

Application security

September 25, 2015

Administrative – submittal instructions

- answer the lab assignment's questions in written report form, as a text, pdf, or Word document file (no obscure formats please)
- email to csci5301@usc.edu
- exact subject title must be "applicationsecuritylab"
- deadline is start of your lab session the following week
- reports not accepted (zero for lab) if
 - late
 - you did not attend the lab (except DEN or prior arrangement)
 - email subject title deviates

Administrative – DETER accounts

- accounts were created last night
- you should have received an auto-generated email message with information and instructions
- please read it and respond to its request to act (change password) within 3 days

Upcoming lab schedule adjustment

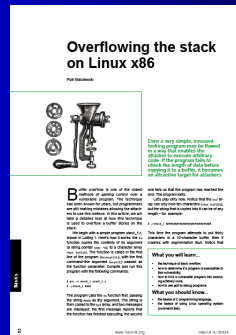
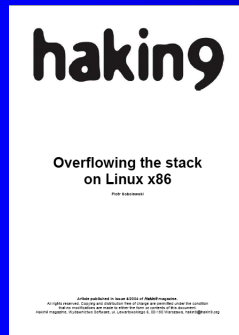
- we will have a capture-the-flag exercise
- among competitive teams
- extending over 5 weeks
- after the midterm, 10/16
- with the help of Prof Jelena Mirkovic
- it will replace one of our scheduled labs
- more to be announced later

Administrative

- refer during upcoming lab to these slides' screenshots
 - recommend you have paper or electronic access to those slides that contain detailed screenshots (lab asks you to mimic screenshot activities)
- use *only* the provided VM environment
(CentOS 4.3 min-gdb)
 - it has been customized a little
 - other platforms/compilers generally *won't work*

Administrative – pre-homework

- advance preparation for this lab
- read through page 8

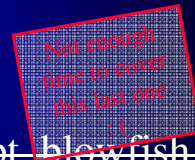


http://sobolewscy.in5.pl/piotr/publikacje/hakin9/stackoverflow_en.pdf
http://www-scf.usc.edu/~csci530l/downloads/stackoverflow_en.pdf

Three aspects of lab

- Generic stack overflow
- heartbleed bounds checking oversight

- ~~sign extension code flaw in crypt_blowfish~~

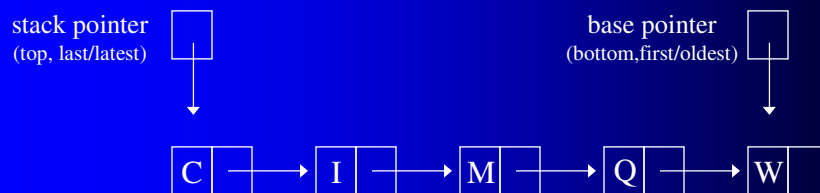


Generic stack buffer overflow

Stack buffer overflow

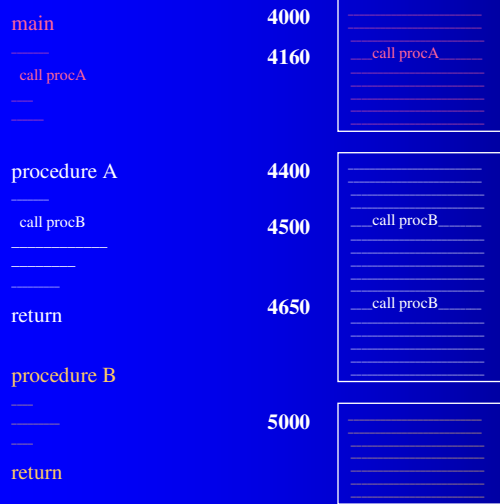
- what's a stack?
- what's an overflow?

Special list: a stack

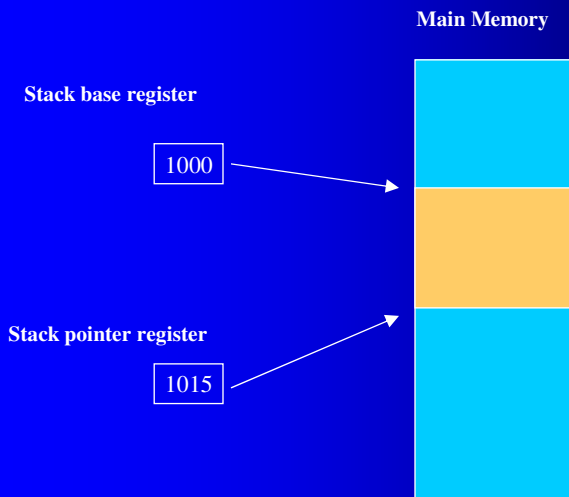


All insertions and deletions occur at one end, the “top.

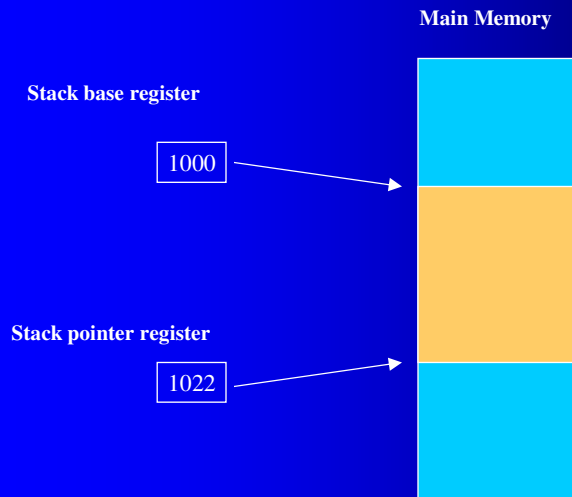
Used for intra-program control flow



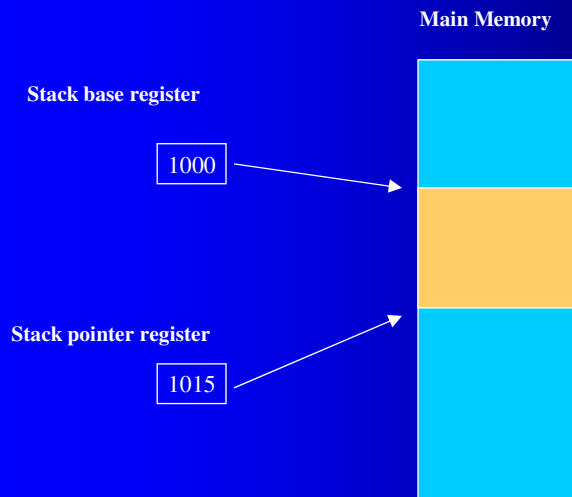
Uses a stack to get back



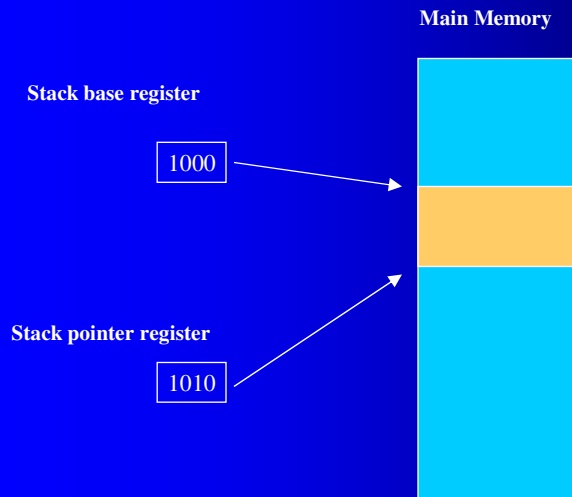
Uses a stack



Uses a stack



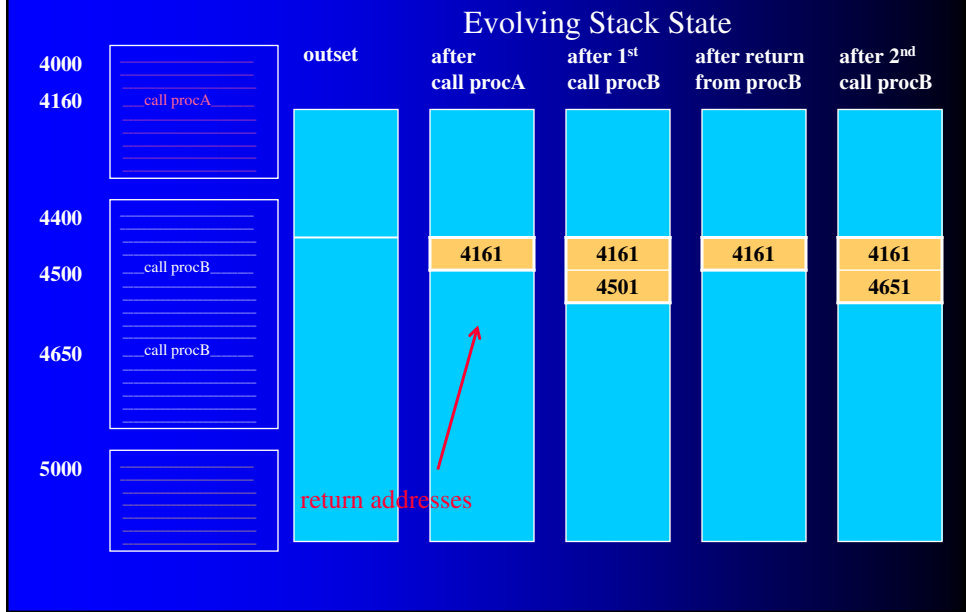
Uses a stack



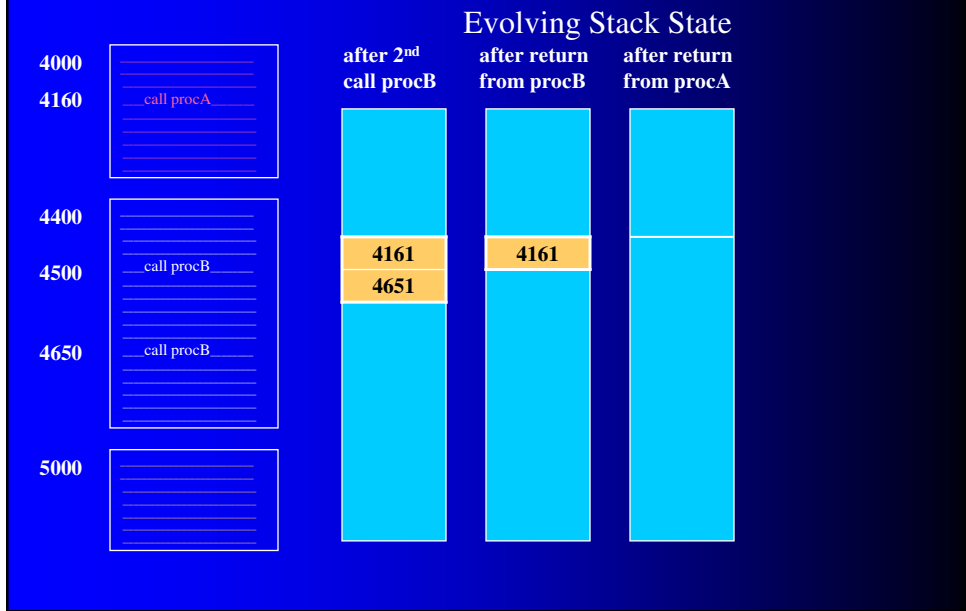
Finding your way back— breadcrumbs & return addresses



Intra-program Flow of control



Intra-program Flow of control

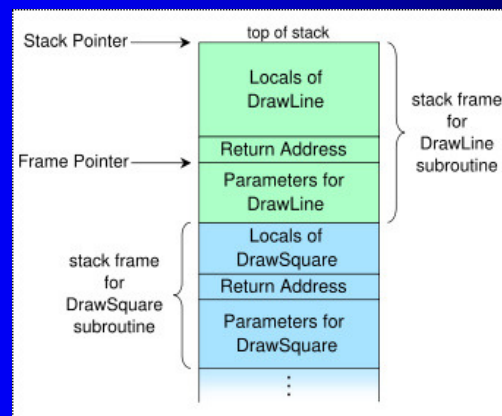


Not only return addresses get “stacked”

- local variables
- frame (intrastack) pointers
- return addresses
- arguments/parameters for called functions

Stack diagram

“For example, if a subroutine named DrawLine is currently running, having just been called by a subroutine DrawSquare, the top part of the call stack might be laid out like this (where the stack is growing towards the top):



From: http://en.wikipedia.org/wiki/Stack_frame#Structure

vars.c – has local variables

```
main (int argc, char *argv[]) {
    int a;
    int b;
    int c;
    a=1;
    b=2;
    c=3;
    printf("the end\n");
}
```

Local variables on the stack

(gdb) list

```
1  main (int argc, char *argv[]) {
2      int a;
3      int b;
4      int c;
5      a=1;
6      b=2;
7      c=3;
8      printf("the end\n");
9  }
```

(gdb) break 6
Breakpoint 1 at 0x804838b: file temp.c, line 6.
(gdb) run
Starting program: /root/.stack/temp

Breakpoint 1, main (argc=1, argv=0xbfe77654) at temp.c:6

```
6      b=2;
(gdb) x/12 $esp
0xbfe775a0: 0x00000000 0x00000000 0xbfe775c8 0x080483c6
0xbfe775b0: 0x0075eff4 0x0075eff4 0x0804949c 0x0075eff4
0xbfe775c0: 0x00000000 0x00000000 0xbfe77628 0x0064bde3
(gdb) next
7      c=3;
(gdb) x/12 $esp
0xbfe775a0: 0x00000000 0x00000000 0xbfe775c8 0x080483c6
0xbfe775b0: 0x0075eff4 0x0075eff4 0x0804949c 0x0075eff4
0xbfe775c0: 0x00000002 0x00000000 0xbfe77628 0x0064bde3
(gdb) next
8      printf("the end\n");
(gdb) x/12 $esp
0xbfe775a0: 0x00000000 0x00000000 0xbfe775c8 0x080483c6
0xbfe775b0: 0x0075eff4 0x0075eff4 0x0804949c 0x0075eff4
0xbfe775c0: 0x00000002 0x00000000 0xbfe77628 0x0064bde3
(gdb)
```

esp=0xbfe775a0 (latest)
ebp=0xbfe775c8 (oldest)

┌ - byte where esp points
└ - byte before where ebp points

↑ stack growth
↓ higher addresses

rvals.c – has a function

```
void fn() {
    printf("now we are in fn\n");
}

main () {
    printf("now we are in main\n");
    fn();
}
```

frame pointers & return addresses

```
root@Snort:~/stack
(gdb) list
1   void fn() {
2       printf("now we are in fn\n");
3   }
4
5   main () {
6       printf("now we are in main\n");
7       fn();
8   }
(gdb) break 7
Breakpoint 1 at 0x80483ac: file rvals.c, line 7.
(gdb) run
Starting program: /root/stack/rvals
now we are in main

Breakpoint 1, main () at rvals.c:7
7   fn()
(gdb) print $esp
$1 = (void *) 0xbfea7950
(gdb) print $ebp
$2 = (void *) 0xbfea7968
(gdb) x/6 $esp
0xbfea7950: 0x0075eff4 0x0075eff4 0x080494c0 0x0075eff4
0xbfea7960: 0x00000000 0x00633ca0
(gdb) break 2
Breakpoint 2 at 0x804836e: file rvals.c, line 2.
(gdb) next

Breakpoint 2, fn () at rvals.c:2
2   printf("now we are in fn\n");
(gdb) print $esp
$3 = (void *) 0xbfea7940
(gdb) print $ebp
$4 = (void *) 0xbfea7948
(gdb) x/10 $esp
0xbfea7940: 0x080484a6 0x00000000 0xbfea7968 0x080483b1
0xbfea7950: 0x0075eff4 0x0075eff4 0x080494c0 0x0075eff4
0xbfea7960: 0x00000000 0x00633ca0
(gdb)
```

Annotations:

- Red circle around `0xbfea7968` in the `$ebp` print output, with a red arrow pointing to it and the text: "pointer to base of current stack/frame (byte preceding stack's first), in register".
- Red circle around `0xbfea7968` in the `x/10 $esp` memory dump, with a red arrow pointing to it and the text: "pointer to base of previous stack/frame, in stack".
- Red arrow pointing to the `0xbfea7968` in the memory dump with the text: "before function call".
- Red arrow pointing to the `0xbfea7968` in the memory dump with the text: "after function call".
- Blue arrow pointing to the `0x080483b1` in the memory dump with the text: "previous frame, intact".

...continued... & return addresses

```
(gdb) print $esp
$3 = (void *) 0xbfea7940
(gdb) print $ebp
$4 = (void *) 0xbfea7948
(gdb) x/10 $esp
0xbfea7940: 0x000484a6 0x00000000 0xbfea7968 0x000483b1
0xbfea7950: 0x0075eff4 0x0075eff4 0x000494c0 0x0075eff4
0xbfea7960: 0x00000000 0x00633ca0
(gdb) disas main
Dump of assembler code for function main:
0x08048380 <main+0>: push %ebp
0x08048381 <main+1>: mov %esp,%ebp
0x08048383 <main+3>: sub $0x8,%esp
0x08048386 <main+6>: and $0xfffff0,%esp
0x08048389 <main+9>: mov $0x0,%eax
0x0804838a <main+10>: add $0xf,%eax
0x08048391 <main+17>: add $0xf,%eax
0x08048394 <main+20>: shr $0x4,%eax
0x08048397 <main+23>: shl $0x4,%eax
0x0804839a <main+26>: sub %eax,%esp
0x0804839c <main+28>: sub $0xc,%esp
0x0804839f <main+31>: push $0x0484a6
0x080483a4 <main+36>: call 0x80482d0
0x080483a9 <main+41>: add $0x10,%esp
0x080483ac <main+44>: call 0x8048368 <fn>
0x080483b1 <main+49>: leave
0x080483b2 <main+50>: ret
End of assembler dump.
(gdb) print $ebp+4
$5 = (void *) 0xbfea794c
(gdb) x 0xbfea794c
0xbfea794c: 0x080483b1
(gdb)
```

breadcrumb!
place to go back
to in calling routine, when done

where to go back to

where you left off

stack_2.c –function parameters to pass

```
void fn(int arg1, int arg2) {
    int x; int y;
    x=3; y=4;
    printf("now we are in fn\n");
}

main () {
    int a; int b;
    a=1; b=2;
    fn(a, b);
}
```

...and args for called functions

```

1 void fn(int arg1, int arg2) {
2     int x; int y;
3     x=3; y=4;
4     printf("now we are in fn\n");
5 }
6
7 main () {
8     int a; int b;
9     a=1; b=2;
10    fn(a, b);
11 }

```

(gdb) break 10
Breakpoint 1 at 0x00483b8: file stack_2.c, line 10.
(gdb) break 4
Breakpoint 2 at 0x004837c: file stack_2.c, line 4.
(gdb) run
Starting program: /root/stack/stack_2

Breakpoint 1, main () at stack_2.c:10
10 fn(a, b);
(gdb) print \$esp
\$1 = (void *) 0xbfed9d10
(gdb) print \$ebp
\$2 = (void *) 0xbfed9d28
(gdb) x/6 \$esp
0xbfed9d10: 0x0075eff4 0x0075eff4 0x080494c4 0x0075eff4
0xbfed9d20: 0x00000002 0x00000001

Breakpoint 2, fn (arg1=1, arg2=2) at stack_2.c:4
4 printf("now we are in fn\n");
(gdb) print \$esp
\$3 = (void *) 0xbfed9cf0
(gdb) print \$ebp
\$4 = (void *) 0xbfed9cf8
(gdb) x/14 \$esp
0xbfed9cf0: 0x00000004 0x00000003 0xbfed9d28 0x080483c6
0xbfed9d00: 0x00000001 0x00000002 0xbfed9d28 0x080483e6
0xbfed9d10: 0x0075eff4 0x0075eff4 0x080494c4 0x0075eff4
0xbfed9d20: 0x00000002 0x00000001

args for fn, placed on stack via main (points to 0x0075eff4)

local vars of main (bottom) and fn (top) (points to 0x00000001)

pointer to base of previous stack frame (points to 0xbfed9cf8)

return address (points to 0x080483c6)

...continued

```

(gdb) print $esp  
$3 = (void *) 0xbfed9cf0  
(gdb) print $ebp  
$4 = (void *) 0xbfed9cf8  
(gdb) x/14 $esp  
0xbfed9cf0: 0x00000004 0x00000003 0xbfed9d28 0x080483c6  
0xbfed9d00: 0x00000001 0x00000002 0xbfed9d28 0x080483e6  
0xbfed9d10: 0x0075eff4 0x0075eff4 0x080494c4 0x0075eff4  
0xbfed9d20: 0x00000002 0x00000001

```

(gdb) disas main
Dump of assembler code for function main:
0x0804838e <main+0>: push %ebp
0x0804838f <main+1>: mov %esp,%ebp
0x08048391 <main+3>: sub \$0x8,%esp
0x08048394 <main+6>: and \$0xffffffff0,%esp
0x08048397 <main+9>: mov \$0x0,%eax
0x0804839c <main+14>: add \$0xf,%eax
0x0804839f <main+17>: add \$0xf,%eax
0x080483a2 <main+20>: shr \$0x4,%eax
0x080483a5 <main+23>: shl \$0x4,%eax
0x080483a8 <main+26>: sub %eax,%esp
0x080483aa <main+28>: movl \$0x1,0xffffffff(%ebp)
0x080483b1 <main+35>: movl \$0x2,0xffffffff(%ebp)
0x080483b8 <main+42>: sub \$0x8,%esp
0x080483bb <main+45>: pushl 0xffffffff(%ebp)
0x080483be <main+48>: pushl 0xffffffff(%ebp)
0x080483c1 <main+51>: call 0x08048368 <fn>
0x080483c6 <main+56>: add \$0x10,%esp
0x080483c9 <main+59>: leave
0x080483ca <main+60>: ret
End of assembler dump.
(gdb)

return address (points to 0x080483c6)

checks out – is the right resumption location to pick up where we left off (points to 0x080483c6)

Return address location formula: \$ebp+4 *

```
(gdb) print $esp
$3 = (void *) 0xbfed9cf0
(gdb) print $ebp
$4 = (void *) 0xbfed9cf8
(gdb) x/14 $esp
0xbfed9cf0: 0x00000004 0x00000003 0xbfed9d28 0x080483c6
0xbfed9d00: 0x00000001 0x00000002 0xbfed9d28 0x080483e6
0xbfed9d10: 0x0075eff4 0x0075eff4 0x080494c4 0x0075eff4
0xbfed9d20: 0x00000002 0x00000001
(gdb) disas main
Dump of assembler code for function main:
0x0804838e <main+0>:  push  %ebp
0x0804838f <main+1>:  mov   %esp,%ebp
0x08048391 <main+3>:  sub  $0x8,%esp
0x08048394 <main+6>:  and  $0xffffffff,%esp
0x08048397 <main+9>:  mov  $0x0,%eax
0x0804839c <main+14>: add  $0xf,%eax
0x0804839f <main+17>: add  $0xf,%eax
0x080483a2 <main+20>: shr  $0x4,%eax
0x080483a5 <main+23>: shl  $0x4,%eax
0x080483a8 <main+26>: sub  %eax,%esp
0x080483aa <main+28>: movl $0x1,0xffffffff(%ebp)
0x080483b1 <main+35>: movl $0x2,0xffffffff(%ebp)
0x080483b8 <main+42>: sub  $0x8,%esp
0x080483bb <main+45>: pushl 0xffffffff(%ebp)
0x080483be <main+48>: pushl 0xffffffff(%ebp)
0x080483c1 <main+51>: call 0x08048368 <fn>
0x080483c6 <main+56>: add  $0x10,%esp
0x080483c9 <main+59>: leave
0x080483ca <main+60>: ret
End of assembler dump.
(gdb)
```

Diagram annotations:
 - A red box highlights the return address `0xbfed9cf8` in the `$ebp` register output.
 - A red arrow points from this box to the memory address `0xbfed9d28` in the `x/14 $esp` dump, with the label `+4=0xbfed9cfc`.
 - A black arrow points to the `0x080483c6` instruction in the disassembly, labeled "return address".
 - A black arrow points to the `add $0x10,%esp` instruction in the disassembly, labeled "checks out – is the right resumption location to pick up where we left off".

*just in case you ever want to overwrite it

stack_1.c – fixed param space, but variable arg len*

```
void fn(char *a) {
    char buf[10];
    strcpy(buf, a);
    printf("the end of fn\n");
}

main (int argc, char *argv[]) {
    fn(argv[1]);
    printf("the end\n");
}
```

*parameter - placeholder variable in function definition for receiving a passed value
 argument – specific value that is passed

Stack separation between argument & return address

```
root@Snorts:~/stack
(gdb) list
1 void fn(char *a) {
2   char buf(10);
3   strcpy(buf, a);
4   printf("the end of fn\n");
5 }
6
7 main (int argc, char *argv[]) {
8   fn(argv[1]);
9   printf("the end\n");
10 }
(gdb) break 3
Breakpoint 1 at 0x80483a6: file stack_1.c, line 3.
(gdb) break 4
Breakpoint 2 at 0x80483b8: file stack_1.c, line 4.
(gdb) run DDDDDDDDD
Starting program: /root/stack/stack_1 DDDDDDDDD
Breakpoint 1, fn (a=0xbfffecl7 "DDDDDDDDDD") at stack_1.c:3
3 strcpy(buf, a);
(gdb) x/12 $esp
0xbfffd530: 0x006344f8 0x00000000 0x00000000 0x00000000
0xbfffd540: 0x0804840c 0x080495e8 0xbfffd578 0x080483f6
0xbfffd550: 0xbfffecl7 0x00000000 0xbfffd578 0x08048426
(gdb) next
Breakpoint 2, fn (a=0xbfffecl7 "DDDDDDDDDD") at stack_1.c:4
4 printf("the end of fn\n");
(gdb) x/12 $esp
0xbfffd530: 0x44444444 0x44444444 0x00004444 0x00000000
0xbfffd540: 0x0804840c 0x080495e8 0xbfffd578 0x080483f6
0xbfffd550: 0xbfffecl7 0x00000000 0xbfffd578 0x08048426
(gdb) print $ebp
$1 = (void *) 0xbfffd548
```

Crafting an attack based on this

- control argument length
 - extend enough to overwrite the return address
- control argument content
 - craft meaningful code into early portion
 - calculate overwritten return address value to backpoint into that code

How?

- this exercise ends with article's page 8
- keep reading, page 9 (extracurricular)...
 - gives a real-world example
 - delivers malicious argument across a network
 - achieves a shell prompt

Please see

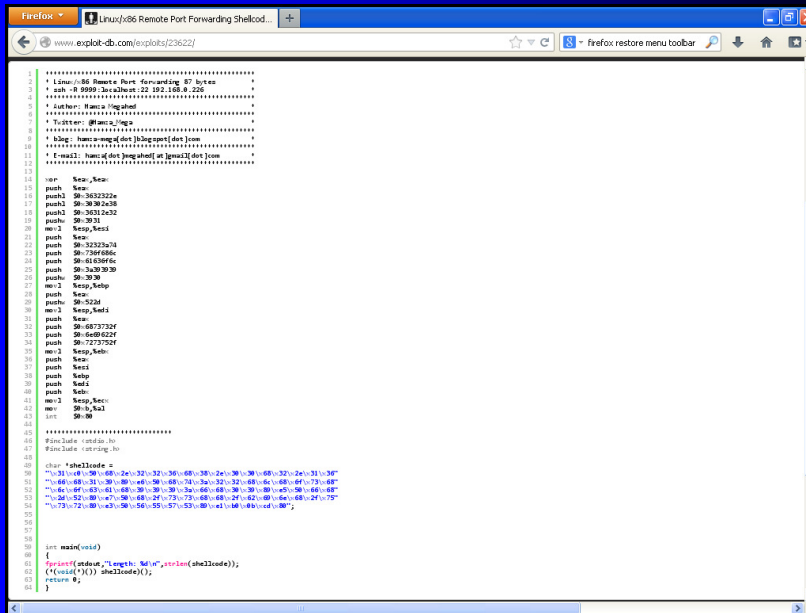
- “Overflowing the stack on Linux x/86”
 - http://www-scf.usc.edu/~csci530l/downloads/stackoverflow_en.pdf
 - originally http://sobolewscy.in5.pl/piotr/publikacje/hakin9/stackoverflow_en.pdf
- GNU debugger (gdb) documentation
 - http://sourceware.org/gdb/current/onlinedocs/gdb.html#SEC_Top

Any other code suffer this feature?

if we knew
about it, no
(it'd be fixed by now)
but we don't,
Yes (lots)



Here's one



What can be done?

- tighten compiler checks
 - this lab might not work with later gcc releases
- perform static code analysis

The screenshot shows the Cppcheck website on the left and a Fedora mailing list post on the right. The website includes navigation links (Home, Wiki, Forum, Issues, Developer info, Online Demo, Project page), a 'Download Now!' button for version 1.32 for Windows, and a list of IDE integrations (CodeBlocks, CodeLite, Eclipse, gedit, Hudson, Jenkins, Tortoise SVN, Mercurial). The mailing list post is titled 'PSA: If you are C/C++ developer, use cppcheck' and is from Rahul Sumbaram. It contains a detailed message about the author's experience with Cppcheck, listing various bugs it caught like uninitialized variables and memory leaks, and provides links to related discussions.

A 2014 "current" event

security system needed in space??

The image shows a screenshot of a Spaceflight Insider article titled 'TWO MARS SPACECRAFT CELEBRATE ONE YEAR IN MARTIAN ORBIT'. The article is dated September 25th, 2015, and features an artist's concept of the MAVEN spacecraft in orbit around Mars. The text mentions that MAVEN started its journey to Mars on Nov. 18, 2013.

9/25/15



JPL DOCUMENT D-68411

JPL Institutional Coding Standard for the C Programming Language

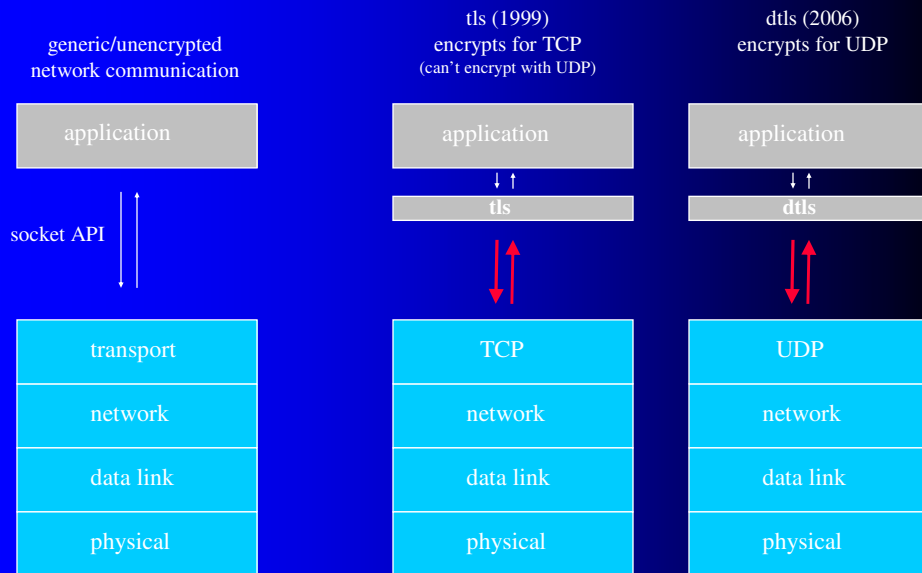
[version edited for external distribution: does not include material copyrighted by MIRA Ltd (i.e., LOC-586) and material copyrighted by the ISO (i.e., Appendix A)]
 Cleared for external distribution on 03/04/09, CL#06-0763.

Version: 1.0
 Date: March 3, 2009

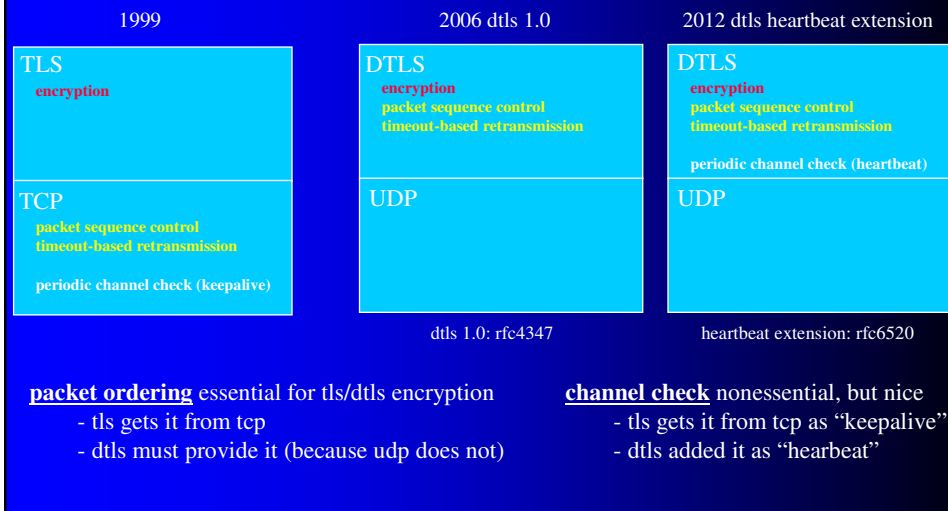
"For instance, an area of memory above the stack limit allocated to each task should be reserved as a safety margin, and filled with a fixed and uncommon bit-pattern. A health task can detect stack overflow anomalies by at regular intervals checking the presence of the bit-pattern for each task. The same principle can be used to protect against buffer overflow, or access to memory outside allocated regions. Critical parameters should similarly be protected in memory by placing safety margins and barrier patterns around them, so that access violations and data corruption can be detected more easily."

heartbleed bounds checking oversight

Encrypting: for TCP vs for UDP



Distribution of function between protocol layers



Heartbeat extension rfc6520

Internet Engineering Task Force (IETF) R. Seggelmann
 Request for Comments: 6520 M. Tuexen
 Category: Standards Track Muenster Univ. of Appl. Sciences
 ISSN: 2070-1721 M. Williams
GWhiz Arts & Sciences
February 2012

Transport Layer Security (TLS) and
 Datagram Transport Layer Security (DTLS) Heartbeat Extension

Abstract

This document describes the Heartbeat Extension for the Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols.

The Heartbeat Extension provides a new protocol for TLS/DTLS allowing the usage of keep-alive functionality without performing a renegotiation and a basis for path MTU (PMTU) discovery for DTLS.

rfc6520 excerpts

“...The Heartbeat protocol is a new protocol running on top of the Record Layer [of ssl]. The protocol itself consists of two message types: HeartbeatRequest and HeartbeatResponse....”

“The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

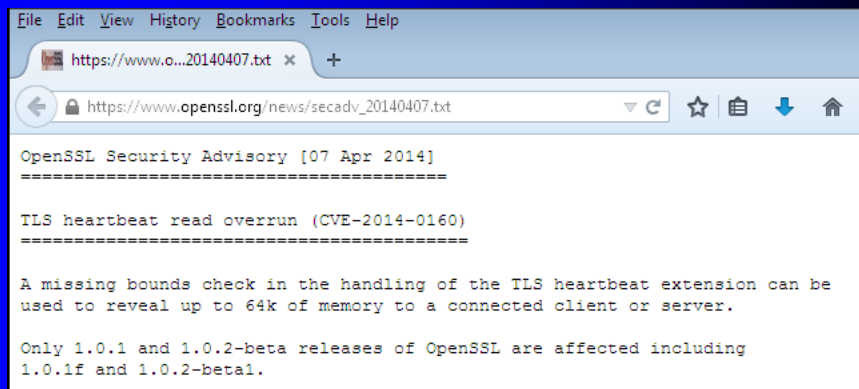
```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

“...payload: The payload consists of arbitrary content.

“...If the payload_length of a received HeartbeatMessage is too large, the received HeartbeatMessage MUST be discarded silently.

“When a HeartbeatRequest message is received ... the receiver MUST send a corresponding HeartbeatResponse message carrying an exact copy of the payload of the received HeartbeatRequest....”

Breaking news...



The effect

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



http://www.theregister.co.uk/2014/04/09/heartbleed_explained/

see also: <https://xkcd.com/1354/>

The fix

<http://pastebin.com/SPP8JVqA>

```

1. $ diff -r openssl-1.0.1f/ssl/t1_lib.c openssl-1.0.1g/ssl/t1_lib.c
2.
3.
4.
5. #ifndef OPENSLL_NO_HEARTBEATS                                #ifndef OPENSLL_NO_HEARTBEATS
6. int                                                         int
7. tls1_process_heartbeat(SSL *s)                             tls1_process_heartbeat(SSL *s)
8. {                                                         {
9.     unsigned char *p = &s->s3->rrec.data[0], *p1;           unsigned char *p = &s->s3->rrec.data[0], *p1;
10.    unsigned short hbtype;                                  unsigned short hbtype;
11.    unsigned int payload;                                    unsigned int payload;
12.    unsigned int padding = 16; /* Use minimum padding */    unsigned int padding = 16; /* Use minimum padding */
13.
14.    /* Read type and payload length first */                <
15.    hbtype = *p++;                                         <
16.    n2s(p, payload);                                       <
17.    p1 = p;                                               <
18.
19.    if (s->msg_callback)                                    if (s->msg_callback)
20.        s->msg_callback(0, s->version, TLS1_RT_HEARTB     s->msg_callback(0, s->version, TLS1_RT_HEARTB
21.        &s->s3->rrec.data[0], s->s3->rrec.len              &s->s3->rrec.data[0], s->s3->rrec.len
22.        s, s->msg_callback_arg);                          s, s->msg_callback_arg);
23.
24.                                                         >
25.                                                         >
26.                                                         >
27.                                                         >
28.                                                         >
29.                                                         >
30.                                                         >
31.                                                         >
32.                                                         >
33. if (hbtype == TLS1_HB_REQUEST)                            if (hbtype == TLS1_HB_REQUEST)

```

```

14.    /* Read type and payload length first */
15.    hbtype = *p++;
16.    n2s(p, payload);
17.    p1 = p;

```

```

25.    /* Read type and payload length first */
26.    if (1 + 2 + 16 > s->s3->rrec.length)
27.        return 0; /* silently discard */
28.    hbtype = *p++;
29.    n2s(p, payload);
30.    if (1 + 2 + payload + 16 > s->s3->rrec.length)
31.        return 0; /* silently discard per RFC 6520 se
32.    p1 = p;

```

Exploitation in the lab

attacker's terminal window, viewing victim memory fetched from victim by heartbleed

send something across to victim, via this form,

Fedora 19 with openssl 1.0.1e here, for CS530

Press the button to invoke the action. The action is to run the "haiku" script.

Enter (or make up) a haiku author: EAZZZZZZZZZY2SEEEEE

that would be recognizable in his memory, if ever seen there.

attacker's browser, viewing page sent from web server on victim (192.168.1.135)

```

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary (openssl_heartbleed) >

```

Don't let browser accept revoked certs require affirmative non-rev check

- server sites remediate by 1-updating OpenSSL 2-revoking certificates (to prevent site impersonation via possible previous heartbleed-exfiltrated private keys)
- only meaningful if client (you!) does his part, i.e., checks for the revocation and honors it
- turn it on in your browser if it supports it
- Firefox does; phones' browsers probably don't

check these

What is mod_nss?

mod_nss is an SSL provider derived from the mod_ssl module for the Apache web server that uses the Network Security Services (NSS) libraries. We started with mod_ssl and replaced the OpenSSL calls with NSS calls.

Why use NSS instead of OpenSSL?

Use what is best for your needs.

What features does mod_nss provide?

Options

General | Data Choices | Network | Update | Certificates

When a server requests my personal certificate:

Select one automatically: Ask me every time

View Certificates | Validation | Security Devices

Certificate Validation

Use the Online Certificate Status Protocol (OCSP) to confirm the current validity of certificates

When an OCSP server connection fails, treat the certificate as invalid

<http://news.netcraft.com/archives/2014/04/24/certificate-revocation-why-browsers-remain-affected-by-heartbleed.html>

My faq

Q. Is this an exploitation of the SSL/TLS protocol?

A. No, it's an exploitation of the OpenSSL implementation of it.

Q. Are there other implementations?

A. Yes for example Mozilla's NSS (Network Security Services) or GnuTLS

Q. How widespread among websites is the use of OpenSSL to provide TLS?

A. Maybe 17.5% of them use OpenSSL for that

<http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html>

Q. Does Apache use OpenSSL for SSL?

A. Yes, if it uses mod_ssl for ssl. But it could use mod_nss and thus NSS's ssl. Usually it installs with mod_ssl by default.

http://directory.fedoraproject.org/wiki/Mod_nss#What_is_mod_nss.3F

Information sources

- code analyses:
 - <http://blog.existentialize.com/diagnosis-of-the-openssl-heartbleed-bug.html>
 - http://www.theregister.co.uk/2014/04/09/heartbleed_explained/
- Security Now podcast - "How the Heartbleeds"
 - audio: <https://media.grc.com/sn/sn-450-lq.mp3>
 - transcript: <https://www.grc.com/sn/sn-450.pdf>
 - shownotes: <https://www.grc.com/sn/sn-450-notes.pdf>
- <http://heartbleed.com>

C language sign extension bug



Case study - a longstanding bug

A hole in crypt_blowfish

By Jake Edge
June 22, 2011

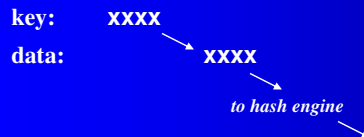
A longstanding bug that was recently found in the `crypt_blowfish` password hashing library highlights the problems that can occur when a bug is found in a widely used low-level library. Because `crypt_blowfish` has been around for so long (this bug is said to go back to 1998 or possibly 1997), it has been used by various other packages (PHP for example) as well as some Linux distributions. The security impact is not likely to be huge, because it only affects passwords with somewhat uncommon characteristics, but the impact on those who have stored hashed passwords generated using the library may be a bit more painful.

- introduced late 90s, noticed then but overlooked ever since
- rediscovered while testing John the Ripper in June 2011
- in the `crypt_blowfish` library
- freely, admirably, immediately admitted, documented, and fixed by the library's author (who is also author of John the Ripper)

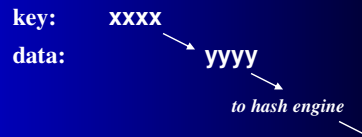
What was the bug?

- 4 bytes of key/password needed to be hashed
 - passed to a char-type parameter variable “key”
 - transferred to long(4-byte)-type variable “data”
- the transfer went bad
 - “data” ended with value *different* from “key”
- resulting hash *not* that of the password

Intent:



Event:



Underlying background issues

- binary signed integer representation
- the bitwise OR operation

Representing signed integers (two's complement method)

Split range in half
 - low value half for zero and positive
 - high value half for negative

	value unsigned		value signed
0000	0	0000	0
0001	1	0001	1
0010	2	0010	2
0011	3	0011	3
0100	4	0100	4
0101	5	0101	5
0110	6	0110	6
0111	7	0111	7
1000	8	1000	-8
1001	9	1001	-7
1010	10	1010	-6
1011	11	1011	-5
1100	12	1100	-4
1101	13	1101	-3
1110	14	1110	-2
1111	15	1111	-1

zero and positive
negative

Widening signed integers "extension" and "sign extension"

2 bits		3 bits		4 bits		8 bits	
unsigned	signed	unsigned	signed	unsigned	signed	unsigned	signed
0	00	0	000	0	0000	0	00000000
1	01	1	001	1	0001	1	00000001
2	10	2	010	2	0010	2	00000010
3	11	3	011	3	0011	3	00000011
		4	100	4	0100	4	00000100
		5	101	5	0101	5	.
		6	110	6	0110	6	.
		7	111	7	0111	7	.
				8	1000	8	125 01111101
				9	1001	9	126 01111110
				10	1010	10	127 01111111
				11	1011	11	128 10000000
				12	1100	12	129 10000001
				13	1101	13	130 10000010
				14	1110	14	131 10000011
				15	1111	15	.
							.
						251	11111011 -5
						252	11111100 -4
						253	11111101 -3
						254	11111110 -2
						255	11111111 -1

To preserve same value, pad left with:

- if positive, **0's** (e.g. +3)
- if negative, **1's** (e.g. -2)

Background: OR operation

- an operation
 - operands (input): 2 bits
 - result (output): 1 bit
- ORing a bit with 0 yields (preserves) that bit
 - $0 \text{ OR } 0 = 0$
 - $1 \text{ OR } 0 = 1$
- ORing a bit with 1 yields 1 unconditionally
 - $0 \text{ OR } 1 = 1$
 - $1 \text{ OR } 1 = 1$

ORing 2 bytes with each other

- no such thing
 - OR is an operation for pairs of bits only
 - not pairs of aces, nor deuces, nor bytes
- “ORing bytes” signifies 8 normal (bitwise) ORs, collectively

“ORing 2 bytes” == 8 of these →

ORing words requires 2 words, of equal length,
to enable ORing their bits

0	1	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	0	1	1	0	1	1

The offending code

I investigated this further, and it turned out to be a source code implementation error.

There is an implementation error in published Blowfish Code. The program chokes on the commented "choke" statement, below:

```

bfinit(char *key,int keybytes)
{
    unsigned long data;
    ...
    j=0;
    ...
    data=0;
    for (k=0;k<4;k++){
        data=(data<<8)|key[j];/* choke*/
        j++;
        if(j==keybytes)
            j=0;
    }
    ...
}

```

It chokes whenever the most significant bit of key[j] is a '1'. For example, if key[j]=0x80, key[j], a signed char, is sign extended to 0xfffff80 before it is ORed with data. For example, when:

```

(j<0x3)==0x3 (that is j=0x3,0x7,0xf, etc.)
--and-
(key[j]&0x80)==0x80 (or when k[j]=0x80,0x81,etc.)

```

data=0xfffff80 (0xfffff81,etc.) upon exit from the above "for(k...)" loop. ORing all of these 1's into data effectively wipes out 3/4 of the key characters! (that is, 3/4 of the key characters are known to be set to 1 when the 4th key byte to be ORed into data has a 1 in the most significant bit.) For a randomly selected 32-bit key, there is a 50% chance that 3/4 of the key could be considered as all '1's, even if they weren't that way to begin with.

offending code loads key's 4 bytes into data

	key[0]	key[1]	key[2]	key[3]
key:	00010001	00100010	01000100	10001000
data:	10101010	10111011	11001100	11011101

by doing this:

data=(data<<8) | key[j]

4 times

←
initial value is
random/garbage

$$\text{data} = (\text{data} \ll 8) \mid \text{key}[j]$$

Observation

- data is 4 bytes wide
- key[j] is only 1 byte
- key[j] is *too short* to OR with data
- so pad (“extend”) it by 24 bits on the left

Operation

1. shift ‘data’ 8 bits left
left byte disappears
right byte zero-filled
2. left-pad key[j] with 24 zeros
3. OR them together
extended key[j]’s zeros preserve data’s leftmost 3 bytes
data’s zeros preserve extended key[j]’s rightmost byte
4. assign result to data

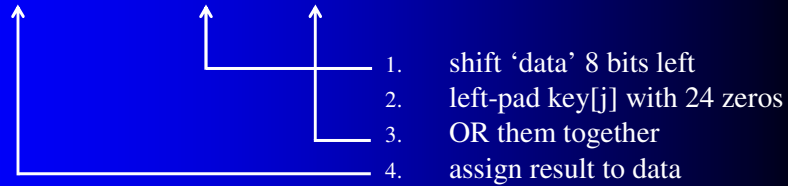
Intended operation of algorithm

initial “key”	00010001	00100010	01000100	10001000
initial “data”	10101010	10111011	11001100	11011101

evolution of “data”:	<i>iteration 1</i>	shift	10111011 11001100 11011101 <i>00000000</i>
		extend	<i>00000000</i> 00000000 00000000 00010001
		or	10111011 11001100 11011101 00010001
00000’s from extend 00000’s from shift	<i>iteration 2</i>	shift	11001100 11011101 00010001 <i>00000000</i>
		extend	<i>00000000</i> 00000000 00000000 00100010
		or	11001100 11011101 00010001 00100010
	<i>iteration 3</i>	shift	11011101 00010001 00100010 <i>00000000</i>
		extend	<i>00000000</i> 00000000 00000000 01000100
		or	11011101 00010001 00100010 01000100
final “data” holds initial “key” →	<i>iteration 4</i>	shift	00010001 00100010 01000100 <i>00000000</i>
		extend	<i>00000000</i> 00000000 00000000 10001000
		or	00010001 00100010 01000100 10001000

step 2: implicit, *lexically invisible*

$\text{data} = (\text{data} \ll 8) \mid \text{key}[j]$



Actual operation of algorithm

initial "key"	00010001	00100010	01000100	10001000
initial "data"	10101010	10111011	11001100	11011101

evolution of "data":					
	<i>iteration 1</i>	shift	10111011 11001100 11011101	00000000	
		extend	00000000 00000000 00000000	00010001	
		or	10111011 11001100 11011101	00010001	
	<i>iteration 2</i>	shift	11001100 11011101 00010001	00000000	
		extend	00000000 00000000 00000000	00100010	
		or	11001100 11011101 00010001	00100010	
	<i>iteration 3</i>	shift	11011101 00010001 00100010	00000000	
		extend	00000000 00000000 00000000	01000100	
		or	11011101 00010001 00100010	01000100	
	<i>iteration 4</i>	shift	00010001 00100010 01000100	00000000	
		extend	11111111 11111111 11111111	10001000	
		or	11111111 11111111 11111111	10001000	

final "data" does *not* hold initial "key" →

problem arises here

A code embodiment

```
[david@emach4 ~]$ cat -n sign-extension-bug.c
1 #include <stdio.h>
2 char key[4] = { 0x11, 0x22, 0x44, 0x88 };
3
4 int main()
5 {
6     int j;
7     unsigned long data;
8     printf("\nEvolution of 'data' as bytes from 'key' are in-shifted from right\n");
9     printf("letters should be progressively replaced by numbers, one byte at a time\n\n");
10
11     data=0xaabbccdd;
12     printf("Initial value of 'data': %08x\n\n", data );
13
14     printf("4 rounds: slide existing bits one byte left, then OR incoming byte into rightmost\n");
15     for (j=0;j<4;j++)
16     {
17         data=(data<<8)|key[j];
18         printf("-----> %08x\n", data );
19         sleep(3);
20     }
21 }
[david@emach4 ~]$ ./sign-extension-bug
Evolution of 'data' as bytes from 'key' are in-shifted from right
letters should be progressively replaced by numbers, one byte at a time
Initial value of 'data': aabbccdd
4 rounds: slide existing bits one byte left, then OR incoming byte into rightmost
-----> bbccdd11
-----> ccdd1122
-----> dd112244
-----> ffffffff ← just-loaded "112244" have been clobbered!
[david@emach4 ~]$ ./sign-extension-bug-fixed ← a fixed version
Evolution of 'data' as bytes from 'key' are in-shifted from right
letters should be progressively replaced by numbers, one byte at a time
Initial value of 'data': aabbccdd
4 rounds: slide existing bits one byte left, then OR incoming byte into rightmost
-----> bbccdd11
-----> ccdd1122
-----> dd112244
-----> 11224488 ← just-loaded "112244" are preserved unmolested
[david@emach4 ~]$
```

Why is this happening?

- because C by default treats char type as signed
- So hex 88 (= bin 10001000) treated is as if
 - decimal -120
 - not decimal 136
- extend from 1 to 4 bytes keeping -120 value needs
 - left-pad with 1
 - not left-pad with 0
- alters the subsequent OR operation

What effects?

- replaces many password characters with FF
- promotes FF to ranks of high predictability
 - along with natural language words
 - along with birthday strings
 - along with pets' names
- eases intelligent brute force cracking task
 - FF-heavy guesses are now rewarding to try a lot

What effects?

"I am wondering ... why I am getting different hashes...."

"...it means we have incorrect (incompatible with OpenBSD's) hashes in the wild..."

"John the Ripper and crypt_blowfish developer Alexander Peslyak (aka Solar Designer) analyzed the effects of the bug and found that some password pairs would hash to the same value with only minimal differences (e.g. "ab&" hashed to the same value as "£"), which would make password cracking easier. A further analysis shows that some characters appearing just before one with the high bit set may be effectively ignored when calculating the hash. That would mean that a simpler password than that given by the user could be used and would still be considered valid—a significant weakening of the user's password.

"It should be noted that Solar Designer has been very forthcoming with details of the problem and its effects."

See: <http://lwn.net/Articles/448699/>
<http://lwn.net/Articles/448723/>
<http://lwn.net/Articles/448725/>

Observations

- a C-language-specific problem
- assembler would be immune
 - left-pad/extension is lexically explicit/inescapable
 - MOVZX, “move zero extend” – use 0s, versus
 - MOVSX, “move sign extend” – use 1s
- will not affect ascii password characters
 - they fall in the “positive” range of signed representation
 - none have the offending, triggering leading 1-bit
 - but not all passwords/keys are human generated ascii

Information sources

- <http://lwn.net/Articles/448699/>
- <http://lwn.net/Articles/448723/>
- <http://lwn.net/Articles/448725/>
- <http://www.schneier.com/blowfish-bug.txt>
- Security Now podcast - “Anatomy of a Security Mistake”
 - audio: <http://media.grc.com/SN/sn-311-lq.mp3>
 - transcript: <http://www.grc.com/sn/sn-311.txt>