Name: ____________________________________________

SID: _____________________________________________

- You may not use any reference materials during this exam.
- Electronic devices, including calculators, cell phones, mp3 players, and laptops are all prohibited.
- You may not use your own scratch paper. The exam has plenty and you can ask for more if needed.
- You may not leave the classroom once the exam has been distributed.
- Communicating with other students in any way is prohibited.

**Academic Honesty:** I understand that if I cheat on this exam in any way, I will receive the maximum possible penalty, including an F in this course.

Signature: _______________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/10</td>
</tr>
<tr>
<td>2</td>
<td>/10</td>
</tr>
<tr>
<td>3</td>
<td>/10</td>
</tr>
<tr>
<td>4</td>
<td>/10</td>
</tr>
<tr>
<td>5</td>
<td>/10</td>
</tr>
<tr>
<td>6</td>
<td>/10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>/60</strong></td>
</tr>
</tbody>
</table>
1 Syscalls from native client untrusted modules

(10 points) Suppose we want to allow Native Client modules to make some system calls, such as \texttt{read(int fd, void *buf, int size)}, directly. Suppose system calls are made by putting the arguments to \texttt{read} in registers $r0$, $r1$, and $r2$, respectively, and then executing a \texttt{syscall} instruction, as in

\begin{verbatim}
mv $r0, 0 // fd = 0
mv $r1, <address of buf>
mv $r2, 5 // read 5 bytes
syscall SYSCALL_NR_READ
\end{verbatim}

What safety checks should be added to this code to make it safe for use in a Native Client module? Give your answer as a snippet of assembly code.

Solution

\begin{verbatim}
; dest, src
mv $r0, <fd>
mv $r1, <buf>
mv $r2, <len>

and $r1, $r1, REGION_MASK
and $r3, $pc, ~REGION_MASK
or $r1, $r1, $r3 ; $r1 points inside the region

or $r3, $r1, REGION_MASK ; $r3: upper bound
add $r4, $r1, $r2
bnle ERR, $r4, $r3

syscall SYSCALL_NR_READ
\end{verbatim}

Grading criteria

Main idea: check bounds of \texttt{buf} inside region. +4 pts.
Assembly code of check the beginning of the \texttt{buf} or force \texttt{buf} inside region. +3 pts
Assembly code of check \texttt{buf}+\texttt{len} inside region. +3 pts
(Your assembly code may use \texttt{shl}, \texttt{shr} or \texttt{cmp})
Checks on \texttt{fd} (not the file) or \texttt{sys call}, 1pt each, up to 10 pts.
Pseudo assembly code without showing details of how to get region bounds, only +1-2 pts
2 Whole-system taint tracking

Run-time taint-tracking systems, such as TaintDroid, track the taintedness of data within one process, but they cannot track taintedness of data that is passed from one process to another. For example, consider a web application that reads tainted data from the network, stores that data in a database server, and later retrieves that data from the database server for further processing, such as generating a web page. Current taint-tracking systems cannot track the taintedness of data as it passes between the web application server and the database server.

Give one advantage and one disadvantage of each of the following possible solutions to this problem (2.5 points each):

- Forbid tainted data to leave any process.
  
  **Advantages:**
  - safe, more conservative
  
  **Disadvantages:**
  - requiring modifying program to sanitize all the data sent to DB
  - breaks program
  - not backward compatible

- Mark all data from the database server as tainted.
  
  **Advantages:**
  - safe, more conservative
  
  **Disadvantages:**
  - as above
  - May unnecessarily taint safe data in DB.

- Extend the operating system, file system, network, etc. to carry taint information for data as it passes from one process to another, to files, and even over the network.
  
  **Advantages:**
  - precise
  - in theory it should be backward compatible
  
  **Disadvantages:**
  - Implementation difficulty
  - Runtime Overhead

- Whenever data leaves the process, the taint-tracking system encodes the data, along with its taint bits, as a base64 string. When the data returns to the process, it is decoded and its taint bits restored. Assume this process can be applied selectively (e.g. to data passed to the database server, but not data sent to clients).
  
  **Advantages:**
  - precise
- medium compatibility
- medium overhead

Disadvantages:

- may break processing in DB if data is opaque in DB
- could break compatibility
- runtime overhead

Grading criteria

Any answer close to one of the advantages/disadvantages, or other reasonable answer, is OK. Two interpretations: does DB decode or treat encoded data as opaque? Both interpretations are valid.
3 Compiler support for limiting data lifetime

Sometimes, programmers attempt to limit data lifetime by explicitly erasing sensitive items when they are no longer needed. However, compilers can thwart the programmer’s efforts, especially when they optimize code.

(5 points) For example, explain what can go wrong with the following:

```c
memset(key, 0, KEY_SIZE);
free(key);
```

Compiler may optimize away the `memset`

Grading criteria

This is the only correct answer. Any other answers do not receive any partial points.

(5 points) On the other hand, programmers may not be able to clear some data. For example, suppose a programmer attempts to erase sensitive data on the stack by zeroing all local variables before returning from any function. Name two locations on the stack, but outside any active stack frame, that might continue to hold sensitive data after a function returns.

- temporaries +2.5pts
- parameters +2.5pts

following answers get 1pt for each

- return address
- bsp/sp/fp
- env vars
- program args
4 Modern notions of principals

Desktop operating systems make access-control decisions based on process’ user IDs. Mobile operating systems make access-control decisions based on process’ application IDs. Web browsers make access-control decisions based on scripts’ origins.

(5 points) Consider a word processor editing two documents – one written by the user and one downloaded from evil.com. What’s the danger here?

Solution

If the evil.com docu exploits a bug in the word processor, it can access and corrupt all data accessible to the word processor, including all the user’s other files on traditional OSs.

Grading Criteria

• both traditional OS and mobile OS are valid consideration.

• simple consequence description such as ”evil docu can access the memory/clipboard of the user’s docu” may receive partial credits. If it does not point out the reason why the consequence occurs (the same uid, the shared word editor process may subject to exploits,etc), it loses 2pts.

(5 points) Consider an OS that attaches three IDs, (userID, appID, originID), to each process and object (such as files) in the system. What should be the default access control rule? What should be the access control rule for a process that has not yet loaded any data? How should a process’ IDs change when it loads data?

Solution

Process: \((userID_p, appID_p, orID_p)\)
Object: \((userID_o, appID_o, orID_o)\).

Access allowed: \(userID_p = userID_o \land appID_p = appID_o \land (orID_p = orID_o \lor orID_p = \perp)\)

Initial: \(orID_p = \perp\)
when process loads data: \(orID_p = orID_o\)

Grading Criteria

not necessarily \(orID_p = \perp\), \(orID_p = null\) is acceptable too.
5 Deputy and incorrect annotations

Consider the following code:

```c
void memzero(char * COUNT(N) p, int size)
{
    int i;
    for (i = 0; i < size; i++) {
        (1) assert(p <= p+i && p+i < p+N)
            (or assert(0 <= i < N))
        p[i] = 0;
    }
}
```

```c
void foo(void)
{
    char A[10];

    (2) assert(N(10) <= 10)
    memzero(A, 10);
}
```

(4 points) Add the run-time checks that Deputy will insert to the code above.

Grading criteria
- Two assertions, each 2 pts.
- For assertion(2), any expression of "size or count N ≤ length of array A" earns full points

(3 points) Suppose the programmer labels the COUNT of p as size/2 (i.e. replace N with size/2). What will happen? Will there be a security bug?

Solution
Assertion (1) will fail. There will not be a security bug.

Grading Criteria
+2.5 for any answer of assertion(1) will fail. +0.5 for no security bug only if the answer points out because assertion(1) will fail. The Answer that at the compiler time the therom prover may catch the bug early may receive points
(3 points) Suppose the programmer labels the COUNT of p as 2*size (i.e. replace N with 2*size). What will happen? Will there be a security bug?

Solution

Assertion (2) will fail. There will not be a security bug.

Grading Criteria

+2.5 for any answer of assertion(2) will fail. +0.5 for no security bug only if the answer points out because assertion(2) will fail. Any analysis based on assertion (1) receives no points.
6 Capability hardware and interrupts

If the hardware supports fat pointers, then we can run all processes in a single address space, so we would no longer need virtual memory hardware to isolate the memory of separate processes.

(5 points) The CPU would still have interrupts, and hence a register holding a pointer to the interrupt handler. However, when the CPU jumps to the interrupt handler, the CPU registers would only contain pointers from the user program, so the interrupt handler would not be able to access any of the kernel’s memory. How would you modify the CPU to enable interrupt handlers to work? Your solution must still use fat pointers, i.e. you can’t just turn off pointer bounds checking when jumping to an interrupt handler.

Solution

A couple of options:

- add an interrupt handler argument register that can hold a pointer to any data needed by interrupt handler

- store the interrupt handler code and any data it needs contiguously in memory and set the IHA register so its bounds encompass both.

(5 points) Does the CPU still need a privileged mode? Why or why not (in one sentence)?

Solution

Yes - Unprivileged code should not be able to work with the IHA (and IHA argument) register.