Jonathan Simpson 100898116

Sorry in advance for the weirdly coloured PDF - the Atom editor is great otherwise.

1. For an operating system of your choosing, to what extent is it a secure operating system? Specifically, does it implement a reference monitor, and is that reference monitor offer complete mediation, is tamperproof, and is verifiable? Be specific. You may refer to hardware features where applicable (but it is not required).

Linux is not a secure operating system. Far from it. Linux does not come with a reference monitor and it's TCB, the linux kernel, has not been verified. There are kernel modules that can offer mediation, but none come with Linux by default. Because the system has not been verified, the system is vulnerable to bugs which could lead to tampering of the system.

Since no reference monitor is available by default with linux, it's up to third parties to provide kernel modules that add a reference monitor. Two popular open source projects that offer a reference monitor are Apparmor and SELinux. Both offer complete mediation of the system calls made by applications. Unfortunately it's not useful to verify Apparmor or SELinux because even if they don't have any bugs, the rest of the linux kernel may have bugs that allow for attackers to bypass the mediation.

The linux kernel is made up of 15 million lines of code. Because of it's sheer size it's not feasible to formally verify that the kernel is correct. About half of the linux kernel's size is taken up by device drivers. These drivers run in the TCB, as well as many other subsystems such as filesystems and networking. A term for a kernel with a lot of subsystems is "monolithic kernel". Due to the sheer codebase size and the number of submodules that are included in the linux kernel, it's prone to bugs. Bugs in the linux kernel can allow an attacker to gain privileges, read sensitive information. These bugs in the TCB can allow the attacker to bypass protection mechanisms.

Linux does work on the premise of discretionary access control by having a shaved down ACL for permissions. Read, write and execute are the actions and user, group and everyone are the group level permissions. This permission model is similar to Multics in that read, write and execute are the access modes, and person and project names correspond to users and groups. Even though Linux uses the same permissions model for file access, it doesn't provide enough security on it's own. Permissions are able to be changed at runtime, therefore it differs from Multic's mandatory access control model.

Linux does not provide a secure operating system. It's popularity was gained by the functionality it offered in exchange for security. The linux kernel is way too large to be formally verified, it doesn't come with a reference monitor by default, though third parties have created them, provides no mediation because there's no reference monitor and is vulnerable to being tampered with because of the possibility of bugs in the TCB.

2. For a computer installation (a computer running software being used in a specific context) where authorized users, and only authorized users, should

have access to information (read, write, or both) stored on a system, describe the following:

- What is the installation?
- What is a specific security threat the installation faces? Remember this threat may involve matters of confidentiality, integrity, and availability. The threat may be hypothetical; it just needs to be plausible given the environment the system is designed or deployed for.
- For this threat, what is one security mechanism that provides at least a partial defense against this threat?
- To what extent is your chosen mechanism's effectiveness potentially compromised due to user and system administrator choices?
- To what extent do you think there are vulnerabilities in the mechanism that could cause it to fail catastrophically?

The installation is a banking system. A bank has a computerized system that keeps track of it's client's accounts and it's balances. Clerks and clients are two authorized users that should be able to access all accounts, and their own accounts, respectively. Clients shouldn't be able to read and/or write other clients accounts since thats a breach of privacy, only their own. Clerks can access each account on behalf of the client since they often assist the client when performing banking operations like setting up a new account.

One security threat this system faces is that of a rogue clerk. A clerk has access to every account and therefore could read and write each account's balance, moving around money without consent from the client. The integrity of the banking system is weakened since the rogue clerk is making changes to client's accounts without their knowledge. Availability can be affected if the rogue clerk were to close a client's accounts, thereby preventing the client from withdrawing money. Confidentiality can also be compromised if the rogue clerk were to export all account and client information and release it on the internet. Banks don't share this information with the public for the client's privacy.

One security mechanism that provides at least a partial defense is authorizing the clerk to operate on the client's account when the client visits a bank branch. Before the clerk can help the client deposit money to their account, the client has to insert their bank card into a card reader before the clerk's banking terminal brings up their account information. This authorization does nothing to prevent authentication of the owner of the bank card, but does prevent a rogue clerk from being able to read and write to every account that the bank contains. The rogue clerk is only able to modify accounts and transfer money when the client is in front of the clerk.

The effectiveness of requiring a banking card to be presented to authorize the clerk to access the client's account is compromised when the clerk has a stack of cards that have been lost or stolen. The rogue clerk can then merrily modify the accounts and transfer money with glee, working around the defense. This authorization would benefit in having multifactor authentication. The banking card represents something you have. Providing a pin at authorization time would be something you know, which would have foiled the stack of lost or stolen bank cards.

There may be a vulnerability that would result in a catastrophic failure if when the bank card doesn't register with the card reader, then the clerk can press a button to get into and modify the client's accounts anyways. This may be put there because of usability reasons. The clerk doesn't want to have the client unable to do business with them if their card doesn't work with the card reader. This completely bypasses the security mechanism for trying to access a client's account. The rogue clerk would still be able to read and write every account belonging to the banking system.

3. In the far, far dystopian future, users who fail to authenticate successfully

to their home security system are fatally electrocuted. Describe possible attacks, defences, and how it would impact the end user.

Due to intergalactic thugs roaming the space streets, houses are equipped with strong security mechanisms to wade of intruders and solicitors. People who fail to authenticate are fatally electrocuted on spot with 10 amps of electricity. Unfortunately false positives are known to occur when someone who lives in the house is unable to authenticate properly. They're fatally electrocuted by their own house.

Defenses have been made to prevent the fatal electric shock from killing the authenticator. The underground black market have worked on making rubber suits that are able to prevent the wearer from feeling the effects of being electrocuted. This allows space thugs and other people to successfully survive an unsuccessful authentication. The wearer of the rubber suit is able to brute force the authentication system by trying the 4 digit pin over and over, where the fatal shock occurs every ten tries.

Attackers are able to break the home security system by connecting the pin pad to the shocking surface using an ultra thick neodymium cable so that when the system delivers a fatal dose of electricity, the electricity is conducted through the cable to the pin pad, effectively frying the electronics of the home security system. Because of safety reasons, the door is unlocked when it is not working, so the attacker gains access.

The end user of such a powerful electric shock security system is the people who try to gain access to the home. People who live in the home should be allowed in. It is possible for someone who lives in the house, but who was accidentally removed from the list of allowed people due to a software bug to get fatally electrocuted when they next attempt to enter the home. Loss of life in a dystopian future occurs often but is still sad to their loved ones and gives the home electrocution security system a bad reputation. Overall, the powerful electrocution defense that it offers keeps more attackers away than the lock and key mechanism so often used by early dwellings on earth.

5. Why do classic buffer overflow attacks (as outlined by Aleph One) generally not work on modern systems? Given these defenses, do buffer overflow attacks remain a threat? Explain.

The original buffer overflow paper written by Aleph One in 1996 has brought to attention the seriousness of the vulnerabilities of buffer overflow attacks. Over the next twenty years operating systems and programming languages have gotten better at preventing buffer overflow attacks using multiple mechanisms. Some of those mechanisms are ASLR and stack canaries. These mechanisms to prevent buffer overflow exploits have made it much harder to exploit buffer overflows, but attacks such as return-to-libc are still prevalent even when these mechanisms are used.

The classic buffer overflow attacks as shown by Aleph One don't work on today's systems because of a number of technologies. Simply, using a higher level language than C, which doesn't allow the user to manage memory gives the application better security since the programmer can't mess up with overwriting memory. Mechanisms such as ASLR randomize the address space of the processes memory. Segments that are loaded into memory are put in different locations at runtime. This prevents buffer overflows from knowing where the address of a specific function or point to jump to in memory is.

Another mechanism that prevents classic buffer overflow exploits from working is stack canaries. Buffer overflow

exploits write to memory beyond what was allocated for the buffer. Beyond the buffer is other local variables and the stack return address. When the program hits a return address when executing code, it looks at the return address in the stack frame and jumps to that position in memory then starts executing code. The buffer overflow exploit wants to overwrite this return address so that the value placed in the stack return address field corresponds with somewhere in the buffer. When the code hits a return, the program jumps into the buffer, executing the contents of the buffer, thereby running untrusted code from the attacker. The attacker now has the ability to run what ever code they choose. A stack canary is a detection mechanism for buffer overflows. Between the local variables and the return address of each stack frame exists a byte or word called a canary. The program sets the value of the canary when creating the stack frame and checks the value of the canary just before the stack frame is removed. If the canary differs from the value it was assigned initially, the program assumes some form of memory corruption occurred and forcefully exits the program. The classic buffer overflow exploit does not work in this situation since it will likely overwrite the canary with a different value when attempting to overwrite the return address.

Buffer overflow attacks still remain a threat because attacks such as return-to-libc exist. This type of attack relies on analyzing the program for small bits of code called gadgets to build programs just by chaining these gadgets together. Libc is so prevalent in programs that it's the perfect candidate for finding and using the gadgets that it contains. If the stack is execute protected then the classical buffer overflow would not work. Instead, return-to-libc would work because the code being executed does not live on the stack, it instead resides in the code segments of libc.

Security mechanisms such as ASLR and stack canaries do help prevent classical buffer overflow exploits. Using safe languages is more popular and therefore provides less of an attack surface for buffer overflow exploits. The buffer overflow exploit ecosystem has evolved to performing advanced attacks using return-to-libc and other methods. It is very rare for the once prevalent classical buffer overflow exploit to be seen, more commonly advanced buffer overflow techniques are required to exploit modern systems.

8. Usability, functionality, privacy, and security are often in conflict in computer systems. Discuss these tradeoffs in the context of one of the following: Android permissions, certificate pinning, two-factor authentication, or anonymous routing.

Two factor authentication offers more security, but at the price of usability. The time taken to perform a second form of authentication can be a slight inconvenience or it can be very costly depending on the situation. The usability disadvantage is often a small compromise when it comes to protecting sensitive data or where authentication of the user is very necessary.

Two-factor authentication is a security mechanism for making authentication more secure. Types of authentication are grouped into three distinct categories: something you know, something you are, and something you have. Normal single factor authentication for websites these days is done with a password that the user enters. This is something you know. Password leaks are now prevalent and therefore the single factor authentication offered by a password isn't enough anymore. Two-factor authentication should have a mix between the three distinct categories. Commonly, two-factor authentication is something you know and something you have. Google, Facebook and many other companies offer two-factor authentication using a password and then a time-based pin code. The time based pin code gives the user more security when logging in. The user can have their password leaked, but an attacker must have the time based pin code to login. The two factor method offers more security than the single factor method, but it comes at a price - usability.

Being asked to enter the password and a pin code every time the user logs into a website or system can be tedious since it takes a few extra seconds of the user's time. In a time sensitive scenario such as "the stock market is crashing and I have to sell all my assets using the e-trade program", the extra time taken to login because of two-factor can cost serious money. The extra security offered by the two-factor authentication may or may not be worth the extra effort required. In a scenario where the government has top secret information, it's important to authenticate multiple times to verify that the person is who they say they are. In a situation where someone is logging onto reddit, having two factor authentication probably isn't that important since the reddit account isn't used to conduct important discussions (though some may think their XKCD discussions are really important.)

Two factor authentication adds a very secure second method of authenticating a user. It also prevents attackers from accessing the user's account if one of the authentication methods have been compromised. The security gains often outweigh the loss of usability when it comes to systems where the user must be authenticated correctly.