Google pays $14,000 for high-risk Chrome security holes

By Ryan Naraine | January 14, 2011, 9:52am PST

Summary
Google has shelled out more than $14,000 in rewards for critical and high-risk vulnerabilities affecting its flagship Chrome web browser.

The latest Google Chrome 8.0.552.237, available for all platforms, patches a total of 16 documented vulnerabilities, including one critical bug for which Google paid the first elite $3133.7 award to researcher Sergey Glazunov.

“Critical bugs are harder to come by in Chrome, but Sergey has done it,” says Google’s Jerome Kersey. “Sergey also collects a $1337 reward and several other rewards at the same time, so congratulations Sergey!,” he added.
Outline

Process Layout

Function Calls

The Heap
**Process Layout in Memory**

- **Stack**
  - grows towards *decreasing* addresses.
  - is initialized at *run-time*.

- **Heap** and **BSS** sections
  - grow towards *increasing* addresses.
  - are initialized at *run-time*.

- **Data** section
  - is initialized at *compile-time*.

- **Text** section
  - holds the program instructions (read-only).
Process Layout in Memory

- **Stack**
  - grows towards *decreasing* addresses.
  - is initialized at *run-time*.

- **Heap** and **BSS** sections
  - grow towards *increasing* addresses.
  - are initialized at *run-time*.

- **Data** section
  - is initialized at *compile-time*.

- **Text** section
  - holds the program instructions (read-only).

---

```
Stack
    ▼
    dynamic
growth

Heap
  ▼
  BSS
  ▼
  Data
  ▼
  Text
  ▼

0xc0000000
high address

0x08048000
low address

uninitialized variables
initialized variables
```
Outline

Process Layout

Function Calls

The Heap
IA-32

Registers

EAX  Accumulator for operands and results data
EBX  Pointer to data in the DS segment
ECX  Counter for string and loop operations
EDX  I/O pointer
ESI  Source pointer for string operations
EDI  Destination pointer for string operations
EBP  Frame pointer
ESP  Stack pointer

Terminology

SFP  saved frame pointer: saved %ebp on the stack
OFP  old frame pointer: old %ebp from the previous stack frame
RIP  return instruction pointer: return address on the stack
### Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>Accumulator for operands and results data</td>
</tr>
<tr>
<td>EBX</td>
<td>Pointer to data in the DS segment</td>
</tr>
<tr>
<td>ECX</td>
<td>Counter for string and loop operations</td>
</tr>
<tr>
<td>EDX</td>
<td>I/O pointer</td>
</tr>
<tr>
<td>ESI</td>
<td>Source pointer for string operations</td>
</tr>
<tr>
<td>EDI</td>
<td>Destination pointer for string operations</td>
</tr>
<tr>
<td>EBP</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>ESP</td>
<td>Stack pointer</td>
</tr>
</tbody>
</table>

### Terminology

- **SFP** saved frame pointer: saved %ebp on the stack
- **OFP** old frame pointer: old %ebp from the previous stack frame
- **RIP** return instruction pointer: return address on the stack
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];

    bar[0] = 'A';
    qux[0] = 0x2a;
}

int main(void)
{
    int i = 1;
    foo(1, 2, 3);

    return 0;
}
main:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    movl $1, -4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12, %esp
    xorl %eax, %eax
    leave
    ret
main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  movl $1,-4(%ebp)
  pushl $3
  pushl $2
  pushl $1
  call foo
  addl $12,%esp
  xorl %eax,%eax
  leave
  ret
Function Calls in Assembler

main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
Function Calls in Assembler

main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  movl $1,-4(%ebp)
  pushl $3
  pushl $2
  pushl $1
  call foo
  addl $12,%esp
  xorl %eax,%eax
  leave
  ret
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
main:
  pushl  %ebp
  movl  %esp,%ebp
  subl  $4,%esp
  movl  $1,-4(%ebp)
  pushl  $3
  pushl  $2
  pushl  $1
  call  foo
  addl  $12,%esp
  xorl  %eax,%eax
  leave
  ret
Function Calls in Assembler

main:

pushl %ebp
movl %esp,%ebp
subl $4,%esp
movl $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call foo
addl $12,%esp
xorl %eax,%eax
leave
ret
foo:
    pushl  %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
foo:

pushl %ebp
movl %esp,%ebp
subl $12,%esp
movl $65,-8(%ebp)
movb $66,-12(%ebp)
leave
ret
foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
Function Calls in Assembler

foo:
  pushl %ebp
  movl %esp,%ebp
  subl $12,%esp
  movl $65,-8(%ebp)
  movb $66,-12(%ebp)
  leave
  ret
Function Calls in Assembler

foo:
  pushl %ebp
  movl %esp,%ebp
  subl $12,%esp
  movl $65,-8(%ebp)
  movb $66,-12(%ebp)
  leave
  ret
foo:
    pushl  %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
Function Calls in Assembler

foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret

leave:
    movl %ebp,%esp
    popl %ebp
foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
Function Calls in Assembler

main:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>pushl %ebp</td>
<td></td>
</tr>
<tr>
<td>movl %esp,%ebp</td>
<td></td>
</tr>
<tr>
<td>subl $4,%esp</td>
<td></td>
</tr>
<tr>
<td>movl $1,-4(%ebp)</td>
<td></td>
</tr>
<tr>
<td>pushl $3</td>
<td></td>
</tr>
<tr>
<td>pushl $2</td>
<td></td>
</tr>
<tr>
<td>pushl $1</td>
<td></td>
</tr>
<tr>
<td>call foo</td>
<td></td>
</tr>
<tr>
<td>addl $12,%esp</td>
<td></td>
</tr>
<tr>
<td>xorl %eax,%eax</td>
<td></td>
</tr>
<tr>
<td>leave</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- ebp
- esp
- rip
- sfp
- ofp
Function Calls in Assembler

```assembly
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
```
Function Calls in Assembler

main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  movl $1,-4(%ebp)
  pushl $3
  pushl $2
  pushl $1
  call foo
  addl $12,%esp
  xorl %eax,%eax
  leave
  ret
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
Function Calls in Assembler

main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
Outline

Process Layout

Function Calls

The Heap
The heap is "[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order." [WJN+95]
The heap is "[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order." [WJN+95]

- ANSI-C functions malloc() and friends are used to manage the heap (glibc uses ptmalloc).
The heap is "[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order." [WJN+95]

- ANSI-C functions malloc() and friends are used to manage the heap (glibc uses ptmalloc).
- Heap memory is organized in chunks that can be allocated, freed, merged, etc.
The heap is "[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order."

[WJN+95]

- ANSI-C functions `malloc()` and friends are used to manage the heap (glibc uses `ptmalloc`).
- Heap memory is organized in **chunks** that can be allocated, freed, merged, etc.
- **Boundary Tags** contain meta information about chunks (size, previous/next pointer, etc.)
  - stored both in the front and end of each chunk.
  - makes consolidating fragmented chunks into bigger chunks very fast.
Chunks in Memory

- allocated chunk
- allocated chunk
- allocated chunk
- free chunk
- allocated chunk
- free chunk
- allocated chunk
- wilderness chunk

The Heap
Understanding Heap Management

**Boundary Tags**

- **prev_size**: size of previous chunk (if free).
- **size**: size in bytes, including overhead.
- **PREV_INUSE**: Status bit; set if previous chunk is allocated.
- **fd/bk**: forward/backward pointer for double links (if free).

The Heap 12 / 15
Understanding Heap Management

Boundary Tags

- **prev_size**: size of previous chunk (if free).
- **size**: size in bytes, including overhead.
- **PREV_INUSE**: Status bit; set if previous chunk is allocated.
- **fd/bk**: *forward/backward pointer* for double links (if free).

![Diagram of heap management](image)

Free chunk

- prev_size
- size
- fd
- bk
- data
- prev_size >= 0

Allocated chunk

- prev_size
- size
- fd
- bk
- data
- prev_size >= 0

Managing Free Chunks

- Free chunks of similar size are grouped into bins.
- fd/bk pointers to navigate through double links.
Understanding Heap Management

Boundary Tags

- `prev_size`: size of previous chunk (if free).
- `size`: size in bytes, including overhead.
- `PREV_INUSE`: Status bit; set if previous chunk is allocated.
- `fd/bk`: `forward/backward pointer` for double links (if free).

Managing Free Chunks

- Free chunks of similar size are grouped into `bins`.
- `fd/bk` pointers to navigate through double links.
Removing Chunks from a Bin: `unlink()`

```c
#define unlink(P, BK, FD) 
{ 
    BK = P->bk; 
    FD = P->fd; 
    FD->bk = BK; 
    BK->fd = FD; 
}
```
Removing Chunks from a Bin: `unlink()`

```c
#define unlink(P, BK, FD) 
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```
Removing Chunks from a Bin: \texttt{unlink()}

```c
#define unlink(P, BK, FD) 
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```
Removing Chunks from a Bin: `unlink()`

```c
#define unlink(P, BK, FD) 
{ 
    BK = P->bk; 
    FD = P->fd; 
    FD->bk = BK; 
    BK->fd = FD; 
}
```
Removing Chunks from a Bin: `unlink()`

```c
#define unlink(P, BK, FD) 
{ 
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```

FD + 12 = BK
Removing Chunks from a Bin: unlink()

```c
#define unlink(P, BK, FD) 
{ 
    BK = P->bk; 
    FD = P->fd; 
    FD->bk = BK; 
    BK->fd = FD; 
}
```

The Heap 13 / 15
Removing Chunks from a Bin: `unlink()`

```c
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```
Paul R. Wilson and Mark S. Johnstone and Michael Neely and David Boles.
Dynamic Storage Allocation: A Survey and Critical Review. 
### IA32 Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>movl Src, Dest</td>
<td>Dest = Src</td>
</tr>
<tr>
<td>addl Src, Dest</td>
<td>Dest = Dest + Src</td>
</tr>
<tr>
<td>subl Src, Dest</td>
<td>Dest = Dest - Src</td>
</tr>
<tr>
<td>imull Src, Dest</td>
<td>Dest = Dest * Src</td>
</tr>
<tr>
<td>sall Src, Dest</td>
<td>Dest = Dest &lt;&lt; Src</td>
</tr>
<tr>
<td>sarl Src, Dest</td>
<td>Dest = Dest &gt;&gt; Src</td>
</tr>
<tr>
<td>xorl Src, Dest</td>
<td>Dest = Dest ^ Src</td>
</tr>
<tr>
<td>andl Src, Dest</td>
<td>Dest = Dest &amp; Src</td>
</tr>
<tr>
<td>orl Src, Dest</td>
<td>Dest = Dest</td>
</tr>
<tr>
<td>incl Dest</td>
<td>Dest = Dest + 1</td>
</tr>
<tr>
<td>decl Dest</td>
<td>Dest = Dest - 1</td>
</tr>
<tr>
<td>negl Dest</td>
<td>Dest = - Dest</td>
</tr>
<tr>
<td>notl Dest</td>
<td>Dest = ~ Dest</td>
</tr>
<tr>
<td>leal Src, Dest</td>
<td>Dest = address of Src</td>
</tr>
<tr>
<td>cmpl Src2, Src1</td>
<td>Sets CCs Src1 - Src2</td>
</tr>
<tr>
<td>testl Src2, Src1</td>
<td>Sets CCs Src1 &amp; Src2</td>
</tr>
<tr>
<td>jmp label</td>
<td>jump</td>
</tr>
<tr>
<td>je label</td>
<td>jump equal</td>
</tr>
<tr>
<td>jne label</td>
<td>jump not equal</td>
</tr>
<tr>
<td>js label</td>
<td>jump negative</td>
</tr>
<tr>
<td>jns label</td>
<td>jump non-negative</td>
</tr>
<tr>
<td>jg label</td>
<td>jump greater (signed)</td>
</tr>
<tr>
<td>jge label</td>
<td>jump greater or equal (signed)</td>
</tr>
<tr>
<td>jl label</td>
<td>jump less (signed)</td>
</tr>
<tr>
<td>jle label</td>
<td>jump less or equal (signed)</td>
</tr>
<tr>
<td>ja label</td>
<td>jump above (unsigned)</td>
</tr>
<tr>
<td>jb label</td>
<td>jump below (unsigned)</td>
</tr>
</tbody>
</table>

### Addressing Modes

#### Immediate

<table>
<thead>
<tr>
<th>$val</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Normal

<table>
<thead>
<tr>
<th>(R)</th>
<th>Mem[Reg[R]]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register R specifies memory address</td>
<td></td>
</tr>
<tr>
<td>movl (%ecx), %eax</td>
<td></td>
</tr>
</tbody>
</table>

#### Displacement

<table>
<thead>
<tr>
<th>D(R)</th>
<th>Mem[Reg[R]+D]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register R specifies start of memory region</td>
<td></td>
</tr>
<tr>
<td>Constant displacement D specifies offset</td>
<td></td>
</tr>
<tr>
<td>movl 8(%ebp), %edx</td>
<td></td>
</tr>
</tbody>
</table>

#### Indexed

<table>
<thead>
<tr>
<th>D(Rb, Ri, S)</th>
<th>Mem[Reg[Rb]+S*Reg[Ri]+D]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant “displacement” 1, 2, or 4 bytes</td>
<td></td>
</tr>
<tr>
<td>Register Rb specifies start of memory region</td>
<td></td>
</tr>
<tr>
<td>Constant displacement D specifies offset</td>
<td></td>
</tr>
<tr>
<td>movl 8(%ebp), %edx</td>
<td></td>
</tr>
</tbody>
</table>

### Condition Codes

- **CF**: Carry Flag
- **ZF**: Zero Flag
- **SF**: Sign Flag
- **OF**: Overflow Flag

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td></td>
</tr>
<tr>
<td>%ecx</td>
<td></td>
</tr>
<tr>
<td>%ebx</td>
<td></td>
</tr>
<tr>
<td>%esi</td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td></td>
</tr>
</tbody>
</table>