Ticket 703: qvm-backup: save backups in AppVM

197-251 minutes

Andrew Sorensen

unread,

Mar 27, 2013, 6:36:35 PM3/27/13

to qubes...@googlegroups.com

I've been working on additions to qvm-core/qubesutils.py to add functionality for compressing files in Dom0 and sending them to an appvm. Right now, I'm just using a hacked up version of qvm-backup to perform the backups, but eventually I want to make that compatible with the new system

My to do the actual copy looks something like this:

compressor = subprocess.Popen (["tar", "-PcOz", file["path"]], stdout=subprocess.PIPE)

subprocess.Popen (["qvm-run", "--pass-io", "-p", appvm, "cat > " + dest_dir + file["basename"] + ".tar.gz"], stdin=compressor.stdout)

(There's an issue with this code: multiple file backups running at the same time, I have yet to fix that).

Here's some questions that would be interesting to get thoughts on:

Should backups to Dom0 be supported at all? (or should it require a working AppVM?) - What about restore?

How should encryption of backups be handled? (Dom0 sounds the best, but how should the user provide their key?) - What about users who create a "backups" appvm and use LUKS on their drive?

How should the command line qvm-backup command work with this additional functionality to backup into an appvm? qvm-backup backups:/mnt/removable/qubes-backups maybe?

What about the case where users have AppVMs already running? (could sync(1) them and pause the domain, then perform the backup)

Joanna Rutkowska

unread,

Mar 27, 2013, 7:17:35 PM3/27/13

to qubes...@googlegroups.com, Andrew Sorensen

On 03/27/13 19:36, Andrew Sorensen wrote:
> I've been working on additions to qvm-core/qubesutils.py to add
> functionality for compressing files in Dom0 and sending them to an appvm.
> Right now, I'm just using a hacked up version of qvm-backup to perform the
> backups, but eventually I want to make that compatible with the new system
>
> My to do the actual copy looks something like this:

> compressor = subprocess.Popen (["tar", "-PcOz", file["path"]],
> stdout=subprocess.PIPE)
> subprocess.Popen (["qvm-run", "--pass-io", "-p", appvm, "cat > " +
> dest_dir + file["basename"] + ".tar.gz"], stdin=compressor.stdout)
>
> (There's an issue with this code: multiple file backups running at the same
> time, I have yet to fix that).
>
> Here's some questions that would be interesting to get thoughts on:
>
> Should backups to Dom0 be supported at all? (or should it require a working
> AppVM?) - What about restore?
>

I see no reason why to block backups to Dom0.

> How should encryption of backups be handled?

Encryption should definitely be done by Dom0. Otherwise the VM where the
backup were to be saved (or from which to be restored) would become as
privileged as Dom0, which would somehow defy the sense of backups to AppVMs.

> (Dom0 sounds the best, but how
> should the user provide their key?) - What about users who create a
> "backups" appvm and use LUKS on their drive?
>

How about the same way as cryptsetup asks for passphrase? And then
passed as a string argument to either backup_prepare() or do_backup(),
to allow the passprase to be passed also from the manager.

> How should the command line qvm-backup command work with this additional
> functionality to backup into an appvm? qvm-backup
> backups:/mnt/removable/qubes-backups maybe?
>

So 'backups' above, I assume, is the AppVM where the encrypted blob of
the backup is to be stored, correct? If so, then it looks good to me.

BTW, I don't really see a reason for using a dedicated 'backups' AppVM.
Instead I would expect people to use something like 'usbvm', or
'personal' as a backup VM (in this example the 'personal' might an AppVM
with access to my home NAS).

> What about the case where users have AppVMs already running? (could sync(1)
> them and pause the domain, then perform the backup)
>

That's a separate issue. I'm not a filesystem expert, so I don't know if
this will always work, but sounds like it might. But again, let's not
mix different functionalists into one task.

joanna.

Outback Dingo

unread,

Mar 27, 2013, 8:52:26 PM3/27/13

to qubes...@googlegroups.com, Andrew Sorensen

be nice if backups could also be sent over remote via https or ssh to a remote server

Joanna Rutkowska

unread,

Mar 27, 2013, 9:56:01 PM3/27/13

to qubes...@googlegroups.com, Outback Dingo, Andrew Sorensen

On 03/27/13 21:52, Outback Dingo wrote:
> be nice if backups could also be sent over remote via https or ssh to a
> remote server
>

What you do with your backup once it makes it to an AppVM of your choice, is totally up to you. You could then use all the Linux networking/storage tools, to send them over SMB, SSH, upload to a WebDAV, S3, and generally god-knows-what-else. But the job of qvm-back is only to store the (encrypted) blob in the selected AppVM. No more, no less.

joanna.

Andrew Sorensen

unread,

Mar 27, 2013, 11:45:27 PM3/27/13

to qubes...@googlegroups.com, Andrew Sorensen

On Wednesday, March 27, 2013 12:17:35 PM UTC-7, joanna wrote:

> On 03/27/13 19:36, Andrew Sorensen wrote:
> > I've been working on additions to qvm-core/qubesutils.py to add
> > functionality for compressing files in Dom0 and sending them to an appvm.
> > Right now, I'm just using a hacked up version of qvm-backup to perform the
> > backups, but eventually I want to make that compatible with the new system
> >
> > My to do the actual copy looks something like this:
> > compressor = subprocess.Popen (["tar", "-PcOz", file["path"]],
> > stdout=subprocess.PIPE)
> > subprocess.Popen (["qvm-run", "--pass-io", "-p", appvm, "cat > " +
> > dest_dir + file["basename"] + ".tar.gz"], stdin=compressor.stdout)
> >
> > (There's an issue with this code: multiple file backups running at the same
> > time, I have yet to fix that).
> >
> > Here's some questions that would be interesting to get thoughts on:
> >
> > Should backups to Dom0 be supported at all? (or should it require a working
> > AppVM?) - What about restore?
> >

> I see no reason why to block backups to Dom0.
>
> > How should encryption of backups be handled?
>
> Encryption should definitely be done by Dom0. Otherwise the VM where the
> backup were to be saved (or from which to be restored) would become as
> privileged as Dom0, which would somehow defy the sense of backups to AppVMs.

I will implement optional encryption in Dom0 then (in my particular use case, I have a separate VM that handles the backups).

> > (Dom0 sounds the best, but how
> > should the user provide their key?) - What about users who create a
> > "backups" appvm and use LUKS on their drive?
> >
>
> How about the same way as cryptsetup asks for passphrase? And then
> passed as a string argument to either backup_prepare() or do_backup(),
> to allow the passprase to be passed also from the manager.

Is it safe to pass passwords through pipes using qvm-run (or does that keep a log somewhere?)

> > How should the command line qvm-backup command work with this additional
> > functionality to backup into an appvm? qvm-backup
> > backups:/mnt/removable/qubes-backups maybe?
> >
>
> So 'backups' above, I assume, is the AppVM where the encrypted blob of
> the backup is to be stored, correct? If so, then it looks good to me.

Yes, that is correct. I was trying to keep to the standard I saw for creating HVMs from CDs.

> BTW, I don't really see a reason for using a dedicated 'backups' AppVM.
> Instead I would expect people to use something like 'usbvm', or
> 'personal' as a backup VM (in this example the 'personal' might an AppVM
> with access to my home NAS).

Okay.

> > What about the case where users have AppVMs already running? (could sync(1)
> > them and pause the domain, then perform the backup)
> >
>
> That's a separate issue. I'm not a filesystem expert, so I don't know if
> this will always work, but sounds like it might. But again, let's not
> mix different functionalists into one task.

I agree. This should be moved to a separate ticket.

> joanna.

Andrew Sorensen

unread,

Mar 27, 2013, 11:47:25 PM3/27/13

to qubes...@googlegroups.com, Outback Dingo, Andrew Sorensen

On Wednesday, March 27, 2013 2:56:01 PM UTC-7, joanna wrote:

> On 03/27/13 21:52, Outback Dingo wrote:
> > be nice if backups could also be sent over remote via https or ssh to a
> > remote server
> >
>
> What you do with your backup once it makes it to an AppVM of your
> choice, is totally up to you. You could then use all the Linux
> networking/storage tools, to send them over SMB, SSH, upload to a
> WebDAV, S3, and generally god-knows-what-else. But the job of qvm-back
> is only to store the (encrypted) blob in the selected AppVM. No more, no
> less.

Currently I'm just piping the compressed files into the AppVM. Should I keep the compression in Dom0, or drop it/make it optional? I assume if we are going to encrypt, then I need to do compression first (eg: in Dom0).

Marek Marczykowski

unread,

Mar 28, 2013, 4:01:45 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen, Outback Dingo

On 28.03.2013 00:47, Andrew Sorensen wrote:
> On Wednesday, March 27, 2013 2:56:01 PM UTC-7, joanna wrote:
>
>> On 03/27/13 21:52, Outback Dingo wrote:
>>> be nice if backups could also be sent over remote via https or ssh to a
>>> remote server
>>>
>>
>> What you do with your backup once it makes it to an AppVM of your
>> choice, is totally up to you. You could then use all the Linux
>> networking/storage tools, to send them over SMB, SSH, upload to a
>> WebDAV, S3, and generally god-knows-what-else. But the job of qvm-back
>> is only to store the (encrypted) blob in the selected AppVM. No more, no
>> less.
>>
>>
> Currently I'm just piping the compressed files into the AppVM. Should I
> keep the compression in Dom0, or drop it/make it optional? I assume if we
> are going to encrypt, then I need to do compression first (eg: in Dom0).

gpg already compress the data.

I see the point in supporting direct sent over https or whatever - this will
not require having space for full backup in AppVM.

So perhaps backup should be _one_ blob, which is sth like:
tar c <list of files to backup> | gpg | qvm-run --pass-io backups 'QUBESRPC
qubes.StoreBackup dom0'

Then qubes.StoreBackup service (configured as any other Qubes RPC service) can

do whatever you want with backup - store on mounted drive, send over the network, pipe to /dev/null...

'dom0' in above command is source VM name.

BTW instead of qvm-run you can use vm.run(..., passio_popen=True), this will return subprocess.Popen return value (so pipes will be available).

--
Best Regards / Pozdrawiam,
Marek Marczykowski
Invisible Things Lab

Marek Marczykowski

unread,

Mar 28, 2013, 4:03:31 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen

On 28.03.2013 00:45, Andrew Sorensen wrote:
> On Wednesday, March 27, 2013 12:17:35 PM UTC-7, joanna wrote:
>
>> On 03/27/13 19:36, Andrew Sorensen wrote:
>>> (Dom0 sounds the best, but how
>>> should the user provide their key?) - What about users who create a
>>> "backups" appvm and use LUKS on their drive?
>>>
>>
>> How about the same way as cryptsetup asks for passphrase? And then
>> passed as a string argument to either backup_prepare() or do_backup(),
>> to allow the passprase to be passed also from the manager.
>>
>>
> Is it safe to pass passwords through pipes using qvm-run (or does that keep
> a log somewhere?)

Why do you want pipe password to other VM? It shouldn't leave dom0.

Andrew Sorensen

unread,

Mar 28, 2013, 4:06:10 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen

Marek Marczykowski

unread,

Mar 28, 2013, 4:10:05 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen

Yes, but it still should be in dom0, VM should get only already encrypted

data. And yes - passing password through pipe should be safe.

Andrew Sorensen

unread,

Mar 28, 2013, 4:13:24 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen

Correct, but this considers the case that the user requires a LUKS device to be decrypted in the AppVM. It's probably best that this responsibility be left to the AppVM, and the backup system left standalone.

Andrew Sorensen

unread,

Mar 28, 2013, 4:14:58 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen, Outback Dingo

On Wednesday, March 27, 2013 9:01:45 PM UTC-7, Marek Marczykowski wrote:

> On 28.03.2013 00:47, Andrew Sorensen wrote:
> > On Wednesday, March 27, 2013 2:56:01 PM UTC-7, joanna wrote:
> >
> >> On 03/27/13 21:52, Outback Dingo wrote:
> >>> be nice if backups could also be sent over remote via https or ssh to a
> >>> remote server
> >>>
> >>
> >> What you do with your backup once it makes it to an AppVM of your
> >> choice, is totally up to you. You could then use all the Linux
> >> networking/storage tools, to send them over SMB, SSH, upload to a
> >> WebDAV, S3, and generally god-knows-what-else. But the job of qvm-back
> >> is only to store the (encrypted) blob in the selected AppVM. No more, no
> >> less.
> >>
> >>
> > Currently I'm just piping the compressed files into the AppVM. Should I
> > keep the compression in Dom0, or drop it/make it optional? I assume if we
> > are going to encrypt, then I need to do compression first (eg: in Dom0).
>
> gpg already compress the data.
>
> I see the point in supporting direct sent over https or whatever - this will
> not require having space for full backup in AppVM.
>
> So perhaps backup should be _one_ blob, which is sth like:
> tar c <list of files to backup> | gpg | qvm-run --pass-io backups 'QUBESRPC
> qubes.StoreBackup dom0'
>
> Then qubes.StoreBackup service (configured as any other Qubes RPC service) can
> do whatever you want with backup - store on mounted drive, send over the
> network, pipe to /dev/null...

I like this idea.

> 'dom0' in above command is source VM name.
>
> BTW instead of qvm-run you can use vm.run(..., passio_popen=True), this will return subprocess.Popen return value (so pipes will be available).

Thanks. I knew there had to be a better way. I'll clean up that code.

Marek Marczykowski

unread,

Mar 28, 2013, 4:45:10 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen, Outback Dingo

Some catches in above approach:
1. enable sparse file support in tar (AFAIR -S option)
2. directory structure in tar: IMHO it should be based on /var/lib/qubes layout, so tar would contain:
./qubes.xml
./appvms/work/private.img
./appvms/work/work.conf
./appvms/work/firewall.xml
(...)
3. VMs selection at restore. This can be some problem: currently we need qubes.xml and list of files to check which VMs are present in backup, then give the user choice about what VMs should be restored, then actual restore. Ideally we'd like not to download the whole backup (about 100GB in my case) twice. Also it's good to allow restore one VM without requiring disk space for whole backup. Concrete use case:
- whole backup takes 120GB
- system has 20GB free
- I want to restore one of my HVM, which takes 10GB, so I remove it first (now have 30GB free)
- here I have enough space for VM to restore, but too small for unpack of whole backup

If you have some idea how to solve 3rd problem, it would be nice. If not, we can leave it for later (anyway backups are mostly to restore whole system after eg. disk crash). Maybe we should ensure right now that backup will have sufficient information to enable such VM selection before unpacking whole backup in the future - like having list of VMs in backup at the beginning of the archive (qubes.xml in backup currently stores all VMs present in system, not only those backed up).

One more thing to consider about restore: if we unpack backup, it will be already copy of backup (original backup will stay intact), so we can move files to destination directories not copy.

Joanna Rutkowska

unread,

Mar 28, 2013, 10:30:16 AM3/28/13

to qubes...@googlegroups.com, Andrew Sorensen

No, I didn't mean LUKS at all! Again, all the encryption, decryption and passphrase management should be done in Dom0 only. What I meant is to use the same function for passphrase prompt as e.g. the cryptsetup uses. Or gpg. Or ssh. Or, perhaps we can even use sscanf() -- I wouldn't be picky about that.

Again, let me reiterate: whatever backup blob leaves dom0 it must already be encrypted and signed. The AppVMs cannot be trusted for anything regarding backup handling! Again, if they were trusted, then they automatically become as privileged as Dom0, which in turn makes it pointless to use a separate AppVM for backups!

And, consequently: whatever we get from the AppVM (the backup blob) should first be verified (gpgv --keyring dom0-backup-keyring?) and only then the unpacking should be attempted. Otherwise we risk that a malicious AppVM might send a malformed backup blob that might exploit Dom0 untaring/uncompressing code.

joanna.

Marek Marczykowski

unread,

Mar 28, 2013, 11:51:28 AM3/28/13

to qubes...@googlegroups.com, Joanna Rutkowska, Andrew Sorensen

The last part could be tricky: how verify backup signature if my keys are in backup itself? Some additional method to import keys for backup, right? So how verify that keys...

For just encryption we can use symmetric encryption (so only passphrase needed). Anyway if someone is able to:
1. encrypt backup using right passphrase (so know passphrase)
and
2. somehow substitute original backup - will probably have access to original backup this way,
then already game is over.
So I think we can stick with simple gpg --symmetric.

Joanna Rutkowska

unread,

Mar 28, 2013, 12:20:46 PM3/28/13

to Marek Marczykowski, qubes...@googlegroups.com, Andrew Sorensen

Good point :)

> For just encryption we can use symmetric encryption (so only passphrase
> needed). Anyway if someone is able to:
> 1. encrypt backup using right passphrase (so know passphrase)
> and
> 2. somehow substitute original backup - will probably have access to original
> backup this way,
> then already game is over.

> So I think we can stick with simple gpg --symmetric.
>

I'm afraid of an attack where some bits in the encrypted blob are (more or less) randomly modified by the attacker, which would result in some (more or less) garbage after decryption. In the best case these would cause a hard-to-detect DoS on the backups, and in the worst case (but rather unlikely) something worse. But the stealth DoS attacks on backups is what I fear the most. Of course a compromised AppVM can always do a DoS on my backup, but otherwise I can easily detect it (and copy the backup from the CDROM or whatever, again).

So, perhaps a simple HMAC would do?

openssl dgst -hmac <passphrase> backup.blob

Perhaps, just to be extra safe the passphrase used for HMAC can be obtained by hashing the actual backup encryption passphrase.

joanna.

Joanna Rutkowska

unread,

Mar 28, 2013, 12:22:15 PM3/28/13

to Marek Marczykowski, qubes...@googlegroups.com, Andrew Sorensen

Ah, and if that wasn't clear: such HMAC would be distributed together with the blob.tgz. Perhaps concatenated on top of it, just to keep it as one file?

j.

Marek Marczykowski

unread,

Mar 28, 2013, 12:26:35 PM3/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Andrew Sorensen

Doesn't gpg already does something like this (checksum of plain data checked during decryption)? If not - indeed such HMAC would be desirable.

Joanna Rutkowska

unread,

Mar 28, 2013, 1:52:33 PM3/28/13

to Marek Marczykowski, qubes...@googlegroups.com, Andrew Sorensen

It doesn't:

Create a test blob (I use no compression to make the experiment easier):

```
[user@work-pub test]$ dd if=/dev/zero of=test.bin bs=1k count=1k
1024+0 records in
1024+0 records out
1048576 bytes (1.0 MB) copied, 0.0060514 s, 173 MB/s
[user@work-pub test]$ gpg -c --compress-level 0 test.bin
[user@work-pub test]$ ll
total 2052
-rw-rw-r-- 1 user user 1048576 Mar 28 13:45 test.bin
-rw-rw-r-- 1 user user 1048625 Mar 28 13:45 test.bin.gpg
[user@work-pub test]$ cp test.bin.gpg test-hacked.bin.gpg

# Let's introduce one byte mutation at a random position in the
encrypted file:

[user@work-pub test]$ dd if=/dev/zero of=test-hacked.bin.gpg bs=1
count=1 seek=666999 conv=notrunc
1+0 records in
1+0 records out
1 byte (1 B) copied, 5.4258e-05 s, 18.4 kB/s

# Let's verify the mutation made it to the encrypted blob indeed:

[user@work-pub test]$ xxd test.bin.gpg > test.bin.gpg.xxd
[user@work-pub test]$ xxd test-hacked.bin.gpg > test-hacked.bin.gpg.xxd
[user@work-pub test]$ diff test.bin.gpg.xxd test-hacked.bin.gpg.xxd
41688c41688
< 00a2d70: 248a 7146 60ba f0fd c42e 3341 f6fc 5eef $.qF`.....3A..^.
---
> 00a2d70: 248a 7146 60ba f000 c42e 3341 f6fc 5eef $.qF`.....3A..^.

# (Nore the zero byte above instead of 0xfd)

# Now, let's try to decrypt it:

[user@work-pub test]$ gpg test-hacked.bin.gpg
gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase
gpg: WARNING: message was not integrity protected

# Aha, we got a warning about no integrity protection. But the same
warning is printed also for the unmutated blob:

[user@work-pub test]$ gpg test.bin.gpg
gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase
File `test.bin' exists. Overwrite? (y/N) n
Enter new filename: test2.bin
gpg: WARNING: message was not integrity protected

# Now let's see how the decrypted blobs were affected:

[user@work-pub test]$ xxd test.bin > test.bin.xxd
[user@work-pub test]$ xxd test-hacked.bin > test-hacked.bin.xxd
[user@work-pub test]$ diff test.bin.xxd test-hacked.bin.xxd
41685,41686c41685,41686
< 00a2d40: 0000 0000 0000 0000 0000 0000 0000 0000 ................
< 00a2d50: 0000 0000 0000 0000 0000 0000 0000 0000 ................
---
```

> 00a2d40: 0000 0000 0000 fd00 0000 0000 0036 46b3 .....6F.
> 00a2d50: 4e0f 442b dc00 0000 0000 0000 0000 0000 N.D+.....

As expected a single byte modification to the encrypted blob caused
multiple mutations in the original, decrypted blob.

Now, the question is -- how to enable integrity protection for gpg -c? I
can't find any info about this in the manual...?

joanna.

Joanna Rutkowska

unread,

Mar 28, 2013, 2:13:17 PM3/28/13

to qubes...@googlegroups.com, Marek Marczykowski, Andrew Sorensen

Apparently the --force-mdc switch seems to be what we need:

[user@work-pub test]$ gpg -c --force-mdc --compress-level 0 test.bin
[user@work-pub test]$ cp test.bin.gpg test-hacked.bin.gpg

[user@work-pub test]$ dd if=/dev/zero of=test-hacked.bin.gpg bs=1
count=1 seek=666999 conv=notrunc
1+0 records in
1+0 records out

1 byte (1 B) copied, 4.4214e-05 s, 22.6 kB/s

[user@work-pub test]$ gpg test-hacked.bin.gpg
gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase

gpg: WARNING: encrypted message has been manipulated!
[user@work-pub test]$ echo $?
2

However, notice how things break down if we used compression and if we
modified the encrypted blob then (specially, superficially enlonging it):

[user@work-pub test]$ gpg -c --force-mdc test.bin
[user@work-pub test]$ ll
total 1028

-rw-rw-r-- 1 user user 1048576 Mar 28 13:45 test.bin

-rw-rw-r-- 1 user user 1115 Mar 28 14:07 test.bin.gpg

[user@work-pub test]$ cp test.bin.gpg test-hacked.bin.gpg

[user@work-pub test]$ dd if=/dev/zero of=test-hacked.bin.gpg bs=1
count=1 seek=666999 conv=notrunc
1+0 records in
1+0 records out

1 byte (1 B) copied, 5.2511e-05 s, 19.0 kB/s
[user@work-pub test]$ ll

```
total 1036
-rw-rw-r-- 1 user user 667000 Mar 28 14:07 test-hacked.bin.gpg

-rw-rw-r-- 1 user user 1048576 Mar 28 13:45 test.bin

-rw-rw-r-- 1 user user 1115 Mar 28 14:07 test.bin.gpg

[user@work-pub test]$ gpg test-hacked.bin.gpg
gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase

gpg: [don't know]: indeterminate length for invalid packet type 10
gpg: mdc_packet with invalid encoding
gpg: decryption failed: invalid packet

gpg: encrypted with 1 passphrase

gpg: assuming IDEA encrypted data
gpg: [don't know]: invalid packet (ctb=47)

gpg: WARNING: message was not integrity protected

gpg: WARNING: multiple plaintexts seen
gpg: handle plaintext failed: unexpected data
gpg: [don't know]: invalid packet (ctb=07)
[user@work-pub test]$ echo $?
2
```

Or lets use some verbosity to see what's happening inside:

```
[user@work-pub test]$ gpg -vv test-hacked.bin.gpg
:symkey enc packet: version 4, cipher 3, s2k 3, hash 2
salt f57aa18be9a30a1a, count 65536 (96)
gpg: CAST5 encrypted data
:encrypted data packet:
length: unknown
mdc_method: 2

gpg: encrypted with 1 passphrase

:compressed packet: algo=1
:literal data packet:
mode b (62), created 1364479642, name="test.bin",
raw data: 1048576 bytes
gpg: original file name='test.bin'
File `test-hacked.bin' exists. Overwrite? (y/N) y
:unknown packet: type 44, length 10
dump: 01 48 ab a7 78 f4 ec 0a 01 48
gpg: [don't know]: indeterminate length for invalid packet type 10
gpg: mdc_packet with invalid encoding
gpg: decryption failed: invalid packet
:encrypted data packet:
length: unknown

gpg: encrypted with 1 passphrase

gpg: assuming IDEA encrypted data
gpg: [don't know]: invalid packet (ctb=47)
gpg: decryption okay
```

gpg: WARNING: message was not integrity protected

:literal data packet:
mode S (53), created 1398146990,
name="\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\
x92SV\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\x
92SV\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\x92SV\x07\xaeG\xcb\xc6\x92",
raw data: 1782 bytes
gpg: original file
name='G SV G SV G SV G SV G SV G SV G SV G SV G SV G SV G SV G SV G SV G
'
gpg: WARNING: multiple plaintexts seen
gpg: handle plaintext failed: unexpected data
gpg: [don't know]: invalid packet (ctb=07)
[user@work-pub test]$

Nice, huh!

So, we see that gpg first parses the blob significantly before first verifying its integrity. This sounds like a very bad idea to me, as it exposes many code paths in the GPG to an attack if the blob was intentionally malformed.

So, I would suggest to use openssl dgst -hmac to verify the integrity of the blob first, and only then to pass it to GPG for decryption. Or perhaps we can also use openssl to handle the decryption? Doesn't sound like a critical decision, but I think it would be more elegant to stick to just one crypto framework -- openssl in that case.

joanna.

Marek Marczykowski

unread,

Mar 28, 2013, 2:16:15 PM3/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Andrew Sorensen

Actually --force-mdc does the job:

[user@testvm gpgtest]$ gpg -c --compress-level 0 --force-mdc test.bin
(...)
[user@testvm gpgtest]$ gpg test-hacked.bin.gpg

gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase

gpg: WARNING: encrypted message has been manipulated!

[user@testvm gpgtest]$ echo $?
2

[user@testvm gpgtest]$ gpg test.bin.gpg

gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase

File `test.bin' exists. Overwrite? (y/N) y

[user@testvm gpgtest]$ echo $?
0

Marek Marczykowski

unread,

Mar 28, 2013, 2:20:12 PM3/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Andrew Sorensen

(...)

Ok, you were first :)

Marek Marczykowski

unread,

Mar 28, 2013, 10:38:48 PM3/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Andrew Sorensen

One possible problem with this approach is requirement for two pass processing
the data:
1. verify HMAC
2. decrypt and further processing
This require local store the file, or download it twice, which can be bad
(check my earlier example use case). Anyway the same data will be already
parsed by openssl during HMAC verification. Do you believe in significant
difference (in terms of security) between "openssl dgst" and "openssl enc -d"?

If we assume trust in openssl, we can use "openssl dgst (...) | openssl enc -d
(...)" and check if data were correct at the end.

The other solution for this problem is split data to chunks and apply HMAC to
each chunk separately. But this will be definitely more complex.

Any ideas? Maybe feature of partial restore without sufficient disk space for
full backup is just to much effort?

Alex Dubois

unread,

Apr 1, 2013, 8:39:42 AM4/1/13

to qubes...@googlegroups.com

Is it because no VM should run during backup/restore that the use of dispVM is not leveraged on to process the hashing&encryption/decryption? It is the default thinking of Qubes to mitigate/limit potential attacks :-)

Alex

Joanna Rutkowska

unread,

Apr 1, 2013, 10:33:04 AM4/1/13

to qubes...@googlegroups.com, Alex Dubois

I don't see how use of a DispVM could provide any benefit in this case? Imagine the gpg that runs in the DispVM (and which is to be used for verification and decryption of the backup blob) gets exploited -- it can now feed *any* backup blob (unverified, etc) to Dom0 and further compromise the whole system -- either by exploiting the next gpg that was to be run in Dom0 (if dom0 was to perform decryption) or, if Dom0 was to relay completely on the DispVM to do the decryption, by providing a malicious backup blob, which, when restored, will being compromised AppVMs, and Dom0 home even. So, we're not gaining anything by moving verification and decryption to DispVM.

joanna.

Alex Dubois

unread,

Apr 2, 2013, 8:19:56 AM4/2/13

to Joanna Rutkowska, qubes...@googlegroups.com

Alex

Very true.

I am a bit stubborn like you... What about hash+OpenSSL in dispVM and gpg in Dom0 (encapsulated)... Easy to suggest but I am yet to provide any patch...

> joanna.
>

Joanna Rutkowska

unread,

Apr 2, 2013, 8:42:16 AM4/2/13

to Alex Dubois, qubes...@googlegroups.com

I don't think you get it :/

There is no point in doing any critical crypto in DispVM. If whatever verification you wanted to do in DispVM is to be considered safe (as e.g. OpenSSL's dgst) then there is no benefit of doing it in DispVM (because it it is "safe" then we can do it in Dom0 as well). If, on the other hand, we assume it is not safe, then there is no benefit of doing it in DispVM either -- because this will not protect our Dom0 from ultimately getting malformed, unverified data.

joanna.

Joanna Rutkowska

unread,

Apr 2, 2013, 11:48:23 PM4/2/13

to Alex Dubois, qubes...@googlegroups.com

[Adding back the list]

On 04/02/13 19:25, Alex Dubois wrote:
>
>
> Alex
>

> On 2 Apr 2013, at 09:42, Joanna Rutkowska

> Unless you try to mitigate a risk by lowering its threat probability.
> OpenSSL digest in one dispVM and gpg digest in a second as the
> probability of both being unsafe (very small) is the product.
>
The above statement is not true, for the reasons I outlined before.

> This is if you feel computing a digest is unsafe. I though I could
> trust such function until you raised the point.
>
That's an incorrect interpretation of what I wrote before. I wrote that the way how gpg parses the incoming file, *before* actually veryfing its digest, is insecure.

joanna.

Andrew Sorensen

unread,

Jul 8, 2013, 1:38:24 AM7/8/13

to qubes...@googlegroups.com

Brief status update:

I have backups (with encryption or compression) to AppVM working. I still need to make adjustments to restore the "backup to a local directory" and "backup without compression or encryption" functions working, in addition to restoration.

The source is available here: https://github.com/AndrewX192/qubes-core

When I finish re-adding the existing backup functionality and add a system to restore the backup I will submit a more formal patch.

On Wednesday, March 27, 2013 11:36:35 AM UTC-7, Andrew Sorensen wrote:

> I've been working on additions to qvm-core/qubesutils.py to add functionality for compressing files in Dom0 and sending them to an appvm. Right now, I'm just using a hacked up version of qvm-backup to perform the backups, but eventually I want to make that compatible with the new system
>
> My to do the actual copy looks something like this:

compressor = subprocess.Popen (["tar", "-PcOz", file["path"]], stdout=subprocess.PIPE)

subprocess.Popen (["qvm-run", "--pass-io", "-p", appvm, "cat > " + dest_dir + file["basename"] + ".tar.gz"], stdin=compressor.stdout)

(There's an issue with this code: multiple file backups running at the same time, I have yet to fix that).

Here's some questions that would be interesting to get thoughts on:

Should backups to Dom0 be supported at all? (or should it require a working AppVM?) - What about restore?

How should encryption of backups be handled? (Dom0 sounds the best, but how should the user provide their key?) - What about users who create a "backups" appvm and use LUKS on their drive?

How should the command line qvm-backup command work with this additional functionality to backup into an appvm? qvm-backup backups:/mnt/removable/qubes-backups maybe?

Joanna Rutkowska

unread,

Aug 1, 2013, 2:23:12 PM8/1/13

to qubes...@googlegroups.com, Andrew Sorensen, Marek Marczykowski

So, one suggestion we recently discussed with Marek:

First, use qubes services instead of qvm-run. This will nicely allow to
setup policy rules regarding who can request the service (e.g. Dom0 only).

Also, take a string in qvm-backup that would specify the target
directory in the dest VMs filesystem where the backup should be stored,
e.g.:

dom0$ qvm-backup --remote-storage backupvm:/mnt/my-nas-storage

Alternaively take the name of the program to run there, e.g. s3put.

So this string should be passed to the service before the rest of the
string. Should make this really flexible and powerful.

joanna.

Marek Marczykowski-Górecki

unread,

Aug 1, 2013, 6:10:53 PM8/1/13

to Joanna Rutkowska, qubes...@googlegroups.com, Andrew Sorensen

On 01.08.2013 16:23, Joanna Rutkowska wrote:
> So, one suggestion we recently discussed with Marek:
>
> First, use qubes services instead of qvm-run. This will nicely allow to
> setup policy rules regarding who can request the service (e.g. Dom0 only).

Just to clarify: calling qubes service from dom0 is just (qvm-)run special
command "QUBESRPC <service name>".

--
Best Regards,
Marek Marczykowski-Górecki
Invisible Things Lab
A: Because it messes up the order in which people normally read text.
Q: Why is top-posting such a bad thing?

Joanna Rutkowska

unread,

Aug 1, 2013, 8:27:19 PM8/1/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Andrew Sorensen

On 08/01/13 20:10, Marek Marczykowski-Górecki wrote:
> On 01.08.2013 16:23, Joanna Rutkowska wrote:
>> So, one suggestion we recently discussed with Marek:
>>
>> First, use qubes services instead of qvm-run. This will nicely allow to
>> setup policy rules regarding who can request the service (e.g. Dom0 only).
>
> Just to clarify: calling qubes service from dom0 is just (qvm-)run special
> command "QUBESRPC <service name>".
>

Well, it is also adding a few helper files (the one with service program
in the VM, and a policy definition in Dom0).

j.

Olivier Médoc

unread,

Aug 8, 2013, 8:56:43 AM8/8/13

to qubes...@googlegroups.com

Hello,

I'm waiting for this new backup feature because I'm sharing new
templates with collegues using SSH through a VM. I'm currently using an
ugly bash script from dom0, and it is not user friendly at all.

Do you need additionnal support for this ticket ?

Olivier

Andrew Sorensen

unread,

Aug 10, 2013, 7:16:27 PM8/10/13

to qubes...@googlegroups.com, Olivier Médoc

Marek Marczykowski-Górecki

unread,

Aug 12, 2013, 2:30:02 AM8/12/13

to Andrew Sorensen, qubes...@googlegroups.com, Olivier Médoc

On 10.08.2013 21:16, Andrew Sorensen wrote:

I've looked briefly at your code and have some comments on backup format.
You create backup as directory full of files (same as the current backup
format), each encrypted separately. The user will prompted for the password
for each such file separately, which is obvious inconvenience. Also having
backup as multiple files makes it difficult to send it directly to some
external service (eg. via ncftpput). Also such multiple files backup discloses
VMs names even if the VMs data is encrypted.

IMHO good solution for all of above problems would be to create backup as one
binary blob, then encrypt it with one gpg process (on the fly). This "one
binary blob" can be single tar archive (the command to create it would be
quite long).

Restore of such one-file backup could be somehow tricky. Its all about disk
space: if you have 128GB SSD disk, and your Qubes VMs weight 100GB, you will
not have space to store full backup *and* just restored system. So restore
should be done on the flight.
The second problem is partial restore (only selected VMs): first you need to
retrieve list of VMs, then the data - only of selected VMs. The common use
case for it would be to restore the data, skipping netvm and firewallvm (which
would exists after system install). But the question is if we want to support
such scenarios in connection with external backups. I think we can answer "no"
here - so if the user want partial restore, he/she needs to provide backup on
some regular local storage connected to dom0. Still some generic settings
could apply, like "Ignore already existing VMs".

So the backup would be something like (python-based pseudocode based on your
code):
tar_cmdline = ["tar", "cS", "-C", "/var/lib/qubes"]
for file in files_to_backup:
tar_cmdline.append(file["path"].strip("/var/lib/qubes/"))
retcode = vm.run(command = "cat > {0}".format(dest_dir + "backup.gpg"),
passio_popen = True)
compressor = subprocess.Popen (["gpg", "-ac", "--force-mdc", "-o-"],
stdout=retcode.stdin)
archiver = subprocess.Popen(tar_cmdline, stdout=compressor.stdin)

"-C" option could be also useful to change current directory of tar _between_
some files (perhaps dom0 home backup case).

The restore part: in this case it is hard extract first qubes.xml, then (only
selected) backup data - so no partial restore. But maybe it isn't problem?
Maybe backup can be extracted as the whole to some directory, then each
directory *move* to the right place? This will be easy to implement based on
the current code: you need only add some code to retrieve full backup (fetch,
decrypt, unpack) from some VM and store it in local directory. Then call

original code on that directory.

Andrew Sorensen

unread,

Aug 12, 2013, 5:53:40 AM8/12/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

On 08/11/13 19:30, Marek Marczykowski-Górecki wrote:
> On 10.08.2013 21:16, Andrew Sorensen wrote:
>> On 08/08/13 01:56, Olivier Médoc wrote:
>>> Hello,
>>>
>>> I'm waiting for this new backup feature because I'm sharing new
>>> templates with collegues using SSH through a VM. I'm currently using
>>> an ugly bash script from dom0, and it is not user friendly at all.
>>>
>>> Do you need additionnal support for this ticket ?
>> I have changes that implement the backup system but not the restore
>> system: https://github.com/AndrewX192/qubes-core
> I've looked briefly at your code and have some comments on backup format.
> You create backup as directory full of files (same as the current backup
> format), each encrypted separately. The user will prompted for the password
> for each such file separately, which is obvious inconvenience. Also having
> backup as multiple files makes it difficult to send it directly to some
> external service (eg. via ncftpput). Also such multiple files backup discloses
> VMs names even if the VMs data is encrypted.

I think we can address the issue of the user needing to provide their password multiple times with gpg-agent. I just need to make sure the agent is started before the backup process is started and that the gpg instances spawned during backup have access to the current instance of the agent.

Despite the issues of information disclosure by encrypting each vm separately, I think we'd be putting some rather unfortunate limitations on the user if the backup system only natively supports backing up to a single file. It really depends on what the user expects to use the backup system for (e.g. disaster recovery or accidental file deletions).

Regardless, I'm not sure what do with qubes.xml - if it is expected that a user will do partial restores, replacing qubes.xml could cause appvms not part of the backup to disappear from the qubes-manager listing.

Olivier Médoc

unread,

Aug 12, 2013, 10:08:05 AM8/12/13

to qubes...@googlegroups.com

Extracting information from a xml file is really straigtforward in python:
import xml.dom.minidom
from xml.dom import Node

dom = xml.dom.minidom.parse(filename)
for vmnode in dom.getElementsByTagName('QubesTemplateVm'):
if vmnode.getAttribute('name') == 'fedora-18-x64':
print "VM xml node",vmnode
for vmnode in dom.getElementsByTagName('QubesAppVm'):
pass
for vmnode in dom.getElementsByTagName('QubesHVm'):
pass
...
You can also use qubes high level tools to load the collection from xml:
backup_collection = QubesVmCollection(store_filename=yourbackupxmlfile)
backup_collection = lock_db_for_reading() # Optionnal ?
backup_collection.load()

Check in qubesutils.py how the select and insert only the required VM metadata.

Marek Marczykowski-Górecki

unread,

Aug 12, 2013, 10:56:06 AM8/12/13

to Andrew Sorensen, qubes...@googlegroups.com, Olivier Médoc

On 12.08.2013 07:53, Andrew Sorensen wrote:
> On 08/11/13 19:30, Marek Marczykowski-Górecki wrote:
>> On 10.08.2013 21:16, Andrew Sorensen wrote:
>>> On 08/08/13 01:56, Olivier Médoc wrote:
>>>> Hello,
>>>>
>>>> I'm waiting for this new backup feature because I'm sharing new
>>>> templates with collegues using SSH through a VM. I'm currently using
>>>> an ugly bash script from dom0, and it is not user friendly at all.
>>>>
>>>> Do you need additionnal support for this ticket ?
>>> I have changes that implement the backup system but not the restore
>>> system: https://github.com/AndrewX192/qubes-core
>> I've looked briefly at your code and have some comments on backup format.
>> You create backup as directory full of files (same as the current backup
>> format), each encrypted separately. The user will prompted for the password
>> for each such file separately, which is obvious inconvenience. Also having
>> backup as multiple files makes it difficult to send it directly to some
>> external service (eg. via ncftpput). Also such multiple files backup discloses
>> VMs names even if the VMs data is encrypted.
> I think we can address the issue of the user needing to provide their
> password multiple times with gpg-agent. I just need to make sure the
> agent is started before the backup process is started and that the gpg
> instances spawned during backup have access to the current instance of
> the agent.

I'm not sure if gpg-agent can be used for symmetric encryption key cache...

Really, encrypted single tar archive is very simple backup format. It is even easier to handle manually (some disaster recovery case) than a bunch of separately encrypted archives.

> Regardless, I'm not sure what do with qubes.xml - if it is expected that
> a user will do partial restores, replacing qubes.xml could cause appvms
> not part of the backup to disappear from the qubes-manager listing.

Olivier already answered this - you can use QubesVmCollection to load
arbitrary qubes.xml file.

Olivier Médoc

unread,

Aug 12, 2013, 12:50:50 PM8/12/13

to qubes...@googlegroups.com

Just for documentation purpose, here are the backup use cases I found / use:

Use case A: Full system restore because of a crash: everything has to be
restored except NetVM and FirewallVMs that are already there.

Use case B: Full system restore, but I still want to restore my NetVM
because my Wireless passwords are stored within. And a VPN access is
probably a similar issue.

Use case C: Restoration of data because of a loss in an AppVM home, or
because of a broken / screwed Template. I wan't to rename my current VM
and restore an old version in order to access to the data. Or maybe
replace it completely.

Use case D: Backup of VMs that are used as Templates, or specific HVMs.
For example, in my case, prepared Windows VMs. In this case, I want to
restore only one VM.

Possible solutions for use cases B-C-D:
Solution 1 :Only the VMs that have specific names can be stored
temporarily in dom0 so that a second pass restoration can occur.
Solution 2 :Some of the VMs need to be splitted, maybe on user demand
during backup.
Solution 3 :To simplify the code of Solution 2, a new backup process can
be initiated for each VM that need to be splitted.
Solution 4 :Users that want these specific use-cases take care of the
VMs they backup and can create backups for single VMs.
Solution 5 :Implementation of a VM Template library mecanism (a new GUI)
that (re)use qubesutils.py and qvm-backup code, backing-up some VMs
separatly.
Solution 6 :Implement a solution allowing on-the-fly
decryption/verification (is on-the-fly verification possible with gpg?),
then select VMs to be restored based on the .xml file (which would be
stored in the beginning of the binary blob), then download again the
full binary blob, extracting only the required VMs. Well, this solution
looks ugly...

Of course, Use case A is the most common one, but I'm ready to implement
something for use case D even if it is for myself (in fact, I have
something with qvm-run|ssh|unencrypted which is not user friendly at all).

I don't see any good solution that solve all the use cases. Maybe you
have some idea ?

Olivier Médoc

unread,

Aug 13, 2013, 9:38:28 AM8/13/13

to qubes...@googlegroups.com

On 08/12/13 07:53, Andrew Sorensen wrote:

> On 08/11/13 19:30, Marek Marczykowski-G﷿recki wrote:
>> On 10.08.2013 21:16, Andrew Sorensen wrote:

Hello,

I tested your backup code. Some notes about optimisations:
- Use gpg -c alone. This way, you will reduce the size of the backup by
using binary data instead of ASCII armor
- Use the tar option --sparse. It will reduce by a factor of two the
time to perform a backup. (in fact the -S option proposed by Marek

Also, I tried to use two Popen calls one for tar and one for gpg2 as
proposed by Marek, linking STDINs to STDOUTs.

I can send you a patch if you are interested. Just say if you are
already working on it so that I don't do it for nothing.

Olivier

Olivier Médoc

unread,

Aug 13, 2013, 3:53:51 PM8/13/13

to qubes...@googlegroups.com

On 08/01/13 22:27, Joanna Rutkowska wrote:

> On 08/01/13 20:10, Marek Marczykowski-Gďž˝recki wrote:
>> On 01.08.2013 16:23, Joanna Rutkowska wrote:
>>> So, one suggestion we recently discussed with Marek:
>>>
>>> First, use qubes services instead of qvm-run. This will nicely allow to
>>> setup policy rules regarding who can request the service (e.g. Dom0 only).
>> Just to clarify: calling qubes service from dom0 is just (qvm-)run special
>> command "QUBESRPC <service name>".
>>
> Well, it is also adding a few helper files (the one with service program
> in the VM, and a policy definition in Dom0).
>
> j.

Playing with QUBESRPC, I have a question:

You can apparently only pass a single argument to the QUBESRPC call
(which is normally source VM ?). How can you pass an argument without
using STDIN (the command you want to run, or the destination of the

backup). Because you then have th send the backup using STDIN.

Or maybe by using some synchronization rule ? (eg: readline once for the
arguments, the rest passing to stdout).

Marek Marczykowski-Górecki

unread,

Aug 13, 2013, 4:05:10 PM8/13/13

to qubes...@googlegroups.com, Olivier Médoc

On 13.08.2013 17:53, Olivier Médoc wrote:
> On 08/01/13 22:27, Joanna Rutkowska wrote:

>> On 08/01/13 20:10, Marek Marczykowski-Górecki wrote:
>>> On 01.08.2013 16:23, Joanna Rutkowska wrote:
>>>> So, one suggestion we recently discussed with Marek:
>>>>
>>>> First, use qubes services instead of qvm-run. This will nicely allow to
>>>> setup policy rules regarding who can request the service (e.g. Dom0 only).
>>> Just to clarify: calling qubes service from dom0 is just (qvm-)run special
>>> command "QUBESRPC <service name>".
>>>
>> Well, it is also adding a few helper files (the one with service program
>> in the VM, and a policy definition in Dom0).
>>
>> j.
> Playing with QUBESRPC, I have a question:
>
> You can apparently only pass a single argument to the QUBESRPC call (which is
> normally source VM ?). How can you pass an argument without using STDIN (the
> command you want to run, or the destination of the backup). Because you then
> have th send the backup using STDIN.

Better don't do that - leave real source domain as this parameter.

> Or maybe by using some synchronization rule ? (eg: readline once for the
> arguments, the rest passing to stdout).

This is way to go, sth like:
sh -c 'read theparameter; exec destination-program $theparameter'

Olivier Médoc

unread,

Aug 13, 2013, 4:15:48 PM8/13/13

to qubes...@googlegroups.com

On 08/13/13 18:05, Marek Marczykowski-Górecki wrote:
> On 13.08.2013 17:53, Olivier Médoc wrote:
>> On 08/01/13 22:27, Joanna Rutkowska wrote:
>>> On 08/01/13 20:10, Marek Marczykowski-Górecki wrote:
>>>> On 01.08.2013 16:23, Joanna Rutkowska wrote:
>>>>> So, one suggestion we recently discussed with Marek:

>>>>>
>>>>> First, use qubes services instead of qvm-run. This will nicely allow to
>>>>> setup policy rules regarding who can request the service (e.g. Dom0 only).
>>>> Just to clarify: calling qubes service from dom0 is just (qvm-)run special
>>>> command "QUBESRPC <service name>".
>>>>
>>> Well, it is also adding a few helper files (the one with service program
>>> in the VM, and a policy definition in Dom0).
>>>
>>> j.
>> Playing with QUBESRPC, I have a question:
>>
>> You can apparently only pass a single argument to the QUBESRPC call (which is
>> normally source VM ?). How can you pass an argument without using STDIN (the
>> command you want to run, or the destination of the backup). Because you then
>> have th send the backup using STDIN.
> Better don't do that - leave real source domain as this parameter.
>
>> Or maybe by using some synchronization rule ? (eg: readline once for the
>> arguments, the rest passing to stdout).
> This is way to go, sth like:
> sh -c 'read theparameter; exec destination-program $theparameter'

Good, I got backup working :)

I go home and check later if it works by sending a ssh command instead
of a directory, but it should work.

- single tarball piped optionally to gpg2 with the optimisations I
discussed earlier
- usage of QUBESRPC
- feedback based on tar --checkpoint piped to a temporary file, and %
computations
- possible to run a command inside the VM instead of the directory
target (coded but to be tested)
- possible to run a backup to dom0 instead of a VM (coded but to be tested)

Olivier Médoc

unread,

Aug 14, 2013, 9:20:20 AM8/14/13

to qubes...@googlegroups.com

Hello,

Please find attached the patches of the features discussed below. These
patches are based on Andrew's repository.

So far, I tested:
- Backup Error handling
- Backup progress feedback
- Backup to dom0
- Backup to a VM directory
- Backup through a VM tool: tested with ssh using multiple arguments and
quotes

So far, the following things are missing:
- Qubes service policies for qubes.Backup (is it normal that it is accepted by default even if I didn't created any policy file ?)
- Restore from a VM
- Partial restore / Backup testing mecanism (to be defined, could be pre-restore, then selection of VMs to be restored, I will try several approaches). Can be achieved by extracting only qubes.xml which is in the first kbytes of the tar file (it also works if it is gpg encrypted).
One issue is that qubes.xml contains all the VMs, not the only one that have been backuped.
- GUI backup/restore adaptation to appvm or encryption selection.

On 08/13/13 18:15, Olivier M&#65533;doc wrote:
> On 08/13/13 18:05, Marek Marczykowski-G&#65533;recki wrote:

>> On 13.08.2013 17:53, Olivier M&#65533;doc wrote:
>>> On 08/01/13 22:27, Joanna Rutkowska wrote:

0001-backup-implemented-use-of-tar-gpg2-instead-of-only-e.patch

0002-backup-improved-performance-by-optimizing-tar-and-gp.patch

0003-backup-implemented-use-of-a-single-tar-file-instead-.patch

0004-backup-implemented-progress-feedback-using-tar-check.patch

0005-backup-major-revamp-of-the-backup-code-to-include-ba.patch

Marek Marczykowski-Górecki

unread,

Aug 14, 2013, 9:36:11 AM8/14/13

to qubes...@googlegroups.com, Olivier Médoc

On 14.08.2013 11:20, Olivier Médoc wrote:
> Hello,
>
> Please find attached the patches of the features discussed below. These
> patches are based on Andrew's repository.
>

> So far, I tested:
> - Backup Error handling
> - Backup progress feedback
> - Backup to dom0
> - Backup to a VM directory
> - Backup through a VM tool: tested with ssh using multiple arguments and quotes
>
> So far, the following things are missing:
> - Qubes service policies for qubes.Backup (is it normal that it is accepted by
> default even if I didn't created any policy file ?)

Actually services called _from dom0_ are always allowed (no policy even checked).

> - Restore from a VM
> - Partial restore / Backup testing mecanism (to be defined, could be
> pre-restore, then selection of VMs to be restored, I will try several
> approaches). Can be achieved by extracting only qubes.xml which is in the
> first kbytes of the tar file (it also works if it is gpg encrypted). One issue
> is that qubes.xml contains all the VMs, not the only one that have been backuped.

Perhaps backup could contain one additional file (at the beginning) with only
list of VMs?
qubes.xml needs to contain (almost) all VMs because of dependencies - if you
backup VM connected to "firewallvm" and based on "fedora-18-x64" template,
qubes.xml must contain also those dependent VMs.

> - GUI backup/restore adaptation to appvm or encryption selection.

So currently to restore such backup one needs to manually download, decrypt
and unpack the archive, right?

Joanna Rutkowska

unread,

Aug 14, 2013, 1:37:31 PM8/14/13

to qubes...@googlegroups.com, Olivier Médoc

On 08/14/13 11:20, Olivier Médoc wrote:
> So far, the following things are missing:
> - Qubes service policies for qubes.Backup (is it normal that it is
> accepted by default even if I didn't created any policy file ?)
> - Restore from a VM
> - Partial restore / Backup testing mecanism (to be defined, could be
> pre-restore, then selection of VMs to be restored, I will try several
> approaches). Can be achieved by extracting only qubes.xml which is in
> the first kbytes of the tar file (it also works if it is gpg encrypted).
> One issue is that qubes.xml contains all the VMs, not the only one that
> have been backuped.

Hm, perhaps we could, when making a backup, re-create the qubes.xml on
the fly so that it only referred the VM's that are in the backup, and
perhaps only the VMs on which those depend (netvms, templates). This way
one could make a backup of just some of the VMs, and hand those to
somebody else, not fearing of also disclosing the names of other VMs and
generally structure of their partitioning...?

This is, however, a separate issue from this ticket, I think.

joanna.

Joanna Rutkowska

unread,

Aug 14, 2013, 3:35:39 PM8/14/13

to qubes...@googlegroups.com, Olivier Médoc

In order to restore only select VMs from a backup that is severed as a
stream from another VM (our case here):

1) We send (encrypted) qubes.xml first, plus signature,
2) The qvm-backup-restore, after receiving this file and verifying the
signature offers a choice to the user which VMs to decrypt (ok, as this
is a command line tool, it should just display the ascii-art table like
it is doing now, and a Y/N prompt -- if the user wanted to exclude some
of the VMs, the tools would be restart with exceptions given as -x
options, as it is now).
3) The qvm-backup-restore continues to receive following files that were
recorded as part of the backup (it doesn't explicitly inform the VM that
it wants to continue receiving it -- just continues to do read() from
the descriptor). In case the files are for a VM that is *not* to be
restored, then this stream of bytes goes into /dev/null, otherwise it is
being stored in Dom0 tmp, then verified and then mv-ed to specific
directory under /var/lib/qubes/.

So all we need is that our restoring VM provides us with a stream
similar to what our qvm-copy-to-vm does. Except that we also need
signatures (HMACs) of some sort for each of those files.

For simplicity, I think there should be no additional Dom0->VM
communication (and then VM->Dom0 responses) -- the VM always servers the
same stream, just that some parts of it go into /dev/null on Dom0 side.

joanna.

Joanna Rutkowska

unread,

Aug 14, 2013, 3:39:41 PM8/14/13

to qubes...@googlegroups.com, Olivier Médoc

So, just to clarify, it's all about preparing the original blob stream
in a smart way (header, qubes.xml, hmac, header, file-XXX, hmac, etc).
This should be done by qvm-backup, of course. Perhaps the qubes.xml
could be re-created on the fly, as I discussed in another message, to
refer to only the actual VMs that are part of the backup (plus deps).
The agent in the VM that serves this back to Dom0 is stupid -- it just
writes this all stream to the descriptor.

joanna.

Marek Marczykowski-Górecki

unread,

Aug 14, 2013, 5:36:55 PM8/14/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

I'm quite against inventing new "file format" for that. For simple reason: to be able to access the data without full Qubes system. In fact backups are for recovery situations.

So perhaps still use tar archive, but place files in the right order - as proposed by Joanna. The question is if it is possible to "pause" unpacking files right after qubes.xml+hmac and wait for user confirmation.

Joanna Rutkowska

unread,

Aug 14, 2013, 7:43:49 PM8/14/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

Ok, so tar with files sequenced like this:

1) qubes.xml
2) qubes.xml's hmac
3) file1
4) file1's hmac
5) etc.

>
> So perhaps still use tar archive, but place files in the right order - as
> proposed by Joanna. The question is if it is possible to "pause" unpacking
> files right after qubes.xml+hmac and wait for user confirmation.
>

The vchan should support it, no? I.e. when the receiver stops read()ing from the descriptor?

j.

Marek Marczykowski-Górecki

unread,

Aug 14, 2013, 7:50:54 PM8/14/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

Yes. The question is how to hold tar process (and when).

Joanna Rutkowska

unread,

Aug 14, 2013, 8:03:57 PM8/14/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

Perhaps we can start tar twice:
1) > head -n XXX | tar x | ... (plus, save the original stream as header)
2) replay the header, then continue reading the socket, and pipe through
tar again.

?

Marek Marczykowski-Górecki

unread,

Aug 14, 2013, 8:09:30 PM8/14/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

This answers only the first question...
But indeed this is a good idea. Maybe tar can be commanded to terminate just
after extracting requested files (qubes.xml, qubes.xml.hmac)?

Andrew Sorensen

unread,

Aug 16, 2013, 5:45:37 AM8/16/13

to qubes...@googlegroups.com, Olivier Médoc

On 08/14/13 02:20, Olivier M&#65533;doc wrote:
> Hello,
>
> Please find attached the patches of the features discussed below.
> These patches are based on Andrew's repository.
>
> So far, I tested:
> - Backup Error handling
> - Backup progress feedback
> - Backup to dom0
> - Backup to a VM directory
> - Backup through a VM tool: tested with ssh using multiple arguments
> and quotes
>
> So far, the following things are missing:
> - Qubes service policies for qubes.Backup (is it normal that it is
> accepted by default even if I didn't created any policy file ?)
> - Restore from a VM
> - Partial restore / Backup testing mecanism (to be defined, could be
> pre-restore, then selection of VMs to be restored, I will try several
> approaches). Can be achieved by extracting only qubes.xml which is in
> the first kbytes of the tar file (it also works if it is gpg
> encrypted). One issue is that qubes.xml contains all the VMs, not the
> only one that have been backuped.
> - GUI backup/restore adaptation to appvm or encryption selection.
>

These patches have been merged into the github repository. I intended to

work on this system further but currently I'm using LVM in my qubes installation to provide support for snapshots and consequently am outside of the scope of the built in backup system.

Olivier Médoc

unread,

Aug 16, 2013, 7:16:00 AM8/16/13

to qubes...@googlegroups.com

Here is a patch for an example of reading the headers (only tested for the dom0 backup). Sorry if the patch is not clean, I'm just testing things.

- I created an additionnal file for the VMs that are in the backup and put this file first in the tar file
- I only read the first 4k of the file, check if it is a gzip (.tar), if not, I try to decrypt it.
- Only untar the file I created during backup (because it is in the first 4k.

Now of course there is no signature verification if the file is unencrypted. When encrypted, I don't know if the verification process is working with gpg considering:
- We enabled --force-mdc during encryption
- We only read a part of the encrypted file and gpg fails during decryption process (but it decrypted the first part).

How can I verify if it works correctly ? Is it sufficient if I just change a byte of the encrypted file and try to decrypt it again ?

For information, tar store files descriptors just before each file, so you have to read the whole file to know its full content.

0001-backup-implemented-mecanism-to-read-only-the-backup-.patch

Joanna Rutkowska

unread,

Aug 16, 2013, 10:03:01 AM8/16/13

to qubes...@googlegroups.com, Olivier Médoc

So, why not re-create qubes.xml instead of adding additional somehow redundant data?

> - I only read the first 4k of the file, check if it is a gzip (.tar), if
> not, I try to decrypt it.

Never ever we should expect Dom0 to perform such a complex operation as gzip -d on an untrusted stream of bytes. gzip is a very complex thing to process. Doing tar -x is somehow acceptable, although I would feel more confident in reusing our special cpio-like format that we came out for

our inter-VM file and dir transfer (used by qvm-copy-to-vm). We created our own packer/unpacker exactly for the purpose of avoiding to trust a more complex tar -x to do the job.

I understand the Marek's argument about keeping backups in a universally understood format (so tar, not Qubes own cpio-like format), although, this tradeoff is not that obvious to me after I thought about it a bit more.

Perhaps we should offer an option to the user which format to use: tar vs. Qubes internal one?

Optionally: how about adjusting our format used in qvm-copy-to-vm to be a compatible subset of either tar or cpio? We would still use our own unpacker.c for handling this (both in qvm-copy-to-vm, as well as in qvm-backup-restore), but we will automatically allow the backups to be also untar-able (or uncpio-able) by vanilla tools.

> - Only untar the file I created during backup (because it is in the
> first 4k.
>

I kinda don't like the idea of choosing some pre-defined lenght, such as 4k. We're operating on stream-like file descriptors, and we could stop reading after any number of bytes (= exact len of file we want).

> Now of course there is no signature verification if the file is
> unencrypted. When encrypted, I don't know if the verification process is
> working with gpg considering:
> - We enabled --force-mdc during encryption
> - We only read a part of the encrypted file and gpg fails during
> decryption process (but it decrypted the first part).
>

Regading the use of gpg -- see the discussion here (and down):

https://groups.google.com/d/msg/qubes-devel/TQr_QcXIVww/5ofgl3KFhAMJ

> How can I verify if it works correctly ? Is it sufficient if I just
> change a byte of the encrypted file and try to decrypt it again ?
>
> For information, tar store files descriptors just before each file, so
> you have to read the whole file to know its full content.
>

Hm, not sure what you mean?

joanna.

Zrubecz Laszlo

unread,

Aug 16, 2013, 11:01:03 AM8/16/13

to qubes...@googlegroups.com

On 16 August 2013 12:03, Joanna Rutkowska <joa...@invisiblethingslab.com> wrote:

> Optionally: how about adjusting our format used in qvm-copy-to-vm to be

> a compatible subset of either tar or cpio? We would still use our own
> unpacker.c for handling this (both in qvm-copy-to-vm, as well as in
> qvm-backup-restore), but we will automatically allow the backups to be
> also untar-able (or uncpio-able) by vanilla tools.

Just jumping in this thread:

This is a really goond option, I would go this way... if it's count :)

--
Zrubi

Olivier Médoc

unread,

Aug 16, 2013, 1:26:34 PM8/16/13

to qubes...@googlegroups.com

On 08/16/13 12:03, Joanna Rutkowska wrote:
> On 08/16/13 09:16, Olivier Mďž˝doc wrote:

>> On 08/14/13 22:09, Marek Marczykowski-Gďž˝recki wrote:
>>> On 14.08.2013 22:03, Joanna Rutkowska wrote:

>>>> On 08/14/13 21:50, Marek Marczykowski-Gďž˝recki wrote:
>>>>> On 14.08.2013 21:43, Joanna Rutkowska wrote:

>>>>>> On 08/14/13 19:36, Marek Marczykowski-Gďž˝recki wrote:
>>>>>>> On 14.08.2013 17:39, Joanna Rutkowska wrote:
>>>>>>>> On 08/14/13 17:35, Joanna Rutkowska wrote:

>>>>>>>>> On 08/12/13 14:50, Olivier Mďž˝doc wrote:

>>>>>>>>>> On 08/12/13 12:56, Marek Marczykowski-Gďž˝recki wrote:
>>>>>>>>>>> On 12.08.2013 07:53, Andrew Sorensen wrote:

>>>>>>>>>>>> On 08/11/13 19:30, Marek Marczykowski-Gďž˝recki wrote:
>>>>>>>>>>>>> On 10.08.2013 21:16, Andrew Sorensen wrote:

Yes, I did that temporarily to quickly be able to test the different
options we are proposing in this thread. Now I'm quite lost in this
thread: How could I know which VMs are inside my backup only based on
qubes.xml ?

>> - I only read the first 4k of the file, check if it is a gzip (.tar), if
>> not, I try to decrypt it.
> Never ever we should expect Dom0 to perform such a complex operation as
> gzip -d on an untrusted stream of bytes. gzip is a very complex thing to
> process. Doing tar -x is somehow acceptable, although I would feel more
> confident in reusing our special cpio-like format that we came out for
> our inter-VM file and dir transfer (used by qvm-copy-to-vm). We created
> our own packer/unpacker exactly for the purpose of avoiding to trust a
> more complex tar -x to do the job.

In fact I'm using the command file to check if there is the gzip
signature. But I'm still running tar -xz on untrusted data. Except if it
is encrypted and if the decryption process allows my to guaranty the

integrity of my backup, in this case the backup should be trusted...

Maybe file is still prone to parsing errors and should not be used at
all. In this case, the user will have to explicitly say if the file is
encrypted or not ?

Maybe we can allow using tar -z only if there is encryption enabled ?

> I understand the Marek's argument about keeping backups in a universally
> understood format (so tar, not Qubes own cpio-like format), although,
> this tradeoff is not that obvious to me after I thought about it a bit more.
>
> Perhaps we should offer an option to the user which format to use: tar
> vs. Qubes internal one?
>
> Optionally: how about adjusting our format used in qvm-copy-to-vm to be
> a compatible subset of either tar or cpio? We would still use our own
> unpacker.c for handling this (both in qvm-copy-to-vm, as well as in
> qvm-backup-restore), but we will automatically allow the backups to be
> also untar-able (or uncpio-able) by vanilla tools.
>
>> - Only untar the file I created during backup (because it is in the
>> first 4k.
>>
> I kinda don't like the idea of choosing some pre-defined lenght, such as
> 4k. We're operating on stream-like file descriptors, and we could stop
> reading after any number of bytes (= exact len of file we want).

No problem with that. It is again only a matter of testing quickly if it
is feasible. I can probably let tar extract qubes.xml until it finished
and then stop reading the stdout. But I don't see how to do it without
reading blocks by blocks. At least I don't have ideas of how to do that
with subprocess.Popen...

>> Now of course there is no signature verification if the file is
>> unencrypted. When encrypted, I don't know if the verification process is
>> working with gpg considering:
>> - We enabled --force-mdc during encryption
>> - We only read a part of the encrypted file and gpg fails during
>> decryption process (but it decrypted the first part).
>>
> Regading the use of gpg -- see the discussion here (and down):
>
> https://groups.google.com/d/msg/qubes-devel/TQr_QcXIVww/5ofgl3KFhAMJ

Thanks, I forgot this point.

So I have to find some other integrity validation mecanism than gpg...
Maybe by switching to openssl as you suggested.

>> How can I verify if it works correctly ? Is it sufficient if I just
>> change a byte of the encrypted file and try to decrypt it again ?
>>
>> For information, tar store files descriptors just before each file, so
>> you have to read the whole file to know its full content.
>>
> Hm, not sure what you mean?

No problem, I have my answer in the gpg discussion. joanna.

Marek Marczykowski-Górecki

unread,

Aug 16, 2013, 8:58:49 PM8/16/13

to qubes...@googlegroups.com, Olivier Médoc

On 16.08.2013 15:26, Olivier Médoc wrote:
> On 08/16/13 12:03, Joanna Rutkowska wrote:

>> On 08/16/13 09:16, Olivier Médoc wrote:

>>> On 08/14/13 22:09, Marek Marczykowski-Górecki wrote:
>>>> On 14.08.2013 22:03, Joanna Rutkowska wrote:
>>>>> Perhaps we can start tar twice:
>>>>> 1) > head -n XXX | tar x | ... (plus, save the original stream as
>>>>> header)
>>>>> 2) replay the header, then continue reading the socket, and pipe through
>>>>> tar again.
>>>> This answers only the first question...
>>>> But indeed this is a good idea. Maybe tar can be commanded to
>>>> terminate just
>>>> after extracting requested files (qubes.xml, qubes.xml.hmac)?
>>>>
>>> Here is a patch for an example of reading the headers (only tested for
>>> the dom0 backup). Sorry if the patch is not clean, I'm just testing things.
>>>
>>> - I created an additionnal file for the VMs that are in the backup and
>>> put this file first in the tar file
>> So, why not re-create qubes.xml instead of adding additional somehow
>> redundant data?
> Yes, I did that temporarily to quickly be able to test the different options
> we are proposing in this thread. Now I'm quite lost in this thread: How could
> I know which VMs are inside my backup only based on qubes.xml ?

You can't. Joanna says about re-creating (during backup) qubes.xml with only VMs contained in the backup. But still some additional VMs must be included in such qubes.xml (like templates, netvms etc) - even if not included in the backup. So this doesn't solve the problem. I'm for the solution with simple VMs list at the beginnig of the archive.

>>> - I only read the first 4k of the file, check if it is a gzip (.tar), if
>>> not, I try to decrypt it.
>> Never ever we should expect Dom0 to perform such a complex operation as
>> gzip -d on an untrusted stream of bytes. gzip is a very complex thing to
>> process. Doing tar -x is somehow acceptable, although I would feel more
>> confident in reusing our special cpio-like format that we came out for
>> our inter-VM file and dir transfer (used by qvm-copy-to-vm). We created
>> our own packer/unpacker exactly for the purpose of avoiding to trust a
>> more complex tar -x to do the job.
> In fact I'm using the command file to check if there is the gzip signature.
> But I'm still running tar -xz on untrusted data. Except if it is encrypted and
> if the decryption process allows my to guaranty the integrity of my backup, in
> this case the backup should be trusted...

IMHO it is ok to run tar -x on the data if backup is not encrypted by gpg.
This means that user have done something else to protect the data (like LUKS).
If not - this is already game over if attacker get hands on this data.

But in case of encrypted backup (gpg) we can't assume any external data
protection, so must somehow verify the data before processing. IMHO gpg (if it
ensures integrity protection) and tar is still ok (tar format isn't much more
complicated than our protocol...). But if gpg doesn't ensures data integrity
(which seems to be the case...), each file must be somehow additionally
verified before further processing - like placing the second file .hmac right
after it. See discussion linked by Joanna.

>
> Maybe file is still prone to parsing errors and should not be used at all. In
> this case, the user will have to explicitly say if the file is encrypted or not ?

Yes, I'd avoid using 'file' tool on untrusted data. Some user-provided option
looks like a good idea.

> Maybe we can allow using tar -z only if there is encryption enabled ?

AFAIR gpg by default compresses the data.

Joanna Rutkowska

unread,

Aug 17, 2013, 8:15:01 AM8/17/13

to qubes...@googlegroups.com, Marek Marczykowski-Górecki, Olivier Médoc

Let me reiterate something I already said in this thread long time ago:
there is NO SENSE in sending an unencrypted backup to some other AppVM.
This is because in that case this other AppVM becomes essentially as
trusted as the Dom0 itself. And this is something we don't want, really.
Especially if the AppVM has networking enabled.

> But in case of encrypted backup (gpg) we can't assume any external data
> protection, so must somehow verify the data before processing. IMHO gpg (if it
> ensures integrity protection) and tar is still ok (tar format isn't much more
> complicated than our protocol...). But if gpg doesn't ensures data integrity
> (which seems to be the case...), each file must be somehow additionally
> verified before further processing - like placing the second file .hmac right
> after it. See discussion linked by Joanna.
>

IIRC, gpg did waaay to much parsing (related decyrpting) before veryfing
the signature, which we agreed was wrong. Openssl seemed better in this
respect.

>>
>> Maybe file is still prone to parsing errors and should not be used at all. In
>> this case, the user will have to explicitly say if the file is encrypted or not ?
>
> Yes, I'd avoid using 'file' tool on untrusted data. Some user-provided option
> looks like a good idea.
>

Let me reiterate: we would like to minimize the amount of parsing done
by Dom0 of any untrusted data. Please always keep in mind, that coming
up with half-baked security solution for Qubes makes no sense, because
in that case one can just use OSX or Windows. Other than security they
have quite an edge on Qubes OS ;)

>> Maybe we can allow using tar -z only if there is encryption enabled ?
>
> AFAIR gpg by default compresses the data.
>

joanna.

Marek Marczykowski-Górecki

unread,

Aug 17, 2013, 2:48:38 PM8/17/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

Yes, but simple tar-ed backup can be used with local storage (eg. attached via
qvm-block and LUKS-ed in dom0). IMHO it's better to switch to single-file
backup format even for local backup - this will for example ease operations on
non-ext4 filesystems (IOW with worse sparse-file support).

>
>> But in case of encrypted backup (gpg) we can't assume any external data
>> protection, so must somehow verify the data before processing. IMHO gpg (if it
>> ensures integrity protection) and tar is still ok (tar format isn't much more
>> complicated than our protocol...). But if gpg doesn't ensures data integrity
>> (which seems to be the case...), each file must be somehow additionally
>> verified before further processing - like placing the second file .hmac right
>> after it. See discussion linked by Joanna.
>>
>
> IIRC, gpg did waaay to much parsing (related decyrpting) before veryfing
> the signature, which we agreed was wrong. Openssl seemed better in this
> respect.

We need to verify initial part of backup (vms list, perhaps also qubes.xml)
before reading the whole blob. This initial part also needs to be encrypted.
So we need sth like:

backup blob -> (1)stream decryption -> (2)select initial part -> (3)verify it

Then select VMs to restore and continue with the rest of data.

Current solution:
(1) - gpg
(2) - hardcoded 4k, but easily doable by selecting exactly two first files
from tar archive
(3) - openssl?

By "gpg did waaay to much parsing (related decyrpting) before veryfing the
signature" you basically mean that gpg isn't usable to encrypt the backup in
our case, right? IIUC ideal solution would be to first verify all the data
intergiry, then decrypt it and pass to restore core. But the "verify all the

data integrity" part needs access to the whole backup blob (so store it somewhere), which isn't doable as I said earlier...
Perhaps some block-based data integrity check (so can be done on the fly), but I'm not aware of any existing solution like this. As said earlier I'd prefer to keep our backup format as simple as possible - IOW to be extractable also with simple tools (without QubesOS).

IMHO the best compromise here is to encrypt/decrypt the data with gpg (symmetric) then use tar for archive and each file keep with attached hmac (additional file right after).
This means that untrusted data is processed by gpg (decryption), then by tar and then by openssl for hmac verification.
Note that if you don't trust gpg in regard of parsing untrusted data (decryption here), probably much more components must be also reimplemented (like dom0 updates - rpm signatures, installation ISO signatures, enigmail for thunderbird etc). So IMO we don't have really the choice about the trust in gpg.

>>>
>>> Maybe file is still prone to parsing errors and should not be used at all. In
>>> this case, the user will have to explicitly say if the file is encrypted or not ?
>>
>> Yes, I'd avoid using 'file' tool on untrusted data. Some user-provided option
>> looks like a good idea.
>>
>
> Let me reiterate: we would like to minimize the amount of parsing done
> by Dom0 of any untrusted data. Please always keep in mind, that coming
> up with half-baked security solution for Qubes makes no sense, because
> in that case one can just use OSX or Windows. Other than security they
> have quite an edge on Qubes OS ;)
>
>>> Maybe we can allow using tar -z only if there is encryption enabled ?
>>
>> AFAIR gpg by default compresses the data.
>>
>
> joanna.
>

Franz

unread,

Aug 17, 2013, 10:50:29 PM8/17/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

Joanna Rutkowska

unread,

Aug 18, 2013, 5:18:48 PM8/18/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

We ain't need no stinkin' gpg to trust!

Perhaps instead of encrypting the whole blob, we should encrypt each file first (vm-lists, qubes.xml, personal/root.img, etc) and then concatenated them together via tar/cpio.

Then, when reading the backup, we first extract files from the tar archive, then verify the hmac's for each files (they should follow each encrypted file in the tar), and only then do the actual decryption.

Again, I would sleep better if we used our own unpacker.c (that we already use for qvm-copy-to-vm) for extraction of those files instead of tar -x. Again, as I wrote somewhere before, perhaps we could just adjust our format to be untar-able by stock tar (yet we will still use our own unpacker.c, as it is the most critical processing we do here).

joanna.

Joanna Rutkowska

unread,

Aug 18, 2013, 5:20:20 PM8/18/13

to Franz, qubes...@googlegroups.com, Olivier Médoc

Nah, we don't really want to make (or support even) 101 addons to Qubes, each of those weakening some part of the system, so that if one installs all of them you end up with something worse than if you used Android or XP ;)

joanna.

Olivier Médoc

unread,

Aug 19, 2013, 7:39:26 AM8/19/13

to qubes...@googlegroups.com

On 08/18/13 19:18, Joanna Rutkowska wrote:

Hello,

I'm starting to understand better what you want to achieve.

> Perhaps instead of encrypting the whole blob, we should encrypt each
> file first (vm-lists, qubes.xml, personal/root.img, etc) and then
> concatenated them together via tar/cpio.

One issue is that during a full backup, the size of dom0 will be doubled. Except if we manage to send each encrypted file to tar/cpio to a virtual PIPE/file without having to store all of them temporarily, but I don't think tar or cpio are able to achieve that. From what I checked, it is possible to concatenate multiple tar archives, but at each time, the tar headers have to be rewritten.

Note that this is not a problem for the HMAC, which can be created for each file.

That is probably why Andrew needed to run the encryption pass AFTER the tar pass, and I kept it like that.

> Then, when reading the backup, we first extract files from the tar
> archive, then verify the hmac's for each files (they should follow each
> encrypted file in the tar), and only then do the actual decryption.
>
> Again, I would sleep better if we used our own unpacker.c (that we
> already use for qvm-copy-to-vm) for extraction of those files instead of
> tar -x. Again, as I wrote somewhere before, perhaps we could just adjust
> our format to be untar-able by stock tar (yet we will still use our own
> unpacker.c, as it is the most critical processing we do here).
>
> joanna.

Well, I think the tar archive format is already quite complicated (my own point of view). Maybe cpio propose several archive formats that are more simple ?

Olivier Médoc

unread,

Aug 19, 2013, 3:06:09 PM8/19/13

to qubes...@googlegroups.com

Before anything else:

Here are the last patch I just tested to make restoration working based on the changes initiated by Andrew: tar + compression + backup_specification_file + gpg2 (option) + AppVM call through qubesrpc (option).

I also started to cleanup the code a little bit.

I think starting from this codebase will be better to test all the things discussed in this thread (because we can test both backup AND restoration).

I plan to start with;
- removing the backup verification using the command 'file'
- testing openssl encryption + verification
- testing openssl HMAC generation (password based ?)
- making restoration working with a single call instead of two calls (one to extract headers, one for the real restoration process)
- encryption of each file separatly "before" using tar ? (even if I have currently no idea how to implement that)
- use of Qubes unpacker.c code ?

On 08/19/13 09:39, Olivier M&#65533;doc wrote:
> On 08/18/13 19:18, Joanna Rutkowska wrote:

0002-backup-reimplemented-restore-function-through-an-App.patch

0003-backup-Added-rpc-restoration-file.patch

Joanna Rutkowska

unread,

Aug 20, 2013, 7:06:50 PM8/20/13

to qubes...@googlegroups.com, Olivier Médoc

On 08/19/13 17:06, Olivier Médoc wrote:
> Before anything else:
>
> Here are the last patch I just tested to make restoration working based
> on the changes initiated by Andrew: tar + compression +
> backup_specification_file + gpg2 (option) + AppVM call through qubesrpc
> (option).
>
> I also started to cleanup the code a little bit.
>
> I think starting from this codebase will be better to test all the
> things discussed in this thread (because we can test both backup AND
> restoration).
>
> I plan to start with;
> - removing the backup verification using the command 'file'
> - testing openssl encryption + verification
> - testing openssl HMAC generation (password based ?)
> - making restoration working with a single call instead of two calls
> (one to extract headers, one for the real restoration process)
> - encryption of each file separatly "before" using tar ? (even if I have
> currently no idea how to implement that)
> - use of Qubes unpacker.c code ?
>

Hi Olivier,

So, because this threat has become a bit of a mess ;) -- could you write again how do you understand and how are you planning to implement the backup finally? Like what will be the structure of the "stream", and how is it going to be processes in Dom0 upon restoring from backup? It surely is better to catch any misunderstandings now, before you actually spend time coding things.

joanna.

> On 08/19/13 09:39, Olivier Médoc wrote:
>> On 08/18/13 19:18, Joanna Rutkowska wrote:

Axon

unread,

Aug 22, 2013, 4:02:08 AM8/22/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

It sounds like the primary dilemma hasn't been resolved, though. Essentially, we have three seemingly incompatible goals:

(1) Do not perform complex operations on untrusted data in dom0.
(2) Do not delegate critical operations to AppVMs.
(3) Do not require initial copying of the entire backup blob to dom0.

And the additional requirement:

(4) Do not require Qubes-specific tools to recover plaintext from the backup.

(1) requires that we verify the integrity and authenticity of the backup before decrypting it in dom0, since failure to do so exposes us to DoS attacks (or worse). (2) means that we can't delegate the tasks of verification or decryption to any AppVM (incl. any DispVM), since any attack which could compromise dom0 could also compromise the AppVM, which could in turn pass compromised data to dom0. Therefore, it must be performed in dom0. In conjunction with (1), this entails that the backup blob must be verified (in dom0) before any complex operations are performed on it. (3) rules out solutions which require an amount of free disk space in dom0 equal to or greater than the total size of the backup blob. (4) is not part of the dilemma proper, but constrains the tools which can be used to those which are not Qubes-specific. This is so that recovery from the backup is possible in the event that no trusted Qubes system is available.

Outcomes:

(A) We achieve (2) and (3), only if we forego (1).
(B) We achieve (1) and (3), only if we forego (2).
(C) We achieve (1) and (2), only if we forego (3).

Summary of Outcomes:

(A): Stream from AppVM --> dom0 decryption --> dom0 verification.
(B): Local access from AppVM --> AppVM verification --> dom0 decryption.
(C): Copy to dom0 --> dom0 verification --> dom0 decryption.

If this is truly an accurate description of the dilemma, and if it is truly a dilemma (i.e. there is no way out; we must choose one of the three outcomes), then the obvious choice from a security perspective is (C), since this is the only outcome of the trio in which we're not sacrificing security (instead, we're sacrificing disk space).

However, it seems like there should be a way out. For example, could we in theory eliminate the need to worry about (1) in the following sort of scenario? Imagine that you have two identical physical machines ready to be restored from your backup blob. The first machine is sacrificial. On the first machine, you do as much parsing or even decryption of the blob in dom0 as you want. You then attempt to verify its integrity. If an attacker has tampered with your blob, then dom0 is now compromised. If not, then you can become sure that the blob is verified. Only if you're sure of the blob's integrity do you then confidently restore from the blob on the second machine. Or is the problem that dom0 might be compromised without our having any way to tell that it is? In that case,

this sort of "disp dom0 clone" idea won't work.

Last point (unrelated to this, but on-topic): I strongly agree with Marek's endorsement of a symmetric cipher for encrypting the backup (the obvious choice is AES), but perhaps you're planning on giving the user a choice about this anyway.

Reason 1: The primary purpose of a backup is so that a single user is able to recover important data in the event of loss. Therefore, there's no need to use a public key to encrypt this data. (Of course, a backup *can* be shared or sent to someone else, but then it's probably more accurate to call it something other than a "backup.")

Reason 2: Backups can and typically *should* be stored or duplicated offsite, which typically means that they will be stored on servers outside the user's control (which, we should assume, makes these servers untrusted). This means that they can be copied without the user's knowledge, stored elsewhere, analyzed, and brute forced for an indefinite period of time. It's becoming increasingly clear that asymmetric ciphers will be considerably less secure than symmetric ciphers against potential advances in quantum computing as time goes on.[1][2][3][4][5][6] Furthermore, the data stored in a whole-system backup is likely to be sensitive for a longer period of time than, e.g., a public-key encrypted email, which makes it more critical to do what we can to maintain backup confidentiality for as long as possible.

[1] Peter W. Shor. 1995. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer".
[2] Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen (eds.) 2009. /Post-quantum cryptography/.
[3] Daniel J. Bernstein. 2009. "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?".
[4] Daniel J. Bernstein. 2010. "Grover vs. McEliece".
[5] http://www.pqcrypto.org
[6] https://www.schneier.com/blog/archives/2008/10/quantum_cryptog.html

Olivier Médoc

unread,

Aug 22, 2013, 6:07:04 AM8/22/13

to qubes...@googlegroups.com

On 08/22/13 06:02, Axon wrote:
> On 08/20/13 12:06, Joanna Rutkowska wrote:

You described exactly the problem I see. One solution I foresee is the solution C, using "checkpoint verification" (verify every n bytes chunck). This would solve the issue number 3 because we still retrieve the whole backup but we can check it and extract the right parts on the fly. That was the initial goal of gpg --force-mdc exept that it seems to do too much parsing.

One example of implementation would be: create a HMAC for the first MB and then every 100MB or 200MB using openssl and a user password. During restore, every 100MB run the verify process on the stream before forwarding it to tar/openssl decryption/gz processes. Tar can then

extract only the required parts of the backup.

I don't know tools to perform that automatically, but I could probably perform it manually using openssl. That is why I tried to quickly finish to implement the complete backup/restore process even with the wrong tools/backup formats. In order to experiment such things, and check the performance issues (who want to wait half a day to perform a backup/restore?).

Now that the data is separated by domain/context (thanks Qubes principles), the user can choose to handle differently the most critical data, with a manual intervention (encrypt himself the backup of his critical VMs, store it on an encrypted hard drive ...). The same way he is able to remove completely network from the critical VMs -- hello vault VM :) --

Joanna Rutkowska

unread,

Aug 27, 2013, 8:08:00 PM8/27/13

to qubes...@googlegroups.com, Olivier Médoc

On 08/22/13 08:07, Olivier Médoc wrote:
> On 08/22/13 06:02, Axon wrote:
>> On 08/20/13 12:06, Joanna Rutkowska wrote:

So, what about the idea I expressed in one of the previous message: do (safe) untar (unpack.c) on the stream, then, on a file-by-file basis, do signature verification before decrypting and parsing each file?

Again, we can use stock tar in the first version, and then move on towards improved unpack.c, the same we use file inter-VM file transfers, only that we modify it so that it was compatible with tar or cpio.

>
>> Summary of Outcomes:
>>
>> (A): Stream from AppVM --> dom0 decryption --> dom0 verification.
>> (B): Local access from AppVM --> AppVM verification --> dom0 decryption.
>> (C): Copy to dom0 --> dom0 verification --> dom0 decryption.
>>
>> If this is truly an accurate description of the dilemma, and if it is
>> truly a dilemma (i.e. there is no way out; we must choose one of the
>> three outcomes), then the obvious choice from a security perspective
>> is (C), since this is the only outcome of the trio in which we're not
>> sacrificing security (instead, we're sacrificing disk space).
>>
>> However, it seems like there should be a way out. For example, could
>> we in theory eliminate the need to worry about (1) in the following
>> sort of scenario? Imagine that you have two identical physical
>> machines ready to be restored from your backup blob. The first machine
>> is sacrificial. On the first machine, you do as much parsing or even
>> decryption of the blob in dom0 as you want. You then attempt to verify
>> its integrity. If an attacker has tampered with your blob, then dom0
>> is now compromised. If not, then you can become sure that the blob is
>> verified. Only if you're sure of the blob's integrity do you then

>> confidently restore from the blob on the second machine. Or is the
>> problem that dom0 might be compromised without our having any way to
>> tell that it is? In that case, this sort of "disp dom0 clone" idea
>> won't work.
>>

Surely it is a problem. Never we should assume that we can easily detect an OS compromise. I spend quite a few years of my life on this problem, and ultimately gave up.

>> Last point (unrelated to this, but on-topic): I strongly agree with
>> Marek's endorsement of a symmetric cipher for encrypting the backup
>> (the obvious choice is AES), but perhaps you're planning on giving the
>> user a choice about this anyway.
>>

Yes, we surely want symmetric crypto (e.g. to resist future quantum cryptoanalisys), but I don't think it changes anything in our backup design, does it? I don't think anybody even suggested to use public key crypto for backup encryption?

joanna.

Axon

unread,

Aug 27, 2013, 8:42:28 PM8/27/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

Ah, yes. Even if, at a given point in time, we were able to detect every kind of compromise we care about, we should not be so foolhardy as to think that no one will ever find any new method of compromise in the future before we do.

>
>>> Last point (unrelated to this, but on-topic): I strongly agree with
>>> Marek's endorsement of a symmetric cipher for encrypting the backup
>>> (the obvious choice is AES), but perhaps you're planning on giving the
>>> user a choice about this anyway.
>>>
>
> Yes, we surely want symmetric crypto (e.g. to resist future quantum
> cryptoanalisys), but I don't think it changes anything in our backup
> design, does it? I don't think anybody even suggested to use public key
> crypto for backup encryption?

You are correct on both. I was just voicing my endorsement of this, but if it was unnecessary and obvious to everyone else already, so much the better. :)

Olivier Médoc

unread,

Aug 28, 2013, 8:39:44 AM8/28/13

to qubes...@googlegroups.com

On 08/27/13 22:08, Joanna Rutkowska wrote:

> On 08/22/13 08:07, Olivier M﷿doc wrote:
>> On 08/22/13 06:02, Axon wrote:
>>> On 08/20/13 12:06, Joanna Rutkowska wrote:

The idea is good, but I have a space problem when preparing the backup.
I have 12G of free space, and the simplest pseudo algorithm would be:
For each file to backup:
Encrypt the file and store it in a temporary file
Create a HMAC and store it along on the temporary file
Tar all the files of the temporary directory and send it to the backup
target through a PIPE (either a file or AppVM file/tool)

About the tar format:
"Physically, an archive consists of a series of file entries
terminated by an end-of-archive entry, which consists of two 512 blocks
of zero bytes.". Also tar should accept without error an archive that
does not contains a end of archive marker.

The idea behind that is that we can programatically remove each end of
archive marker at the end of each tar passe (in dom0), so that the file
in the AppVM is never closed. Such a file should be readable by tar,
even if there is no end of archive marker (which we can keep for the
last file anyway). This is basically what the tar --concatenate option
performs, except that we don't want to run tar on the target VM.

In consequence, the algorithm could be:
For each file to backup:
Encrypt the file and send it to the PIPE
Read the PIPE to perform a HMAC
Read the PIPE to forward it to tar | backup target (File or AppVM)
Remove the End-of-archive data from the PIPE when tar exits
Send the HMAC to tar | backup target after removing the
End-of-archive data

Send Two blocks of 512 zero bytes to the PIPE and close the PIPE.

This way, the backup could be performed on the fly without filling much
the dom0 space. We would read the files a single time, but would have to
manage the PIPE by blocs programatically, which could cause performance
issues.

The simplest would be to use a modified tar version that does not
creates end of archive marker. I didn't found any option for this.

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 8:47:43 AM8/28/13

to qubes...@googlegroups.com, Olivier Médoc

On 28.08.2013 10:39, Olivier Médoc wrote:
> On 08/27/13 22:08, Joanna Rutkowska wrote:

>> On 08/22/13 08:07, Olivier Médoc wrote:
>>> On 08/22/13 06:02, Axon wrote:
>>>> On 08/20/13 12:06, Joanna Rutkowska wrote:

This idea looks perfectly fine to me. How serious the performance issue is?
Can you do some simple test (even simple cat-like program in the same language
as backup software - python, right)?

Or perhaps use 'head -c -1024'?

> The simplest would be to use a modified tar version that does not creates end
> of archive marker. I didn't found any option for this.

I also haven't found such option...

Olivier Médoc

unread,

Aug 28, 2013, 8:49:34 AM8/28/13

to qubes...@googlegroups.com

In fact, I can apparently concatenate all tar files. This will create an
archive that does not follow the tar POSIX format, but that can be read
back using the option --ignore-zeros to skip end-of-archive markers.

Joanna Rutkowska

unread,

Aug 28, 2013, 9:04:06 AM8/28/13

to qubes...@googlegroups.com, Olivier Médoc

On 08/28/13 10:49, Olivier Médoc wrote:

> On 08/28/13 10:39, Olivier Médoc wrote:
>> On 08/27/13 22:08, Joanna Rutkowska wrote:

>>> On 08/22/13 08:07, Olivier Médoc wrote:
>>>> On 08/22/13 06:02, Axon wrote:
>>>>> On 08/20/13 12:06, Joanna Rutkowska wrote:

Hmmm, actually why do we not want to run tar in the BackupVM? If we send
it pairs of (encrypted file, hmac file), then it should be ok?

Later, we can also expect the BackupVM to do untaring and just sending
us file pairs (encrypted file, hmac file), perhaps even using our
qubes.FileCopy service?

joanna.

Joanna Rutkowska

unread,

Aug 28, 2013, 9:15:13 AM8/28/13

to Olivier Médoc, qubes...@googlegroups.com

On 08/28/13 11:14, Joanna Rutkowska wrote:
> On 08/28/13 11:11, Olivier Médoc wrote:

>> Actually, sending a single stream allow to ignore the AppVM activity,
>> and to perform a single AppVM call. How can you differentiate two
>> separate files in a stream ? Except by creating a new protocol or using
>> a specific file format.
>>
> We already have a protocol for sending files between VMs in Qubes...
>
>> The other point is that it allows the AppVM to do what it wants with the
>> stream even if tar is not installed (eg: Windows/MacOS HVM), including
>> sending it over the network.
>
> The BackupVM can always do that.
>
>> This is also better if dom0 ensure the
>> "consistency" of the backup file format, instead of delegating it to an
>> AppVM.
>>
>
>
> I don't see any security problem with that -- the interpretation of any
> backup will start with:
> 1) veryfing qubes.xml.gpg signature
> 2) decrypting, reading and parsing the qubes.xml
> 3) veryfing, decrypting and parsing other files according to the info in
> the (verified) qubes.xml.
>
> joanna.
>

Adding list.

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 9:22:17 AM8/28/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

But this means that BackupVM will unpack (tar) archive just to pack it back
(perhaps in different format). Senseless for me.

Tar format is so simple, that IMHO we can add it support to our unpack.c. And
in the first version just use existing tar command.

BTW tar support sparse files, which our unpack.c currently doesn't.

Joanna Rutkowska

unread,

Aug 28, 2013, 9:56:28 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

No, it will be adding files to archive on the fly, while receving them,
one after another (well, two, after twos).

> Tar format is so simple, that IMHO we can add it support to our unpack.c. And
> in the first version just use existing tar command.
>
> BTW tar support sparse files, which our unpack.c currently doesn't.
>

Complications, right?

j.

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 10:07:05 AM8/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

But how BackupVM (or dom0) will know where the file boundary is? Either some
headers will be needed, or stateful service (so separate transfers will be
concatenated to single archive). Seriously, don't complicate simple things...

>> Tar format is so simple, that IMHO we can add it support to our unpack.c. And
>> in the first version just use existing tar command.
>>
>> BTW tar support sparse files, which our unpack.c currently doesn't.
>>
> Complications, right?

At some point we must support sparse files, otherwise backup will take *much
more* than original system (and will not fit back during restore). But this
can be done at any layer, especially inside of encrypted *and "signed"* file.

Joanna Rutkowska

unread,

Aug 28, 2013, 10:17:08 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

We already have this mechanism, qubes.FileCopy...?

>>> Tar format is so simple, that IMHO we can add it support to our unpack.c. And
>>> in the first version just use existing tar command.
>>>
>>> BTW tar support sparse files, which our unpack.c currently doesn't.
>>>
>> Complications, right?
>
> At some point we must support sparse files, otherwise backup will take *much
> more* than original system (and will not fit back during restore). But this
> can be done at any layer, especially inside of encrypted *and "signed"* file.

>

So, we can first do tar (in Dom0) in order to support sparse'ness, then
gpg encrypt it, and only then process it further. Sounds good.

j.

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 10:22:11 AM8/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

But using it (unmodified qubes.FileCopy) stores the file in BackupVM, which
can be undesired (think of 50GB standalone VM). Modifying the service to pack
tar on the fly is just "repacker", which is senseless to me.

>>>> Tar format is so simple, that IMHO we can add it support to our unpack.c. And
>>>> in the first version just use existing tar command.
>>>>
>>>> BTW tar support sparse files, which our unpack.c currently doesn't.
>>>>
>>> Complications, right?
>>
>> At some point we must support sparse files, otherwise backup will take *much
>> more* than original system (and will not fit back during restore). But this
>> can be done at any layer, especially inside of encrypted *and "signed"* file.
>>
> So, we can first do tar (in Dom0) in order to support sparse'ness, then
> gpg encrypt it, and only then process it further. Sounds good.
>
> j.
>

Joanna Rutkowska

unread,

Aug 28, 2013, 10:25:24 AM8/28/13

to qubes...@googlegroups.com, Marek Marczykowski-Górecki, Olivier Médoc

In case of restoration, this is not a problem, because we choose which
files to restore and need to save them anyway.

In case of preparing a backup -- we can (easily, I guess) modify
unpack.c to not store the file on disk but instead pipe it to the
receiving process (which we will be starting to handle the backup stream
anyway).

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 10:31:32 AM8/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

But for what? Just to use somewhere our file transfer protocol? IMO tar _protocol_ is as good as unpack.c for this purpose and we can use existing tar implementation for most places. The only place where using our unpack.c is beneficial is unpacking during restore - here we can easily add tar format support to our unpack.c (or its clone).

Joanna Rutkowska

unread,

Aug 28, 2013, 10:34:37 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

So, why not get rid of unpack.c everywhere and use tar instead, then, if we trust it so much?

> The only place where using our unpack.c is
> beneficial is unpacking during restore - here we can easily add tar format
> support to our unpack.c (or its clone).
>

j.

Joanna Rutkowska

unread,

Aug 28, 2013, 10:39:42 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

I know this is crude, but still:

[user@work-pub src]$ pwd
/home/user/Downloads/tar-1.26/src
[user@work-pub src]$ wc -l *.c | tail -1
19870 total

[user@qubes qrexec-lib]$ pwd
/home/user/qubes/qubes-R2/linux-utils/qrexec-lib
[user@qubes qrexec-lib]$ wc -l unpack.c
157 unpack.c

:P

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 10:40:50 AM8/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

I said about data format, not implementation itself. In this case (unpacking the backup) we need only small portion of its features, so its like our

protocol only with different places in file header for file name, size, attrs
etc. Read the next paragraph...

>> The only place where using our unpack.c is
>> beneficial is unpacking during restore - here we can easily add tar format
>> support to our unpack.c (or its clone).
>>
>
> j.
>

Joanna Rutkowska

unread,

Aug 28, 2013, 10:48:51 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

So, IIUC, we have two choices:
1) We do tar in Dom0 (using the standard tar), piece by piece and keep
sending this to the BackupVM.

When restoring we receive the tarred stream in Dom0, and use our
modified unpack.c to extract single files, piece by piece.

This requires some programmatic work on unpack.c -- making it support
tar format. Probably easy, but how long it will take to make sure it is
impeccable? You suggest to use tar in the "first version", which means
about 1000x more code. We know that we will never have time to go back
and replace standard tar with improved unpack.c

2) We keep sending file to BackupVM. In the "first version", which just
save those files on the BackupVM filesystem, and then we let the local
handling process to do whatever it wants with them, presumably tar them
all together and pipe out somewhere (to NAS, S3, wherever).

In the next version we might modify unpack.c not to store it on the
local filesystem, but to pipe it directly to the handling process. Or
perhaps we can do that in the "first version", because it is an easy
modification, not requiring exhausting testing of handling of a new format.

j.

Joanna Rutkowska

unread,

Aug 28, 2013, 10:53:01 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

In this repsect, I would use cpio:

/home/user/Downloads/cpio-2.11/src
[user@work-pub src]$ wc -l * | tail -1
9225 total

Still about 10x more code than our unpack.c...

Joanna Rutkowska

unread,

Aug 28, 2013, 10:57:07 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

Hm, we can actually tar them in Dom0 when making the backup (as in #1),
but when restoring we can do the untaring still in the BackupVM, and
send the files to Dom0 using our qubes.FileCopy. In this case it is not
a problem that the files will be saved on the filesystem.

Perhaps this solution combines the advantages of the above two?

j.

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 11:09:37 AM8/28/13

to Joanna Rutkowska, qubes...@googlegroups.com, Olivier Médoc

Sounds good to me. Still needs tar->our protocol converter, but only in one
direction. The converter probably will be as simple as adding tar format
support to our unpack.c, but in this use case it will not be part of TCB,
which is good.

The "first version" can store untared backup in some temporary directory and
then send it using qubes.FileCopy to dom0. This will need substantial amount
of space in BackupVM... But for testing it should be enough, and writing then
converter should be a simple task.

Olivier, does above sounds sensible to you? Does this solve all the "backup
file format" problem?

Joanna Rutkowska

unread,

Aug 28, 2013, 11:29:01 AM8/28/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

On 08/28/13 13:09, Marek Marczykowski-Górecki wrote:
>> Hm, we can actually tar them in Dom0 when making the backup (as in #1),
>> but when restoring we can do the untaring still in the BackupVM, and
>> send the files to Dom0 using our qubes.FileCopy. In this case it is not
>> a problem that the files will be saved on the filesystem.
>>
>> Perhaps this solution combines the advantages of the above two?
>
> Sounds good to me. Still needs tar->our protocol converter, but only in one
> direction. The converter probably will be as simple as adding tar format

> support to our unpack.c, but in this use case it will not be part of TCB,
> which is good.
>

Exactly.

> The "first version" can store untared backup in some temporary directory and
> then send it using qubes.FileCopy to dom0. This will need substantial amount
> of space in BackupVM... But for testing it should be enough, and writing then
> converter should be a simple task.
>

Right. Then all that is need is to make sure we can use our
qubes.FileCopy to send files that are received from a stdin, instead of
from a file on disk.

> Olivier, does above sounds sensible to you? Does this solve all the "backup
> file format" problem?
>

So, to summary, to make sure we all get this right:

For each file that is to be part of the backup, Dom0 does the following:

1) tar's it using the standard tar -- this is in order to preserve the
file's sparse'ness

2) gpg encrypts it with some symmetric key/passphrase, supplied by the
user. The output is then piped, again, to tar (still in Dom0), whose
output goes directly to the BackupVM (via qrexec). Do we need a modified
tar here so that it doesn't add an EOF marker after each file?

3) Finally, for each file it also creates an HMAC for it in a form of a
separate file. The passphrase is needed here again to generate/verify
the HMAC. This hmac file is also piped to tar and later to the BackupVM.

Now, in order to restore the backup:

The BackupVM runs tar on the input stream and for each extracted file:
1) it stores the file and its hmac on the BackupVM fs,
2) use qvm-copy-to-vm to send those two files to Dom0. In the next
version we will modify qvm-copy-to-vm so that it took input from stdin,
remove the necessity to temporary store files on BackupVM fs. Or perhaps
this could be done immediately in the first version, because it sounds
easy, actually, doesn't it?

Dom0 receives those files via a (modified) qubes.FileCopy -- actually we
could name it qubes.Dom0BackupRestore), expects that the first two
files to be send this way to be: qubes.xml.gpg and qubes.xml.hmac -- it
stores those two somewhere (but imposes some restriction on max size to
protect against simple DoS attack which would be sending out very large
qubes.xml.gpg or qubes.xml.hmac).

1) It now asks the user for the passphrase, and uses it to verify the
hmac, and, if that's ok, decrypts the qubes.xml.gpg.

2) It now parses qubes.xml and proceeds to receiving the actual files of
the backup in a loop. Hm, would be nice if qubes.xml contained the sizes
of each of the file, again, so that we could avoid simple DoS attacks in

this step (the assumption here would be that the size of the gpg
encrypted file is no longer than the size of the unencrypted file).

3) Each received file is stored locally, verified, decrypted, and then
moved to /var/lib/qubes/* accordingly, just like in a normal restore.

joanna.

Olivier Médoc

unread,

Aug 28, 2013, 1:50:36 PM8/28/13

to qubes...@googlegroups.com

On 08/28/13 13:29, Joanna Rutkowska wrote:

I'm currently testing points 1 and 3. Point 2 still needs additionnal
work on my part, such as asking for a passphrase and running an
additional tar pass.

After additionnal tests, I prefer not using a modified version of tar.
Testing untar using the --ignore-zeros option seems to works. I tested
removing by hand the EOF in the backup code, but it looks quite
complicated (need to wait for buffers to fill up before sending data to
the AppVM).

> Now, in order to restore the backup:
>
> The BackupVM runs tar on the input stream and for each extracted file:
> 1) it stores the file and its hmac on the BackupVM fs,
> 2) use qvm-copy-to-vm to send those two files to Dom0. In the next
> version we will modify qvm-copy-to-vm so that it took input from stdin,
> remove the necessity to temporary store files on BackupVM fs. Or perhaps
> this could be done immediately in the first version, because it sounds
> easy, actually, doesn't it?
>
> Dom0 receives those files via a (modified) qubes.FileCopy -- actually we
> could name it qubes.Dom0BackupRestore), expects that the first two
> files to be send this way to be: qubes.xml.gpg and qubes.xml.hmac -- it
> stores those two somewhere (but imposes some restriction on max size to
> protect against simple DoS attack which would be sending out very large
> qubes.xml.gpg or qubes.xml.hmac).
>
> 1) It now asks the user for the passphrase, and uses it to verify the
> hmac, and, if that's ok, decrypts the qubes.xml.gpg.
>
> 2) It now parses qubes.xml and proceeds to receiving the actual files of
> the backup in a loop. Hm, would be nice if qubes.xml contained the sizes
> of each of the file, again, so that we could avoid simple DoS attacks in
> this step (the assumption here would be that the size of the gpg
> encrypted file is no longer than the size of the unencrypted file).

Back on your idea of cleaning up qubes.xml, we can add the size found
during the backup preparation, and an additionnal information to know if
the vm is actually stored in the backup or not. This would remove the
need for an additionnal backup specification file.

Joanna Rutkowska

unread,

Aug 28, 2013, 2:59:33 PM8/28/13

to qubes...@googlegroups.com, Olivier Médoc

On 08/28/13 15:50, Olivier Médoc wrote:
> On 08/28/13 13:29, Joanna Rutkowska wrote:

Ah, that's even better then. Somewhere in the README or Wiki we would just need to make it clear that in order to manually extract backup one would need to pass this --ignore-zeros option. I guess we should just have a "Manual backup restoration" section, where to describe this.

Yes, sounds good. So the only question is -- do we want to re-build qubes.xml that we store with the backup, so that we remove all the information about VMs that do not go into the backup (apart for VMs on which those in the backup depend, such as netvms, or templates), or should we use the complete qubes.xml and, for each VM, just add a property about whether it is in the backup. The first option seems not only more privacy-preserving, but also more elegant.

>>
>> 3) Each received file is stored locally, verified, decrypted, and then
>> moved to /var/lib/qubes/* accordingly, just like in a normal restore.
>>
>> joanna.
>>
>

joanna.

Marek Marczykowski-Górecki

unread,

Aug 28, 2013, 3:48:23 PM8/28/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

But we can't clean up qubes.xml from all VMs not included in the backup. qubes.xml needs to be consistent, so at least must include template, netvm and firewallvm. Rebuilding qubes.xml to include all dependent VMs can be quite complicated (I'm sure that we will forget about e.g. UpdateVM).
So I'm for the option with additional attributes.

Olivier Médoc

unread,

Aug 29, 2013, 7:47:30 AM8/29/13

to qubes...@googlegroups.com

On 08/28/13 16:59, Joanna Rutkowska wrote:

> On 08/28/13 15:50, Olivier M&#65533;doc wrote:
>> On 08/28/13 13:29, Joanna Rutkowska wrote:

Now that I think about it, once the HMAC is ready, I write it to
/var/lib/qubes, and run tar to send it to the AppVM.

This must be the same one a file is encrypted, except that this file can
take 40G (eg for a private.img). Again the same issue. Maybe somebody
has hints about a solution. That could be:

The tar file format require to know the file size (5th field of the tar
header), which means one need to completely write a file before knowing
its size, thus breaking the stream.

- Simple solution: There must be at least as much free space in dom0 as
the biggest file found in the backup. If not, warn the user.

- Change the backup file format solution:
In this case, either we have to find a "stream" file format, either we
have to split the stream in blocks, and reconstruct it later. For
example, we can make tar think that the tape-length is 2G (hello fat32
:)), then encrypt and create hmac for each blob.

As a side note, it is possible to ask the tar subprocess to wait
(os.kill(subprocess_pid, signal.SIGSTOP), os.kill(subprocess_pid,
signal.SIGCONT)), if the free space is too low...

Joanna Rutkowska

unread,

Aug 29, 2013, 7:54:42 AM8/29/13

to qubes...@googlegroups.com, Olivier Médoc

On 08/29/13 09:47, &#65533; wrote:
> On 08/28/13 16:59, Joanna Rutkowska wrote:

>> On 08/28/13 15:50, Olivier Médoc wrote:
>>> On 08/28/13 13:29, Joanna Rutkowska wrote:

I'd rather prefer that we don't "pollute" the system's /var/lib/qubes
with additional helper files?

> This must be the same one a file is encrypted, except that this file can
> take 40G (eg for a private.img). Again the same issue. Maybe somebody
> has hints about a solution.

I'm not sure what you mean here? Please rephrase.

j.

Marek Marczykowski-Górecki

unread,

Aug 29, 2013, 8:53:58 AM8/29/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

Tar needs to know file size prior to archiving, so we can't simply pipe output from gpg to tar, we must store encrypted file somewhere.
It's ok for me to require disk space for it in the first version (gpg can compress the data so it will be slightly smaller).

In the later version we can split the file into 2GB blocks (right after gpg, before hmac and second tar).

Regarding place for temporary files, you can use /var/tmp (not /tmp, because it is on tmpfs so in RAM).

Joanna Rutkowska

unread,

Aug 29, 2013, 8:55:20 AM8/29/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com, Olivier Médoc

Does cpio makes any difference here?

Olivier Médoc

unread,

Aug 29, 2013, 9:01:37 AM8/29/13

to qubes...@googlegroups.com

We can work in /tmp, except that /tmp is often managed using tmpfs making complicated to manage big files.

>> This must be the same one a file is encrypted, except that this file can
>> take 40G (eg for a private.img). Again the same issue. Maybe somebody
>> has hints about a solution.
> I'm not sure what you mean here? Please rephrase.

Sorry, I should have removed this first paragraph...

As I said below, the tar file format need to know the file size when creating an archive, this means it has to be a real file instead of an archive.

The tests I performed until now are like this:
tar -cO --sparse private.img | hmac > private.img.hmac
| qvm-run AppVM QUBESRPC qubes.Backup

tar -cO private.img.hmac | qvm-run AppVM QUBESRPC qubes.Backup

Now if I want to encrypt the file, and then tar it again before sending it using qvm-run, I need to store private.img.encrypted locally instead of sending it directly to the first qvm-run. This creates new free space problems, because of tar constraints.

- Tar cannot take file content in STDIN, only the file name. This is because tar archive files and need to know the filename, the size, the

file ACLs before creating the archive. This is the same for cpio.
- I checked using a named pipe, but tar just backup the "special file" instead of backing up the file content.
- I checked using spit archive. Tar apparently supports spitting the archive. This would change the algorithm to:

tar -cO --sparse --tape-length 2G private.img
cat private.img.tar.01 | encrypt | hmac >
private.img.tar.01.hmac
> private.img.tar.01.encrypted

tar -cO private.img.tar.01.encrypted | qvm-run AppVM QUBESRPC qubes.Backup
tar -cO private.img.tar.01.hmac | qvm-run AppVM QUBESRPC qubes.Backup

With one possible optimisation because tar wait for input before creating the second file. In this case, I can do:

tar -cO --sparse --tape-length 2G private.img | encrypt | hmac >
private.img.tar.01.hmac
> private.img.tar.01.encrypted

tar -cO private.img.tar.01.encrypted | qvm-run AppVM QUBESRPC qubes.Backup
tar -cO private.img.tar.01.hmac | qvm-run AppVM QUBESRPC qubes.Backup

For instance, the backup format would be something like:

qubes.xml
qubes.xml.hmac
appvms/OSX/OSX.conf.encrypted
appvms/OSX/OSX.conf.encrypted.hmac
appvms/OSX/private.img.tar.01.encrypted
appvms/OSX/private.img.tar.01.encrypted.hmac
appvms/OSX/private.img.tar.02.encrypted
appvms/OSX/private.img.tar.02.encrypted.hmac
appvms/OSX/private.img.tar.03.encrypted
appvms/OSX/private.img.tar.03.encrypted.hmac
appvms/OSX/root.img.tar.01.encrypted
appvms/OSX/root.img.tar.01.encrypted.hmac
appvms/OSX/root.img.tar.02.encrypted
appvms/OSX/root.img.tar.02.encrypted.hmac
...

Olivier Médoc

unread,

Aug 29, 2013, 9:02:27 AM8/29/13

to qubes...@googlegroups.com

Sadly, cpio does not change this problem.

Marek Marczykowski-Górecki

unread,

Aug 29, 2013, 9:13:03 AM8/29/13

to qubes...@googlegroups.com, Olivier Médoc

On 29.08.2013 11:01, Olivier Médoc wrote:
> On 08/29/13 09:54, Joanna Rutkowska wrote:
>> On 08/29/13 09:47, [illegible] wrote:
>>> On 08/28/13 16:59, Joanna Rutkowska wrote:
>>>> On 08/28/13 15:50, Olivier Médoc wrote:
>>>>> On 08/28/13 13:29, Joanna Rutkowska wrote:
>>>>>> On 08/28/13 13:09, Marek Marczykowski-Górecki wrote:
>>>>>>> Hm, we can actually tar them in Dom0 when making the backup (as in
>>>>>>> #1),
>>>>>>> but when restoring we can do the untaring still in the BackupVM, and
>>>>>>> send the files to Dom0 using our qubes.FileCopy. In this case it
>>>>>>> is not
>>>>>>> a problem that the files will be saved on the filesystem.
>>>>>>>
>>>>>>> Perhaps this solution combines the advantages of the above two?
>>>>>> Sounds good to me. Still needs tar->our protocol converter, but only
>>>>>> in one
>>>>>> direction. The converter probably will be as simple as adding tar
>>>>>> format
>>>>>> support to our unpack.c, but in this use case it will not be part of
>>>>>> TCB,
>>>>>> which is good.
>>>>>>
>>>>> Exactly.
>>>>>
>>>>> The "first version" can store untared backup in some temporary
>>>>> directory and
>>>>> then send it using qubes.FileCopy to dom0. This will need substantial
>>>>> amount
>>>>> of space in BackupVM... But for testing it should be enough, and
>>>>> writing then
>>>>> converter should be a simple task.
>>>>>
>>>> Right. Then all that is need is to make sure we can use our
>>>> qubes.FileCopy to send files that are received from a stdin, instead of
>>>> from a file on disk.
>>>>
>>> Olivier, does above sounds sensible to you? Does this solve all the
>>> "backup
>>> file format" problem?
>>>
>> So, to summary, to make sure we all get this right:
>>
>> For each file that is to be part of the backup, Dom0 does the
>> following:
>>
>> 1) tar's it using the standard tar -- this is in order to preserve the
>> file's sparse'ness
>>
>> 2) gpg encrypts it with some symmetric key/passphrase, supplied by the
>> user. The output is then piped, again, to tar (still in Dom0), whose
>> output goes directly to the BackupVM (via qrexec). Do we need a
>> modified
>> tar here so that it doesn't add an EOF marker after each file?
>>

>>>>>> 3) Finally, for each file it also creates an HMAC for it in a form of a
>>>>>> separate file. The passphrase is needed here again to generate/verify
>>>>>> the HMAC. This hmac file is also piped to tar and later to the
>>>>>> BackupVM.
>>> Now that I think about it, once the HMAC is ready, I write it to
>>> /var/lib/qubes, and run tar to send it to the AppVM.
>>>
>> I'd rather prefer that we don't "pollute" the system's /var/lib/qubes
>> with additional helper files?
> We can work in /tmp, except that /tmp is often managed using tmpfs making
> complicated to manage big files.
>
>>> This must be the same one a file is encrypted, except that this file can
>>> take 40G (eg for a private.img). Again the same issue. Maybe somebody
>>> has hints about a solution.
>> I'm not sure what you mean here? Please rephrase.
> Sorry, I should have removed this first paragraph...
>
> As I said below, the tar file format need to know the file size when creating
> an archive, this means it has to be a real file instead of an archive.
>
> The tests I performed until now are like this:
> tar -cO --sparse private.img | hmac > private.img.hmac
> | qvm-run AppVM QUBESRPC qubes.Backup
>
> tar -cO private.img.hmac | qvm-run AppVM QUBESRPC qubes.Backup
>
> Now if I want to encrypt the file, and then tar it again before sending it
> using qvm-run, I need to store private.img.encrypted locally instead of
> sending it directly to the first qvm-run. This creates new free space
> problems, because of tar constraints.
>
> - Tar cannot take file content in STDIN, only the file name. This is because
> tar archive files and need to know the filename, the size, the file ACLs
> before creating the archive. This is the same for cpio.
> - I checked using a named pipe, but tar just backup the "special file" instead
> of backing up the file content.
> - I checked using spit archive. Tar apparently supports spitting the archive.
> This would change the algorithm to:
>
> tar -cO --sparse --tape-length 2G private.img
> cat private.img.tar.01 | encrypt | hmac >
> private.img.tar.01.hmac
>> private.img.tar.01.encrypted
>
> tar -cO private.img.tar.01.encrypted | qvm-run AppVM QUBESRPC qubes.Backup
> tar -cO private.img.tar.01.hmac | qvm-run AppVM QUBESRPC qubes.Backup
>
> With one possible optimisation because tar wait for input before creating the
> second file. In this case, I can do:
>
> tar -cO --sparse --tape-length 2G private.img | encrypt | hmac >
> private.img.tar.01.hmac
>> private.img.tar.01.encrypted
>
> tar -cO private.img.tar.01.encrypted | qvm-run AppVM QUBESRPC qubes.Backup
> tar -cO private.img.tar.01.hmac | qvm-run AppVM QUBESRPC qubes.Backup

>
> For instance, the backup format would be something like:
>
> qubes.xml
> qubes.xml.hmac
> appvms/OSX/OSX.conf.encrypted
> appvms/OSX/OSX.conf.encrypted.hmac
> appvms/OSX/private.img.tar.01.encrypted
> appvms/OSX/private.img.tar.01.encrypted.hmac
> appvms/OSX/private.img.tar.02.encrypted
> appvms/OSX/private.img.tar.02.encrypted.hmac
> appvms/OSX/private.img.tar.03.encrypted
> appvms/OSX/private.img.tar.03.encrypted.hmac
> appvms/OSX/root.img.tar.01.encrypted
> appvms/OSX/root.img.tar.01.encrypted.hmac
> appvms/OSX/root.img.tar.02.encrypted
> appvms/OSX/root.img.tar.02.encrypted.hmac
> ...

Looks good to me.

--
Best Regards,
Marek Marczykowski-Górecki
Invisible Things Lab
A: Because it messes up the order in which people normally read text.
Q: Why is top-posting such a bad thing?

Marek Marczykowski-Górecki

unread,

Aug 29, 2013, 9:34:39 AM8/29/13

to qubes...@googlegroups.com, Joanna Rutkowska, Olivier Médoc

On 29.08.2013 10:53, Marek Marczykowski-Górecki wrote:

>>> This must be the same one a file is encrypted, except that this file can
>>> take 40G (eg for a private.img). Again the same issue. Maybe somebody
>>> has hints about a solution.
>>
>> I'm not sure what you mean here? Please rephrase.
>

> Tar needs to know file size prior to archiving, so we can't simply pipe output
> from gpg to tar, we must store encrypted file somewhere.
> It's ok for me to require disk space for it in the first version (gpg can
> compress the data so it will be slightly smaller).

I've just checked on 43G private.img, it compressed down to 22G.

Olivier Médoc

unread,

Sep 10, 2013, 7:47:42 AM9/10/13

to qubes...@googlegroups.com

So finally, I got something working. Handling tar with tape-length option was quite tricky and I had to use a named pipe because tar close my proc stdout after the first chunk...

The only thing is that I continued on my previous code so for each appvm, I tar the whole directory instead of each file separatly:

qubes.xml
qubes.xml.hmac
appvms/OSX.000
appvms/OSX.000.hmac
appvms/OSX.001
appvms/OSX.001.hmac
appvms/ubuntu.000
appvms/ubuntu.000.hmac
appvms/ubuntu.001
appvms/ubuntu.001.hmac
...

Some notes:
- I tested it both with / without encryption using an AppVM, and using a dom0 directory
- I'm currently able to restore manually using tar / openssl. Now I have to implement the restore process
- I didn't implemented the changes we discussed earlier with qubes.xml (adding information to know which VMs are in the backup)
- The last tar pass sending data to the appvm uses the --posix option to ensure that the initial extraction can be done using simple tools
- Using some paralellism, it seems that the backup process is not too long (considering tar | encryption | sending to appvm). For instance, it takes 7 minutes for a 3GB backup of my test vm. This is the same before implementing multiple tar passes.
- This paralellism also allows to remove temporary files each time it has been sent to the appvm
- Now that the files can be verified, the first tar passe (tar_sparse) can probably use gzip to compress data, but I don't know the performance overhead.
- Do to some limitation in input/output, the user passphrase can probably appears when running ps, because I had to put the passphrase on the command line.

0004-backup-code-cleanup.patch

0005-backup-implemented-backup-mecanism-using-tar_sparse-.patch

0006-backup-introduced-a-second-tar-pass-to-send-encrypte.patch

0007-backup-use-a-thread-to-send-data-to-AppVM-in-paralle.patch

0008-backup-multiple-fixes-for-the-backup-process-includi.patch

Marek Marczykowski-Górecki

unread,

Sep 11, 2013, 7:22:47 PM9/11/13

to qubes...@googlegroups.com, Olivier Médoc

Great!

> Handling tar with tape-length option was
> quite tricky and I had to use a named pipe because tar close my proc stdout
> after the first chunk...
>
> The only thing is that I continued on my previous code so for each appvm, I
> tar the whole directory instead of each file separatly:
>
> qubes.xml
> qubes.xml.hmac
> appvms/OSX.000
> appvms/OSX.000.hmac
> appvms/OSX.001
> appvms/OSX.001.hmac
> appvms/ubuntu.000
> appvms/ubuntu.000.hmac
> appvms/ubuntu.001
> appvms/ubuntu.001.hmac
> ...

I think it's ok.

> Some notes:
> - I tested it both with / without encryption using an AppVM, and using a dom0
> directory
> - I'm currently able to restore manually using tar / openssl. Now I have to
> implement the restore process

Can you write some draft instruction how to do it? Just to be able easily test
the solution.

> - I didn't implemented the changes we discussed earlier with qubes.xml (adding
> information to know which VMs are in the backup)
> - The last tar pass sending data to the appvm uses the --posix option to
> ensure that the initial extraction can be done using simple tools
> - Using some paralellism, it seems that the backup process is not too long
> (considering tar | encryption | sending to appvm). For instance, it takes 7
> minutes for a 3GB backup of my test vm. This is the same before implementing

> multiple tar passes.
> - This paralellism also allows to remove temporary files each time it has been
> sent to the appvm

:)

> - Now that the files can be verified, the first tar passe (tar_sparse) can
> probably use gzip to compress data, but I don't know the performance overhead.
> - Do to some limitation in input/output, the user passphrase can probably
> appears when running ps, because I had to put the passphrase on the command line.

Perhaps use "env:sth" for password?

Can you push all the patches to some git server?

Olivier Médoc

unread,

Sep 13, 2013, 9:30:09 AM9/13/13

to qubes...@googlegroups.com

1) Extract the first tar pass:
$ tar -i -xvf ../qubes-backup-2013-10-10-085338
qubes.xml.000
qubes.xml.000.hmac
vm-templates/archlinux-x64-spec-2013.000
vm-templates/archlinux-x64-spec-2013.000.hmac
vm-templates/archlinux-x64-spec-2013.001
vm-templates/archlinux-x64-spec-2013.000.hmac
...

2) Verify the files
$ cat qubes.xml.000.hmac
(stdin)= cb021bef1a444dacbe0d5203caa40d0b323e86dc
$ openssl dgst -hmac yourbackuppassphrase qubes.xml.000
HMAC-SHA1(qubes.xml.000)= cb021bef1a444dacbe0d5203caa40d0b323e86dc

3) Decrypt all the files (if the files are encrypted):
$ openssl enc -d -pass pass:yourbackuppassphrase -aes-256-cbc -in
qubes.xml.000 -out qubes.xml.dec.000
$ openssl enc -d -pass pass:yourbackuppassphrase -aes-256-cbc -in
vm-templates/archlinux-x64-spec-2013.000 -out
vm-templates/archlinux-x64-spec-2013.dec.000
$ openssl enc -d -pass pass:yourbackuppassphrase -aes-256-cbc -in
vm-templates/archlinux-x64-spec-2013.001 -out
vm-templates/archlinux-x64-spec-2013.dec.001
...

4) Untar the backup (2nd pass)
$ tar --tape-length 1000000 -xvf qubes.xml.dec.000
$ tar --tape-length 1000000 -xvf
vm-templates/archlinux-x64-spec-2013.dec.000
vm-templates/archlinux-x64-spec-2013/
vm-templates/archlinux-x64-spec-2013/root-cow.img
vm-templates/archlinux-x64-spec-2013/updates.stat
vm-templates/archlinux-x64-spec-2013/apps/

vm-templates/archlinux-x64-spec-2013/apps/archlinux-x64-spec-2013-qubes-appmenu-select.desktop
vm-templates/archlinux-x64-spec-2013/apps/archlinux-x64-spec-2013-vm.directory
vm-templates/archlinux-x64-spec-2013/apps/archlinux-x64-spec-2013-xfce4-terminal.desktop
vm-templates/archlinux-x64-spec-2013/firewall.xml
vm-templates/archlinux-x64-spec-2013/icon.png
vm-templates/archlinux-x64-spec-2013/whitelisted-appmenus.list
vm-templates/archlinux-x64-spec-2013/netvm-whitelisted-appmenus.list
vm-templates/archlinux-x64-spec-2013/apps.templates/
vm-templates/archlinux-x64-spec-2013/apps.templates/xgpsspeed.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/xgps.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/qtconfig4.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/qgit.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/xterm.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/uxterm.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/policytool.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/bvnc.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/gimp.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/firefox.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/avahi-discover.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/bssh.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/google-earth.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/jconsole.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/xfce4-about.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/leafpad.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/wireshark.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/meld.desktop
vm-templates/archlinux-x64-spec-2013/apps.templates/xfce4-terminal.desktop
vm-templates/archlinux-x64-spec-2013/root.img
Prepare volume #2 for `vm-templates/archlinux-x64-spec-2013.dec.000' and
hit return: n vm-templates/archlinux-x64-spec-2013.dec.001
Prepare volume #3 for `vm-templates/archlinux-x64-spec-2013.dec.001' and
hit return: n vm-templates/archlinux-x64-spec-2013.dec.002
vm-templates/archlinux-x64-spec-2013/archlinux-x64-spec-2013.conf
vm-templates/archlinux-x64-spec-2013/root-cow.img.old
vm-templates/archlinux-x64-spec-2013/clean-volatile.img.tar
vm-templates/archlinux-x64-spec-2013/private.img
vm-templates/archlinux-x64-spec-2013/volatile.img
vm-templates/archlinux-x64-spec-2013/vm-whitelisted-appmenus.list
...

>> - I didn't implemented the changes we discussed earlier with qubes.xml (adding
>> information to know which VMs are in the backup)
>> - The last tar pass sending data to the appvm uses the --posix option to
>> ensure that the initial extraction can be done using simple tools
>> - Using some paralellism, it seems that the backup process is not too long
>> (considering tar | encryption | sending to appvm). For instance, it takes 7
>> minutes for a 3GB backup of my test vm. This is the same before implementing
>> multiple tar passes.
>> - This paralellism also allows to remove temporary files each time it has been
>> sent to the appvm
> :)
>
>> - Now that the files can be verified, the first tar passe (tar_sparse) can
>> probably use gzip to compress data, but I don't know the performance overhead.
>> - Do to some limitation in input/output, the user passphrase can probably
>> appears when running ps, because I had to put the passphrase on the command line.
> Perhaps use "env:sth" for password?

>
> Can you push all the patches to some git server?

Here is, I finally created a github account:
https://github.com/ptitdoc/qubes-core

>

Joanna Rutkowska

unread,

Sep 13, 2013, 11:09:40 AM9/13/13

to qubes...@googlegroups.com, Olivier Médoc

Olivier, so can you describe the actual flow of qvm-backup-restore?

joanna.

Olivier Médoc

unread,

Sep 13, 2013, 1:56:11 PM9/13/13

to qubes...@googlegroups.com

Hello,
I didn't implemented the new restore functions yet. I will try to follow
your recommendations (which can be found somewhere in this thread).

>
> joanna.
>

Olivier Médoc

unread,

Sep 27, 2013, 7:58:40 AM9/27/13

to qubes...@googlegroups.com

On 09/13/13 13:09, Joanna Rutkowska wrote:

Some progress with the restore functions, I also started to add the new
backup features to the GUI:
https://github.com/ptitdoc/qubes-core.git
https://github.com/ptitdoc/qubes-manager.git

The restore flow is currently:

Restore_header: appvm-cat | tar_posix_extraction | file_verification |
decrypt_file | extract_qubes.xml

1st thread: appvm-cat | tar_posix_extraction | file_verification |
forward_to_2nd_thread

2nd thread: decrypt_file | extract_file

If you look at the code, there are still a lot of fixme (implement
restore with encryption ...). I also still have to deal with a safer
program for the first extraction:
- Using a safer tool such as cpio or adapting unpack.c (the archive is
normally POSIX compatible and most of the fields can probably be ignored).
- as suggested by Joana this first extraction can be run inside the
appvm. Meaning I have to write a piece of code to extract each files,
send them one by one to dom0 (this is not allowed through
qvm-copy-to-vm), and dealing with the Appvm size.

>
> joanna.
>

Marek Marczykowski-Górecki

unread,

Sep 27, 2013, 2:56:20 PM9/27/13

to qubes...@googlegroups.com, Olivier Médoc

Some minor comment for now: you set vm.backup_content=True, but nowhere set it
to False. Either set it to False for all the VMs just before, or operate on
copy of qubes.xml so original will still have False here.

Anyway it looks promising :)

> https://github.com/ptitdoc/qubes-manager.git
>
> The restore flow is currently:
>
> Restore_header: appvm-cat | tar_posix_extraction | file_verification |
> decrypt_file | extract_qubes.xml
>
> 1st thread: appvm-cat | tar_posix_extraction | file_verification |
> forward_to_2nd_thread
> 2nd thread: decrypt_file | extract_file
>
> If you look at the code, there are still a lot of fixme (implement restore
> with encryption ...). I also still have to deal with a safer program for the
> first extraction:
> - Using a safer tool such as cpio or adapting unpack.c (the archive is
> normally POSIX compatible and most of the fields can probably be ignored).
> - as suggested by Joana this first extraction can be run inside the appvm.
> Meaning I have to write a piece of code to extract each files, send them one
> by one to dom0 (this is not allowed through qvm-copy-to-vm), and dealing with
> the Appvm size.

Actually this can be done as tar->"unpack.c format" converter. I'll write it
in some spare time, but for now you can exctact tar in appvm and then copy
them using unmodified qvm-copy-to-vm (or qfile-agent directly).

Olivier Médoc

unread,

Sep 27, 2013, 5:53:24 PM9/27/13

to qubes...@googlegroups.com

Good point.

> Anyway it looks promising :)
>
>> https://github.com/ptitdoc/qubes-manager.git
>>
>> The restore flow is currently:
>>
>> Restore_header: appvm-cat | tar_posix_extraction | file_verification |
>> decrypt_file | extract_qubes.xml
>>
>> 1st thread: appvm-cat | tar_posix_extraction | file_verification |
>> forward_to_2nd_thread
>> 2nd thread: decrypt_file | extract_file
>>
>> If you look at the code, there are still a lot of fixme (implement restore
>> with encryption ...). I also still have to deal with a safer program for the
>> first extraction:
>> - Using a safer tool such as cpio or adapting unpack.c (the archive is
>> normally POSIX compatible and most of the fields can probably be ignored).
>> - as suggested by Joana this first extraction can be run inside the appvm.
>> Meaning I have to write a piece of code to extract each files, send them one
>> by one to dom0 (this is not allowed through qvm-copy-to-vm), and dealing with
>> the Appvm size.
> Actually this can be done as tar->"unpack.c format" converter. I'll write it
> in some spare time, but for now you can exctact tar in appvm and then copy
> them using unmodified qvm-copy-to-vm (or qfile-agent directly).

Again, I don't really get how to use qvm-copy-to-vm to copy to dom0. I
through I was forbidden by design, and that no tool exists for that
(except retrieving data/file content directly qvm-run output).

Joanna Rutkowska

unread,

Sep 27, 2013, 10:31:01 PM9/27/13

to qubes...@googlegroups.com, Olivier Médoc

I'm sorry, but it's not clear to me which operations are performed in
which VM (backkupvm vs. Dom0) -- can you elaborate?

> If you look at the code, there are still a lot of fixme (implement
> restore with encryption ...). I also still have to deal with a safer
> program for the first extraction:
> - Using a safer tool such as cpio or adapting unpack.c (the archive is
> normally POSIX compatible and most of the fields can probably be ignored).
> - as suggested by Joana this first extraction can be run inside the
> appvm. Meaning I have to write a piece of code to extract each files,
> send them one by one to dom0 (this is not allowed through
> qvm-copy-to-vm), and dealing with the Appvm size.

>
>

j.

Olivier Médoc

unread,

Sep 28, 2013, 9:36:16 AM9/28/13

to qubes...@googlegroups.com

Restore_header: AppVM:cat_backup_file | dom0:tar_posix_extraction |
dom0:file_verification | dom0:decrypt_file | dom0:extract_qubes.xml

Restore_backup:
1st thread: AppVM:cat_backup_file | dom0:tar_posix_extraction |
dom0:file_verification | dom0:forward_to_2nd_thread
2nd thread: dom0:decrypt_file | dom0:extract_file

Now I'm trying to put the tar_posix_extraction process in an appvm, but
I didn't managed to use qvm-copy-to-vm for that (only able to copy files
to others VMs, but not to dom0)

Marek Marczykowski-Górecki

unread,

Sep 28, 2013, 2:32:29 PM9/28/13

to qubes...@googlegroups.com, Olivier Médoc

Check dom0 update mechanism:
VM: /usr/lib/qubes/qubes-download-dom0-updates.sh
dom0: /usr/libexec/qubes/qubes-receive-updates (or where it was on R2B2...)

Olivier Médoc

unread,

Sep 29, 2013, 8:54:18 AM9/29/13

to Marek Marczykowski-Górecki, qubes...@googlegroups.com

If I understood correctly, if I manage to make unpack.c compatible with
the PaX format, then I can easily use a mecanism similar than
qubes-receive-update instead of extracting everything to dom0 ?

I found the PAX archive project which only takes 350 lines of code for
PAX parsing. I started to import this code to unpack.c, disabling all
unused things such as FIFO, special character files...

Is it ok for you knowing that PAX is little bit different than your
initial format ? (PAX headers are pure ASCII :
http://www.mkssoftware.com/docs/man4/pax.4.asp)

Marek Marczykowski-Górecki

unread,

Sep 29, 2013, 11:29:05 AM9/29/13

to Olivier Médoc, qubes...@googlegroups.com, Joanna Rutkowska

Sounds good to me. Joanna, what do you think?

Olivier Médoc

unread,

Sep 29, 2013, 6:05:34 PM9/29/13

to qubes...@googlegroups.com

Just a side note: I choosed PAX because it is the POSIX archive format,
and because it is the format of my backup (tar --posix).

Another solution would be to write a small pax2qfile that sends directly
an archive to any vm. This tool could allow:
- to send on the fly files of a tar archive (in pax format) to another
appvm (or dom0)
- to apply filtering rules on the archive content allowing to send only
part of the archive

Joanna Rutkowska

unread,

Sep 30, 2013, 2:17:16 PM9/30/13

to Marek Marczykowski-Górecki, Olivier Médoc, qubes...@googlegroups.com

On 09/29/13 13:29, Marek Marczykowski-Górecki wrote:

Looks to complex for me: Extended Header? ASCII used for headers, which
require atoi()-like conversions, instead of simple bin struct's --
doesn't look good.

I'd rather go for tar2unpack converter (running in an AppVM) and keep
the current unpack unchanged in Dom0.

joanna.

Olivier Médoc

unread,

Sep 30, 2013, 3:16:14 PM9/30/13

to qubes...@googlegroups.com

On 09/30/13 16:17, Joanna Rutkowska wrote:
> On 09/29/13 13:29, Marek Marczykowski-G&#65533;recki wrote:
>> On 29.09.2013 10:54, Olivier M&#65533;doc wrote:

>>> On 09/28/13 16:32, Marek Marczykowski-G&#65533;recki wrote:
>>>> On 27.09.2013 19:53, Olivier M&#65533;doc wrote:
>>>>> On 09/27/13 16:56, Marek Marczykowski-G&#65533;recki wrote:

Plus several exceptions in the parsing flow to handle specific implementations / compatibility with other formats...

> I'd rather go for tar2unpack converter (running in an AppVM) and keep
> the current unpack unchanged in Dom0.

Good for me (yeah I understood, in the AppVM :P),

Then I will get onto tar2unpack, that will simplify the restore flow.

Do not hesitate if you feel that I should integrate specific features to this tool (actually, I only plan to decode pax archives). So that I can think about it in advance.

> joanna.
>

Olivier Médoc

unread,

Oct 10, 2013, 7:13:42 AM10/10/13

to qubes...@googlegroups.com

Good new, I finally have tar2qfile working at least for simple regular files. I cloned core-agent-linux repository for that:
https://github.com/ptitdoc/core-agent-linux.git

Now I need to test it with real backup files.

By the way, I have one additionnal questions:

I found that for dom0 updates, the update-vm call back the dom0 using a qubes_rpc call. This allows qubes policy to be triggered by dom0 in order to validate if the updatevm is allowed to transfert data in dom0. Should I do that for backups ? Because actually, I get data directly from qvm-run stdout.

>> joanna.
>>
>

Marek Marczykowski-Górecki

unread,

Oct 10, 2013, 1:16:36 PM10/10/13

to qubes...@googlegroups.com, Olivier Médoc

On 10.10.2013 09:13, Olivier Médoc wrote:

> On 09/30/13 17:16, Olivier Médoc wrote:
>> On 09/30/13 16:17, Joanna Rutkowska wrote:

>>> On 09/29/13 13:29, Marek Marczykowski-Górecki wrote:
>>>> On 29.09.2013 10:54, Olivier Médoc wrote:
>>>>> On 09/28/13 16:32, Marek Marczykowski-Górecki wrote:
>>>>>> On 27.09.2013 19:53, Olivier Médoc wrote:
>>>>>>> On 09/27/13 16:56, Marek Marczykowski-Górecki wrote:

:)

> Now I need to test it with real backup files.
>
> By the way, I have one additionnal questions:
>
> I found that for dom0 updates, the update-vm call back the dom0 using a
> qubes_rpc call. This allows qubes policy to be triggered by dom0 in order to
> validate if the updatevm is allowed to transfert data in dom0. Should I do
> that for backups ? Because actually, I get data directly from qvm-run stdout.

Backups are already initiated from dom0, so IMHO no need for additional verify
here. For updates it is done this way b/c there are (was?) some cases when
updatevm send updates in some automatic manner.

Marek Marczykowski-Górecki

unread,

Nov 8, 2013, 10:13:42 PM11/8/13

to qubes...@googlegroups.com, Olivier Médoc

Olivier, do you have any commits not pushed to the above repo (or
qubes-core.git)? I'm in process of merging this, testing, fixing some issues,
finishing some functions.
Especially there was a big change in repository layout between beta 2 and beta
3. I've already converted your code to the new layout.
As soon as I get the new code working, I'll push it to my git repo.

Andrew David Wong

unread,

Dec 30, 2016, 5:58:19 AM12/30/16

to Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

On 2013-08-18 10:18, Joanna Rutkowska wrote:
> On 08/17/13 16:48, Marek Marczykowski-Górecki wrote:
>>> IIRC, gpg did waaay to much parsing (related decyrpting) before
>>> veryfing
>>>> the signature, which we agreed was wrong. Openssl seemed
>>>> better in this respect.
>> [...]
>>
>> By "gpg did waaay to much parsing (related decyrpting) before
>> veryfing the signature" you basically mean that gpg isn't usable

>> to encrypt the backup in our case, right? IIUC ideal solution
>> would be to first verify all the data intergiry, then decrypt it
>> and pass to restore core. But the "verify all the data integrity"
>> part needs access to the whole backup blob (so store it
>> somewhere), which isn't doable as I said earlier... Perhaps some
>> block-based data integrity check (so can be done on the fly),
>> but I'm not aware of any existing solution like this. As said
>> earlier I'd prefer to keep our backup format as simple as
>> possible - IOW to be extractable also with simple tools (without
>> QubesOS).
>>
>> IMHO the best compromise here is to encrypt/decrypt the data with
>> gpg (symmetric) then use tar for archive and each file keep with
>> attached hmac (additional file right after). This means that
>> untrusted data is processed by gpg (decryption), then by tar and
>> then by openssl for hmac verification. Note that if you don't
>> trust gpg in regard of parsing untrusted data (decryption here),
>> probably much more components must be also reimplemented (like
>> dom0 updates - rpm signatures, installation ISO signatures,
>> enigmail for thunderbird etc). So IMO we don't have really the
>> choice about the trust in gpg.
>>
>
> We ain't need no stinkin' gpg to trust!
>
> [...]

After meditating on this thread for a little over three years, I'd like to revive it, because I think Marek made an important point here, and I don't quite understand Joanna's response.

If we don't trust GPG in our backup system because it does too much parsing of untrusted data before verification (which I believe is correct), then why do we trust it in our dom0 update system?

As a reminder, take a look at what we say about the security of the dom0 update system.[1] We tout the "separation of duties" between dom0 and the UpdateVM as a security feature (which it certainly is), but we also note that we're implicitly trusting GPG in dom0 not to be exploitable by a malicious package when it attempts to verify that package.

The only asymmetry between the two cases that I can see is that in the case of dom0 updates, the packages are not encrypted, only signed, whereas we have to assume that the backup blob may be encrypted. Therefore, our position (viz., distrusting GPG with respect to backups while trusting it with respect to dom0 updates) is consistent only if it is the case that too much pre-parsing is done when handling a signed-and-encrypted blob, whereas an acceptable amount of pre-parsing is done on an only-signed blob.

This condition, in turn, is likely to hold only if GPG handles signed-and-encrypted blobs by *first* attempting to decrypt the blob, *then* attempting to verify the integrity of the plaintext.[2] Otherwise, there should be no significant difference between the two cases in the amount of pre-parsing performed when verifying a blob. If the first step is always verification, then the amount of parsing done

before the first step should be basically the same whether the thing being verified happens to be plaintext or ciphertext.

[1] https://www.qubes-os.org/doc/software-update-dom0/#how-is-software-updated-securely-in-dom0

By the way, the last sentence in this section sounds false to me: "While we could, in theory, write a custom solution, it would only be effective if Qubes repos included all of the regular TemplateVM distro's updates, and this would be far too costly for us to maintain." This doesn't follow, since TemplateVMs are less trusted than dom0. A more secure custom solution for dom0 updates would not have to be applied to TemplateVMs in order to make Qubes as a whole more secure (by making dom0 less vulnerable to attacks via malicious update packages).

Perhaps we meant something slightly different here: All the normal distro packages *for dom0* (e.g., Fedora 23 packages, in the case of R3.2) would have to be included in the Qubes repos in order for such a solution to be effective. Even this would be true only if the custom solution were repo- or package-dependent, and only if we wanted to guarantee secure updates using the new method for all of those packages. (There's a lot to be said for making dom0 smaller, and minimizing the number of packages installed there accords with that notion.)

[2] This would seem to be a significant design flaw (perhaps a flaw in the OpenPGP spec).[3] It means that GPG does either mac-then-encrypt or mac-and-encrypt rather than the superior encrypt-then-mac.[4]

[3] http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.html

[4] http://www.daemonology.net/blog/2009-06-24-encrypt-then-mac.html

- --
Andrew David Wong (Axon)
Community Manager, Qubes OS
https://www.qubes-os.org
-----BEGIN PGP SIGNATURE-----

iQIcBAEBCgAGBQJYZfdrAAoJENtN07w5UDAwjt4QAM0U67EaippB0pYpBirxg6ER
qzwsHFEF7vSyqi7942jrZ5SUGSLMNU4j9IK6MmqD2XAp1qzSG0KGc6R7T1azFvSc
YxeFLdSnD0Y4k1UTJNmSTqMe+L8Bmud0U84aATswI0E7u8whnElPfoXrk2GJYkNJ
B8CzFWgcPSxxloqJbbXU7ueaeNL766QtPnMYNmgDBzR5zuZ8hYzgfMOoGXQKq69T
G5/rTRjqKeipkpunEmsJzqLLV3DdnnF4x81zPVsKrrNPOzmtMTQ4ihdhwFDGZiGf
PMxJdbHW7v1YmcC0T6BDmry7K3Nb3YKewIo2fv1eF8EIKMhWV0DcwhBqJzUkxtGZ
D+owcoDCitcYMRAT3DPuwLSZ1LPW6hvN98wtib/4Yw5PjYReJEieScbhg4dD5mE/
2SvidPXGaYhuQm9AZjYeSlbD5VWWBaE5XGoSkigNXdXwr1lsyUpp8o6q77MBb47M
3D4BqW0rmNJ9KMmfMmHI048YgP+dlblkeU/O9wnjdUSe6dayy37xMb7rIQMjUXv0
mA2IGWmXHHh0HAD4qN2gnpbDZ6jmwed1xPiq3koCgI2aYG0rcfHHiJzRkII1Bp0D
2asnGu7XA/vHfe5xPMEsCGpo35xLPesjrwDldzf1cup7KmyyqMiCeKxB9hgSd3+t
FcdLc+/lNrcxJ+nafOJu
=uFbd
-----END PGP SIGNATURE-----

Olivier Médoc

unread,

Dec 30, 2016, 7:54:42 AM12/30/16

to qubes...@googlegroups.com

I think the assumption was that PGP makes too much parsing during decryption/decoding of an encrypted blob. In fact it can do decryption, decompression which means more string/data manipulation routines and more potential bugs or vulnerabilities.

On this assumption, PGP has been used first to check the signature of encrypted blobs (using a HMAC algorithm based on a user provided passphrase), then and only then any program could be selected to decrypt/decompress the data. In this case this can also be PGP.

In most linux distribution PGP is used to verify packages before reading package data or extracting packages, but the reason is essentially validating package integrity and chain of trust. Not protecting against parsing=bugs=vulnerabilities.

The question may be whether one should choose PGP or another tool, but in fact the important thing is how Encryption / Signature / Trust mechanisms are used whatever the selected or combined tool, and PGP can sign then encrypt or encrypt then sign.

One interesting example is the history of package signature in Archlinux:

Archlinux only choose recently - considering its history - to sign packages and it was during years considered as unnecessary by Archlinux developpers. Now it verify packages (trust), and *optionnally* the package database that contains metadata of all packages (Actually, I think that there is no signature available for the core package database). This means that this database that involve a lot of parsing is not verified, but packages are.

I don't checked how fedora package update exactly works, especially package database files, however I remember that packages are downloaded in an AppVM (usually the default firewall vm). The package signatures could then be verified against PGP fedora PGP keys before being imported in dom0, and are definitively verified in dom0 before generating a package database file only for dom0 [1].

[1] /usr/libexec/qubes/qubes-receive-updates

Andrew David Wong

unread,

Dec 30, 2016, 10:30:42 AM12/30/16

to Olivier Médoc, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

But this only matters if decryption is performed *before* HMAC verification. If it's not -- if, instead, HMAC verification is performed first -- then there will be one of two results: Either the verification will succeed, in which case the data is trustworthy, and

decryption is safe no matter how much parsing it involves; or
verification fails, in which case the data is untrustworthy, and there
is no reason to attempt decryption or any further parsing at all.

> On this assumption, PGP has been used first to check the signature
> of encrypted blobs (using a HMAC algorithm based on a user
> provided passphrase), then and only then any program could be
> selected to decrypt/decompress the data. In this case this can also
> be PGP.
>

If that's true, then our design is inconsistent, for we trust GPG in
the case of verifying packages but not in the case of verifying
backups, even though the amount of pre-parsing should be the same in
both cases.

> In most linux distribution PGP is used to verify packages before
> reading package data or extracting packages, but the reason is
> essentially validating package integrity and chain of trust. Not
> protecting against parsing=bugs=vulnerabilities.
>

Nonetheless, protecting against parsing/bugs/vulnerabilities applies
just as much (indeed, more) to packages. An attacker may attempt to
craft a malformed package that, when parsed by GPG for verification,
exploits a bug that allows it to take over dom0. This attack vector is
more likely than a malformed backup blob, since a package can be
distributed concurrently to thousands of users through a single repo,
whereas the storage locations of user backups are highly individual,
and most backups are never restored.

> The question may be whether one should choose PGP or another tool,
> but in fact the important thing is how Encryption / Signature /
> Trust mechanisms are used whatever the selected or combined tool,
> and PGP can sign then encrypt or encrypt then sign.
>

I disagree. If GPG, when used *only* for verification, pre-parses to
such a degree that we consider it insecure for backup verification,
then we ought, on pain of inconsistency, to regard it as insecure
for any comparably security-sensitive verification task, including
for package verification.

> One interesting example is the history of package signature in
> Archlinux:
>
> Archlinux only choose recently - considering its history - to sign
> packages and it was during years considered as unnecessary by
> Archlinux developpers. Now it verify packages (trust), and
> *optionnally* the package database that contains metadata of all
> packages (Actually, I think that there is no signature available
> for the core package database). This means that this database that
> involve a lot of parsing is not verified, but packages are.
>
> I don't checked how fedora package update exactly works,
> especially package database files, however I remember that packages
> are downloaded in an AppVM (usually the default firewall vm). The
> package signatures could then be verified against PGP fedora PGP

> keys before being imported in dom0, and are definitively verified
> in dom0 before generating a package database file only for dom0
> [1].
>
> [1] /usr/libexec/qubes/qubes-receive-updates
>

As Joanna argued previously in this thread, it does little good to verify the signature of an untrusted file in an AppVM as an intermediate step before transferring it to dom0, since if the verification process allows for the compromise of that AppVM, we cannot trust it to accurately report the verification result of the file that may have compromised it.

iQIcBAEBCgAGBQJYZjc2AAoJENtN07w5UDAw2cAQALwnPDMnvDlzWImpBnKEBc6P
4ASokf5CFuaBE+8hV+y5FFBnZQmAV/FM+x6uQfe+LKNYx6YuE0xp2YgutAIYXOri
ala0PJm/wJx8fMyci6CUN7CtI+gqlGoamcH33agRQJ999n0H4Bq6q+3ExXy3ru4Z
39Xxcy//NmGeIgb1IbxlDhCgQZKlzKQtdsLXPTnQoKBwjWceCwBDcrddW6BLUxxk
edXgxUM0+/Lpy472IiNf7yz0XExMpSTy0eciSC2/980HtZOWcMAMGmPYmVY+d+78
jSTO+fIyIJN//wZ15n0HLoNfEyNAX98FvgbX084tIWn+XfSGUb17t+INy0oAaR5c
UQeH0A5YJiuLb2NYJJaXz7s3yIXm437FdkMGbx8jPAQX6ajv00QiCn5vxysFoCwA
f56UTzm+oZdNH7OVstBiEmA+Z9zRP7H5PQoMrwzA60Re4AsyL9kB13yCPIVmds+G
D/IdsESLACpY8nllcgo3idlxFOF7MM6FyEkbafVSXYtCNijr2aCRquojSlD3chb1
2LsuVWxdW+tDOZFvlFl94nkDAiPbSHG7QwT7oBKye1uIaA76edmExNIcFXXl5jig
u/d5ge5yn9rYrCV4Eb1Am8+xtSpWCn2uUuqMM5cFBpUNViywQLLB5uVtRxvTgtMK
iz7rLSR8fE56B2K5c/xA
=H5/W
-----END PGP SIGNATURE-----

Olivier Médoc

unread,

Dec 30, 2016, 12:43:16 PM12/30/16

to qubes...@googlegroups.com

Sorry, there is something I do not understand:

The restore mechanism is working like this: verify HMAC first, then cancel the operation if the data is untrustworthy.
Most package managers are working like this: verify PGP signature first, then tell the user the package is invalid if the signature is wrong or untrustworthy and stop doing things.

So it is consistent.

You should interpret "GPG is no trusted to parse package data" as "we should not give untrusted data to GPG decryption/decompression routines".

>
> > In most linux distribution PGP is used to verify packages before
> > reading package data or extracting packages, but the reason is
> > essentially validating package integrity and chain of trust. Not
> > protecting against parsing=bugs=vulnerabilities.
>
>
> Nonetheless, protecting against parsing/bugs/vulnerabilities applies

> just as much (indeed, more) to packages. An attacker may attempt to
> craft a malformed package that, when parsed by GPG for verification,
> exploits a bug that allows it to take over dom0. This attack vector is
> more likely than a malformed backup blob, since a package can be
> distributed concurrently to thousands of users through a single repo,
> whereas the storage locations of user backups are highly individual,
> and most backups are never restored.
>
> > The question may be whether one should choose PGP or another tool,
> > but in fact the important thing is how Encryption / Signature /
> > Trust mechanisms are used whatever the selected or combined tool,
> > and PGP can sign then encrypt or encrypt then sign.
>
>
> I disagree. If GPG, when used *only* for verification, pre-parses to
> such a degree that we consider it insecure for backup verification,
> then we ought, on pain of inconsistency, to regard it as insecure
> for any comparably security-sensitive verification task, including
> for package verification.

That is true, however, if you verify that the data is from a trusted source first, you limit the risk of the data being forged to trigger a vulnerability in the rest of your code. The verification routine can still be vulnerable, but this reduce your risks of being compromized, and the amount of source code that you should review.

Chris Laprise

unread,

Dec 31, 2016, 3:32:35 AM12/31/16

to Andrew David Wong, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

On 12/30/2016 12:58 AM, Andrew David Wong wrote:
>
> After meditating on this thread for a little over three years, I'd
> like to revive it, because I think Marek made an important point here,
> and I don't quite understand Joanna's response.
>
> If we don't trust GPG in our backup system because it does too much
> parsing of untrusted data before verification (which I believe is
> correct), then why do we trust it in our dom0 update system?

Ultimately, only Joanna can decide whether some guardian functions are suitable for Qubes. But I don't necessarily buy into her assertion about gpg verify: Watch some verbose output and declare 'Hark! Complexity!'. Is it /that/ worrisome for gpg to iterate over X message segments and decide if each is valid?

"If we don't trust GPG in our backup system" ...can it be trusted anywhere else?

Remember, a number of us are defending pgp/gpg in social media for general use. IIRC, Joanna recently has. Does it make sense to assume Qubes backups or updates are special cases? This issue should be taken to the GnuPG and OpenPGP fora.

> Perhaps we meant something slightly different here: All the normal
> distro packages *for dom0* (e.g., Fedora 23 packages, in the case of
> R3.2) would have to be included in the Qubes repos in order for such a
> solution to be effective. Even this would be true only if the custom
> solution were repo- or package-dependent, and only if we wanted to
> guarantee secure updates using the new method for all of those
> packages. (There's a lot to be said for making dom0 smaller, and
> minimizing the number of packages installed there accords with that
> notion.)
>
> [2] This would seem to be a significant design flaw (perhaps a flaw in
> the OpenPGP spec).[3] It means that GPG does either mac-then-encrypt
> or mac-and-encrypt rather than the superior encrypt-then-mac.[4]
>
> [3] http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.html
>
> [4] http://www.daemonology.net/blog/2009-06-24-encrypt-then-mac.html

No. 3 is really old. Does it still apply? Is there really no option that results in encrypt-then-mac?

No. 4 is interesting, but no mention of PGP and it seems focused on AES.

On update packages -- Since they're not encrypted (only signed), does it matter? Furthermore, its the repo manifests that are signed (at least on Debian).

Chris

Andrew David Wong

unread,

Dec 31, 2016, 4:25:52 PM12/31/16

to Olivier Médoc, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

I think you're forgetting that we decided not to use GPG in the backup system for precisely the reasons being discussed here (i.e., too much pre-parsing). Instead, we use OpenSSL for HMAC verification. Meanwhile, we continue to use GPG for package verification.

> You should interpret "GPG is no trusted to parse package data" as
> "we should not give untrusted data to GPG decryption/decompression
> routines".
>

But that doesn't reflect our actual position. We distrust GPG for untrusted data *verification* in the backup system. If we didn't, then we would be using GPG in the backup system.

This point is not in dispute. We all agree that verification prior to any *further* parsing is superior to no verification at all. The question is whether the amount of parsing that GPG does *prior to verification* is acceptable for security-critical components. If it's

not, then we shouldn't use it in any of them (including dom0 updates).
If it is, then we should feel free to use it any of them (including
the backup system).

>>
>>> One interesting example is the history of package signature in
>>> Archlinux:
>>
>>> Archlinux only choose recently - considering its history - to
>>> sign packages and it was during years considered as unnecessary
>>> by Archlinux developpers. Now it verify packages (trust), and
>>> *optionnally* the package database that contains metadata of
>>> all packages (Actually, I think that there is no signature
>>> available for the core package database). This means that this
>>> database that involve a lot of parsing is not verified, but
>>> packages are.
>>
>>> I don't checked how fedora package update exactly works,
>>> especially package database files, however I remember that
>>> packages are downloaded in an AppVM (usually the default
>>> firewall vm). The package signatures could then be verified
>>> against PGP fedora PGP keys before being imported in dom0, and
>>> are definitively verified in dom0 before generating a package
>>> database file only for dom0 [1].
>>
>>> [1] /usr/libexec/qubes/qubes-receive-updates
>>
>>
>> As Joanna argued previously in this thread, it does little good
>> to verify the signature of an untrusted file in an AppVM as an
>> intermediate step before transferring it to dom0, since if the
>> verification process allows for the compromise of that AppVM, we
>> cannot trust it to accurately report the verification result of
>> the file that may have compromised it.
>>

iQIcBAEBCgAGBQJYZ9wEAAoJENtN07w5UDAwtxwP/0POqTWEH0rux6xI57ugk/5W
KRkAhRhPJwegT3BnXklWAwyPgs5UYGJ0RaeqFO58FErnQFpVyMwN5CZmBmhwvYBt
8SpG6rL7eHFQfI7YV+uKW+8qNm9oFz8CuRgasIhpAj77yGMwYX28eTC+N2e1mzjN
bU343lPFtqPIYUbWcviAoYiBRyXK/3keYUoBkgK0SbBCdeSSbmuV5fLqbKzW+gEB
EMzT/Ye+Cv08BMoyTm3ddqx34XzNY5cn/8rnB1Geiz1ANdox27FcbFAtECp1lXVC
wqQqb8Ey2NyyBgy74o9cPmyI71fPmj2nTQUsEBMEZE3xOaZrjs/4OKib0MoZ4L7v
wYW6x5M+5w/7+tHNruGFJc+2UmiFU6KvcjVgX89jw3KXu2EEXjr2VLUVne17ahkU
CwF6tWKFrxNkYdJIBWfMrb3zOfLx7txXJIrK1qYNZ0H8ELu9vBMDotyCrZN0vvn/
2iiPEfBU73EI09h2n3iz9msWFvq3fNpsV3eRPY3HWoOxDw3gveu/DyhBMsORzaic
OwYR6rC4p49Z00hjwmTrym5pfxPxgSIh9hGkxIu2q6byhkSPw2UkiQY7Kw6UlzHU
T2TEwGKllxP023UouMZDKQBYuwLTpUKwybhJ6vL3npiy9sK/7oVS2KVEDvyJM8ik
Nwxj7YjngeKzbpb2lhsT
=ulSw
-----END PGP SIGNATURE-----

Andrew David Wong

unread,

Dec 31, 2016, 4:43:46 PM12/31/16

to Chris Laprise, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

On 2016-12-30 19:32, Chris Laprise wrote:
> On 12/30/2016 12:58 AM, Andrew David Wong wrote:
>>
>> After meditating on this thread for a little over three years,
>> I'd like to revive it, because I think Marek made an important
>> point here, and I don't quite understand Joanna's response.
>>
>> If we don't trust GPG in our backup system because it does too
>> much parsing of untrusted data before verification (which I
>> believe is correct), then why do we trust it in our dom0 update
>> system?
>
> Ultimately, only Joanna can decide whether some guardian functions
> are suitable for Qubes. But I don't necessarily buy into her
> assertion about gpg verify: Watch some verbose output and declare
> 'Hark! Complexity!'. Is it /that/ worrisome for gpg to iterate over
> X message segments and decide if each is valid?
>

Interesting point. I wish I knew.

> "If we don't trust GPG in our backup system" ...can it be trusted
> anywhere else?
>
> Remember, a number of us are defending pgp/gpg in social media for
> general use. IIRC, Joanna recently has. Does it make sense to
> assume Qubes backups or updates are special cases? This issue
> should be taken to the GnuPG and OpenPGP fora.
>

Yes, I think the argument can be made that Qubes backups and dom0
updates are both special cases in Qubes, since both directly concern
the security of dom0. By contrast, using GPG in an AppVM is far less
security-critical, and the risk is mitigated even further with Split
GPG.

>
>> Perhaps we meant something slightly different here: All the
>> normal distro packages *for dom0* (e.g., Fedora 23 packages, in
>> the case of R3.2) would have to be included in the Qubes repos in
>> order for such a solution to be effective. Even this would be
>> true only if the custom solution were repo- or package-dependent,
>> and only if we wanted to guarantee secure updates using the new
>> method for all of those packages. (There's a lot to be said for
>> making dom0 smaller, and minimizing the number of packages
>> installed there accords with that notion.)
>>
>> [2] This would seem to be a significant design flaw (perhaps a
>> flaw in the OpenPGP spec).[3] It means that GPG does either
>> mac-then-encrypt or mac-and-encrypt rather than the superior
>> encrypt-then-mac.[4]
>>
>> [3] http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.html

>>
>> [4]
>> http://www.daemonology.net/blog/2009-06-24-encrypt-then-mac.html
>
> No. 3 is really old. Does it still apply? Is there really no option
> that results in encrypt-then-mac?
>

No idea.

> No. 4 is interesting, but no mention of PGP and it seems focused on
> AES.
>

IANAC, but I think this is a general cryptographic principle. I don't see any reason to think that it's implementation- or algorithm-specific. (BTW, we do use AES as the default encryption algorithm in Qubes backups.)

> On update packages -- Since they're not encrypted (only signed),
> does it matter? Furthermore, its the repo manifests that are signed
> (at least on Debian).
>

Indeed, it does. That's the central point here. If GPG does too much pre-parsing before verification, and if that's why we rejected it for the backup system, then we ought, on pain of inconsistency, to reject it for the same reason in the dom0 update system.

iQIcBAEBCgAGBQJYZ+AxAAoJENtN07w5UDAwnN4QAKptm9zQoNuIS/bQY5ibBYrV
Omrb9w5Is1YO4Jfl5Ci+F/qBEC9vavJst3j8HrT03VgZHehNNnCnvZke91znjo9Z
jM/bKfiPInCkSlNmQBCa/ihriDQcpFm0h4Kcec4VmWcLadxR8xb5xK1lkfpj6NvS
nQIQJB1oHPuMSrIDCDCJITHszW2TBeQptal3fS7NNHswgehjkLkJ+vuoGCJEZM+h
aom6e5RewW/uzk+UcvdMS7h17PNeRFU8WP5o96g+bJrkS5XVgL7/jZ7N5kcmjZel
TTZ/mqjORtjUm1QjMNlxbqrxK2nOw9hgQ3PI5pilRoUVhZ0lxojTk7uCI8gL4L2N
oNXqs3VN4wVVqfFiZxu7vxGGo6DK6WgaKUfVyUgql5Tp4IDQPVZDrTENB18mM6ER
7oZ9JEtfM5tdogc1DsOqAYGU8BdbMQB90byzv9wPxllyQ14yVUs8BqCpQATnOOlf
ubl9QKzjRuNvKahxwjRG0OZ/MT+SFwXVkjOPj/+E3m/3zgeYyQ9qtBV1rRkW0UAE
gWQyUhnU43LD+Q0uYoUSoWluFF8MwkGzKPIZphjgW9hMTQgppzmvuuM9rVKafdQX
Na1FTGONoDcfbmPsSG51Y/7kl+I3+wZ64d5hHNwTXuDKvcQWADiysIx9RXxDkFwq
AKMjT73pfEjcreMGRqU2
=2bhn
-----END PGP SIGNATURE-----

Olivier Médoc

unread,

Dec 31, 2016, 8:20:49 PM12/31/16

to qubes...@googlegroups.com

Ooops, I actually forgot. That's why I did not understood your arguments :D

In this case, it possible that openssl has been selected because it supports HMAC (password verification instead of public key) ? Because I still feel that the discussion about trust was related to the usage (verification before any parsing) and that it did not specifically outed the PGP tools because of that.

One good point however is that multiplying verification tools if they have the same purpose increase the attack surface. If you use both PGP and OpenSSL, you double the chances that a vulnerability is discovered in one of those tools.

I agree that this could be investigated. If there is security critical code such as signature verification and that there are a lot of tools available, selecting the most hardened one and stripping it to avoid bad usage is probably a good idea.

Andrew David Wong

unread,

Dec 31, 2016, 10:56:12 PM12/31/16

to Olivier Médoc, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

Understandable, given that it's been over three years. :)

> In this case, it possible that openssl has been selected because it
> supports HMAC (password verification instead of public key) ?
> Because I still feel that the discussion about trust was related to
> the usage (verification before any parsing) and that it did not
> specifically outed the PGP tools because of that.
>

Re-read this thread if you don't believe me. :)

iQIcBAEBCgAGBQJYaDd7AAoJENtN07w5UDAwulwP/RWhz48K9uOPAVMOyX7Y1eJQ
0AcZuwK0x2vJU/vajGOEyGBvXLvFwXF55s5x+gxdfc5I08sJXkgO5eq8wdNNU1ul
qU4m9WElPc9cTxVMTCsfk4gVA/ZtQQLtggW5AGAPk5PbKk0eHNuqBrEeD574ba06
c2SveUKy62Ck6XgabaBPH3Pm1tCvLS3oR/RL/EK+qEPycQxBKBb9M6YXSyXZTMh5
Zw6TDoeiJcGqwKxI+o2AgBHnZi2ANB9p7+XaCSZlH7wcY/1oQzH8UdBnao7biybI
M3XtZ7BvoetvkK02NuCp2Pbf2nBcbyuFSlGMCHNGvJrpr9O8U7MJNeyFEym6qQK6
17rctpp/iCl66XUpC76dbmIerG12i+qNQ85HqSzLaLK0FCA1NzsxI4xrRHTOOXbl
pe9EVL3B0DkMX6IkTbSuKGA9UU1Kl1maQdNEOkgeimKdPitqfyiLfcR2SQoQhRBO
NkZyDL13vp8YdYrs0U7rgEC5jRT6ZrFwg79GCvVXpC9Y7QsMOTt1+FmJem7Y87CE
izYm9HKbEfyXzoFfDmsOzUJkMkX+l9f5DssWoiOTRaU80sG4f0dlGh8EZWWjHCRX
+Ae0eodJ7UR6BzX/3h5L3JV731MljZzO7KaxhPcKPI56dEl/oEEbp6xAZj9XOlt3
bCapE1UGt1BUZoVHNu5q
=cLFO
-----END PGP SIGNATURE-----

Chris Laprise

unread,

Jan 1, 2017, 11:20:09 PM1/1/17

to Andrew David Wong, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

On 12/31/2016 11:43 AM, Andrew David Wong wrote:
> -----BEGIN PGP SIGNED MESSAGE-----
> Hash: SHA512
>
> On 2016-12-30 19:32, Chris Laprise wrote:

>> On 12/30/2016 12:58 AM, Andrew David Wong wrote:
>>> After meditating on this thread for a little over three years,
>>> I'd like to revive it, because I think Marek made an important
>>> point here, and I don't quite understand Joanna's response.
>>>
>>> If we don't trust GPG in our backup system because it does too
>>> much parsing of untrusted data before verification (which I
>>> believe is correct), then why do we trust it in our dom0 update
>>> system?
>> Ultimately, only Joanna can decide whether some guardian functions
>> are suitable for Qubes. But I don't necessarily buy into her
>> assertion about gpg verify: Watch some verbose output and declare
>> 'Hark! Complexity!'. Is it /that/ worrisome for gpg to iterate over
>> X message segments and decide if each is valid?
>>
> Interesting point. I wish I knew.

If Qubes devs are going to start making commitments of time and effort based on such claims, and disrupting the admin aspect of using Qubes as well, I'd advise you to back them up with something rigorous before leading the community down that path.

Laboring under the assumption that GPG is bad, you won't even know if your Qubes-curated Fedora/Debian packages are good unless you get those projects to switch to your alternative as well.

>
>> "If we don't trust GPG in our backup system" ...can it be trusted
>> anywhere else?
>>
>> Remember, a number of us are defending pgp/gpg in social media for
>> general use. IIRC, Joanna recently has. Does it make sense to
>> assume Qubes backups or updates are special cases? This issue
>> should be taken to the GnuPG and OpenPGP fora.
>>
> Yes, I think the argument can be made that Qubes backups and dom0
> updates are both special cases in Qubes, since both directly concern
> the security of dom0. By contrast, using GPG in an AppVM is far less
> security-critical, and the risk is mitigated even further with Split
> GPG.

GPG is used by developers, whistleblowers, activists and journalists on X number of platforms. Some of those platforms use it for secure system updates. Sorry, but this aspect of Qubes isn't special --- When non-Qubes users use GPG in whatever role, it is just as critical to *their* security.

I worry that a foolish consistency may be setting in.

Chris

Andrew David Wong

unread,

Jan 2, 2017, 1:22:38 AM1/2/17

to Chris Laprise, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

On 2017-01-01 15:20, Chris Laprise wrote:
> On 12/31/2016 11:43 AM, Andrew David Wong wrote:
>> On 2016-12-30 19:32, Chris Laprise wrote:
>>> On 12/30/2016 12:58 AM, Andrew David Wong wrote:
>>>> After meditating on this thread for a little over three years,
>>>> I'd like to revive it, because I think Marek made an important
>>>> point here, and I don't quite understand Joanna's response.
>>>>
>>>> If we don't trust GPG in our backup system because it does too
>>>> much parsing of untrusted data before verification (which I
>>>> believe is correct), then why do we trust it in our dom0 update
>>>> system?
>>> Ultimately, only Joanna can decide whether some guardian functions
>>> are suitable for Qubes. But I don't necessarily buy into her
>>> assertion about gpg verify: Watch some verbose output and declare
>>> 'Hark! Complexity!'. Is it /that/ worrisome for gpg to iterate over
>>> X message segments and decide if each is valid?
>>>
>> Interesting point. I wish I knew.
>
> If Qubes devs are going to start making commitments of time and effort
> based on such claims, and disrupting the admin aspect of using Qubes as
> well, I'd advise you to back them up with something rigorous before
> leading the community down that path.
>

You're misinterpreting the nature of this conversation. It's just a
discussion. No decisions or commitments have been made. I'm primarily
asking questions. The claims I'm making are conditional in nature. It's
quite possible that no action is warranted, but that's by no means obvious.

> Laboring under the assumption that GPG is bad, you won't even know if
> your Qubes-curated Fedora/Debian packages are good unless you get those
> projects to switch to your alternative as well.
>

First, I'm not laboring under that assumption. It's one of the things
about which I'm inquiring.

Second, that's not necessarily true. There could be solutions that don't
require other projects to change anything (for example, a hypothetical
system in which we first compute the hash of a package with minimal
pre-parsing, compare the hash value to a distributed database, then, if
it matches, proceed to verify the package's signature with GPG).

>>
>>> "If we don't trust GPG in our backup system" ...can it be trusted
>>> anywhere else?
>>>
>>> Remember, a number of us are defending pgp/gpg in social media for
>>> general use. IIRC, Joanna recently has. Does it make sense to
>>> assume Qubes backups or updates are special cases? This issue
>>> should be taken to the GnuPG and OpenPGP fora.
>>>
>> Yes, I think the argument can be made that Qubes backups and dom0

>> updates are both special cases in Qubes, since both directly concern
>> the security of dom0. By contrast, using GPG in an AppVM is far less
>> security-critical, and the risk is mitigated even further with Split
>> GPG.
>
> GPG is used by developers, whistleblowers, activists and journalists on
> X number of platforms. Some of those platforms use it for secure system
> updates. Sorry, but this aspect of Qubes isn't special --- When
> non-Qubes users use GPG in whatever role, it is just as critical to
> *their* security.
>

You misunderstand me. I clearly stated that these may be special cases
of GPG usage *in Qubes* compared to other kinds of GPG usage *in Qubes*.
The claim is not that the security of Qubes backups and dom0 updates is
more important than the security of GPG usage in non-Qubes contexts.
Rather, it's simply that Qubes backups and dom0 updates are special
cases *within Qubes* since they directly concern the security of dom0.

No offense, but this reads like a feeble jab attempting to mask the
absence of a real argument. I encourage you to say what you mean and
mean what you say. An unwillingness to engage in a clear and direct
rational discussion will get us nowhere.

For what it's worth, I have absolutely no problem with differences of
opinion or with being told that I'm wrong. Here's an example of a
dissenting response that I'd actually be inclined to agree with:

"Sure, our position is probably inconsistent, but it's not worth trying
to change it. Here's why: Even if GPG does too much pre-parsing when
verifying packages, it would be exceedingly difficult for an attacker to
exploit it because [reasons], and it would be extremely difficult to
fix. By contrast, it would be much easier to attack Qubes users through
[other vectors], so we should just live with the inconsistency for now
and focus our efforts on those other vectors, which will take up all of
our time for the foreseeable future. To put it another way: GPG may
theoretically be exploitable, but it's still one of the safest parts of
the system. So, this isn't even the bad kind of inconsistency. It just
means that the backup system is *extra* secure (i.e., more conservative
than it needs to be). That's not a bad thing, but it doesn't follow that
other components, like the update system, also have to be more
conservative than they need to be."

iQIcBAEBCgAGBQJYaasxAAoJENtN07w5UDAw8QMQAIlXtgpeT4hkjoxpgxDT9Cja
szqeRLV34jWyTApH01twr9ZjaUlBGnYpdnCz9tMlJk9k0Cg+7g8UHssnPIcF0+vV
KOxJL+jc2GlitnuTfhwqOd2P91eENy5QCYegePZuiR13bzozNu40vCI/jWiNv7vV
1B8jjG9eV2NC5H/BdB1lf7qkfk/yEcuWPnI7K/NO5tmElm1hSqd5KMzj+hP/rILa
+277W3XzAP2H+a9JOKo98p/CAREOnXC5qJNEQfFmgyGHiViFBcT3rHctGGp3tm/9
Vq/VJrJD/WoAapOtiM0Mfd4uohrOUc3tNoIyXEdrG5J89UzHv6RLUzP5Bl1OJi2D
V5NkKAVczjfvPXIlxS7wz3dlO41Aj9RAfPcymuXWMOvnEmH308fosKIsrfFYBro5
pw412UxL5d5b95K4y+FYc7tCy180gV6EPX1nNJQCIrxfTNJ8KxMBkkE0MJbFetSl
2YS/xGVnmhvDd8xcET2Yv1nfVO6VKG0zNcQffB3Jc44kwI8aFBhfKffpLxb14Bvu
7lDD1h8QHRU+GUnkq1eZIRJNOLPFwTYdazjP8yRGgw+2Gm4HWnW3e5JgszPXdjKd
IHe4+IcmNXlq/s+YrsBCjtAD7yn8+OJUcsG4zuZJf+UEZZgonuSaNi7hCzJ3AOWw
U+O1dUst2P/mC3QrBONn
=AiLu
-----END PGP SIGNATURE-----

Chris Laprise

unread,

Jan 6, 2017, 5:35:45 AM1/6/17

to Andrew David Wong, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

Given this thread was used to make implementation decisions in
qmv-backup, the suggestion of extending this assumption about GPG to
updates looked more than hypothetical to me (and being about multi-party
transactions, it happens to involve far more effort and complexity than
backups). Thanks for the clarification. :)

>> Laboring under the assumption that GPG is bad, you won't even know if
>> your Qubes-curated Fedora/Debian packages are good unless you get those
>> projects to switch to your alternative as well.
>>
> First, I'm not laboring under that assumption. It's one of the things
> about which I'm inquiring.

> Second, that's not necessarily true. There could be solutions that don't
> require other projects to change anything (for example, a hypothetical
> system in which we first compute the hash of a package with minimal
> pre-parsing, compare the hash value to a distributed database, then, if
> it matches, proceed to verify the package's signature with GPG).

If the Qubes Project is not authoring some of the packages, it sounds
unlikely to work as a primary step in verifying integrity. This may
/necessarily/ be true until the upstream suppliers for dom0 change.

Assuming GPG is an attack vector (a BIG assumption), I don't think
having users compare hashes with each other over HTTPS links, for
example, is going to suffice in addition to GPG if the former won't
suffice on its own. We would need some way of obtaining upstream
packages without any invocation of GPG and then verify them somehow
without invoking GPG. After that point (assuming at least some packages
will need to be compiled) we need to deal with the much more real threat
of the /compiler/ being targeted for attack. That is all /prior/ to the
step of generating a hash and uploading it to a database.

--

I will say that if moving away from GPG does turn out to be warranted
(though I'll make a guess that it won't be), I'd suggest replacing it in
the development procedures first before trying it with general
distribution---fundamental changes should be practiced by developers
first. Beyond that, trialling a new verification procedure on the ISO
download page would make sense, since the image taken as a whole is a
work authored by the Qubes Project.

Chris

Andrew David Wong

unread,

Jan 6, 2017, 11:24:24 AM1/6/17

to Chris Laprise, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

It sounds like you may be focusing exclusively on the hypothetical
example at the expense of the general point, but for the sake of
discussion:

> If the Qubes Project is not authoring some of the packages, it
> sounds unlikely to work as a primary step in verifying integrity.
> This may /necessarily/ be true until the upstream suppliers for
> dom0 change.
>

I don't see why. Maybe we're thinking of different things. Here's the
general idea:

https://github.com/QubesOS/qubes-issues/issues/2524

> Assuming GPG is an attack vector (a BIG assumption), I don't think
> having users compare hashes with each other over HTTPS links, for
> example, is going to suffice in addition to GPG if the former
> won't suffice on its own.

It's not supposed to. The idea is that hashing with minimal
pre-parsing is a safe first-pass that mitigates (but does not
eliminate) the risk of GPG verification (with pre-parsing). GPG
verification is not replaced; it's still required.

> We would need some way of obtaining upstream packages without any
> invocation of GPG and then verify them somehow without invoking
> GPG.

That's the whole idea. Roughly:

1. Download <package>.
2. $ sha512sum <package>.
3. Check hash.

If match:

4. $ gpg --verify <package>.

If no match:

5. $ rm <package>.

> After that point (assuming at least some packages will need to be
> compiled) we need to deal with the much more real threat of the
> /compiler/ being targeted for attack.

This seems like an orthogonal threat that exists either way, though.

> That is all /prior/ to the step of generating a hash and uploading
> it to a database.
>
> --
>

> I will say that if moving away from GPG does turn out to be
> warranted (though I'll make a guess that it won't be), I'd suggest
> replacing it in the development procedures first before trying it
> with general distribution---fundamental changes should be
> practiced by developers first.

Indeed, that's why I'm such a fan of this:

https://github.com/QubesOS/qubes-issues/issues/2524

> Beyond that, trialling a new verification procedure on the ISO
> download page would make sense, since the image taken as a whole
> is a work authored by the Qubes Project.
>
> Chris
>

iQIcBAEBCgAGBQJYb35TAAoJENtN07w5UDAwGMwP/03gNojFlXxodsFm6BzRLA5/
Wa/x1Q3N3ZyGoz48adyctybdC1Q97/x6kJiB+sXQE1TGl/uTopOSLXMv4SMlFShy
XbUPAXpgz+zUIrz+R+85ODU5RpCbYu+Vw9S2FIlb6ccjFpy82DzXSI/Ew4mA2YHT
w+gTMmuIkMqJN4/Vp+XahHpPL6fwOdheXGsrgUQMtZBntsMTkUW6hG9DiTL2vp/V
h1dKYV0+VEaZ+7J67HhTELkVddKzAnSVJLKw6T3TDa2QK+CuqGkNAdVZFMYmbdPS
gQTy/EFi1Hga+QB9AL+tLaaW/vHk3xC3CRT/f0YPwlwUpM304koCE9VHvUIa1K26
YRYORM/aex76hmUSN+//dAyBiGExi1cUSJe1/pZR6xHli7JLTDvgvLVxHKp5wy/G
C/ncvelj6TvZEScpVoCZlxewqZOiL5qOyebHN1aNDAa9tDvUwR5mr7tmfhjt8tvt
nntooc8HkrVwg2x2cRIV99Ee9aopzsamF8G5gVtG9Uw6GYwPm2/9l1RaQ7FNohOU
j1j0moU2dcL3wfQEJ0mvjff+lE/CxfwTgbLf0bsviLtzHGlvsv3/oNwgDGpxxv6l
vXUdfeleQ29P5vD/Cv7tGu/tkbL5AbLUUSAbo2IS3uGl+ucscilZ2SRMdpi5vFQ2
DJdTU3/pKWfao8LXqs/P
=ul19
-----END PGP SIGNATURE-----

Chris Laprise

unread,

Jan 6, 2017, 8:28:10 PM1/6/17

to Andrew David Wong, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

On 01/06/2017 06:24 AM, Andrew David Wong wrote:
>
> It sounds like you may be focusing exclusively on the hypothetical
> example at the expense of the general point, but for the sake of
> discussion:
>
>> If the Qubes Project is not authoring some of the packages, it
>> sounds unlikely to work as a primary step in verifying integrity.
>> This may /necessarily/ be true until the upstream suppliers for
>> dom0 change.
>>
> I don't see why. Maybe we're thinking of different things. Here's the
> general idea:
>
> https://github.com/QubesOS/qubes-issues/issues/2524
>
>> Assuming GPG is an attack vector (a BIG assumption), I don't think

>> having users compare hashes with each other over HTTPS links, for
>> example, is going to suffice in addition to GPG if the former
>> won't suffice on its own.
> It's not supposed to. The idea is that hashing with minimal
> pre-parsing is a safe first-pass that mitigates (but does not
> eliminate) the risk of GPG verification (with pre-parsing). GPG
> verification is not replaced; it's still required.

If neither GPG nor 'collaborative hashing' is considered safe on its
own, and if GPG is a supposed attack vector, then the weaknesses of the
two procedures could *increase* risk instead of canceling it out.

This 'great new idea' may be unique for good reasons, the primary one
being that simple GPG verification is considered safe.

>
>> We would need some way of obtaining upstream packages without any
>> invocation of GPG and then verify them somehow without invoking
>> GPG.
> That's the whole idea. Roughly:
>
> 1. Download <package>.
> 2. $ sha512sum <package>.
> 3. Check hash.
>
> If match:
>
> 4. $ gpg --verify <package>.
>
> If no match:
>
> 5. $ rm <package>.

What happens if you're the first one to hash a package? You don't get to
use GPG at all? Or the order gets reversed... GPG is used before hashing
and exploit might ensue?

Is the hash uploaded before or after running GPG?

>> After that point (assuming at least some packages will need to be
>> compiled) we need to deal with the much more real threat of the
>> /compiler/ being targeted for attack.
> This seems like an orthogonal threat that exists either way, though.

It doesn't really exist either way, does it? A big reason why I'm
debating the thinking behind this is that its based on a low-quality
assessment done here:

https://groups.google.com/d/msg/qubes-devel/TQr_QcXIVww/SHLNsoPgWTAJ

That is the whole 'I don't trust GPG' rationale right there. My takeaway
from it could be that verbose mode gives me super-powers of perception
with hardly any effort ...or not.

>
>> That is all /prior/ to the step of generating a hash and uploading
>> it to a database.
>>

>> --
>>
>> I will say that if moving away from GPG does turn out to be
>> warranted (though I'll make a guess that it won't be), I'd suggest
>> replacing it in the development procedures first before trying it
>> with general distribution---fundamental changes should be
>> practiced by developers first.
> Indeed, that's why I'm such a fan of this:
>
> https://github.com/QubesOS/qubes-issues/issues/2524

I didn't see where hashes are shared and compared. Are they?

Chris

Andrew David Wong

unread,

Jan 7, 2017, 8:18:40 AM1/7/17

to Chris Laprise, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

On 2017-01-06 12:27, Chris Laprise wrote:
> On 01/06/2017 06:24 AM, Andrew David Wong wrote:
>>
>> It sounds like you may be focusing exclusively on the
>> hypothetical example at the expense of the general point, but for
>> the sake of discussion:
>>
>>> If the Qubes Project is not authoring some of the packages, it
>>> sounds unlikely to work as a primary step in verifying
>>> integrity. This may /necessarily/ be true until the upstream
>>> suppliers for dom0 change.
>>>
>> I don't see why. Maybe we're thinking of different things. Here's
>> the general idea:
>>
>> https://github.com/QubesOS/qubes-issues/issues/2524
>>
>>> Assuming GPG is an attack vector (a BIG assumption), I don't
>>> think having users compare hashes with each other over HTTPS
>>> links, for example, is going to suffice in addition to GPG if
>>> the former won't suffice on its own.
>> It's not supposed to. The idea is that hashing with minimal
>> pre-parsing is a safe first-pass that mitigates (but does not
>> eliminate) the risk of GPG verification (with pre-parsing). GPG
>> verification is not replaced; it's still required.
>
> If neither GPG nor 'collaborative hashing' is considered safe on
> its own, and if GPG is a supposed attack vector, then the
> weaknesses of the two procedures could *increase* risk

How so?

> instead of canceling it out.

I never said anything about "canceling it out."

> This 'great new idea' may be unique for good reasons,

Why did you put that in quotation marks? I never said it was a "great new idea." It's just a hypothetical example that I used to illustrate a more general point in the foregoing discussion.

> the primary one being that simple GPG verification is considered
> safe.

Well, there are many things that are considered safe by the general public (and even by the general computer security community) that are not considered safe in Qubes. For example, our position of distrusting the infrastructure seems to regard as unsafe (or at least insufficiently trustworthy) much of what most people consider safe (e.g., HTTPS).

It's at least *possible* that the GPG pre-parsing issue indicates a genuine risk. A lot of security software is considered safe until bugs are later discovered.

>>
>>> We would need some way of obtaining upstream packages without
>>> any invocation of GPG and then verify them somehow without
>>> invoking GPG.
>> That's the whole idea. Roughly:
>>
>> 1. Download <package>. 2. $ sha512sum <package>. 3. Check hash.
>>
>> If match:
>>
>> 4. $ gpg --verify <package>.
>>
>> If no match:
>>
>> 5. $ rm <package>.
>
> What happens if you're the first one to hash a package? You don't
> get to use GPG at all?

I suppose it depends on what your goals are. If you're the package maintainer, maybe you're providing the first hash to the database. If you're trying to be helpful, maybe you're finding packages without hashes in the database so that you can contribute them. If you're an end user, maybe you decide to wait a few hours to see if any hashes for that package get added.

> Or the order gets reversed... GPG is used before hashing and
> exploit might ensue?
>

If there's a script performing this sequence of operations, it's hard to see why the order would get reversed. In general, in any security-sensitive software, very bad things can happen if the sequence of operations somehow start getting arbitrarily reversed.

> Is the hash uploaded before or after running GPG?
>

That depends on the implementation. Perhaps something like:

1. Download the database.
2. Hash your package.
3. Check your hash against the database.
4. If you find a match, gpg --verify the package.
5. If verification is successful, upload your hash from step 2 to the database.

>
>>> After that point (assuming at least some packages will need to
>>> be compiled) we need to deal with the much more real threat of
>>> the /compiler/ being targeted for attack.
>> This seems like an orthogonal threat that exists either way,
>> though.
>
> It doesn't really exist either way, does it?

Why not? Assume the verification scheme of your choice. A package either succeeds verification under that scheme, or it does not. If it succeeds, then it should be trustworthy (since you've verified it as originating from a source you've decided to trust). If it proceeds to attack your compiler, then you've misplaced your trust, and you should no longer trust that source. If it does not succeed verification, then you should not attempt to compile it, since it does not originate from a source you've decided to trust.

> A big reason why I'm debating the thinking behind this is that its
> based on a low-quality assessment done here:
>
> https://groups.google.com/d/msg/qubes-devel/TQr_QcXIVww/SHLNsoPgWTAJ
>
> That is the whole 'I don't trust GPG' rationale right there. My
> takeaway from it could be that verbose mode gives me super-powers
> of perception with hardly any effort ...or not.
>

I don't follow your reasoning. Would you mind laying out your argument more explicitly?

>>
>>> That is all /prior/ to the step of generating a hash and
>>> uploading it to a database.
>>>
>>> --
>>>
>>> I will say that if moving away from GPG does turn out to be
>>> warranted (though I'll make a guess that it won't be), I'd
>>> suggest replacing it in the development procedures first before
>>> trying it with general distribution---fundamental changes
>>> should be practiced by developers first.
>> Indeed, that's why I'm such a fan of this:
>>
>> https://github.com/QubesOS/qubes-issues/issues/2524
>

> I didn't see where hashes are shared and compared. Are they?
>

No, #2524 has to come first. The part about sharing and comparing
hashes is just a hypothetical example inspired by #2524.

iQIcBAEBCgAGBQJYcKRQAAoJENtN07w5UDAwqxAP/0RtFEHz3F6PoK9xSH+V5qX5
821US8y9AWRjhC93/KbkKzpnHgyzUU7h4ed6Iw01pqW/24uhDr3gyo9bTebmmDhj
jiB9w4we2sgVs0kzGzE1T5fELJ/g6fTsgCUEOJmKEmQl0S22Nh7XxS/i3gl6Fx5k
gx0KebdY7/XyvPgx74JnrFdRe4XukB5uRE7p4gteBLzLbXtvepZk27gh0j9mZcT+
+H76LcrTioYqoepCEjx8hGEmfbOMZTpKd4MiVk3orpmy9Xu8fcbi4vH5AegxUpgt
Nes18alCRve4n5rdclEVZ3yS11Vxh02YdxiHCIC6MHGZdhWZPh6yFrQdT0U/MPE0
FQngphinS7F+SwVHWJTRcc2uj4EsCIUsbq6IvqS8I971Y5BF1MEIAPK9Nri9e/Yj
T8QnaHvEbRcoBQi67P6uR8RXePA5xhW+kHggWRDlGe5GnqQ9Gljy1sZbA8v2TSLk
iwJWeEHZZyXNZ0os83OVFU2a97XbiFPwbqHVFDRGRofleiFHbSn89//MOTjUropo
mPwj6RK0ga++SHLs39PxqYZcB1aG3rVQIK3WFpts+xdV+GoVUYyZFJgsgW39ciKp
78dRNOKAzg7Yy9hkF8AODu4UtgNV/1OXVnVLSIxW6NpgWc9wCRUEABkBGPI6ghFd
ozQ1m6H5G8t3/Y/ttcuA
=HEUu
-----END PGP SIGNATURE-----

Chris Laprise

unread,

Jan 7, 2017, 4:48:24 PM1/7/17

to Andrew David Wong, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

I think its up to advocates to state how it does *not* increase risk
when adding complexity to a process. Doubly so if discrete verification
steps are performed across networks without signatures.

>> instead of canceling it out.
> I never said anything about "canceling it out."

Well, if its not intended to cancel out some of the supposed 'risk'
(single quotes) of verifying with GPG, then what is it?

>> This 'great new idea' may be unique for good reasons,
> Why did you put that in quotation marks? I never said it was a "great
> new idea." It's just a hypothetical example that I used to illustrate
> a more general point in the foregoing discussion.

I used single 'ironic' quotes because they're my words and I don't
believe them to be true.

>> the primary one being that simple GPG verification is considered
>> safe.
> Well, there are many things that are considered safe by the general
> public (and even by the general computer security community) that are
> not considered safe in Qubes. For example, our position of distrusting
> the infrastructure seems to regard as unsafe (or at least
> insufficiently trustworthy) much of what most people consider safe
> (e.g., HTTPS).

Distrust of infrastructure is applicable to this subject as well, and
its where the focus needs to stay if the debate is to get anywhere.

Focusing on hashing operations performed in a single computer doesn't get anywhere.

As for what's considered safe "in Qubes" (supposing you mean the community or the project) I'll just say I don't think concern should be mistaken for expertise, especially when it comes to crypto tools. Qubes' particular expertise lies in domain isolation, and our expert on that matter is Joanna. That was enough to start and maintain an unusual new OS like Qubes because regular PCs were lacking in high-quality isolation. But all that does not bestow any special insight about crypto tools; that is a different animal and the stakes (and interest level) for people using crypto on different platforms is just as high.

Maybe Joanna, Marek or someone else could analyze the non-crypto parts of verification in GPG or the OpenPGP standard, and the analysis might be credible, but I don't see that happening. If it did, I would welcome it.

What I do not want is to see the Qubes community become a part of the anti-PGP bandwagon and acquire a boondoggle in the process. Its already bad enough we have a Fedora update process in dom0 that allows MITM to selectively withhold updates to packages, including Xen.

>
> It's at least *possible* that the GPG pre-parsing issue indicates a
> genuine risk. A lot of security software is considered safe until bugs
> are later discovered.

Every year multiple bugs are found in Xen's isolation. Does that mean I should run Xen inside a different type of container to 'help' it? I think its the design complexity thats really at issue.

>>>> We would need some way of obtaining upstream packages without
>>>> any invocation of GPG and then verify them somehow without
>>>> invoking GPG.
>>> That's the whole idea. Roughly:
>>>
>>> 1. Download <package>. 2. $ sha512sum <package>. 3. Check hash.
>>>
>>> If match:
>>>
>>> 4. $ gpg --verify <package>.
>>>
>>> If no match:
>>>
>>> 5. $ rm <package>.
>> What happens if you're the first one to hash a package? You don't
>> get to use GPG at all?
>
> I suppose it depends on what your goals are. If you're the package
> maintainer, maybe you're providing the first hash to the database. If
> you're trying to be helpful, maybe you're finding packages without
> hashes in the database so that you can contribute them. If you're an
> end user, maybe you decide to wait a few hours to see if any hashes
> for that package get added.

IOW, such a process would have to come with a ton of disclaimers and caveats about its consistency and effectiveness. In addition, the awful taint of 'exploitable' public key crypto will have no place in the

upload or download process?

Here's an idea :) Why don't you *sign* the hashes? :))

You know... for safety...

I love the original idea of distributed verification laid out by ITL. Somehow (rightly, I think) I assumed all that would be secured by cryptographic signatures. What's being described here (so far) is neither distributed nor secure; It lacks an acknowledgement of the basic, high-risk design obstacles associated with multi-party communications. When distributed verification for the development process is figured out, then at that point we may be in a position to extend it to enhance dom0 updates.

The way I see it: If the vague, corrosive distrust of GPG/PGP continues, the only options will be to eventually review the code or move to another signing tool.

Chris

Andrew David Wong

unread,

Jan 8, 2017, 4:45:49 PM1/8/17

to Chris Laprise, Joanna Rutkowska, Marek Marczykowski-Górecki, qubes...@googlegroups.com

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

Hashing a file with minimal pre-parsing doesn't, ex hypothesi, expose the system to any risks associated with parsing an untrusted file, so no risk from this vector is added over GPG verification.

>
>>> instead of canceling it out.
>> I never said anything about "canceling it out."
>

It only *mitigates* the risk; it doesn't "cancel it out." The idea is that you do two passes: The first pass is a quick and dirty (but safe) check. The second is a slow and careful (but possibly risky) check. Using two passes makes the procedure safer for you and more difficult for your adversary.

Doesn't all the best security software come with such disclaimers and caveats (Tor, GPG, Whonix, Qubes, etc.)?

> In addition, the awful taint of 'exploitable' public key crypto
> will have no place in the upload or download process?

I'm not sure how you got the idea that I think of public key crypto in those terms, but nothing could be further from the truth.

> Here's an idea :) Why don't you *sign* the hashes? :))
>
> You know... for safety...

>

I think that could be a great idea, similar to this:

https://github.com/rootkovska/codehash.db

> I love the original idea of distributed verification laid out by
> ITL. Somehow (rightly, I think) I assumed all that would be secured
> by cryptographic signatures. What's being described here (so far)
> is neither distributed nor secure; It lacks an acknowledgement of
> the basic, high-risk design obstacles associated with multi-party
> communications. When distributed verification for the development
> process is figured out, then at that point we may be in a position
> to extend it to enhance dom0 updates.
>
> The way I see it: If the vague, corrosive distrust of GPG/PGP
> continues, the only options will be to eventually review the code
> or move to another signing tool.

You seem to think that I have something against GPG or PGP. I certainly don't. As a matter of fact, I count myself among their biggest fans. This was just (supposed to be) a thoughtful discussion about whether GPG, as a tool, can or should be more defensive when it verifies untrusted input.

In any case, I think this discussion has run the course of its usefulness (for me, anyway), so I'm taking my leave from it.

iQIcBAEBCgAGBQJYcmyIAAoJENtN07w5UDAwSEUQAMLjJqctVbeVqbmFOChvqVmV
THVluTnM1eDIV22ViqmFYjxmU/GJJtrBmBxtHqIKoYmYAY0OtGevfFCnkgrmOoKv
KnDfm5js5mLciB12jwcLRdxhBNuVORzU290w5EjXmMpsmdqLBwn4e8j9D9UMKJTf
7194kVnGWNvbvOTj1zZ7aqAbTG1eJBi5wZ433PDHmK/V06//Gidqq/jyQdNDczF7
cuc5VshDg/HaqDpFWSF6M8lLRJ7N9x1iSw6h8Gpc4Be24pKSGKILU9lMzCA5jDjj
rN+NQsOR+4B9vHvjbq5v+rWrOm2Acq6oyxrqAVvgRBgeV8sRpR5KcCwcpHnhmnW5
Jd1i5G64oowSjJ2xS8Ixp70Hj3WzA13SKmzQlRz5Z+wslZ0Uk81OjSmsGQ8V+/k7
WjQu0IFxw0da5fW0/wTiMSinBmR2PplS2oz/IVEoh8CU+TFqiOsuL9PQx8vVq7it
DMF+yfauWM0KNGRDS1kCXMef771Kd5pjUnBC3pvBvYsmM4ULO68woVuYyidvgNpz
p0wRry8PibKDZ9FJ1h0mA39fP5nbhnXJ/+O0wy24GmW5jrZ72UysBSCn7FZ/RxiR
FpRMqhmabbx3hq6MCz/lIOQvHEyc+XJrTjwTpnyBndrjs+VoEOq2A2Xhxod31f+2
iZOoFU4uRyooi/ByOPcX
=i9K4
-----END PGP SIGNATURE-----

Ángel

unread,

Jan 10, 2017, 11:40:03 PM1/10/17

to qubes...@googlegroups.com

Chris Laprise wrote:
> It doesn't really exist either way, does it? A big reason why I'm
> debating the thinking behind this is that its based on a low-quality
> assessment done here:
>
> https://groups.google.com/d/msg/qubes-devel/TQr_QcXIVww/SHLNsoPgWTAJ
>

> That is the whole 'I don't trust GPG' rationale right there. My
> takeaway from it could be that verbose mode gives me super-powers of
> perception with hardly any effort ...or not.

I think this thread digressed too much.

Note that the revived thread deals just with package verification. You are not dealing with full gpg files. Actually, I would expect the upstream package manager to use gpgv, not gpg, thus reducing the attack surface.

Per the man page:
> It is somewhat smaller than the fully-blown gpg and uses a different
> (and simpler) way to check that the public keys used to make the signature are
> valid.

and indeed on a cursory look, it is clear that it handles a only a limited subset of packets (PKT_SIGNATURE/PKT_ONEPASS_SIG, PKT_PLAINTEXT, PKT_COMPRESSED and PKT_GPG_CONTROL),
although I agree it is still quite complex. :-(

Jean-Philippe Ouellet

unread,

Jan 20, 2017, 8:13:23 AM1/20/17

to qubes-devel, Joanna Rutkowska, Andrew David Wong, Marek Marczykowski-Górecki

I would like to bring this list's attention to the availability of what I believe to be a good non-OpenPGP solution to the problem of cryptographically verifying code.

The OpenBSD community has had very similar discussions internally several years ago, and they resulted in the implementation of a minimal non-OpenPGP signature creation & verification tool called signify, using the NaCl primitives [1]. It has been successfully used for package, release, and advisory signing for several years now, and has been audited both within and without the OpenBSD community.

[1]: https://nacl.cr.yp.to/

Its implementation is small enough that some years ago I personally read the whole thing down to libc, (minus the crypto code which came directly from reference implementations from well-regarded cryptographers), and doing so was not a large undertaking. By contrast, I could barely even keep all the state in my head to properly review even the ASN.1 parser which GnuPG must (by nature of OpenPGP) use to parse attacker-controlled input before any pgp-specific data structures can be extracted, before the key can be identified, before it can be used to verify the rest of the data, and then just hope that all other tools parse in the exact same way, and that no later steps ever trust any data that was not previously verified. Current langsec research empirically suggests such constructions are dangerous and should be avoided [2] regardless of how skilled and careful their implementor(s) may be. The popularity of GnuPG among certain communities is irrelevant to discussions of implementation safety.

[2]: http://langsec.org/papers/langsec-cwes-secdev2016.pdf

For more info on signify, I'd recommend reading it's man page [3], this post by its author [4], this (old) HN discussion [5], and of course also it's source code [6] [7] which is purposefully small and easy to audit (and more importantly, has been!).

[3]: http://man.openbsd.org/signify
[4]: http://www.tedunangst.com/flak/post/signify
[5]: https://news.ycombinator.com/item?id=9708120
[6]: http://cvsweb.openbsd.org/cgi-bin/cvsweb/src/usr.bin/signify/
[7]: http://cvsweb.openbsd.org/cgi-bin/cvsweb/~checkout~/src/usr.bin/signify/signify.c

and note also that it is not just some obscure one-off format/tool, but rather has been seen much review and community adoption outside *BSD circles, including several portable versions (with [8] being the best maintained) and several format-compatible independent implementations including a library [9] and slightly extended tool (with the addition of signed metadata) named minisign [10] written by the maintainer of libsodium (the portable version of NaCl).

[8]: https://github.com/aperezdc/signify
[9]: https://github.com/vstakhov/asignify
[10]: https://jedisct1.github.io/minisign/

Side note: an additional pragmatic consideration which also initially motivated the development of signify was OpenBSD's desire to fit the tool in their installer ramdisk's kernel image, and the desire to have public keys of human-manageable length to trivialize distribution and verification by humans. For example, once while at a conference I easily copied down the entire key on a piece of paper from a trusted developer's own trusted piece of paper.

And in the interest of full disclosure (because potential bias), I should also note that I used to be the maintainer of the OS X port of signify [11].

[11]: https://github.com/jpouellet/signify-osx

Marek Marczykowski-Górecki

unread,

Jan 20, 2017, 10:41:04 AM1/20/17

to Jean-Philippe Ouellet, qubes-devel, Joanna Rutkowska, Andrew David Wong

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

On Fri, Jan 20, 2017 at 03:12:54AM -0500, Jean-Philippe Ouellet wrote:
> I would like to bring this list's attention to the availability of
> what I believe to be a good non-OpenPGP solution to the problem of
> cryptographically verifying code.
>
> The OpenBSD community has had very similar discussions internally
> several years ago, and they resulted in the implementation of a
> minimal non-OpenPGP signature creation & verification tool called

> signify, using the NaCl primitives [1]. It has been successfully used
> for package, release, and advisory signing for several years now, and
> has been audited both within and without the OpenBSD community.

While the tool looks interesting, it solve somehow different problem
than we have. The tool is only about signing data, with asymmetric
crypto (so a key pair is needed). But in our case, we have problem with
both integrity protection and encryption (in the old backup format, key
derivation is weak). And also, IMO asymmetric is not the way to go here.
It's about backup, which, by definition, should contain data necessary
to restore your system, even in case of total system failure (or theft).
If restoring the data would need additional key, you'll need to store
this key somewhere, securely - so probably encrypted somewhere. And here
we're back at the initial problem.

- --

Best Regards,
Marek Marczykowski-Górecki
Invisible Things Lab
A: Because it messes up the order in which people normally read text.
Q: Why is top-posting such a bad thing?

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2

iQEcBAEBCAAGBQJYgek5AAoJENuP0xzK19csuTcH/RpD2xU7o2Gld71cGdRtbyGH
lYdaC4XyuVIFcHHZQ2vh+hnWoishit2tKmi0YOi9kDHLqa7eZnDTtOYwmjIf8HCc
ZAKLcL2ar7hdY/Jj3u3AyEd9tDd0MYZWTkMBcO7f6stugS3ROyAISX9mSWV/YhTT
wH9iFuseumxATf7I18wReRpjIRdjmlXZqn5GT6ItmFsIVMBArP8uIjWE5UJiEax7
T1n4Z2ANpkn15SknVLaRWQgtFGN/g6neDqP9BL3kmsQiYrDiosCMyQ5cWsV9dF5C
TJ+2xYkc+NX+bxC6P5wyKBmqHol7U5KlasDiiptUX83s/zfFs4p1+aJ1Kt4Rcto=
=z1Qw
-----END PGP SIGNATURE-----

Jean-Philippe Ouellet

unread,

Jan 20, 2017, 9:45:31 PM1/20/17

to Marek Marczykowski-Górecki, qubes-devel, Joanna Rutkowska, Andrew David Wong

On Fri, Jan 20, 2017 at 5:40 AM, Marek Marczykowski-Górecki
<marm...@invisiblethingslab.com> wrote:
> -----BEGIN PGP SIGNED MESSAGE-----
> Hash: SHA256
>
> On Fri, Jan 20, 2017 at 03:12:54AM -0500, Jean-Philippe Ouellet wrote:
>> I would like to bring this list's attention to the availability of
>> what I believe to be a good non-OpenPGP solution to the problem of
>> cryptographically verifying code.
>>
>> The OpenBSD community has had very similar discussions internally
>> several years ago, and they resulted in the implementation of a
>> minimal non-OpenPGP signature creation & verification tool called
>> signify, using the NaCl primitives [1]. It has been successfully used
>> for package, release, and advisory signing for several years now, and

>> has been audited both within and without the OpenBSD community.
>
> While the tool looks interesting, it solve somehow different problem
> than we have. The tool is only about signing data, with asymmetric
> crypto (so a key pair is needed). But in our case, we have problem with
> both integrity protection and encryption

Indeed. I propose it mainly in the context of dom0 package
verification, where we have the inconsistency of trusting pgp there
while explicitly avoiding doing so elsewhere in Qubes, as discussed
previously in this thread. Sorry for the ambiguity.

Marek Marczykowski-Górecki

unread,

Jan 20, 2017, 11:15:49 PM1/20/17

to Jean-Philippe Ouellet, qubes-devel, Joanna Rutkowska, Andrew David Wong

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

Ah I see. Then indeed that makes some sense (disclaimer: haven't looked
at the tool yet - only read your description). But there is one
precondition to even think about something like this: we'd need to build
all dom0 packages ourself. Currently we use Fedora packages directly.
And those packages are signed with gpg.

- --
Best Regards,
Marek Marczykowski-Górecki
Invisible Things Lab
A: Because it messes up the order in which people normally read text.
Q: Why is top-posting such a bad thing?
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2

iQEcBAEBCAAGBQJYgpoeAAoJENuP0xzK19csMwIH/jRbcPTPwnXZoY81oPKUv8/Y
Y/jizAan07yBW67ZK9JItGNndSrveiSeC6e6sx/ndlcmfzb3mL/5VcABBwggecBI
mkKIHzDh8S2Oono11YqdZmLJok4pOv2j+M8tiaf/BRa8teuUDlaOHqmjnHDWXatH
RZe7nbzFrCgggDKAYySdYPgMIM2Ec0WC0kiukGSsVCoie3b6iimX9ss3akA85FiI
EbcLVUs00u3Ao92B7+CfJBuhaH4tvIdm87eI+/1AmO0l4WwwIEP4M+daeUsUV6+T
0W0oChq2hRYKIeI+vWoJ4EpxTsiCYqQ6V3gk4QsBkuSLLogBoQ5+lmD1un5GWSI=
=8Ctw
-----END PGP SIGNATURE-----

Jean-Philippe Ouellet

unread,

Jan 21, 2017, 1:20:36 AM1/21/17

to Marek Marczykowski-Górecki, qubes-devel

On Fri, Jan 20, 2017 at 6:15 PM, Marek Marczykowski-Górecki
<marm...@invisiblethingslab.com> wrote:

> But there is one precondition to even think about something like this: we'd need to build
> _all_ dom0 packages ourself. Currently we use Fedora packages directly.
> And those packages are signed with gpg.

Indeed. And there's still the fact that the sources are likely
ultimately verified with pgp...

OpenBSD is satisfied with a trust-on-first-use model (source archive
hash is first measured and recorded by the port maintainer, and later
verified to match when built officially on trusted infrastructure).
This provides at least some checking across time and networks.

The cost of needing to build everything yourself was a non-issue for
the *BSDs because they have significantly greater control over their
respective ecosystems than in GNU/Linux land, and have already been
providing binary packages of everything themselves for well over a
decade.