Proactive bug finding

Take advantage of the Debian architecture to find bugs

Sam Hocevar (*sam@zoy.org*)

22 June 2007
DebConf'07, Edinburgh

Summary

- find bugs in the sources
  - search the code
  - gcc warnings explained
  - build & buildd logs
- bugs in binary packages
  - Debian QA tools
  - find regressions
- runtime bugs
  - test suites
  - fuzzing

Part 1

find bugs in the sources

Code search

- **full audits**
  - expensive (time, skills, tools)
  - only worth it for critical components
  - Debian Security Audit Project
    `http://www.debian.org/security/audit/`
  - definitely needed, but...
- **partial audits, quick skimming**
  - cheap, fast, automated
  - 20% of the energy to find 80% of the bugs
  - by no ways perfect, but finding a bug never hurts
  - grep the code, grep the build logs

A common bug (1)

```
void write_long(char *buffer, long i)
{
    long * tmp = buffer;
    tmp[0] = i;
}
```

- may work depending on **buffer**, but may crash at random on arm, sparc...
- **gcc** emits a warning:

```
warning: initialization from
  incompatible pointer type
```

A common bug (2)

```
void write_long(char *buffer, long i)
{
    long * tmp = (long *)buffer;
    tmp[0] = i;
}
```

- generates the same code
- gcc doesn't say anything!
- this kind of bug needs to be searched directly in the code:

```
grep '( *long *\(int *\|\)\*)'
```

A common bug (3)

```
void write_long(char *buffer, long i)
{
    memcpy(buffer, &i, 4);
}
```

- no alignment issues now
- gcc will inline memcpy() for speed
- but will only write half the long integer on sparc, amd64...
- cannot be automatically found, but a few clever regexes can help

A common bug (4)

```
void write_long(char *buffer, long i)
{
    memcpy(buffer, &i, sizeof(i));
}
```

- this is one correct way to do it
- bonus hint: use #include <stdint.h>
  - guaranteed int8_t, int32_t, etc.
  - use it in new applications
  - can be useful to port old i386 applications

Static code search

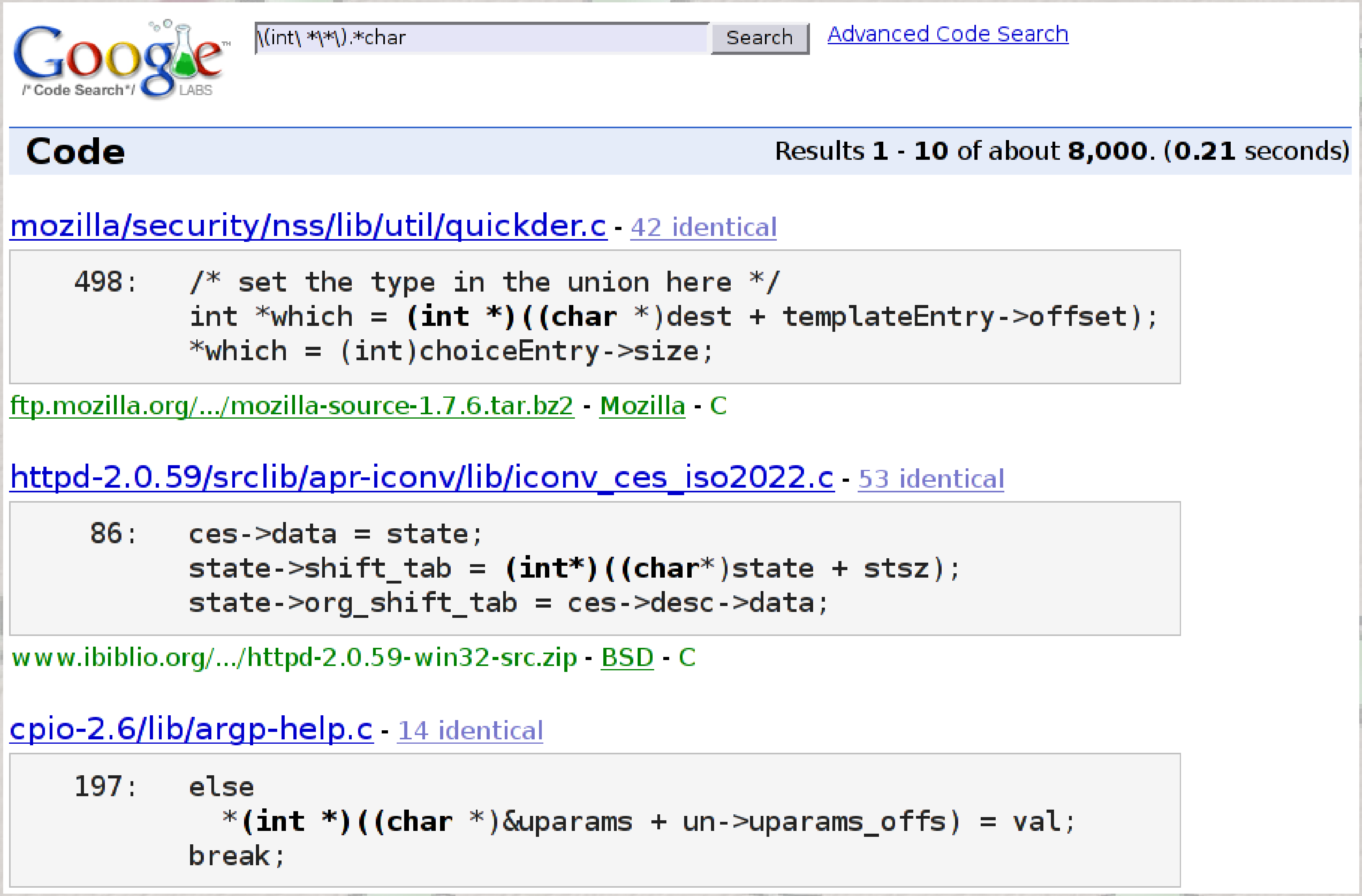
- unpack the whole source archive
  1. get a big hard drive (at least 110GiB)
  2. use debmirror
  3. untar everything
  - don't forget tarball-in-tarball packages
- grep through the code
  - think of all filenames (.C, .cpp, .c++, .cxx...)
  - or just grep through everything for safety
- use trial and error to think of ways to get rid of false positives

Google Code Search (1)

- `http://codesearch.google.com/`

- uses regexes, not usual Google syntax
- incredibly fast
- has limitations, though
  - far from having all the code that's in Debian
  - no multiline search
  - no easy way to ignore false positives

Google Code Search (2)



\(int\ *\)*\).*char

Code Results **1** - **10** of about **8,000**. (**0.21** seconds)

mozilla/security/nss/lib/util/quickder.c - 42 identical

```
  498:   /* set the type in the union here */
         int *which = (int *)((char *)dest + templateEntry->offset);
         *which = (int)choiceEntry->size;
```

ftp.mozilla.org/.../mozilla-source-1.7.6.tar.bz2 - Mozilla - C

httpd-2.0.59/srclib/apr-iconv/lib/iconv_ces_iso2022.c - 53 identical

```
   86:   ces->data = state;
         state->shift_tab = (int*)((char*)state + stsz);
         state->org_shift_tab = ces->desc->data;
```

www.ibiblio.org/.../httpd-2.0.59-win32-src.zip - BSD - C

cpio-2.6/lib/argp-help.c - 14 identical

```
  197:   else
           *(int *)((char *)&uparams + un->uparams_offs) = val;
         break;
```

Static analysis tools in Debian

- **rats**
  - does C, C++, PHP, Perl, Python
  - rather limited but still finds a lot of things
- **pscan**
  - only C, focuses on format strings
- **jlint**
  - checks Java code
- **pychecker**
  - checks Python code
- Google "static code analysis" for more

Compiler warnings

- what do they tell?
  - ambiguities, errors in the code
  - not always bugs
  - but they're emitted for a reason
- why should you look at them?
  - because your upstream doesn't have access to our variety of different architectures
- know what they mean first
  - blindly bypassing them could create bugs

Activate compiler warnings (1)

- which warnings do I want?
  - gcc has some warnings by default
  - you <u>always</u> want `-Wall`
  - `-W` can be useful
    - unused arguments
    - weird C or C++ constructs
  - lots of other useful ones
    - `-Wpointer-arith -Wcast-align -Wshadow -Wnested-externs -Wstrict-prototypes -Waggregate-return -Wmissing-prototypes -Wcast-qual -Wsign-compare...`
- seldom activated by upstream

Activate compiler warnings (2)

- autotools packages
  - in debian/rules:

```
CFLAGS="-Wall -W -Whatever -g"
CFLAGS="$(CFLAGS)" ./configure ...
```

  - the package may override the flags
- other packages
  - on a case by case basis
  - usually setting CFLAGS at build time works

Activate compiler warnings (3)

- what if upstream doesn't cooperate?
  - weird build systems
  - output redirected to /dev/null (*eg.* libtool)
- makewrap: `LD_PRELOAD` mechanism
  - `LD_PRELOAD=makewrap.so debian/rules`

  - wraps calls to `execve()`, `execvp()`
  - adds missing compiler warning flags
  - prevents `/dev/null` redirection
  - will be released soon(ish)

makewrap in action

```
        then mv -f ".deps/libmp4_plugin_la-mp4.Tpo" ".deps/libmp4_plugin_la-mp4.Plo"; \
        else rm -f ".deps/libmp4_plugin_la-mp4.Tpo"; exit 1; \
        fi
mkdir .libs
 ia64-linux-gnu-gcc -DHAVE_CONFIG_H -I. -I. -I../../.. -DSYS_LINUX -I../../../include -D_FILE_
*** makewrap warning *** "ia64-linux-gnu-gcc" called with "-Wall", adding "-W -Wsign-compare"
if /bin/sh ../../../libtool --mode=compile ia64-linux-gnu-gcc -DHAVE_CONFIG_H -I. -I. -I../../
          -c -o libmp4_plugin_la-libmp4.lo `test -f 'libmp4.c' || echo './'`libmp4.c; \
        then mv -f ".deps/libmp4_plugin_la-libmp4.Tpo" ".deps/libmp4_plugin_la-libmp4.Plo"; \
        else rm -f ".deps/libmp4_plugin_la-libmp4.Tpo"; exit 1; \
        fi
 ia64-linux-gnu-gcc -DHAVE_CONFIG_H -I. -I. -I../../.. -DSYS_LINUX -I../../../include -D_FILE_
*** makewrap warning *** "ia64-linux-gnu-gcc" called with "-Wall", adding "-W -Wsign-compare"
libmp4.c: In function 'MP4_ReadBox_url':
libmp4.c:698: warning: comparison between signed and unsigned
libmp4.c:698: warning: signed and unsigned type in conditional expression
libmp4.c:698: warning: comparison between signed and unsigned
libmp4.c:698: warning: signed and unsigned type in conditional expression
libmp4.c:698: warning: comparison between signed and unsigned
libmp4.c:698: warning: signed and unsigned type in conditional expression
libmp4.c: In function 'MP4_ReadBox_urn':
libmp4.c:720: warning: comparison between signed and unsigned
libmp4.c:720: warning: signed and unsigned type in conditional expression
libmp4.c:720: warning: comparison between signed and unsigned
libmp4.c:720: warning: signed and unsigned type in conditional expression
libmp4.c:720: warning: comparison between signed and unsigned
libmp4.c:720: warning: signed and unsigned type in conditional expression
```

Other compiler warnings (1)

- `implicit declaration of function ‘foo’`
  - usually a missing header include

  - compiler will assume `foo()` returns `int`
  - what if `foo()` actually returns a pointer?

  - compiler will infer argument types
  - what if an implicit cast was expected?

Other compiler warnings (2)

- **`suggest parentheses around assignment used as truth value`**
  - not a bug, but ignoring it could make you ignore other bugs
  - if you mean `if(x = 5)`, use `if((x = 5))`
- **`‘x’ might be used uninitialized in this function`**
  - only static variables are initialised to zero

Why use the buildd logs?

- all the data is in one place
  - `http://buildd.debian.org/`
  - text, easily greppable
- they have all the architectures
  - ...except yours; it would be nice to have our own build logs available, too
  - builds are not necessarily consistent across architectures (pointer sizes vary, system headers vary)

Part 2

bugs in binary packages

Debian QA tools: lintian

- what it does
  - checks source and binary packages
  - interprets the Debian policy
  - machine-readable output

```
E: libk1: old-fsf-address-in-copyright-file
W: libk1: shlib-without-dependency-information
  lib/libk.so.1
E: libk1: shlib-with-executable-bit
  lib/libk.so.1 0755
```

  - easily automated (`lintian.debian.org`)

Create lintian checks

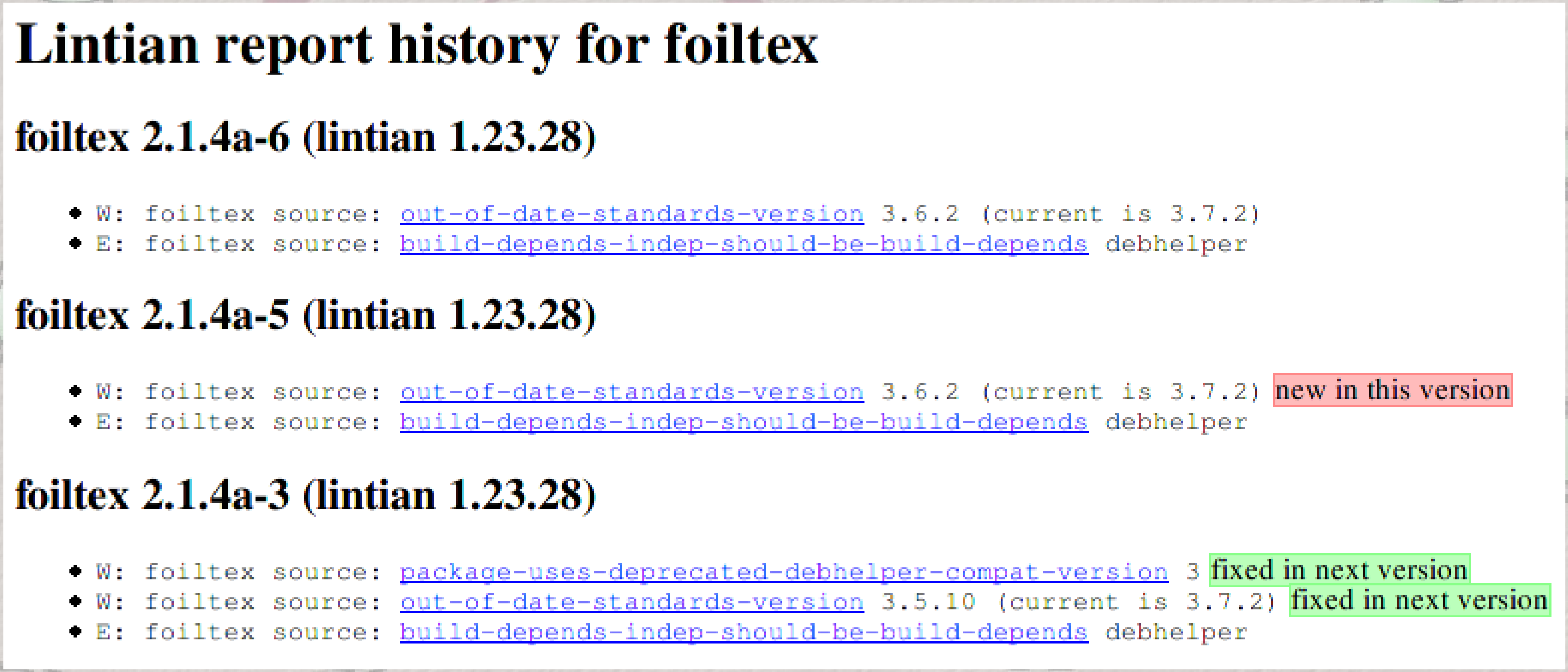
- the lintian process
  - unpacks packages in a laboratory
  - adds meta-information to the lab (list of scripts, objdump information...)
  - runs checks on the lab contents
- what is a check?
  - /usr/share/lintian/checks/blah
    - Perl code implementing run()
    - runs on the lab contents
    - calls the tag subroutine when errors are found
  - /usr/share/lintian/checks/blah.desc
    - verbose description of the tags

Improve lintian.debian.org

- add a history to answer useful queries
  - which warnings/errors appeared in my last upload? in the last lintian upgrade?
  - which package uploads fixed a given tag?
  - which packages saw the same tags appear? can I help fix them the same way?
- how to implement this?
  - SQL database
  - use mole?
  - proof of concept here:

`svn://svn.debian.org/svn/sam-hocevar/lintian`

New interface example



Lintian report history for foiltex

foiltex 2.1.4a-6 (lintian 1.23.28)

- W: foiltex source: out-of-date-standards-version 3.6.2 (current is 3.7.2)
- E: foiltex source: build-depends-indep-should-be-build-depends debhelper

foiltex 2.1.4a-5 (lintian 1.23.28)

- W: foiltex source: out-of-date-standards-version 3.6.2 (current is 3.7.2) new in this version
- E: foiltex source: build-depends-indep-should-be-build-depends debhelper

foiltex 2.1.4a-3 (lintian 1.23.28)

- W: foiltex source: package-uses-deprecated-debhelper-compat-version 3 fixed in next version
- W: foiltex source: out-of-date-standards-version 3.5.10 (current is 3.7.2) fixed in next version
- E: foiltex source: build-depends-indep-should-be-build-depends debhelper

Debian QA tools: linda

- very similar to lintian
  - same output format
  - different language (Python)
  - slightly different checks
- which one should I use?
  - both, of course

Create `linda` checks

- the `linda` process
  - similar to `lintian` (lab + checks)
- `linda` checks
  - `/usr/share/linda/checks/blah.py`
    - Python class deriving from `LindaChecker`
    - runs on the lab contents
    - calls `signal_error` when errors are found
  - `/usr/share/linda/data/blah.data`
    - list of tag types (warnings, errors...)
  - `/usr/share/linda/po/{en,de,..}.gmo`
    - verbose and i18n'ed descriptions of the tags

Why create new checks?

- it's not only about the policy
  - general QA stuff
  - transitions
- examples
  - packages with a menu file but no `.desktop`
  - packages with no icons
  - `X-Vcs` control fields
  - some ignored `DEB_BUILD_OPTIONS` flags
  - extract font copyright information
  - [insert your own personal crusade here]

Debian QA tools: piuparts

- how does it work?
  - debootstraps a minimal system
  - installs package
  - removes package
  - tests for cruft or errors
  - can check upgrades or mass-upgrades
- it takes time, but you should use it!

  (come on, everyone already knows you don't test your own packages)

Extending piuparts

- why?
  - because the framework is here
  - check for robustness before the user can
- what?
  - corrupt /var/cache, see what happens
  - check packages with /bin/sh set to zsh, bash...
  - not necessarily "bugs" for the policy, but often worth fixing
- how?
  - I don't know yet...

Part 3

runtime bugs

Test suites

- upstream software sometimes has them
  - can be activated at build time? do it!
  - tired of rebuilding your package? implement `DEB_BUILD_OPTIONS=nocheck` (#416450)
  - try to remain cross-buildable

```
ifeq ($(DEB_BUILD_GNU_TYPE), $(DEB_HOST_GNU_TYPE))
    $(MAKE) -C testsuite
endif
```

- you can create one yourself
  - not really your job
  - but bugs linked with other packages might reappear

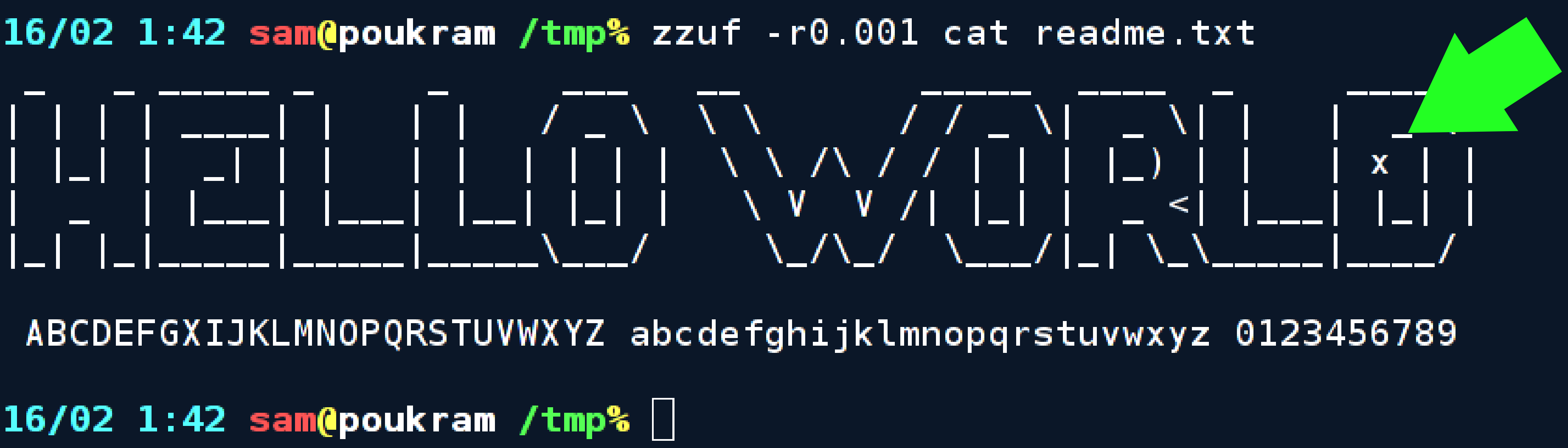
Fuzzing

- the idea
  - alter a program's input and watch its behaviour
- expose bugs
  - data is often user-contributed (web, e-mail)
  - file parsers, interpreters are complicated
  - can have security implications
- quick
  - still not the ultimate bug-finding solution
  - but any bug found is worth fixing

Presenting zzuf

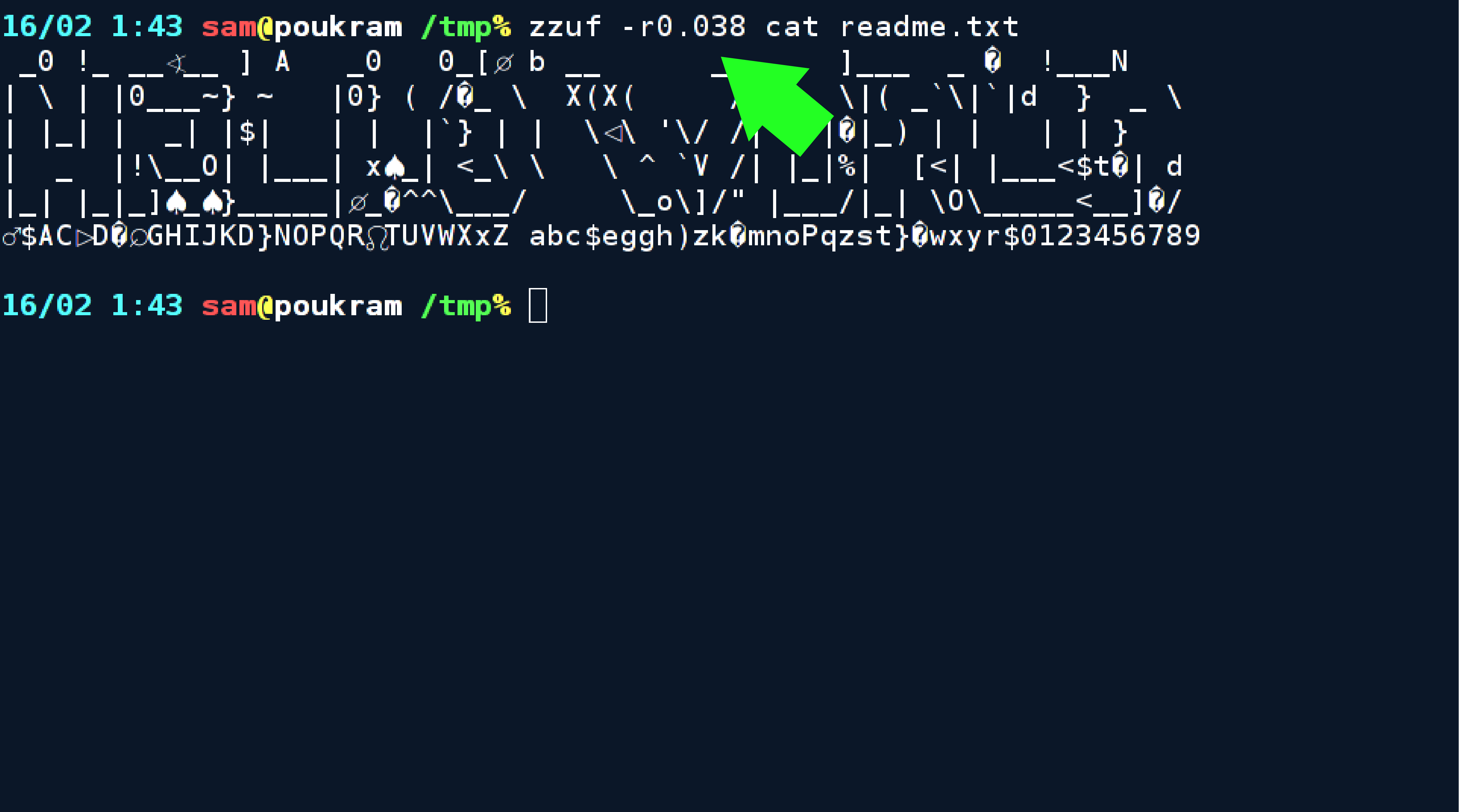
- **LD_PRELOAD fuzzing approach**
  - no modification or recompilation required
  - can fuzz files, but also DVDs, network...
- **fully automated**
  - checks for SIGSEGV, SIGABRT...
  - checks for memory usage
  - checks for infinite loops
- **reproducible behaviour**
  - can be used in batch mode until a bug is found
  - ideal for testsuites

zzuf example (1) - cat



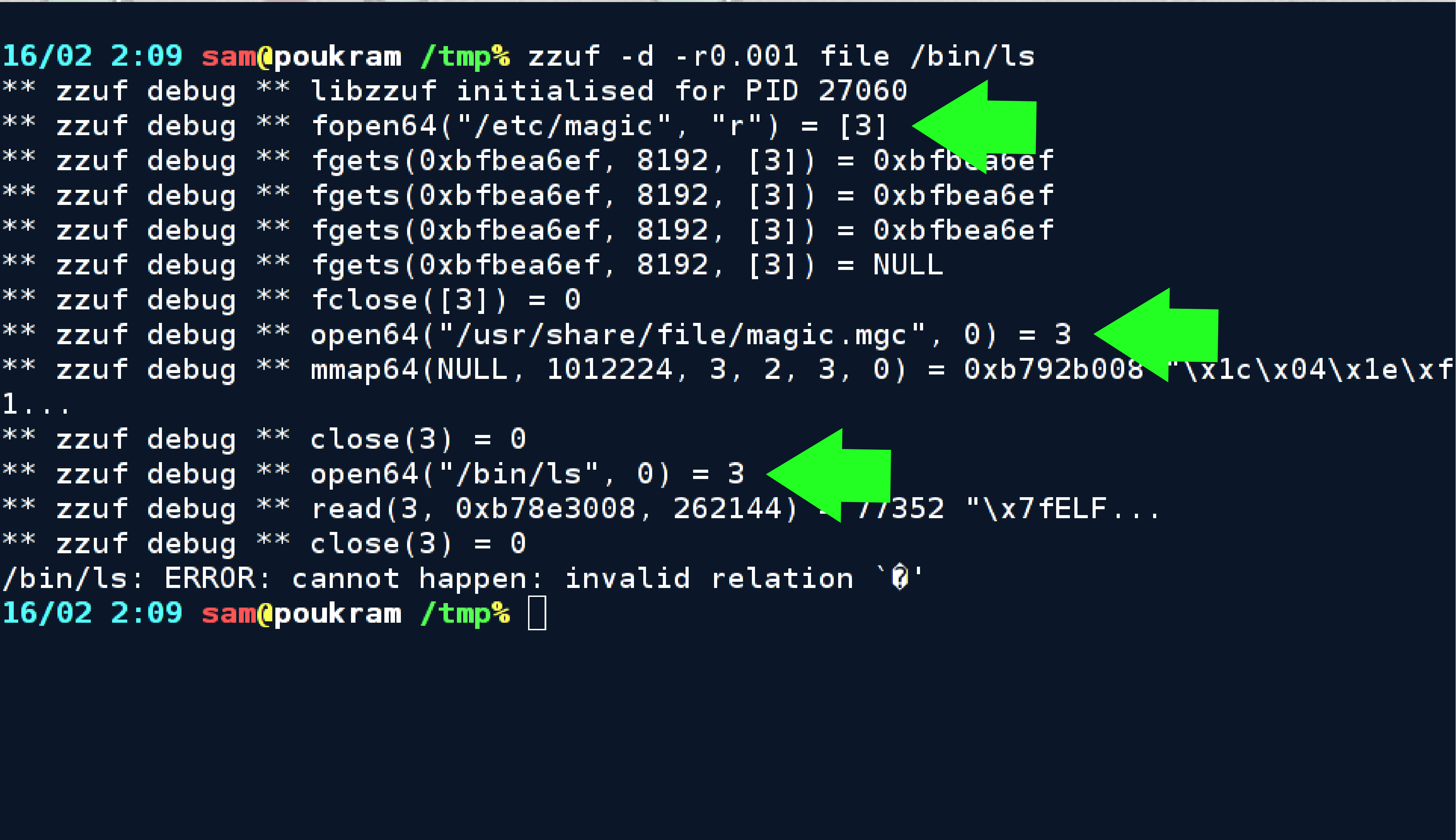
```
16/02 1:42 sam@poukram /tmp% zzuf -r0.001 cat readme.txt
 _   _ _____ _     _     ___   __        _____  ____  _     ____
| | | | ____| |   | |   / _ \  \ \      / / _ \|  _ \| |   |  _
| |_| |  _| | |   | |  | | | |  \ \ /\ / / | | | |_) | |   | x | |
|  _  | |___| |___| |__| |_| |   \ V  V /| |_| |  _ <| |___| |_| |
|_| |_|_____|_____|_____\___/     \_/\_/  \___/|_| \_\_____|____/

 ABCDEFGXIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 0123456789

16/02 1:42 sam@poukram /tmp% 
```

zzuf example (2) - cat



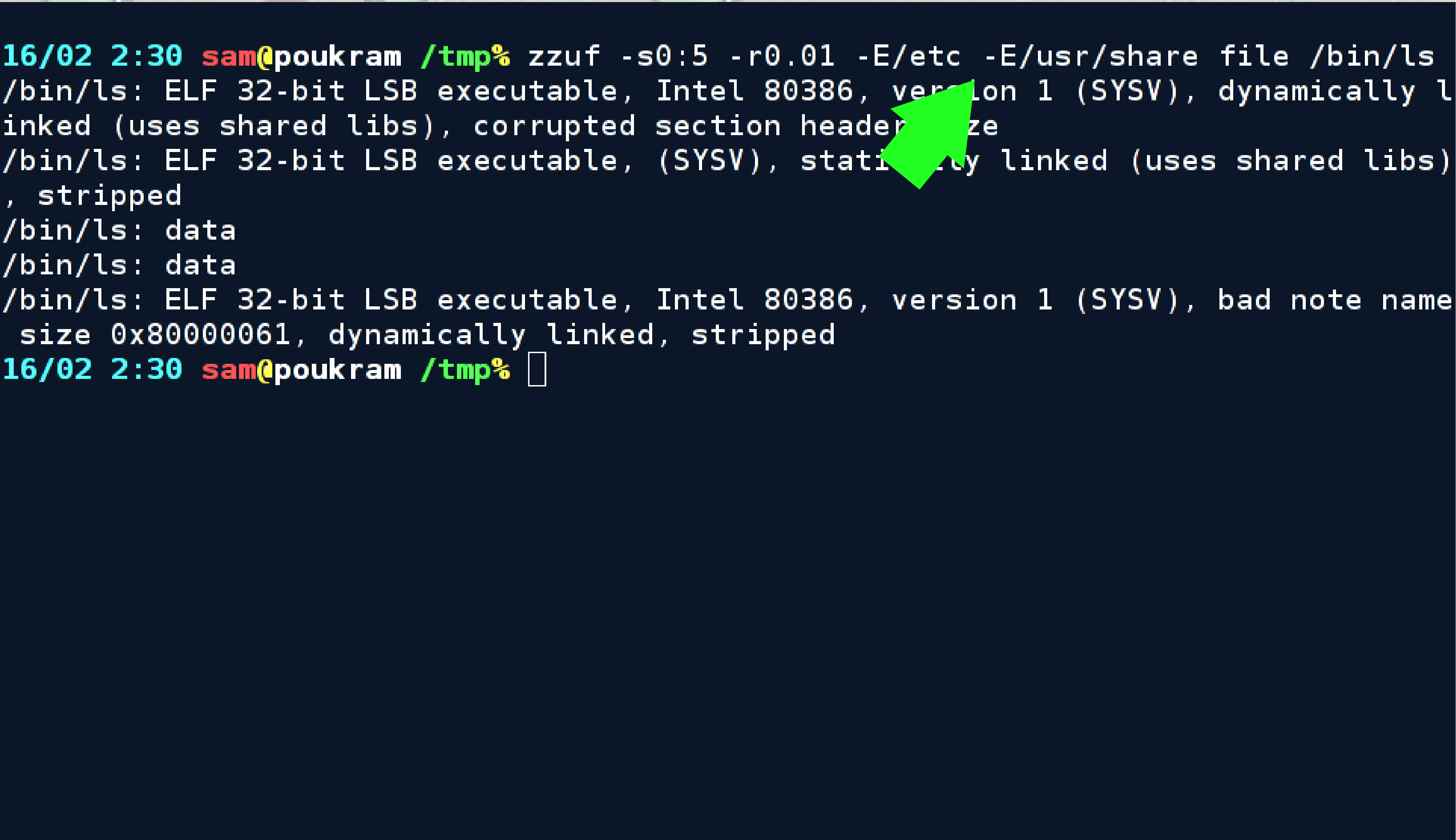
```
16/02 1:43 sam@poukram /tmp% zzuf -r0.038 cat readme.txt
 _0 !_ __⊀__ ] A   _0   0_[∅ b __      _      ]___  _ �  !___N
| \ | |0___~} ~   |0} ( /�_ \  X(X(      /    \|( _`\|`|d  }  _ \
| |_| |  _| |$|   | |  |`} | |  \◁\ '\/ /|   |�|_) | |    | | }
|  _  |!\__O| |___| x♠_| <_\ \   \ ^ `V /| |_|%|  [<| |___<$t�| d
|_| |_|_]♠_♠}______|∅_�^^\___/     \_o\]/" |___/|_| \O\______<__]�/
♂$AC▷D�⌕GHIJKD}NOPQR♌TUVWXxZ abc$eggh)zk�mnoPqzst}�wxyr$0123456789

16/02 1:43 sam@poukram /tmp% ▯
```

zzuf example (3) - file



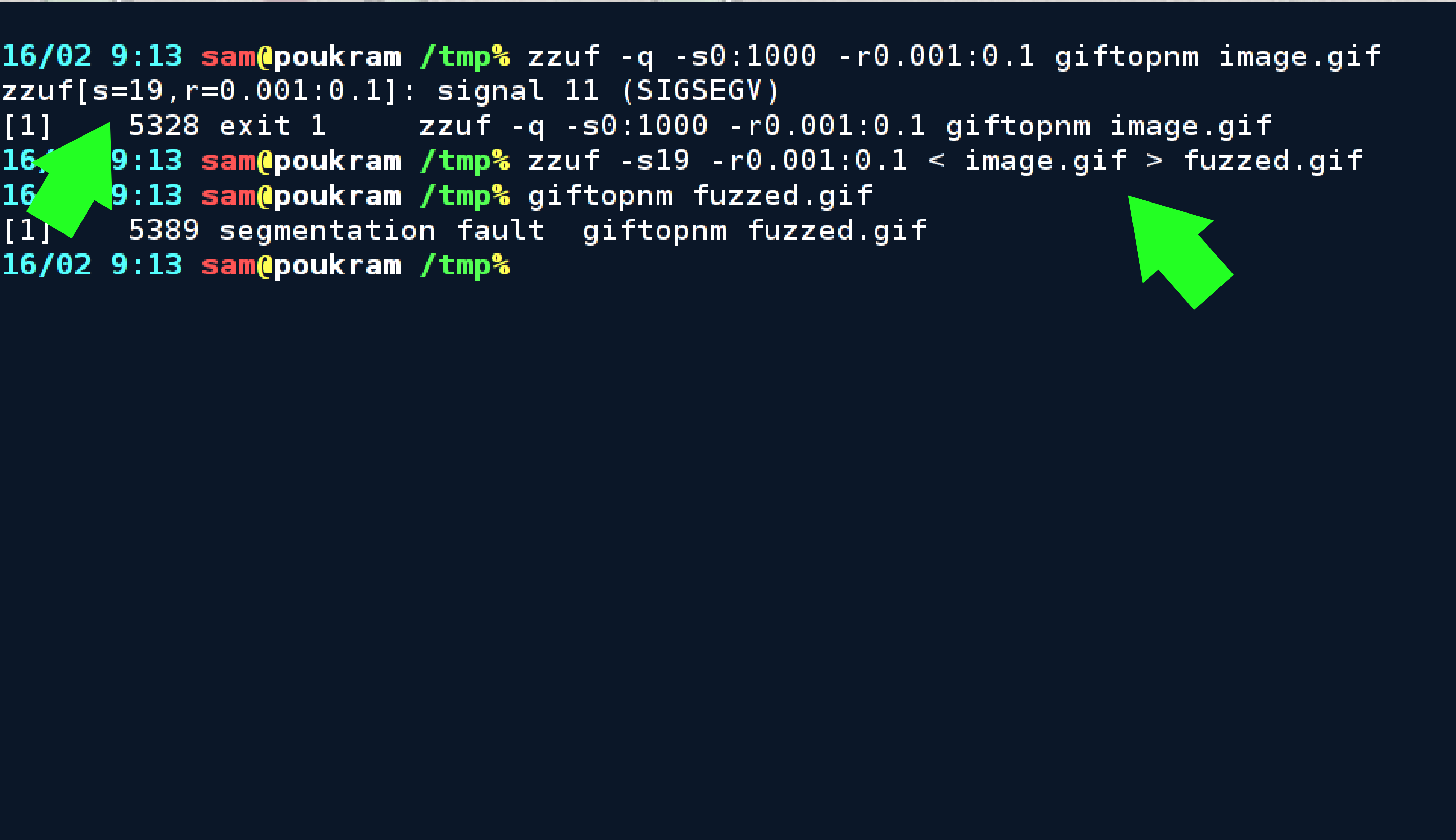
```
16/02 2:09 sam@poukram /tmp% zzuf -d -r0.001 file /bin/ls
** zzuf debug ** libzzuf initialised for PID 27060
** zzuf debug ** fopen64("/etc/magic", "r") = [3]
** zzuf debug ** fgets(0xbfbea6ef, 8192, [3]) = 0xbfbea6ef
** zzuf debug ** fgets(0xbfbea6ef, 8192, [3]) = 0xbfbea6ef
** zzuf debug ** fgets(0xbfbea6ef, 8192, [3]) = 0xbfbea6ef
** zzuf debug ** fgets(0xbfbea6ef, 8192, [3]) = NULL
** zzuf debug ** fclose([3]) = 0
** zzuf debug ** open64("/usr/share/file/magic.mgc", 0) = 3
** zzuf debug ** mmap64(NULL, 1012224, 3, 2, 3, 0) = 0xb792b008 "\x1c\x04\x1e\xf
1...
** zzuf debug ** close(3) = 0
** zzuf debug ** open64("/bin/ls", 0) = 3
** zzuf debug ** read(3, 0xb78e3008, 262144) = 77352 "\x7fELF...
** zzuf debug ** close(3) = 0
/bin/ls: ERROR: cannot happen: invalid relation `�'
16/02 2:09 sam@poukram /tmp% 
```

zzuf example (4) - file



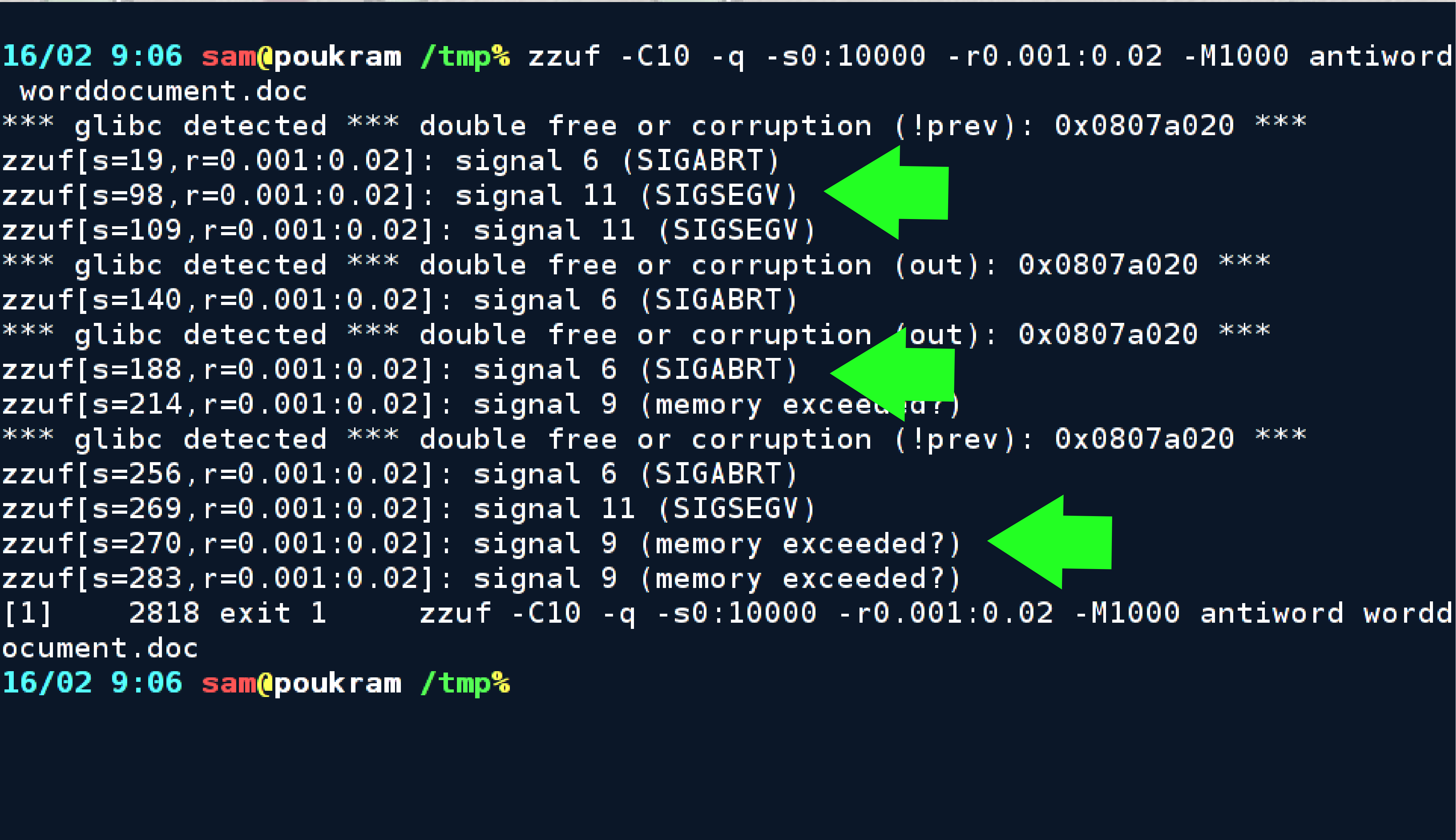
```
16/02 2:30 sam@poukram /tmp% zzuf -s0:5 -r0.01 -E/etc -E/usr/share file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, ver  ion 1 (SYSV), dynamically l
inked (uses shared libs), corrupted section header  ze
/bin/ls: ELF 32-bit LSB executable, (SYSV), stati   ly linked (uses shared libs)
, stripped
/bin/ls: data
/bin/ls: data
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), bad note name
 size 0x80000061, dynamically linked, stripped
16/02 2:30 sam@poukram /tmp% 
```

zzuf example (5) - giftopnm



```
16/02 9:13 sam@poukram /tmp% zzuf -q -s0:1000 -r0.001:0.1 giftopnm image.gif
zzuf[s=19,r=0.001:0.1]: signal 11 (SIGSEGV)
[1]    5328 exit 1     zzuf -q -s0:1000 -r0.001:0.1 giftopnm image.gif
16/02 9:13 sam@poukram /tmp% zzuf -s19 -r0.001:0.1 < image.gif > fuzzed.gif
16/02 9:13 sam@poukram /tmp% giftopnm fuzzed.gif
[1]    5389 segmentation fault  giftopnm fuzzed.gif
16/02 9:13 sam@poukram /tmp%
```

zzuf example (6) - antiword



```
16/02 9:06 sam@poukram /tmp% zzuf -C10 -q -s0:10000 -r0.001:0.02 -M1000 antiword
 worddocument.doc
*** glibc detected *** double free or corruption (!prev): 0x0807a020 ***
zzuf[s=19,r=0.001:0.02]: signal 6 (SIGABRT)
zzuf[s=98,r=0.001:0.02]: signal 11 (SIGSEGV)
zzuf[s=109,r=0.001:0.02]: signal 11 (SIGSEGV)
*** glibc detected *** double free or corruption (out): 0x0807a020 ***
zzuf[s=140,r=0.001:0.02]: signal 6 (SIGABRT)
*** glibc detected *** double free or corruption (out): 0x0807a020 ***
zzuf[s=188,r=0.001:0.02]: signal 6 (SIGABRT)
zzuf[s=214,r=0.001:0.02]: signal 9 (memory exceeded?)
*** glibc detected *** double free or corruption (!prev): 0x0807a020 ***
zzuf[s=256,r=0.001:0.02]: signal 6 (SIGABRT)
zzuf[s=269,r=0.001:0.02]: signal 11 (SIGSEGV)
zzuf[s=270,r=0.001:0.02]: signal 9 (memory exceeded?)
zzuf[s=283,r=0.001:0.02]: signal 9 (memory exceeded?)
[1]    2818 exit 1     zzuf -C10 -q -s0:10000 -r0.001:0.02 -M1000 antiword wordd
ocument.doc
16/02 9:06 sam@poukram /tmp%
```

Other fuzzing software

- hachoir
  - http://hachoir.org/
  - multiple purpose fuzzing, like zzuf
  - far cleverer than random fuzzing, attacks with knowledge of the file format
  - has parsers for many file formats
- WebFuzzer (SQL injection, XSS), ISIC (IP stacks), SPIKEFile, radiusfuzzer, fuzz, netsed (network)...
- Google for “fuzzing”, “fuzz testing”, “fault injection”...

Fuzzing as a testsuite

- why do this?
  - cheap way to create a testsuite
  - build-depend on a fuzzer, test at build-time
  - we have different architectures with different bugs and behaviours
  - using a different random seed each time means better chances to find a bug
- a few warnings
  - be reasonable, don't stress the buildds!
  - think before deciding to make the build fail

Test suites for GUI apps

- use the xvfb package
  - has an xvfb-run script
- warnings
  - you may need additional build dependencies
  - be sure your application exits!

Thanks!

- Any questions?

Slides available on
  http://sam.zoy.org/lectures/