem could arise. It could be argued that the admin called it upon himself by running OpenDNSSEC while changing the HSM PIN. Still, the risk should be properly documented IMHO, because the admin does not (want to) know the internal structure of OpenDNSSEC.

The PIN mechanism is PKCS #11 compliant

The PIN store deviates from PKCS #11 in the string formats supported. Where PKCS #11 specifies UTF8 character strings with an explicit notion of length, the PIN storage assumes a C-string format, terminated by a zero character. Since such characters can occur as part of UTF8 strings, their formats are incompatible.

Suggestion: Change the format of the PIN storage to one that incorporates an explicit length, or change the characters from bytes to uint32_t or so. Map as appropriate on reading and writing, and/or change their I/O format as well.

Note that no checks are made whether the top bit in the PIN is set; this means that UTF8 character strings could be constructed, thus leading to a `C_Login` call that attempts to read more than the available memory, possible leading to all sorts of leaks, due to the interpretation of the string length as an UTF8 length in `C_Login`. Relying on `getpass` to never return content with the high bit set seems dangerous to me.

The old mode of operation works, a nice transition path exists

I checked that the setting of a `<PIN/>` in `conf.xml` works to override the PIN Storage mechanism. This is a nice path of transition, where a user can remove this line after assuring themselves of the PIN daemon facilities. This means that the OpenDNSSEC daemons can be bootstrapped with a notification to the admin about supplying the PIN.

Usage of the mechanism is not counter-intuitive

For programmers, the `data` argument to `hsm_prompt_pin` and `hsm_block_pin` will be awkward, as it is not used. Is this a result of using a generic `pin_callback` perhaps?

Suggestion: Remove the parameter, or explain what future or other potential use it has and how it should be safely called until that use is made available.

For end users, the absense of a mirrorred command `ods-hsmutil logout` would feel awkward. Given that the utility encapsulates access to the shared memory segment for the PIN storage, it would make sense to do the same with its cleanup.

Suggestion: Include a `logout` instruction to `ods-hsmutil`.

I am wondering if users could need `login` and `logout` from individual repositories. This is not currently supported.