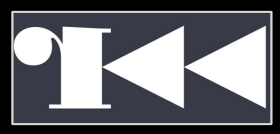


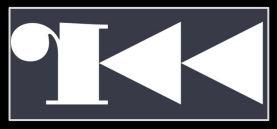
R2CON 2021

Debugging & Bugfixing r2

@unixfreaxjp

Cyber Emergency Center - LAC / LACERT

Research material of radare.org project

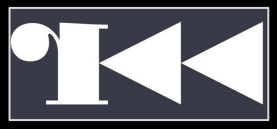


DISCLAIMER:

1. I wrote this training slide as a **blue-teamer** based on my r2 know-how & experience as radare2 users and tester, as a share-back knowledge to fellow blue teams in radare2 community in dealing bugs and how to handle them properly.

2. The talk is meant to be a non-operational with no-attribution and it is written to be as conceptual for education purpose; contains basic knowhow of radare2 tools in the unix like OS and other platforms.

3. The material is based on radare2.org base research we have, and there is no data nor information from other speaker's profession or from other groups is included in any of these slides.

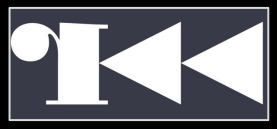


The contents

“Debugging & Bugfixing r2”

@unixfreaxjp

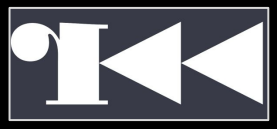
1. Introduction
2. Why testing is needed
   - Definition, Bugfix steps, target etc
   - Requirements
     - Human: QC human spec
     - Stuff: OS, Arch, exotic
     - Dedicated time
   - What to test
     - Packages, usage, approach
     - Dealing with plugins
3. Practical test
   - Knowing the limit
   - Testing in action
4. Bug hunting
   - Playbook to handle bugs
5. Report
   - Test/Bug reports leads to next features in r2 miletones
   - Examples



LAC
ともに、イキル

Chapter one Introduction

“In r2land we use our cyber knowledge for a good purpose”



Introduction

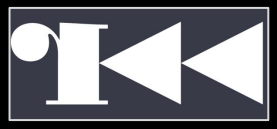
In this presentattion you are introduced to understand the development cycle of radare dev, how the testing and bugfix playing a big roles in it.

And it will be explained further how to handle situation when the bug occurred before or during analysis has been done with the radare2 binaries and its commonly used plugin.

The handling is the key, with the correct steps you all can help the radare2 developer team to fix the bug quickly by following the presented way.

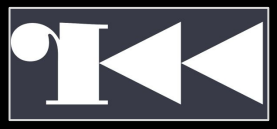
By helping in testing of radare2 products you are contributing much for the stability and maturity of the radare2 projects.

I am speaking about my experience on the subject and sharing this knowledge to you for the regeneration and continuation of the radare2.



About me (@unixfreaxjp)

1. Just another security guy:
   - We help cyber incident victims at Cyber Emergency Center of LACERT Tokyo, Japan, I work as RE & cyber threat intrusion analyst.
   - Security lecturer & activist from team LACERT in FIRST dot org
   - Write a lot of *“r2 related stuff”* in blog.malwaremustdie.org
2. The radare2 community:
   - Helping trufae at r2 works for stable version’s test / bug hunt / QC.
   - Old “radare (without 2)” user since 2007, switched to “radar (without2)” at FreeBSD port (2011) & joined radare2 dev at Github community from 2012-2013
   - Sharing “howto” use radare2 in social media since 2012
   - Co-builder of 1st radare2 full decompiler preinstall OS (TsurugiLinux) Co-maintainer of R2JP (radare2 Japan community).

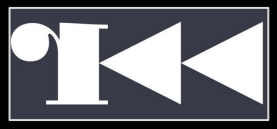


LAC
ともに、イキル

About my other radare2 talks & its sequels

I have executed a roadmap on sharing practical r2 know-how on binary analysis in a series of talks on sequel of events below:

Year	Event	Theme	Description
2017	AVTOKYO	Intro of radare2 in Japan	First JP workshop radare2 & R2JP est.
2018	R2CON	Unpacking a non-unpackable	ELF custom packed binary dissection r2
2019	HACKLU	Linux Fileless injection	Post exploitation today on Linux systems
2019	SECCON	Decompiling in NIX shells	Forensics & binary analysis w/shell tools
2020 (Spt)	R2CON	Okay, so you don't like shellcode too?	Shellcode (part1 / beginner) For radare2 users
2020 (Oct)	ROOTCON	A deeper diving on shellcode..	Shellcode (part2 / advanced) for blue team DFIR analysts
2021	SECCON	Shellcode workshop 11h	r2 & shellcode analysis (Japanese, online)
2021	Texas Cyber	Shellcode briefs 2h	r2 & shellcode analysis (English, online)
2022	TBD	Radare2 forensics workhop	TBD



The contents

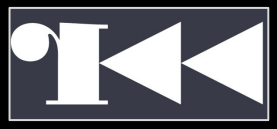
"Debugging & Bugfixing r2"

@unixfreaxjp

1. Introduction
2. Why testing is needed
   - Definition, Bugfix steps, target etc
   - Requirements
     - Human: QC human spec
     - Stuff: OS, Arch, exotic
     - Dedicated time
   - What to test
     - Packages, usage, approach
     - Dealing with plugins
3. Practical test
   - Knowing the limit
   - Testing in action
4. Bug hunting
   - Playbook to handle bugs
5. Report
   - Test/Bug reports leads to next features in r2 miletones
   - Examples

Chapter two Why testing is needed

"All good knowhow starts from the basics"



Why testing in r2 is needed & how?

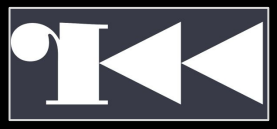
Testing is an important steps for stable released versions in radare2 dev programs. Radare2 development is defined by set of milestones, that each milestones is a new base of development explaining new base of features to be implemented in next release, to be added w/more dev'ed features.

For example in radare2 version 5.4.2, it has below meaning:

- Major version is 5, w/ new base milestone (see the github)
- Which is developed w/ additional features 4 times (hence the minor version = 4)
- And has been maintained for bugfixes 2 times since Minor ver release.

For the testing department every version means:

{ Major (5) . Minor(4) } . "2" <= this is what testing resulted



LAC
ともに、イキル

Requirement for testing

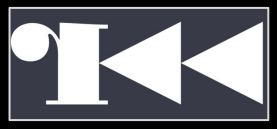
Testing requirements needed are:

- Target (supported OS, architecture of CPU)
- Who?
  - CI (Travis)
  - Human (anyone with skillset)
- Time (Human)
  - Generic platforms (Servers & Desktops) = overall 1 day (～24h)
    - x86/AMD(32/64) WinOSX/Linux, ARM/MIPS(32/64) Linux/BSD
  - Mobile generics
    - Android = 1 day (～24h)
    - iOS = ask pancake :)
  - Extended platform test (included embedded platforms)
    - Embedded CPU build test = 1 CPU per day (～24h)
  - Blind bugfix support = ～ +/- 1 week in average per bug

So… Who does the test?

Travis CI

```
Worker information                                                                         worker_info
                                                                                                          0.01s
Build system information                                                                   system_info

                                                                                                          1.19s
docker stop/waiting                                                                        docker_mtu
resolvconf stop/waiting                                                                    resolvconf
$ sudo service docker start                                                                services       3.02s
$ git clone --depth=50 --branch=master https://github.com/radareorg/radare2.git radareorg/radare2   git.checkout   3.19s

                                                                                                          0.01s
Setting environment variables from repository settings
$ export GITHUB_TOKEN=[secure]

Setting environment variables from .travis.yml
$ export COMPILER_NAME=gcc
$ export INSTALL_SYSTEM=meson
$ export ASAN=1
$ export ASAN_OPTIONS=detect_odr_violation=0
$ export LSAN_OPTIONS=detect_leaks=0
$ export CFLAGS="-DR2_ASSERT_STDOUT=1"

$ bash -c 'echo $BASH_VERSION'
4.3.11(1)-release

$ if [ "${FUZZIT}" == "1" ]; then                                                          before_install   22.97s
$ export PR_NAME=$(echo $TRAVIS_PULL_REQUEST_SLUG | cut -d'/' -f1)                                        0.01s
The command "export PR_NAME=$(echo $TRAVIS_PULL_REQUEST_SLUG | cut -d'/' -f1)" exited with 0.
$ if [ "$TRAVIS_OS_NAME" != "osx" ] && [ "$NODOCKER" != "1" ]; then                                      0.04s
  $SHELL ./travis-extract-var.sh | tee ${TRAVIS_BUILD_DIR}/docker-env.sh
fi

export TRAVIS_ARCH=amd64
export TRAVIS_FILTERED=redirect_io
export TRAVIS_STACK_JOB_BOARD_REGISTER=/.job-board-register.yml
export TRAVIS_STACK_LANGUAGES=__garnet__\ c\ c++\ clojure\ cplusplus\ cpp\ default\ go\ groovy\ java\ node_js\ php\ pure_java\
python\ ruby\ scala
export TRAVIS_TEST_RESULT=0
export TRAVIS_COMMIT=5dd63b77fbf9c2bbdff2a5ca677ba6567a3c64e1
export TRAVIS_APT_PROXY=http://build-cache.travisci.net
export TRAVIS_OS_NAME=linux
export TRAVIS_INTERNAL_RUBY_REGEX=\^ruby-\(2\\.\[0-4\]\\.\[0-9\]\|1\\.9\\.3\)
export TRAVIS_UID=2000
```



LAC
ともに、イキル

So… Who does the test?

Travis CI - Merit vs Demerit

Merit:

- Faster check
- Github integrated - automation testing in every/after merge of new code
- Useful for developer to instantly check/adjust their works w/trial-error

Demerit:

- Supporting only limited specific build environment configuration of a platform only
- Not supporting one OS with having multiple ways to setup the build environments
- Missing mostly build bugs
- Developer tends to not maintaining their CI distribution much

So… Who does the test?

Human

unixfreaxjp commented on May 13, 2016 • edited

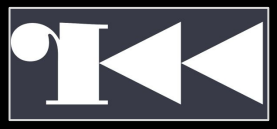
PoC:

Explanation: I found ${OS} and ${CPU} the values were weird in powerpc tests, then I use freebsd since everyone can see the it clearly how different the values were.

The proposed fix:
unixfreaxjp@ `1a4418f` #diff-e2d5a00791bce9a01f99bc6fd613a39d

Pull requested:
`1a4418f`

Thank you very much in advance!



LAC
ともに、イキル

So… Who does the test?

Human - Merit vs Demerit

Merit:

- More thorough check than CI
- Can be a team, which:
  - the bigger test platform resources the bigger scope of tests can be performed
  - can delegate test works & platforms maintenance
- Can cover testing for build, bugs-fixing and troubleshooting
- **Capable to do QC or QA for the stable versions**

Demerit:

- Needs dedication, resource, skillset and time
- Can not be done instantly and needs regularly planned

The “human” test process of radare2 major versions 5

The roadmap to a stability & portability in a r2 version

5.0.0 - version released <= base feature released
5.1.0 - feature addition released <= feature’s development

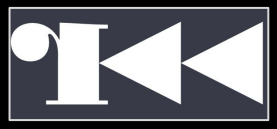
5.2.0 - feature addition release <= testing performed
5.2.1 - stable, bug fix released <= full test result, stable

5.3.0 - feature addition released <= testing performed
5.3.1 - stable, bug fix released <= full test result, stable

5.4.0 - feature addition released <= testing performed
5.4.1 - 1st bug fix goes to git <= testing performed
5.4.2 - stable, bug fix released <= full test result, lastest-stable

Legend:

Alert 1
Alert 2
Alert 3
All cleared



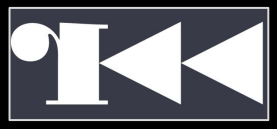
OS used in tests

Supported OS for generic tests are:

- Mac OS X
- Linux
- FreeBSD / OpenBSD / NetBSD
- Windows
- Android
- (for iOS ask pancake)

Which supported OS versions are:

- Long term supported OS (we don’t test unsupported OS)
- Specific device / OS distro that is having specific OS version can be supported seperately with dedicated maintainer i.e. Gameboy, ReaktOS, Tsurugi Linux, Linux Hobbyist with “Exotic CPU”



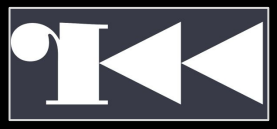
CPU Architecture in tests

Architecture used for regular tests are:

- Intel 32
- AMD 64
- ARM 32 64
- MIPS 32 64
- PPC 32. 64
- Windows 32 64

Extention tests:

- ARM M1 - Darwin OSX (64bits)
- ARM Cortex 57a Linux Debian (64bits)
- ARMv5te Kirkwood FreeBSD (64bits)
- etc



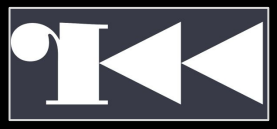
LAC
ともに、イキル

Approach in radare2 regular tests

Production test
- Aiming new feature checks

Bug fixing test
- Pinning a kown bug
  - On one platform
  - On overall platforms

Stability & portability test
- Compatibility on OS, Architecture & Devices
- Backup and restore



LAC
ともに、イキル

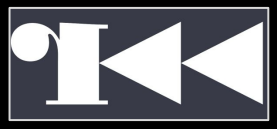
Dealing with r2 plugins

Understanding r2pm

- How r2pm install the plugin
- Where all of the stuff after a plugin is installed
- Where is the source of the plugin and which commit is now installed
- How to backup your plugins
- How that plugin is integrated to r2.. Read the r2book:

All packages are located in radare2-pm repository, and have very simple text format.

```
R2PM_BEGIN

R2PM_GIT "https://github.com/user/mycpu"
R2PM_DESC "[r2-arch] MYCPU disassembler and analyzer plugins"

R2PM_INSTALL() {
    ${MAKE} clean
    ${MAKE} all || exit 1
    ${MAKE} install R2PM_PLUGDIR="${R2PM_PLUGDIR}"
}

R2PM_UNINSTALL() {
    rm -f "${R2PM_PLUGDIR}/asm_mycpu."*
    rm -f "${R2PM_PLUGDIR}/anal_mycpu."*
}

R2PM_END
```



The contents

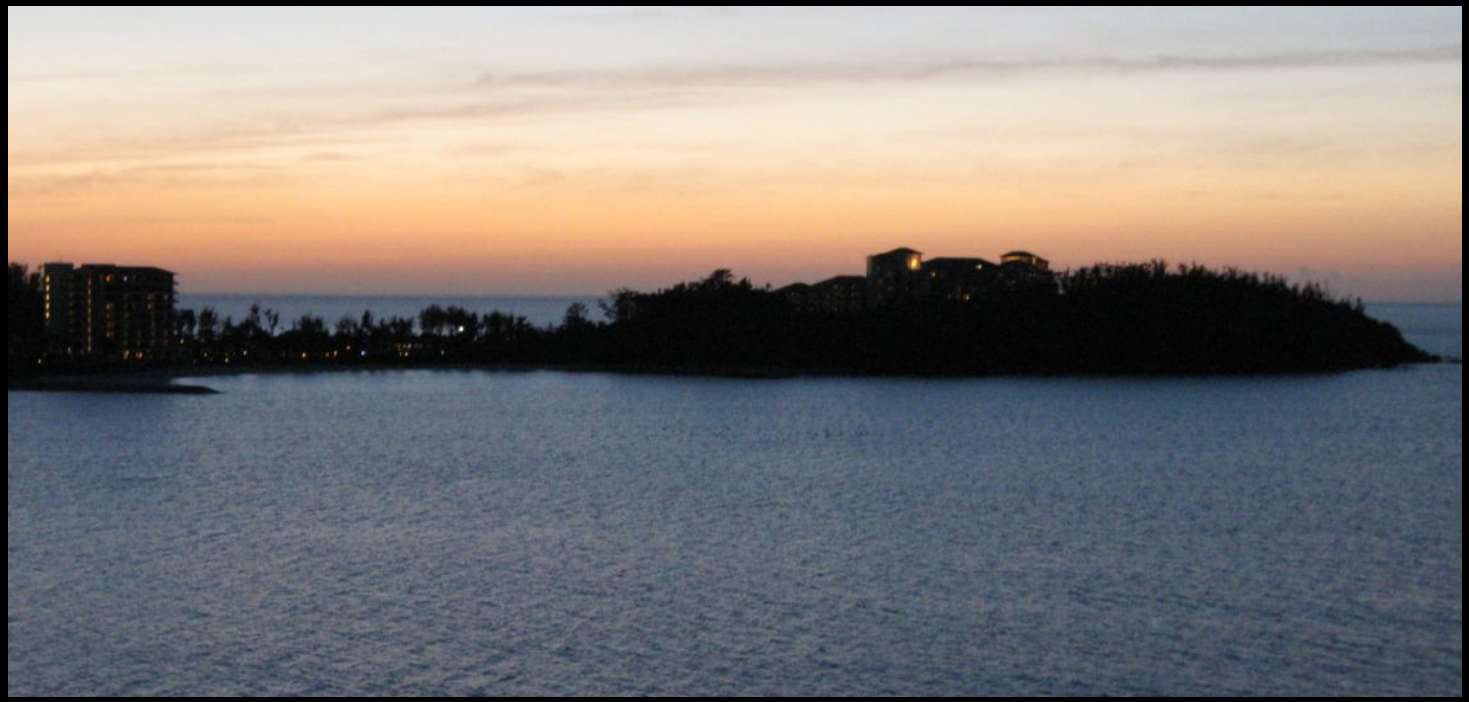
“Debugging & Bugfixing r2”

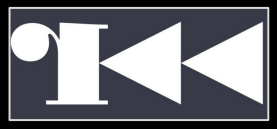
@unixfreaxjp

1. Introduction
2. Why testing is needed
   - Definition, Bugfix steps, target etc
   - Requirements
     - Human: QC human spec
     - Stuff: OS, Arch, exotic
     - Dedicated time
   - What to test
     - Packages, usage, approach
     - Dealing with plugins
3. Practical test
   - Knowing the limit
   - Testing in action
4. Bug hunting
   - Playbook to handle bugs
5. Report
   - Test/Bug reports leads to next features in r2 miletones
   - Examples

Chapter three Practical test

"A journey of thousand miles starts from a step"





Knowing the limitation for radare2 fuctionalities

Several limitation in analysis can be misunderstood as bug if, i.e. :

Debugging

- Several systems doesn’t support software breakpoints
  And some cpu has limit of hardware breakpoints, i.e. in x86 < 5
- You can ruin a process opcode if you don’t e io.cache =1

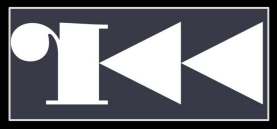
ESIL

- You can’t force ESIL to do backtrace without setting the aeip

Plugins

- R2dec: You can’t run pdd on non-intel platforms
- R2ghidra: You have to install r2ghidra integration to r2 via radare-pm

etc..



Demonstration

Build test

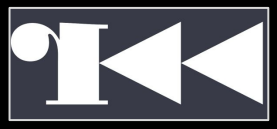
- System wide installation
- User space installation

Basic tests

- Interface (pipes, etc commands, visual, HUD)
- Static analysis (parser on several arch binaries)
- Debugging (invoked from -d and ood)
- ESIL

Test build reporting

- (several examples)



The contents

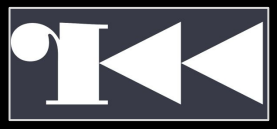
"Debugging & Bugfixing r2"

@unixfreaxjp

1. Introduction
2. Why testing is needed
   - Definition, Bugfix steps, target etc
   - Requirements
     - Human: QC human spec
     - Stuff: OS, Arch, exotic
     - Dedicated time
   - What to test
     - Packages, usage, approach
     - Dealing with plugins
3. Practical test
   - Knowing the limit
   - Testing in action
4. Bug hunting
   - Playbook to handle bugs
5. Report
   - Test/Bug reports leads to next features in r2 miletones
   - Examples

Chapter four Bug hunting

“The best prepared team wins - learn from the best”



Playbook to handle a bug in radare2 project

What packages to test:

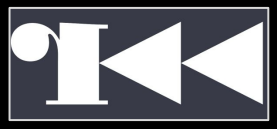
- radare2 core bins
- commonly used plugins

What usage functionality to test:

- binary analysis
  - via radare2 binary (main interface)
  - via components (rabin2, rax2, ragg2, rahash2 etc)
- forensics (commands , load test)

How to QC Testing approach (r2core)

- build fail
- basic functionality interface (parser, cmd, visual, HUD)
- basic functios(ESIL, debugging, carving)
- standard regression in usage (limited to know bugs)



Playbook to handle a bug in radare2 project

(cont'd)

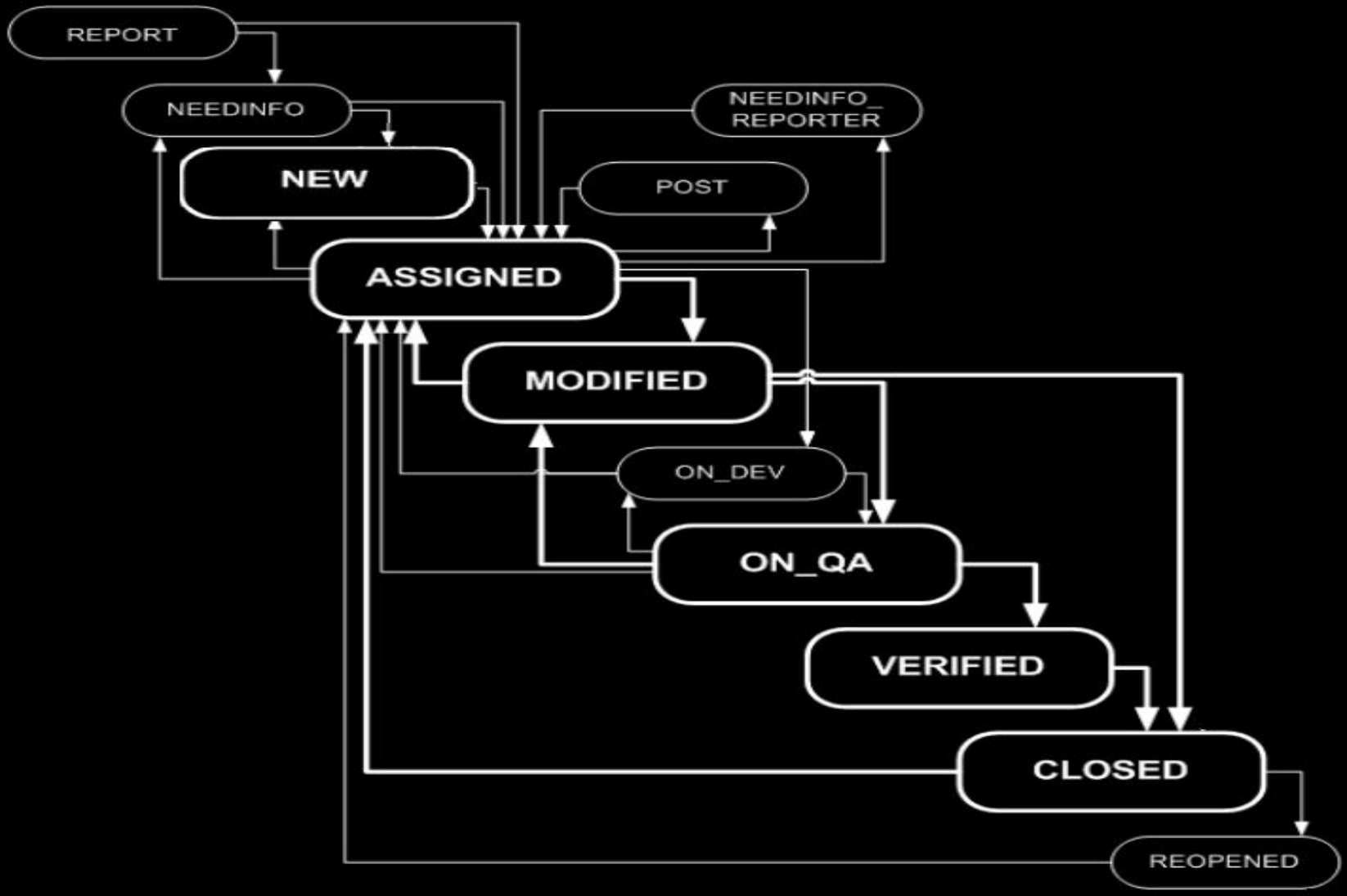
How to test plugins

- knowing every plugin matched to specific r2 versions
- test plugins that are (only) maintained by radare2 main stream
- start with commonly used plugins (r2ghidra, r2frida, etc)

Knowing the limitations so far:

- we can't test them all : plugins , user interface
- few distribution support plugins and interface
- you can trace the build commit openly but not for plugin..

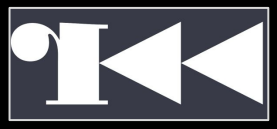
Current bugfix scheme in radare2 Github



Bug-fix process model based on RHEL 6

Knowhow you need to troubleshoot bugs

Always refer to what pancake is working with, it is described in this slides.



LAC
ともに、イキル

Knowhow you need to troubleshoot bugs

Understanding : Directory structure (i.e.: /libr /binr /shlr ..)

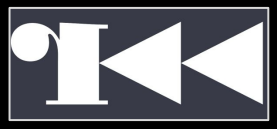
Directory Structure

One common concern for beginners is the way the code is structured, it aims to be simple:

- Libr : directory containing all the libraries of r2 (one per directory)
- Binr: main executables that will be installed later
- Shlr: shared / external / 3rd party libraries
  - shlr/capstone

Each directory under libr can have none or some of the following:

- D: data associated with the module
- P: plugins to be loaded when initializing the library
- T: test files and scripts



Knowhow you need to troubleshoot bugs

Understanding : Code related structure

Code Structure

- Public APIs are prefixed with R_API, you can grep for that
- Public function names start with r_modulename, for example: r_util or r_core
- Lifecycle involves _new() and _free() functions
  - When plugins are in need, the _use() function should be called
- RUtil is the standard library for radare2, that way we abstract from the OS
- RMain provides the main functions of all the r2 tools, busybox style!
  - So you can call rabin2 from r2
- Tight dependency between modules
- SDB - external project sync with r2 for source compat reasons
- Capstone and vector35

Knowhow you need to troubleshoot bugs

Understanding : Base building process

Building

The recommended way to install r2 from git:

- Git clone https://github.com/radareorg/radare2 && cd radare2
- sys/install.sh or sys/user.sh

The standard to do test build

Support for both build systems

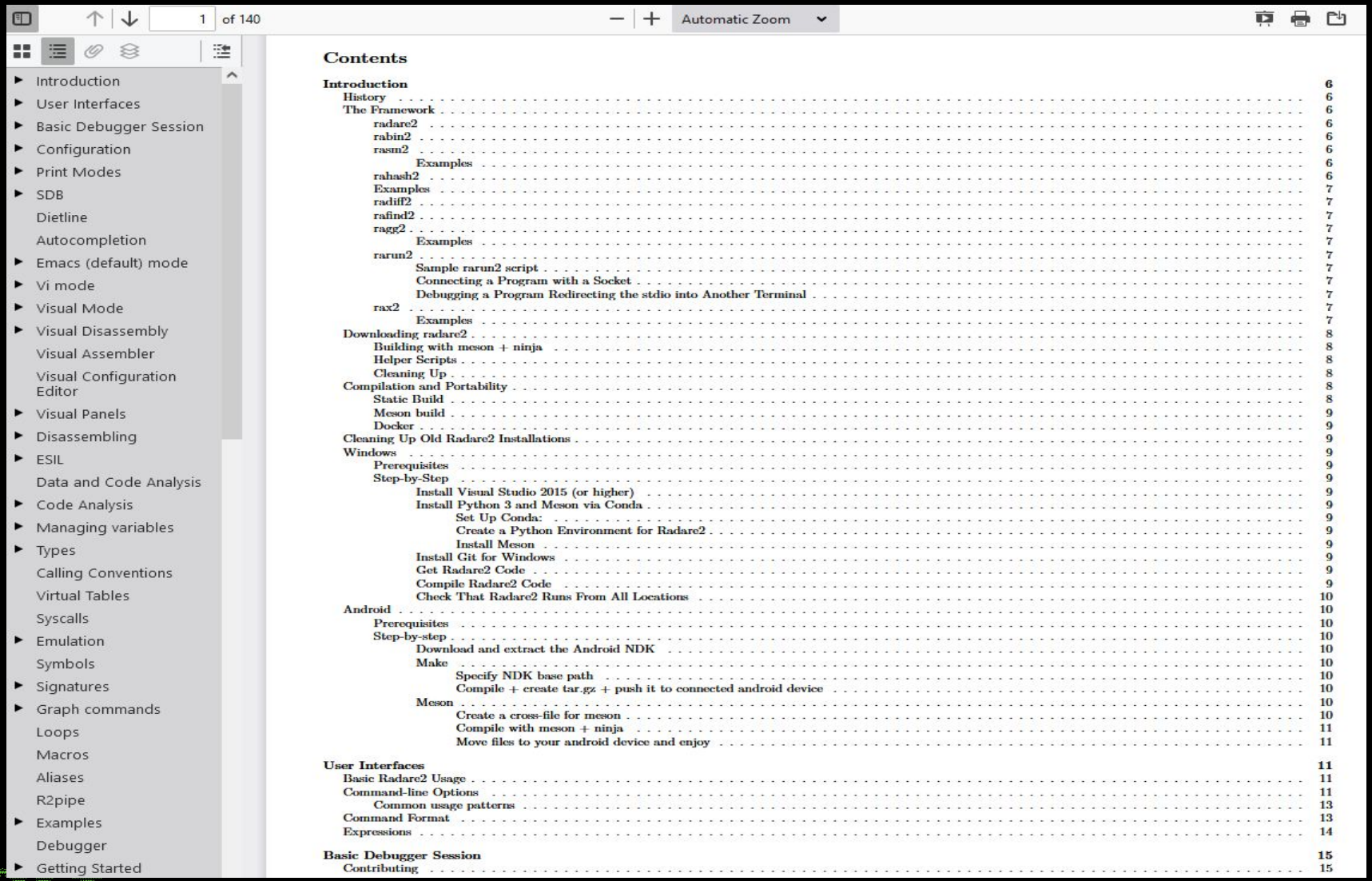
- ACR (./configure, make)
- Meson (ninja, visual studio)

C standard use to test code miss- syntax

Compilers supported

- Tcc, Clang, Gcc, MSVC, and probably more if they support C90

Use more reference - r2book



Contents

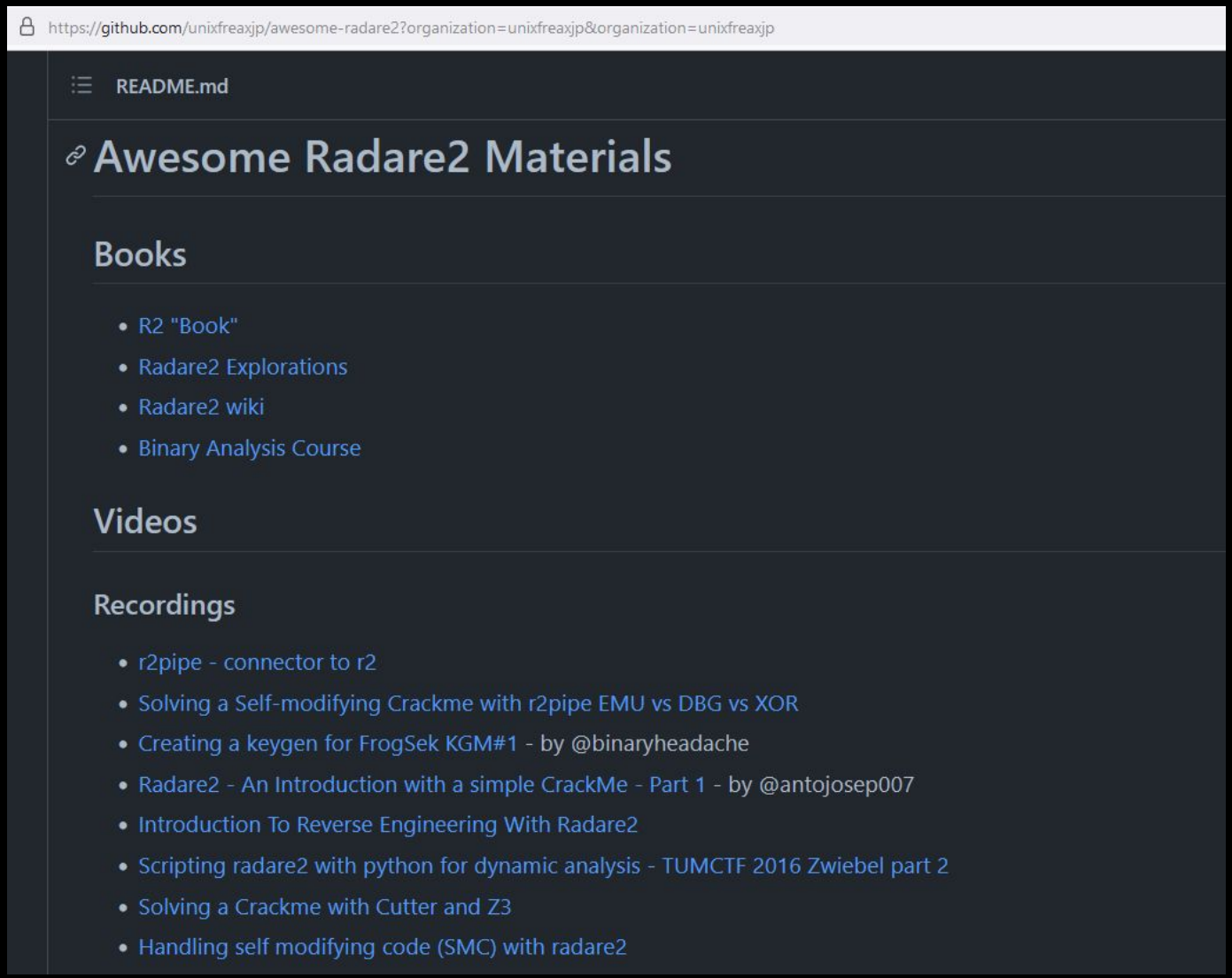
Use more reference - r2 project's resources links

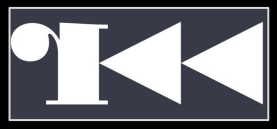
https://github.com/radareorg/r2jp

README.md

- site https://www.radare.org/ or http://rada.re/
- twitter @radareorg https://twitter.com/radareorg
- releases https://github.com/radare/radare2/releases
- dev/source: https://github.com/radare/radare2
- doc http://rada.re/vdoc/
- book https://radare.gitbooks.io/radare2book/content/
- wiki https://r2wiki.readthedocs.io/en/latest/
- etc documentaton http://radare.today/posts/radare2-is-documented/
- blog1 http://radare.today
- blog2 https://radareorg.github.io/blog/
- installers http://radare.mikelloc.com/list
- r2con conference https://rada.re/con/ or https://github.com/radareorg/r2con
- web demo http://cloud.rada.re (今メインテナンス中)
- ctf tips http://radare.today/posts/using-radare2/
- cheatsheet https://github.com/radareorg/radare2/blob/master/doc/intro.md
- radare2 UNIX servers compatibility info, ～April 2020
- devcode CLang checks (Jenkins) http://ci.radare.org/job/radare2-scan-build/
- practical aspects of using r2 aka radare2 exploration
- referece card https://radare.gitbooks.io/radare2book/content/refcard/intro.html

Use more reference - or this one...



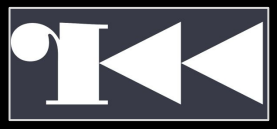


The contents

"Debugging & Bugfixing r2"

@unixfreaxjp

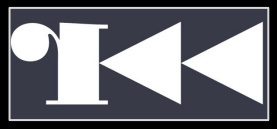
1. Introduction
2. Why testing is needed
   - Definition, Bugfix steps, target etc
   - Requirements
     - Human: QC human spec
     - Stuff: OS, Arch, exotic
     - Dedicated time
   - What to test
     - Packages, usage, approach
     - Dealing with plugins
3. Practical test
   - Knowing the limit
   - Testing in action
4. Bug hunting
   - Playbook to handle bug
5. Report
   - Test/Bug reports leads to next features in r2 miletones
   - Examples



LAC
ともに、イキル

Chapter five Report

“Documentation is everything..”

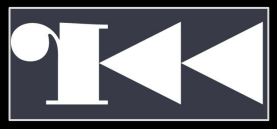


LAC
ともに、イキル

Bug Reporting

What to be included in a bug report:

- The background story
- Environment information
- The bug visualization
  - Screenshot
  - Video
  - Smart playback shell interface
- How did you install radare2
- Description of bug affected components
- The bug regeneration information (commands, setup, usage etc)
- (optional) The problem source
- (optional) Advise on how to bugfix
- (optional) The sample, scripts, etc data that caused the bug



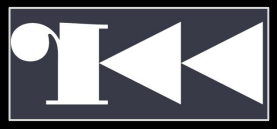
LAC
ともに、イキル

Bug Reporting - Environment

- r2 version
- OS version
- suppl:
  Date, etc

Environment:

```
$
$ date
Fri Jan 29 22:12:34 JST 2021
$
$ r2 -v
radare2 5.1.0 0 @ linux-arm-64 git.
commit: HEAD build: 2021-01-29__19:16:55
$
$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
VERSION_CODENAME=stretch
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
$
$ uname -a
Linux arm64-cortex-a57 4.9.0-14-arm64 #1 SMP Debian 4.9.246-2 (2020-12-17) aarch64 GNU/Linux
$
```



LAC
ともに、イキル

Bug Reporting - Visualization

- Screenshot
- Video
- Playback

  etc

Requireement (ports):

```
devel/gmake
```

Fixed screenshot:

```
$
$ date ; uname -msv
Tue Jan 19 17:32:45 JST 2021
FreeBSD FreeBSD 12.2-RELEASE-p1 GENERIC  amd64
$
$ sudo gmake purge
rm -rf "/usr/local/share/doc/radare2"
cd man ; for FILE in *.1 ; do rm -f "/usr/local/share/man/man1/
rm -f "/usr/local/share/man/man1/r2.1"
rm -f "/usr/local/lib/libr_"*".a"
rm -f "/usr/local/lib/pkgconfig/r_"*.pc
rm -rf "/usr/local/include/libr"
rm -f "/usr/local/lib/radare2/5.0.1-git/-"*
rm -f ~/bin/"rabin2" rm -f ~/bin/"radare2" rm -f ~/bin/"radiff2"
~/bin/"rarun2" rm -f ~/bin/"rasign2" rm -f ~/bin/"rasm2" rm -f
dent" rm -f ~/bin/"r2r"
rm -f ~/bin/r2
rmdir ~/bin
rmdir: /root/bin: No such file or directory
gmake: [Makefile:312: user-uninstall] Error 1 (ignored)
for FILE in rabin2 radare2 radiff2 rafind2 ragg2 rahash2 rarun2
n/$FILE" ; done
rm -f "/usr/local/bin/ragg2-cc"
rm -f "/usr/local/bin/r2"
rm -f "/usr/local/lib/libr_"*
rm -f "/usr/local/lib/libr2"*".so"
rm -rf "/usr/local/lib/radare2"
rm -rf "/usr/local/include/libr"
```

Bug Reporting - Clear bug description

Bug details

1. Summary:

During `make uninstall` and `make purge` in radare2 core components, these `make` options (`uninstall|deinstall` or `purge`) are having execution error in FreeBSD OS, the removal process for targeted binaries and library files has not been working as expected (not cleanly uninstalled and purged).

Tested r2 base version snapshot:

```
radare2 5.0.1-git 25317 @ linux-x86-32 git.5.0.0
commit: b4b48cbf86f99dd3952074c96660df926f81a5e3 build: 2020-12-26__17:33:55

radare2 5.0.1-git 25317 @ freebsd-x86-32 git.5.0.0
commit: b4b48cbf86f99dd3952074c96660df926f81a5e3 build: 2020-12-26__17:41:35
```

OS:

```
FreeBSD 12.2-RELEASE
arch: 64bit & 32bit
Compilers: Clang, Cmake, GCC, ACR supported
```

2. Problem:

This is a typical incompatibility `make` syntax under ifeq (for freebsd), in example declarations like `DL_LIBS=` or `CAPSTONE_LDFLAGS=` at `ifeq freebsd` part, FreeBSD `make` will need further operators, it affects scripts in `config-user.mk`, `global.mk`, `Makefile`, `config.mk` which some of them were automated created by related ACR files.

Let me elaborate further: The current used `ifreq` syntax is not for `make` but for `gmake` in FreeBSD OS. As the result, the make script will be exit with error flag, process is halted, and the uninstall (deinstall) and purge further deletion actions will not be executed.

Bug Reporting - Solution & PoC

3. Solution:

Theer are two options as solution:

- Need a an adjustment in ACR files for FreeBSD's make ifreq syntax, for example at `config-user.mk.acr` on these bwlow lines, as well as in `global.mk` and `configure.acr`.
  - https://github.com/radareorg/radare2/blob/master/config-user.mk.acr#L51
  - https://github.com/radareorg/radare2/blob/master/config-user.mk.acr#L55
  - https://github.com/radareorg/radare2/blob/master/config-user.mk.acr#L64
- Keep the current ACR and adding requirement installation of `devel/gmake` for radare2 build for FreeBSD users.

4. PoC/Log/screenshots/etc

Current error in `make uninstall` and `make purge` in FreeBSD:

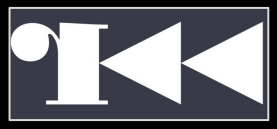
```
$
$ freebsd-version
12.2-RELEASE-p2
$ date
Sat Dec 26 18:09:54 JST 2020
$
$ r2 -v
radare2 5.0.1-git 25317 @ freebsd-x86-64 git.5.0.0
commit: b4b48cbf86f99dd3952074c96660df926f81a5e3 build: 2020-12-26__17:23:29
$
$ sudo make uninstall
make: "/usr/home/test/radare2-5.0.0-git-last/config-user.mk" line 51: Missing dependency operator
make: "/usr/home/test/radare2-5.0.0-git-last/config-user.mk" line 52: Missing dependency operator
make: "/usr/home/test/radare2-5.0.0-git-last/config-user.mk" line 54: Need an operator
```

Bug Reporting - Workaround

3. Workaround:

The problem has workaround by rollback radare2 v5.0.1-git(v5.0.0+last-dev) into commit: b4b48cbf86f99dd3952074c96660df926f81a5e3 and please use r2ghidra commit: a356affa58320b44d92f35c0c1ed

```
sym.imp.verrx(2, arg1, &var_20h_2);
uVar2 = fcn.00202900();
if (*(int32_t *)0x205808 < 1) {
    iVar6 = 0;
} else {
    *(int32_t *)0x205808 = *(int32_t *)0x205808 + -1;
    iVar6 = *(int64_t *)(*(int64_t *)0x205810 + 8);
    *(int64_t *)0x205810 = *(int64_t *)0x205810 + 8;
}
iVar1 = *(int32_t *)0x205808;
iVar3 = fcn.00202770(iVar6);
if (iVar3 == 0x402) {
    if (iVar1 < 1) {
        iVar6 = 0;
    } else {
        *(int32_t *)0x205808 = iVar1 + -1;
        iVar6 = *(int64_t *)(*(int64_t *)0x205810 + 8);
        *(int64_t *)0x205810 = *(int64_t *)0x205810 + 8;
    }
    uVar4 = fcn.00202770(iVar6);
    uVar5 = fcn.002026d0(uVar4);
    uVar2 = (uint32_t)((uVar5 | uVar2) != 0);
```



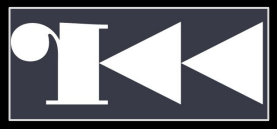
LAC
ともに、イキル

Bug Reporting

(Several examples of the explained points below)

- The background story
- Environment information
- The bug visualization
  - Screenshot
  - Video
  - Smart playback shell interface
- How did you install radare2
- Description of bug affected components
- The bug regeneration information (commands, setup, usage etc)
- (optional) The problem source
- (optional) Advise on how to bugfix
- (optional) The sample that caused the bug

Question(s)?



Salutation and thank you

Thanks for watching and help us to test radare2!

Kudos: pancake & greets to other friends in r2 community

Quiz: Which area of Japan were these photographs taken?

You can reach me at twitter by DM to @malwaremustd1e

@unixfreaxjp Oct, 2021 - r2jp, mmd