Homework 4 Solutions

CS161 Computer Security, Fall 2008
Assigned 11/12/08
Due 11/19/08

For your solutions you should submit a hard copy; either hand written pages stapled together or a print out of a typeset document[1].

1. *Attestation using Trusted Computing Hardware* **[2 points]**

   Recall that the trusted computing architecture includes a hardware TPM in every machine. The TPM contains a secret signing key and the resulting signatures are used for attestation. Do all computer hardware TPM chips require to have a distinct signing key? Explain why.

   Solution. Yes, all TPM chips should have distinct signing keys. Suppose two computers, say A and B, have the same keys and A is a compromised machine running a malicious software stack under attacker's control. Whenever A recieves a remote attestation request, it could forward the attestation request to B (which runs a valid software stack) and forward B's response as the attestation for A. This gives a false assurance of A's software stack integrity to a remote party, while A is running a compromised stack.

2. *Trusted Boot using Trusted Computing Hardware* **[3 points]**

   Recall that the trusted computing architecture includes a hardware TPM in every machine. Suppose the TPM does not have any persistent *protected* storage other than those needed for storing the cryptographic keys. This would still permit remote attestation applications, as the attesting process requires no protected persistent storage. Would it also permit secure boot? If yes, how would you design a secure system. If no, explain why.

   Solution. No, it does not permit secure boot. For secure boot the TPM computes the integrity hash of the n^{th} layer while booting, and compares it to the integrity hash of the valid software that it expects to be running at that layer. This requires it to store the valid hashes on persistent storage, and further, ensure that the integrity checking hashes are never tampered with by the software layers. Hence, without protected storage in the TPM, this is not possible.

3. *Inline Reference Monitors* **[10 points]**

   Recall that *address sandboxing* is the policy of confining all memory reads and writes made by an untrusted module to a pre-determined segment in memory. It is possible to enforce address sandboxing using software fault isolation (SFI) – SFI embeds an inline reference monitor to confine all memory accesses to within one segment in the process address space.

   Consider a target architecture has only two memory access instructions, `load` and `store`. `load [r1],r2` reads the value from memory pointed to by register `r1` into register `r2`, and, `store [r1],r2` writes the value of register `r2` to memory pointed to by register `r1`. The target architecture has only one unconditional control transfer instruction, `jmp [r1]`, that transfers control to the address stored in `r1`. All instruction on the architecture are 8 bytes in length. You may assume other instructions exist if you need them to solve this question, but be sure to define them before using them in your answers.

   (a) **[3 points]** The following checks for the enforcing address sandboxing policy are inserted before each `load [r1],r2` and `store [r1],r2` instruction.

   ```
   r1 = r1 & 0x00000fff
   r1 = r1 | 0xccccc000
   ```

[1] LaTeX is the most suitable tool for typesetting mathematical documents, but other use of other editors are perfectly acceptable

The following checks are inserted before `jmp [r1]` instructions :

`r1 = r1 & 0x00000fff`

`r1 = r1 | 0xbbbbb000`

This implementation of the address sandboxing scheme aims to confine memory reads and writes to the memory range `0xccccc000 - 0xcccccfff`, and the untrusted code with inline checks is placed in the memory segment `0xbbbbb000 - 0xbbbbbfff`. Explain why this implementation still allows a malicious code to write memory outside the intended range. Outline an alternate implementation that is secure, and explain why your proposed implementation is safe.

Solution. . The checks permit an attacker to jump in the middle of inline monitor checks for load/store operations. Thus, and attacker can set r1 to a value of its choice, and jump right past the instrumented checks straight to the access instruction.

An alternate implementation should use dedicated registers instead of r1, which are only used by the inline reference monitor checks, as outlined in the lecture slides, and reproduced below :

```
dr1 = r1 & 0x00000fff         ; mask r1
dr1 = dr1 | 0xccccc000
load [dr1], r2
```

The "dr1" register is used only by IRM checks, and is initialized approporately by the IRM. You can verify that no matter where the control jumps, at the execution of load instruction, "dr1" will always have an address in the sandboxed range.

(b) In parts (b.1) and (b.2) you are asked to design an inline reference monitor scheme, similar to SFI, with a different policy than address sandboxing. The new policy is to prevent an untrusted module from reading/writing data within one segment, starting at address `0xabcde000` and ending at `0xabcdefff`.

(b.1) **[3 points]** Show the inline reference monitor checks, if any, that may be needed before the `load`, `store` and `jmp` instructions in the untrusted code. Clearly state use of any dedicated registers and/or memory in your scheme.

Solution. Here we show the instrumentation for `load [r1], r2`; store is similar.

```
dr1 = r1 & 0xfffff000     ; mask r1
dr2 = r1 & 0x00000fff     ; get offset in segment
cmp dr1 0xabcde000        ; check if segment is in prohibited range
jeq error_label           ; if yes, exit.
dr2 = dr1 | dr2           ; construct original address
load [dr2], r2
```

`error_label` is an exit point of the program, where execution halts. No checks are need for jump instructions.

(b.2) **[4 points]** How does your scheme prevent a malicious module from bypassing the inserted checks using indirect control transfer instructions in its code? Briefly explain why are the checks sufficient to ensure the new policy.

Solution. "dr1" and "dr2" are dedicated registers. "r1" is never used after the second IRM check. So, as long the verifier ensures that dr1 and dr2 have correct values, the attacker can not violate the policy by setting "r1" and jumping anywhere after the second check. If it jumps between the first and second check, dr2 will always be correct – the reason is that "dr1" is ensured to never have a value `0xadcde000` if control reaches the load.

4. *System Call Interposition* **[10 points]**

Consider the following program running on a typical UNIX system.

For conciseness, the code only show the system calls made by the program as $S1$, $S2$ …, $S7$. $S0$ denotes a network `write` system, $S1$ denotes a network `read`, $S2$ is a file `read` system call, $S3$ is a file `write`, $S4$ is a file `open` system call, $S5$ is file `close`, $S6$ is a `getruid` and $S7$ is `setuid`.

```
S0;
while (..) {
     S1;
     if (...) S2;
     else S3;
     if (... S4 ...) ... ;
     else S2;
     S5;
}
S6;
S7;
```

(a) **[2 points]** How can system call interposition prevent an attacker from exploiting this program to execute a shell (via the `exec` system call).

Solution. By analysing the body of the code, it is clear that the code makes no `exec` system calls during legitimate program execution. System call interposition can flag an exploit as soon as exec is invoked.

(b) **[6 points]** Based on the program code, you notice that the program can only execute a certain sequence of system calls under benign operation. For instance, the program never performs a network `read` or `write` after the `setuid` system call. Write a regular expression that captures all legitimate sequences of system calls that the program shown above can make. Express your answer using regular expression operators (concatenation, alternation and Kleene's star) over the set of symbols ($S1$, $S2$ . . ., $S7$ and ϵ).

Solution. $S0.\ (S1.\ (S2 \mid S3).\ S4.\ (\epsilon \mid S2).\ S5.)^{*}\ S6.\ S7$.

(c) **[2 points]** How does your regular expression model in part (b) help enforce the following policy using system call interposition : *No network writes after file read.* Outline one example of a threat that this policy defends against.

Solution. The system call interposition based monitor ensures that all sequences of system calls invoked by the program at runtime match the regular expression, i.e., the regular expression defines a policy for all valid sequences of system calls that the monitor can observe. $S0$ (network write) can never follow $S2$ (file read) in any sequence that matches the regular expression – thus, a system call monitor that allows system call sequences that match the regular expression will automatically enforce this policy.

Consider an attack where the attacker compromises this application, then reads a confidential file and sends it contents over the network. Syscall interposition with the above policy prevents this attack on this program.

5. *Privilege Separation* **[10 points]**

Foomail is a UNIX electronic mail transfer program – it manages sending and receiving mails on a multi-user UNIX system. In this question, you are given a high-level design description of the functions that *foomail* performs. Note that it requires superuser privileges, but many of its operations handle immense amount of untrusted data. Your task is to redesign the `foomail` program with several trusted and untrusted components using the principle of least privilege.

System Resources. The set of resources that are used by `foomail` along with a brief description is outlined below.

- *Mail Queue (Q)* : An operating system resource that stores incoming and outgoing mails, waiting to be processed, for all users on the system.
- *Mail Spool Directory (SD)* : A special directory containing a separate file for each user where the mail transfer program stores the received mail for that user.
- *SMTP Port on local machine (P)* : Port 25 is the SMTP port on which the mail transfer program binds and listens for incoming mail delivery connections.

- *Other high-numbered ports on local ports (N)* : `foomail` needs to bind and connect to high-numbered ports on local machine to access the network for communication with SMTP mail servers on remote hosts.
- *DNS (D)* : Access to a DNS server on a remote host on the network, which resolves network host names to IP addresses.
- `/etc/hosts` *file* : Privileged file (read/write access allowed only to superuser, read-only access to users on the system), that is a local database to lookup IP addresses for network hostnames.
- `/etc/passwd` *file* : Privileged file (read/write access allowed only to superuser, read-only access to users on the system), containing general user information (in addition to other sensitve information) about each user on the system.
- `.forward` *file* : A user-specific file (owned by a user), that contains information about how to forward email received at one email address to another.

Access to Q, SD, and binding to port P requires superuser permissions. Access permissions to files are as outlined in the above description; other resources can be accessed by processes running as unprivileged (non-superuser) users.

Operations performed by foomail. When the `foomail` process runs, it roughly needs to perform the following tasks, not necessarily in the order defined below:

- Bind and listen on local SMTP port P for receiving mail.
- Upon receiving mail from the network port P, it updates Q with the arrived mail.
- Read Q to fetch incoming mail that has not been processed.
- Access user information from `/etc/passwd` to determine the recipient's information.
- Read `.forward` files of a user to check if mail is to be delivered.
- Consult SD to place received mail for a user in his/her mail file.
- Write outgoing mails from user's mail file in SD to Q for further processing.
- Read Q to get outgoing mail.
- To send outgoing mail to another remote SMTP mail server, `foomail` first needs to get an IP for the hostname of the remote SMTP mail server. It first reads `/etc/hosts` to resolve the network hostname to an IP address.
- To send outgoing mail to another remote SMTP mail server, it first needs to get an IP for the hostname of the remote SMTP mail server. If `/etc/hosts` can not resolve the hostname of target SMTP server, it makes a network request to a DNS server to get the destination IP.
- Send outgoing mail to destination SMTP server by opening a network connection to the IP address on the remote SMTP server.

(a) Split the monolithic design of `foomail` into 2 or more individual components, that minimize the privileges required for each component. Label each of the components as A,B,C,D . . . and so on.

Solution. The goal is to give minimum priveleges to each component. Therefore, we should try to identify individual functions and assign each function to a separate component.

(a.1) **[4 points]** Describe the functionality provided by each component in no more than 2 sentences per component.

Solution.

- A - Queue Handler: Runs as superuser, only reads/writes files to queue.
- B - Directory Manager - Runs as superuser, creates and deletes files in the directory.
- C - Incoming mail daemon - Binds to port 25 and listens for new incoming mail connection. Drops superuser privileges after bind.
- D - DNS proxy - Handles all network traffic to and from the DNS server. Needs superuser privleges.

- E - Network proxy - Manages accesses to the high numbered ports. Needs no superuser privileges.
- F - Local DNS resolver - Performs accesses to `/etc/hosts/`. Needs no superuser privileges.
- G - GECOS extractor - Accesses user information in `/etc/passwd`. Needs no superuser privileges.
- H - Forward file reader - Reads .forward files for a user. Needs no superuser privileges.

(a.2) **[4 points]** Write down for each component the kind of access (read, write, read-and-write, or none) for each of the resources described above, as a matrix with rows as the component labels and columns as resource names.

	Q SD	P	N	D	etc hosts	etc passwd	.forward	
A	RW							
B		RW						
C			R					
D					RW			
E				RW				
F						R		
G							R	
H								R

(a.3) **[2 points]** In at most 3 sentences, precisely state what security mechanisms would you use to enforce restrictions you have outlined in part (a.2) on components.

Solution. Place each component as a separate process, each running with indicated user privileges. Inter-component communication would be through IPC mechanisms. System call interposition should be used to restrict access privileges to file and network resources, as indicated by the above matrix.

(b) **[5 points]** Consider the component, say `X`, that reads `/etc/passwd` which runs at normal user privileges. `X` is passed the name of the recipient by the component that receives the email from the network, and `X` then retrieves the user's account information from `/etc/passwd`. Suppose a remote attacker discovers a buffer overflow in `X`, which it can exploit remotely by providing a very long recipient user name. Based on the permission restrictions imposed by your solution in part (a.2), can the attacker exploit this bug to leak sensitive information in `/etc/passwd` about all users to the remote attacker directly via the network?

Solution. No. X does not have network access as per the imposed policy.

If not, then consider a slightly different setting. Suppose the user has installed a malicious network game application on the same computer as the one running `foomail`. It turns out that the remote attacker who knows about the buffer overflow in X, is also the author of the malicious game. The malicious game has been restricted access only to the network, and has no access to files in `/etc`. Can the remote attacker manage to learn the sensitive information in `/etc/passwd` in this setting?

Solution. Yes, the remote attacker can use covert channels to communicate the information between the compromised component and the network game. Recall from class that it is possible for the attacker to communicate a "1" by doing a CPU intensive operation at a pre-decided time – the listner can simply monitor the CPU load to determine at the same time to check if module X wishes to communicate "1" or "0".