Digitalization of Kernel Diversion from the Upstream

To minimize local code modifications

Hisao Munakata

Linux Foundation Consumer Electronics working group

April 4th 2016

Who am I ?

- From an embedded SoC provider company Renesas
- Linux Foundation CE[1] working Gr. Steering committee and AG member
- LF/CEWG LTSI[2] project initiator member
- An Advisory Board and major contributor of AGL[3]
- Leads dedicated upstream development team at Renesas
- And, supports customers who develop automotive IVI products

[1] CE = Consumer Electronics
[2] LTSI =Long Term Support Initiative
[3] AGL =Automotive Grade Linux

Renesas contributes for kernel upstream development

Most active 4.5 employers

By changesets

Intel	1734	14.4%
(Unknown)	975	8.1%
Red Hat	732	6.1%
Linaro	723	6.0%
(None)	628	5.2%
Samsung	513	4.3%
SUSE	382	3.2%
Atmel	380	3.2%
Renesas Electronics	360	3.0%
IBM	346	2.9%
AMD	283	2.4%
Mellanox	275	2.3%
(Consultant)	245	2.0%
Broadcom	208	1.7%
Oracle	179	1.5%
Google	160	1.3%
Texas Instruments	152	1.3%
Huawei Technologies	141	1.2%
NVidia	137	1.1%
ARM	127	1.1%

By lines changed

Red Hat	83657	12.1%
Intel	80160	11.6%
AMD	74673	10.8%
Texas Instruments	41808	6.1%
(Unknown)	27958	4.1%
IBM	25433	3.7%
Linaro	22198	3.2%
(None)	21929	3.2%
Mellanox	19558	2.8%
Samsung	19190	2.8%
Renesas Electronics	17964	2.6%
(Consultant)	15593	2.3%
NVidia	15038	2.2%
Freescale	13964	2.0%
Code Aurora Forum	13514	2.0%
Atmel	10845	1.6%
Realtek	10090	1.5%
Rockchip	9735	1.4%
Huawei Technologies	7992	1.2%
Broadcom	7930	1.2%

http://lwn.net/Articles/679289/

Did you care for purity of your Linux BSP

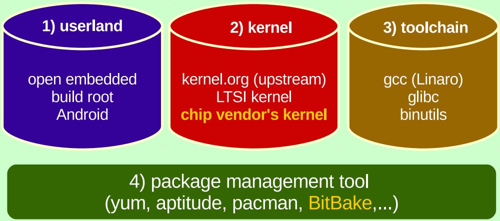
common embedded Linux issues caused by in-house kernel

Embedded Linux development issues-1 : no de-facto distribution

Various distribution exist for multiple target

- Desktop : Ubuntu, Fedora, Debian
- Smartphone : Android AOSP
- Game : Steam OS
- Server : Red Hat, SUSE, Oracle
- Cloud : Chrome OS
- R&D : Arch Linux, Gentoo
- General embedded : ?

Embedded Linux application program development environment



As there is no turn-key style binary distribution like PC, embedded Linux developer needs to combine kernel and other components together to create the own development environment.

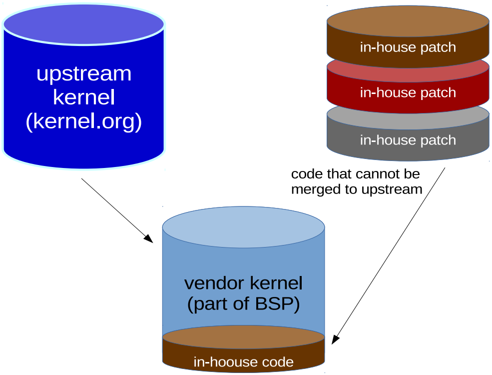
Contents of Embedded Linux distribution

Many embedded Linux developers still rely on SoC vendor’s kernel

Embedded Linux development issues-2 : quality of vendor's kernel

Why kernel may contain in-house code?

- in-house code = not from the upstream
- Already merged in later version kernel
- Dirty quick workaround
- Rejected by the community
  - break existing upstream code
  - contaminate with upstream design
  - designed for specific environment
  - poor C coding



Vendor Linux BSP likely contains dirty code

Vendor's BSP kernel may contain in-house code that troubles you

Embedded Linux development issues-3 : security patch adoption

Security (=software virus protection) is no more Windows's PC only risk

- Common Vulnerabilities and Exposures (CVE) information is available at `https://cve.mitre.org/`
- Community provides (some of) security-fix as a LongTerm-Stable (LTS)
- LTS security-fix patch is designed for native upstream kernel code
- Security-patch delivery becomes mandatory service for the end-user

- Security rating = frequency of security-fix patch release
- LTS security-fix patch may conflict with in-house kernel code

In-house kernel modification will result severe security risk

Embedded Linux development issues-4 : kernel version migration

New product surely requires new kernel

- Modern application requires newly supported advanced kernel API
  i.e. CMA, DMABUF, KDBUS,...
- You need to manipulate state-of-art device to make your new product
- New peripheral device interface support may be requested
  i.e. USB3.0, Bluetooth low-energy, EthernetAVB...
- New file system may be demanded to support a large volume
- Advanced security framework becomes mandatory criteria

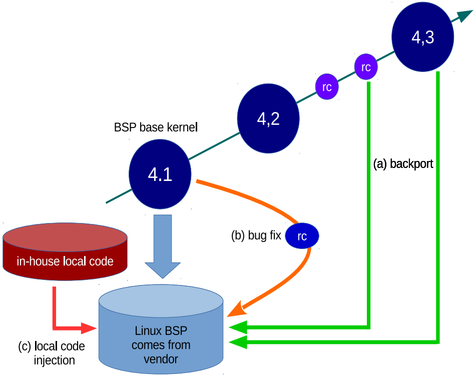
Local modification (even optimization) breaks kernel upgradability

Sanity assessment for the vendor kernel

We need to assess in-house patch risk level (clean, safe and dirty)

in-house code category

- a) Early adoption (clean)
  - Backport from newer upstream code
  - Early adoption from -rc or -next
- b) Minor fix (relatively safe)
  - small bug-fix against mainlined code
  - self-containing code adoption
- c) Rewrite/break existing code (dirty)
  - replace an existing upstream code



3 different code flows to create vendor BSP

The severity of each in-house patch depends on its characteristics

Standard BSP BOM does not contain in-house patch risk indicator

Typical Linux BSP BOM does not tell its sanity

- Kernel version is introduced, however...
- No information provided about
  - Referenced kernel tree information
  - Delta against the upstream kernel code
  - Description of vendor kernel file structure
  - Description of in-house kernel patch
  - Security patch delivery scheme
- Very hard to determine the sanity of vendor BSP kernel from a current standard BSP BOM

Overall BSP sanity score		95

Summary

base kernel version		4.4.6
total file		aa,aaa (100 %)
upstream code		bb,bbb (bb %)
in-house	bug-fix	xxx (XX %)
	new feature	yy (yy %)
	replaced	zzz (zz %)

Detail

file name	origin	delta	status
111.c	upstream	0	-
222.c	in-house	small	bug-fix
333.c	in-house	large	not mainlined
444.c	in-house	large	rewrite
555.c	in-house	middle	add feature

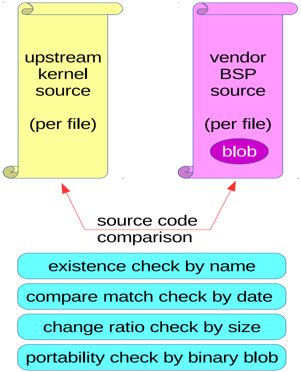
Image of “BSP certification of contents document”

We want to define and create “BSP certification of contents document”

How can we assess the vendor BSP kernel sanity?

upstream kernel vs. vendor kernel per file comparison

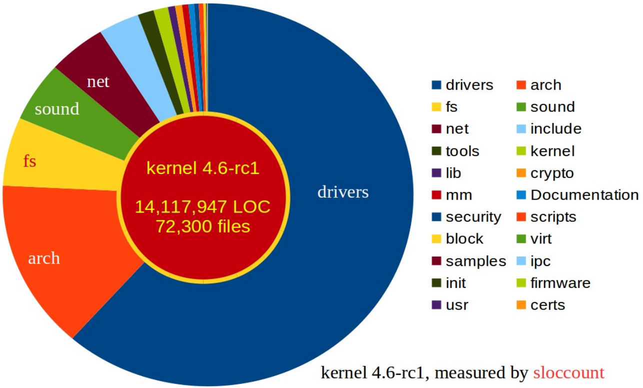
- File name
  - Detect locally added or deleted files
  - Scan later upstream kernel to determine a backport
- Time stamp / file size
  - Can find modified which file was edited
  - diff command (or git diff) helps change scale detection
- Binary blobs
  - Use of binary blob cause future serous migration trouble



We can determine the vendor kernel risk from the code, however...

Linux kernel source code comparison cannot be a human job

category	SLOC	%
drivers	8,681,715	61.5%
arch	2,014,724	14.3%
fs	817,753	5.8%
sound	713,625	5.1%
net	650,675	4.6%
include	455,320	3.2%
tools	180,123	1.3%
kernel	160,402	1.1%
lib	80,811	0.6%
crypto	76,430	0.5%
mm	71,728	0.5%
Documentation	61,129	0.4%
security	51,300	0.4%
scripts	50,680	0.4%
block	24,960	0.2%
virt	8,128	0.1%
samples	6,973	0.0%
ipc	6,221	0.0%
init	2,691	0.0%
firmware	1,877	0.0%
usr	558	0.0%
certs	124	0.0%
total	14,117,947	100%



kernel 4.6-rc1, measured by sloccount

"Automated code analysis tool" is mandatory to assess BSP vendor kernel as Linux kernel contains huge scale C source code.

Computer aided BSP kernel sanity check

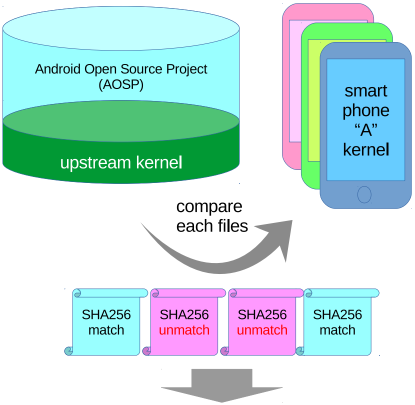
upstream code match detection

Original yaminabe method (SHA256 hash based file comparison)

Original yaminabe file comparison procedure

- use SHA256 for hash value calculation
- upstream kernel file number count – (A)
- calculate hash of original kernel files – (B)
- calculate hash of BSP kernel files – (C)
- compare (B) and (C) to determine locally modified file from the upstream kernel
- count modified files number – (D)
- (D)/(A) gives BSP sanity index value

yaminabe only detects match or unmatch



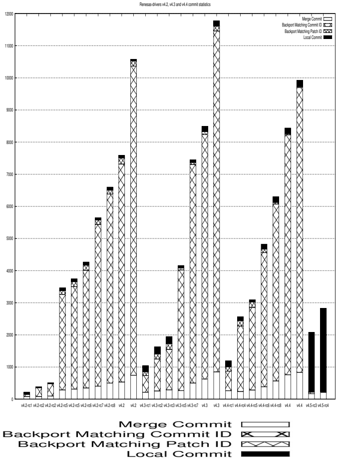
yaminabe project detected that almost all phone producers modified (rewrote or replaced) camera related kernel code to make a real product.

git id trace method (git patch-id and commit-id comparison)

Scan and compare patch-id and commit-id by the script

- Premise: vendor kernel managed by patch and git
- Scan vendor kernel patch-id to create search list
- Write a custom script to scan upstream git commit-id
- Check if patch-id exist in upstream kernel git
- Count in-house orphan patch and upstream patch
- Get an accurate in-house code ratio and trends
- Can trace backport patch from later upstream

Need to write a dedicated script for each kernel



upstream "code match" method summary

We can determine how many in-house patches are applied in the vendor kernel

- IMHO, 100% upstream code BSP is not realistic for embedded device
- Thus, we need to measure the risk of each vendor BSP kernel code.

- Pros. of code match scan
  - relatively fast and easy
  - good for encourage people to send more code to the upstream
- Cons. of code match scan
  - cannot measure the magnitude of each local-code risk
  - cannot distinguish which vendor BSP is clean and sanity

We really need to deep dive into the risk assessment of unmatched file

TLSH based yaminabe2(=yb2) method

yaminabe2 (=yb2) : Vendor kernel risk assessment challenge

yaminabe2 (=yb2) project motivation and expected outcome

- Collaborative work with Mr.Armijn Hemel (following the original yaminabe)

- Code scanner tool to compare upstream and production kernel code
- Combine TLSH (A Locality Sensitive Hash) method to measure the risk
- yb2 aims to grab a reasonably reliable score without deep code analysis

- Aiming open source so that anyone can measure the vendor kernel risk
- Hope this tool encourage everyone to minimize risk caused by local code

yb2 aims digitizing the vendor kernel risk using TLSH technology

yaminabe2 utilizes TLSH (A Locality Sensitive Hash) method

regular hash algorithm (for yb,yb2)

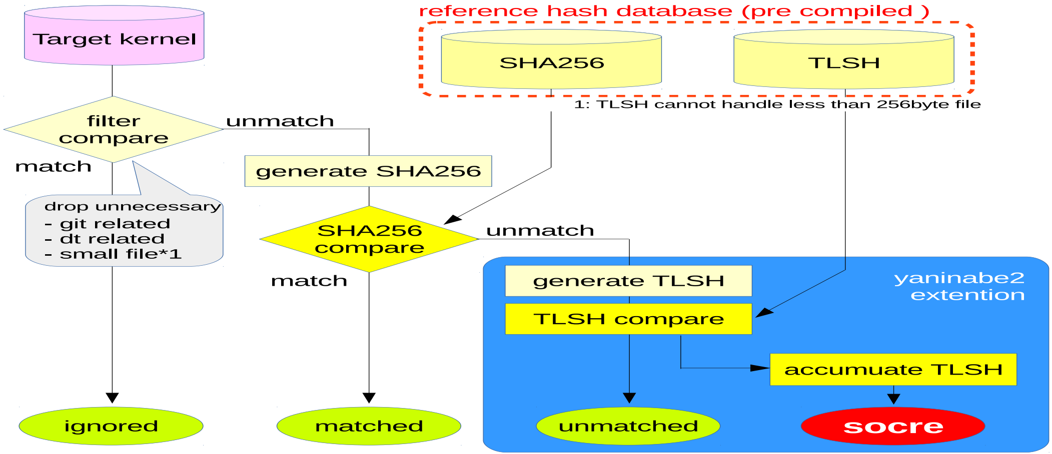
- sha1,md5,sha256...
- Small difference (even 1 byte) generate completely different value
- Designed for the file identification
- linux standard feature
- light weight and fast
- for file falsification check

A Locality Sensitive Hash (for yb2)

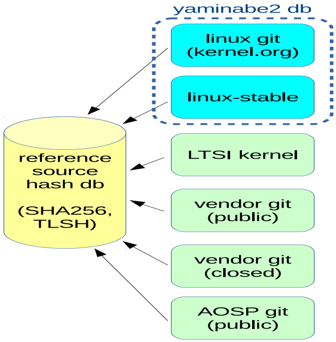
- TLSH (Trendmicro LSH, opensource)
- Similar file generate closer value
- Designed for file locality detection
- Need custom installation to use
- Relatively slow, more computing
- For file diff distance check
- Can find closest files pair

TLSH can show the numeric similarity indicator of unmatched files

yaminabe2 file comparison process flow (SHA256, TLSH combined)



Use of the reference code database (code origin is configurable)



You can input whatever source as you want

You can add whatever git tree you want to compare

- linux upstream git
- linux-stable git
- LTSI kernel git
- vendor kernel public git
- closed vendor source git (if you have an access)
- OSS project git (AOSP, Tizen,...)
- others, if any

yb2 compared linux(upstream) and linux-stable tree as a reference

yaminabe2 programs and sample reference data

yaminabe2 contents

- python script and config
  - gittlsh.py : script to explode Git repositories and store metadata like SHA256 and TLSH checksums out of band
  - gittreecompare.py : script to compare two tags in Git repositories and compute a TLSH score
  - sourceverifier.py : script for both the Yaminabe and Yaminabe2 projects
  - sourceverify.config : configuration file used for the Python scripts
- pre-compiled database (xz archived size / extracted size)
  - db contains upstream (Linus's tree) and linux-stable (Greg's tree)
  - kernelgit.sqlite3 (472M / 2G) : TLSH data
  - kerneldb.sqlite3 (863M / 11G) : SHA256 data + package data

download from `http://http://elinux.org/Yaminabe2` (data ready, contents under construction)

It’s time to play yaminabe2 on your machine

preparation-1 : install TLSH to your computer (1/2)

1. grab tlsh from github: `https://github.com/trendmicro/tlsh`
   we used version b53fef82c579906d6a6234bccfc3536c5abd28f0
2. unpack the ZIP file or simply cd into the Git checkout
3. Change the following in CMakeLists.txt (option)
   1. set(TLSH_BUCKETS_128 1) to set(TLSH_BUCKETS_256 1)
   2. set(TLSH_CHECKSUM_1B 1) to set(TLSH_CHECKSUM_3B 1)
   These changes make the scores reported more fine grained.
4. $ sh make.sh
   Note: the unit tests will fail if the CMakeLists.txt file is changed.
   This is expected, as they don't expect the settings to be changed.

preparation-1 : install TLSH to your computer (2/2)

5. cd py_ext;
6. python setup.py build
7. su -c 'python setup.py install'
8. check if the module is installed, type "import tlsh" into python prompt

```
tlsh_install_test ×
munakata@muna-E450:~/yb2$ python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import tlsh
>>> |
```

9. If there is no error message the module is successfully installed.

preparation-2 : Edit reference database configuration (1/5)

- Initially, I strongly recommend to start play with pre-compiled yb2 database that we prepared before start creating your database.
- If you decided to use pre-compiled database, still you need to read following config sections to reflect your database file locations.
- As initial whole kernel source TLSH hash generation cause huge amount of CPU workloads[4], I suggest following
  1. Use high performance machine (multi-thread helps hash calculation)
  2. Use ram-disk (4G min, 8G ideal) to store reference source
  3. Place git command on ram-disk, too

[4] File comparison does not require whole TLSH hash generation

preparation-2 : Edit reference database configuration (2/5)

[sourceverify] section of "sourceverify.config"

- database: SHA256 + package info. database location
- tlshdatabase: TLSH database location
- trusted: list trusted project group here
- scanlicense: license scan option, not used, set to "no"

*sourceverify.config ×

```
[sourceverify]
database = /home/munakata/yb2b/master.sqlite3
tlshdatabase = /media/ramdisk/kernelgit.sqlite3
trusted = linux|kernel
scanlicense = no
verbose = yes
```

preparation-2 : Edit reference database configuration (3/5)

[global] section of "sourceverify.config"

- gitdatabase: What differs from upper database location setting?
- processors: CPU thread allocation, set (amount of CPU threads) - 1
- gitpath: GIT executable file location, specify this if you locate it in ram-disk
- optimizedb: database size optimization
- statebackupdir: location of state cache file (optional)

*sourceverify.config

```
### DEFINITIONS FOR THE DATABASE CREATION SCRIPT ###
[global]
gitdatabase = /home/armijn/yaminabe2/backports/kernelgit.sqlite3
processors = 7
gitpath = /ramdisk/git
optimizedb = yes
#statebackupdir =
```

preparation-2 : Edit reference database configuration (4/5)

Note : Following configurations are only required for initial reference db creation

[(reference git)] section of "sourceverify.config"

- type: = project
- enabled: yes=use this reference, no=ignore this reference
- project: reference group name
- gitdirs: reference source location
- ramdisk; yes=use ram-disk
- revisionlogpath:
- restorestate: yes=use state cache
- statefile: state cache file location
- priority: reference tree priority, 1=highest weight
- giturl: git repo location
- trustedrepository: if this is untrusted tree, set this to "no"

preparation-2 : Edit reference database configuration (5/5)

Note : Following configurations are only required for initial reference db creation

*sourceverify.config ×

```
[linux]
type = project
enabled = yes
project = linux
gitdirs = /home/munakata/source/linux
ramdisk = yes
revisionlogpath = /tmp/gitrevlist
restorestate = yes
statefile = /tmp/seendict-linux
priority = 1
giturl = git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
trustedrepository = yes
```

preparation-3 : Execute reference database generation

database generation options

- Extract pre-build database
  - pre-build database is XZ compressed (=.xz), use "unxz" to extract
- Scan execution error
  - If you hit an error saying "ImportError: No module named magic"
  - To solve this you need to install "python-magic"

Start reference DB file generation w/gittlsh.py

```
$ python gittlsh.py -c ./sourceverify.config
```

* Initial db creation may take 4 to 12 hours, depends on the size and the machine

* Supplemental creation on top of the pre-compiled takes much shorter period

Yaminabe2 execution and trial result

Running yaminabe2 scan on Renesas R-Car BSP

Now let's run the very first yaminabe2 file scan

My file placement (reference database, scan target source,...)

- /home/munakata/yb2b/master.sqlite3 : SHA256 database on HDD
- /media/ramdisk/kernelgit.sqlite3 : TLSH database copied to ramdisk (8G)
- TLSH db contains kernel upstream (Linus's tree) and linux-stable (Greg's tree)
- gitdirs = /home/munakata/source/linux : latest upstream kernel source
- Adobe file placement settings are reflected to "sourceverify.config"
- /home/munakata/source/renesas-backport/ : scan target source

Start yaminabe2 code scan process w/sourceverifier.py

$ python sourceverifier.py -c sourceverify.config -s /home/munakata/source/renesas-backport/

How yaminabe2 (=sourceverifier.py) terminal output looks like

```
munakata@muna-E450:~/yb2b$ python sourceverifier.py -c sourceverify.config -s /home/munakata/source/renesas-backport/

SCANNING 36603 files
864 FILES NOT FOUND IN DATABASE
COMPUTING AND COMPARING TLSH OF FILES NOT FOUND IN DATABASE

CLOSEST REVISION FOR drivers/base/dma-contiguous.c IS 7ee793a62fa8c544f8b844e6e87b2d8e8836b219 FROM
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 17

CLOSEST REVISION FOR drivers/gpu/drm/drm_vm.c IS f435046d38af631920b299455db9e95dfc06d055 FROM
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 5

CLOSEST REVISION FOR arch/arm/mach-shmobile/headsmp.S IS cc61591e45c0457139ddd4cd7e57f75928acaaf2 FROM
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 210

CLOSEST REVISION FOR drivers/staging/lttng/wrapper/writeback.h IS 9e5c353510b26500bd6b8309823ac9ef2837b761 FROM
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 372h

CLOSEST REVISION FOR drivers/gpu/drm/rcar-du/rcar_du_kms.c IS 8bed5cc765ffdd61b59f8405d38b377f5a7f0920 FROM
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 63
```

How yaminabe2 (=sourceverifier.py) terminal output looks like

munakata@muna-E450:~/yb2b$ python sourceverifier.py -c sourceverify.config -s /home/munakata/source/renesas-backport/

SCANNING 36603 files
864 FILES NOT FOUND IN DATABASE
COMPUTING AND COMPARING TLSH OF FILES NOT FOUND IN DATABASE

CLOSEST REVISION FOR drivers/base/dma-contiguous.c IS 7ee793a62fa8c544f8b844e6e87b2d8e8836b219 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 17

CLOSEST REVISION FOR drivers/gpu/drm/drm_vm.c IS f435046d38af631920b299455db9e95dfc06d055 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 5

CLOSEST REVISION FOR arch/arm/mach-shmobile/headsmp.S IS cc61591e45c0457139ddd4cd7e57f75928acaaf2 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 210

CLOSEST REVISION FOR drivers/staging/lttng/wrapper/writeback.h IS 9e5c353510b26500bd6b8309823ac9ef2837b761 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 372h

CLOSEST REVISION FOR drivers/gpu/drm/rcar-du/rcar_du_kms.c IS 8bed5cc765ffdd61b59f8405d38b377f5a7f0920 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 63

How yaminabe2 (=sourceverifier.py) terminal output looks like

munakata@muna-E450:~/yb2b$ python sourceverifier.py -c sourceverify.config -s /home/munakata/source/renesas-backport/

SCANNING 36603 files
864 FILES NOT FOUND IN DATABASE
COMPUTING AND COMPARING TLSH OF FILES NOT FOUND IN DATABASE

846 / 36,603 = 2.3% --- in-house code rate

CLOSEST REVISION FOR drivers/base/dma-contiguous.c IS 7ee793a62fa8c544f8b844e6e87b2d8e8836b219 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 17

CLOSEST REVISION FOR drivers/gpu/drm/drm_vm.c IS f435046d38af631920b299455db9e95dfc06d055 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 5

CLOSEST REVISION FOR arch/arm/mach-shmobile/headsmp.S IS cc61591e45c0457139ddd4cd7e57f75928acaaf2 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 210

CLOSEST REVISION FOR drivers/staging/lttng/wrapper/writeback.h IS 9e5c353510b26500bd6b8309823ac9ef2837b761 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 372h

CLOSEST REVISION FOR drivers/gpu/drm/rcar-du/rcar_du_kms.c IS 8bed5cc765ffdd61b59f8405d38b377f5a7f0920 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 63

What TLSH hash delta tells you about two file's similarity?

Delta of TLSH hash represents FP rate of 2 files

- Identical pair filtered by the SHA256 hash match
- Then, create a unmatched list and calculate TLSH hash
- TLSH hash delta represents compared file's similarity, smaller delta means two files are closed
- FP rate = false positive ratio, =false alarm ratio
- 60 > means relatively closed, minor difference
- 61 to 150 means have some similarity, but modified
- > 150 means limited similarity, almost different

http://www.academia.edu/7833902/TLSH_-A_Locality_Sensitive_Hash

TLSH		
Score	FP rate	Detect rate
< 300	79.30%	98.80%
< 250	69.06%	98.80%
< 200	50.10%	98.80%
< 150	24.33%	98.10%
< 100	6.43%	94.50%
< 90	4.49%	92.30%
< 80	2.93%	89.00%
< 70	1.84%	83.60%
< 60	1.09%	76.00%
< 50	0.52%	65.30%
< 40	0.70%	49.60%
< 30	0.00181%	32.20%
< 20	0.00181%	17.30%
< 10	0.00181%	6.40%

How yaminabe2 (=sourceverifier.py) terminal output looks like

munakata@muna-E450:~/yb2b$ python sourceverifier.py -c sourceverify.config -s /home/munakata/source/renesas-backport/

SCANNING 36603 files
864 FILES NOT FOUND IN DATABASE
COMPUTING AND COMPARING TLSH OF FILES NOT FOUND IN DATABASE

846 / 36,603 = 2.3% --- in-house code rate

CLOSEST REVISION FOR drivers/base/dma-contiguous.c IS 7ee793a62fa8c544f8b844e6e87b2d8e8836b219 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 17

CLOSEST REVISION FOR drivers/gpu/drm/drm_vm.c IS f435046d38af631920b299455db9e95dfc06d055 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 5

clean

CLOSEST REVISION FOR arch/arm/mach-shmobile/headsmp.S IS cc61591e45c0457139ddd4cd7e57f75928acaaf2 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 210

dirty

CLOSEST REVISION FOR drivers/staging/lttng/wrapper/writeback.h IS 9e5c353510b26500bd6b8309823ac9ef2837b761 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 372

CLOSEST REVISION FOR drivers/gpu/drm/rcar-du/rcar_du_kms.c IS 8bed5cc765ffdd61b59f8405d38b377f5a7f0920 FROM git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git WITH DISTANCE 63

OK

yaminabe2 BSP brief sanity scoring output (current shape)

Originally we aimed to create "BSP certification of contents document"

Overall BSP sanity score 95

Summary

base kernel version		4.4.6
total file		aa,aaa (100 %)
upstream code		bb,bbb (bb %)
in-house	bug-fix	xxx (XX %)
	new feature	yy (yy %)
	replaced	zzz (zz %)

Detail

file name	origin	delta	status
111.c	upstream	0	-
222.c	in-house	small	bug-fix
333.c	in-house	large	not mainlined
444.c	in-house	large	rewrite
555.c	in-house	middle	add feature

<BSP certification of contents document>

```
=======================================
               SUMMARY
=======================================
FILES SCANNED: 40434
FILES FOUND IN UPSTREAM RELEASE: 39846
FILES NOT FOUND IN UPSTREAM RELEASE: 588
TOTAL DISTANCE: 5721
IDENTICAL FILES IN GIT: 660
NOT MATCHED IN GIT: 6
UNDETERMINED IN GIT: 0
0-60: 52
61-150: 7
over 150: 6
```

<yaminabe2 BSP scoring output>

Some R-Car Linux BSP sanity analysis

R-Car generation2 (kernel 3.10) yaminabe2 trial run

rcar-gen2 / v1.1.0	63,616
rcar-gen2 / v1.2.0	63,879
rcar-gen2 / v1.3.0	65,451
rcar-gen2 / v1.4.0	65,469
rcar-gen2 / v1.5.0	66,267
rcar-gen2 / v1.6.0	70,320
rcar-gen2 / v1.6.1	70,473
rcar-gen2 / v1.7.0	71,308
rcar-gen2 / v1.8.0	71,518
rcar-gen2 / v1.9.0	72,097
rcar-gen2 / v1.9.1	72,112
rcar-gen2 / v1.9.2	72,111
rcar-gen2 / v1.9.3	72,098
rcar-gen2 / v1.9.4	72,178
rcar-gen2 / v1.9.5	72,200
rcar-gen2 / v1.9.6	72,242

yaminabe2 scan result for R-Car BSP

R-Car gen2 (H2/M2/E2) BSP status

- Based on LTSI-3.10 kernel
- Upstream 3.10 does not support R-Car gen2 due to its release timing
- Due to that, the distance is relatively big
- After release, distance becomes bigger
- This is caused by local bug-fix code

R-Car gen2 BSP (3.10) average distance was 70,000

R-Car generation3 (kernel 4.3 to 4.5) yaminabe2 trial run

```
comparing tags rcar-3.0.0 and v4.3-rc1
FILES SCANNED: 40393
TOTAL DISTANCE: 154704
IDENTICAL FILES: 37161
0-60: 2684
61-150: 268
over 150: 95

...

comparing tags rcar-3.2.0 and v4.5
FILES SCANNED: 41392
TOTAL DISTANCE: 110693
IDENTICAL FILES: 38654
0-60: 2401
61-150: 139
over 150: 65
```

yaminabe2 scan result for R-Car BSP

R-Car gen3 (H3) BSP status

- Keep chasing latest upstream ver. now
- Plans to lands on LTSI-2017 (LTSI-2017 ver not fixed yet)
- Device support became available at v4.5
- Then, the distance dramatically dropped
- Keep continue to eliminate local-patch

gen3 BSP distance should be less than gen2

We doubt why current gen3 distance is bigger than gen2 now

R-Car generation3 yaminabe2 trial run2 (update)

```
python sourceverifier.py -c sourceverify.config -s /home/munakata/source/renesas-bsp/
```

```
=======================================
                SUMMARY
=======================================
FILES SCANNED: 41434
FILES FOUND IN UPSTREAM RELEASE: 40350
FILES NOT FOUND IN UPSTREAM RELEASE: 1084
TOTAL DISTANCE: 19323
IDENTICAL FILES IN GIT: 921
NOT MATCHED IN GIT: 27
UNDETERMINED IN GIT: 1
0-60: 94
61-150: 22
over 150: 19
```

yaminabe2 rescan result for R-Car BSP

R-Car gen3 (H3) BSP status (retry)

- Retried after ELC2016 presentation
- use updated database (inc. v4.5 kernel)
- update renesas-bsp git information
- re-run with revised script

Now we got much smaller number around 20k

outcome and lesson learned

yaminabe2 achievement: How in-house kernel risk digitalizad

description

- Utilizing TLSH mechanism, yaminabe2 start telling interesting indicator that reflects BSP kernel healthiness
- We need to verify the risk of local patch by the distance number (currently set to 60 and 150) given by yaminabe2.
- Also, we need to tune reference database setting to focus on the risk of local code (eliminating unrelated arch code, etc.)
- We could opensource the initial yaminabe2 program for the public review. We need feedback to improve the value of this trial.

conclusion

Conclusion

- Many embedded Linux developers rely on SoC vendor's BSP and its kernel may contain in-house code. And it might cause various security, migration issues. We need some computer aided vendor kernel assessment tool.
- We can compare file match between upstream kernel and vendor BSP kernel. However, it is not sufficient to assess how unmatched files diverted from the upstream (=dirty) from that information.
- We adopted TLSH (Locality Sensitive Hash) to measure the distance of in-house code in yaminabe2 project. And successfully it starts telling some score regarding vendor kernel sanity. use this tool to consult vendor kernel patch risk.
- Database generation script, file comparison script and trial reference database that contains upstream kernel code can be download for your trial.

Call for action and future work candidates

Call for action

- Run yaminabe2 file scan for your BSP kernel to consult the risk
- Configure your reference database to get more precise result
- Encourage your business partner to eliminate dirty in-house code

Future work (so far just an idea for yaminabe3)

- Do further verification of the accuracy of TLSH value
- Improve reporting (=post processor) feature so that anyone can
- Do further study for Renesas R-Car BSP verification

Resources

- yaminabe2 intro (scripts, pre-compiled reference database)
  - http://www.elinuxwiki.org/yaminabe2
- TLSH
  - https://github.com/trendmicro/tlsh
  - https://github.com/trendmicro/tlsh/blob/master/TLSH_Introduction.pdf
  - https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf
- Renesas R-Car BSP seed code
  - gen2 : https://git.kernel.org/cgit/linux/kernel/git/horms/renesas-backport.git/
  - gen3 : https://git.kernel.org/cgit/linux/kernel/git/horms/renesas-bsp.git/