Token Kidnapping's Revenge

Author: Cesar Cerrudo
(cesar.at.argeniss.dot.com)

Table of contents

Abstract

This document describes some Microsoft Windows elevation of privilege vulnerabilities, how they were found with the use of simple tools and how they can be exploited. Starting with a little security issue that then leads to more significant vulnerabilities finding. All the vulnerabilities detailed here are not publicly know at the time of this document release.

Introduction

Token Kidnapping [1] is the name of a research I did some time ago, it consisted of security issues and techniques that allowed elevation of privileges on all recent Windows operating systems. On Windows 2003 and XP it allowed to elevate to Local System account from any account that had impersonation rights. On Windows Vista and 2008 it allowed to elevate to Local System account from Network Service and Local Service accounts.

The old Token Kidnapping issues were fixed by Microsoft. This new research, presented in this document, demonstrates that those Microsoft fixes were not enough and elevation of privileges is still possible on all Windows versions.

Many people wonder how security research is performed, what tools are used, how is the process of vulnerability finding, etc. Based on that I tried to make this document as practical and detailed as possible in order to the reader can learn and easily understand it.

Some theory

Before starting we need to understand some theory, people with enough knowledge could skip this part.

Impersonation

Is the ability of a thread to execute using different security information than the process that owns the thread. Threads impersonate in order to run code under another user account, all ACL checks are done against the impersonated users. Impersonation can only be performed by processes with the following privilege: "Impersonate a client after authentication" (SeImpersonatePrivilege). When a thread impersonates it has an associated impersonation token.

Token

An access token is an object that describes the security context of a process or thread. It includes the identity and privileges of the user account associated with the process or thread. They can be Primary or Impersonation tokens, Primary ones are those that are assigned to processes, Impersonation ones are those that can be get when impersonation occurs. There are four impersonation levels: SecurityAnonymous, SecurityIdentity, SecurityImpersonation, SecurityDelegation. Impersonation takes place mostly during Inter Process Communication (IPC) using Local Procedure Call (LPC), Named Pipes, etc. Impersonation can be limited by clients by setting proper options on the calling APIs.

Windows 2003 and XP services security

Services run under Local System, Network Service, Local Service and regular user accounts. All services can impersonate. Some Windows services that impersonate privileged accounts are protected, they are created with "special" permissions, for instance a service running under Network Service account can't access a protected service running under the same account. This protection was introduced as a patch to fix the issues detailed in my previous Token Kidnapping research [1]. Before this patch it was possible to elevate privileges by getting privileged impersonation tokens from other processes, the patch restricts processes to access some other processes running under the same account that have privileged impersonation tokens.

Windows 7, Vista and 2008 R1 & R2 services security

There are lots of security improvements in latest Windows versions, there are new protections such as:
- *Session 0 isolation:* protect against Shatter attacks [5] by running services in a different session (session 0) than regular user processes.
- *Least privilege:* allow to run Windows services with only the minimum required privileges.
- *Per service SID:* each service process has a unique security identification, this allows service processes to be armored. Service running under "X" account can't access other service resources no matter the other service is running under the same "X" account.
- *Write restricted token:* services can have write access to resources only if explicitly granted to the service SID, logon SID, Everyone SID or write-restricted SID.
- *Restricted network access:* services can only accept and make connections on specified ports and protocols. Services can be restricted to have no network access.

This is implemented as firewall rules that can't be disabled after service starts.
- In Windows 7 and 2008 R2, IIS 7.5 worker processes don't run any more under Network Service account by default as they did on Windows 2008 R1 and Windows 2003. Now they run under a special account named DefaultAppPool. This provides more protection since web applications can't access processes running under Network Service account nor their resources. But DefaultAppPool account has the same privileges as Network Service account, it can impersonate.

The Tools

Let's describe the tools that will be used:

- Process Explorer (ProcExp): this tool displays information about all Windows processes, by selecting a process you can see information such as: Process ID, Windows objects handles opened and their names, user name that the process is running under, processes and objects DACL, etc.
- Process Monitor (ProcMon): this tool displays information about registry, file system and network access by Windows processes.
- WinDbg: it's a user mode and kernel mode debugger for Windows, part of Debugging tools for Windows.
- Registry Editor (Regedit): Windows tool to display and edit Windows registry.

Finding the vulnerabilities

I was waiting for Windows 7 (Win7) RC to take a quick look at it, mostly for finding low hanging fruit security issues since I didn't have much free time. I wanted to check also if there were some new issues similar to the ones described in my previous Token Kidnapping research [1], basically issues that would allow to elevate privileges and to bypass new protections.

After Win7 RC was released I got a copy, installed it and started to take a look at it. I ran ProcExp and looked at processes checking for DACLs issues on services, processes and on process objects such as threads, shared sections, mutexes, etc. Everything looked good so far.

After a while I couldn't find anything interesting by just clicking around in ProcExp. I remembered I had found some little issue on Windows 2008 R1 (Win2k8) and I thought I should check if it was still present on Win7.
The issue is that Telephony service (TapiSrv) has a process object handle from some service that runs under Local System account and the handle has DuplicateHandle privileges on it. This means that Telephony service process can duplicate any handle from this other process. Telephony service runs under Network Service account and it could, for instance, duplicate a Local System impersonation token handle from the other service process and use it to elevate privileges.
The issue is not important since in order to exploit it you must first exploit some vulnerability on Telephony service. But it's security issue anyways that can be exploited to bypass new Windows services protections. So I tested it on Win7 and it was still present there, that was good news, it meant Win7 wasn't perfect.

I continued looking around a little more at Win7 but couldn't find any low hanging fruit, I decided to focus on the only issue I had so far. I didn't know any details about the Telephony service issue, why "sometimes" it had that process handle with those privileges? it was a mystery for me. I say "sometimes" because in some tests the process handle wasn't there. I had to research more the issue.

I thought about what I knew about Telephony service:
- It provides functionality for programs that controls telephony devices: modems, VoIP, etc.
- It doesn't run by default.
- Any user can start it by issuing a "net start tapisrv" command.
- It runs under Network Service account on Win2k8 R1 & R2, Vista and Win7 and it runs under Local System account in WinXP and Win2k3.
- It has had some remote and local vulnerabilities in the past.

I needed to know more about inner and outer workings of that service, what files and registry keys it uses, how it communicates with other processes, what applications use its functionality, etc. I always start by trying the easiest tests that don't require much time and effort. I thought I would start by looking at file and registry interaction.
File Monitor and Registry Monitor tools would help, I checked the web for new versions and I found ProcMon which was a better tool for the job, you can monitor registry, files and network access with the same tool. After installing ProcMon I ran it and set a filter to just display Telephony service process activities. This service runs under a Generic Host Process for Win32 Services (svchost.exe process) together with other services. Using ProcExp I identified the svchost.exe process hosting Telephony service and got its process ID (PID), then I used this PID to create a filter in ProcMon to just display all activities related only to that process.

The tool was ready I just needed to find a way to interact with Telephony service to force it to access files, registry, network, etc. I started by stopping the service running: "net stop tapisrv" and then started it: "net start tapisrv", obviously this produced a lot of file and registry access activity. I got dozens of file and registry access items to analyze, but what I would look for?

The first step was to try to quickly understand what the service was doing, what registry keys and files it accessed and why. This could be a little difficult if you don't have enough Windows OS knowledge but with some effort you can quickly get practice and experience.
I didn't understand everything about TapiSrv actions, luckily ignorance makes you asking some questions and try to answer them in a way that makes some sense.

I saw TapiSrv was accessing HKLM\Software\Microsoft\Tracing\tapisrv registry key, I haven't seen that registry key before so it got my attention. I ran Regedit and located the key. The first thing that came to my mind was to check key ACL permissions, there was a surprise there. Network Service, Local Service and Users accounts had the same permissions and they included the "Set Value" permission. Clearly there was an issue there since any user could manipulate values that then are read and used by privileged processes. A little smile was being drown in my face, I started to think the issue was exploitable and I knew how to exploit it but I had to confirm it.

I looked at the subkeys under HKLM\Software\Microsoft\Tracing key, they were many, I found that they had the same or similar names as Windows services which made me think that many Windows services used that key.

At that time I didn't need to know what those registry keys were used for, it was enough knowing that they were read and possible written by some Windows services. Later when researching another vulnerability not mentioned here I found out that those registry keys are used by a tracing functionality implemented by some services. This functionality logs errors, debug messages, etc. related to the services. Services using this functionality automatically monitor the registry key for changes so if there is a change all the key values are automatically read again by the services.

Let's see now why I though the permissions issue was exploitable and I knew how to exploit it. If you look at the registry values under *"HKLM\Software\Microsoft\Tracing\tapisrv"* you will find out one value named *"FileDirectory"* that has a default value of *"%windir%\tracing"*, that

value is a Windows folder name that is probably read by services and then used to access files. I located and opened the folder with Windows Explorer but it was empty. Looking at the items displayed by ProcMon I saw that there weren't items showing access to that folder. Then with a new look at the key values I saw *"EnableFileTracing"* value which had a value of *"0"*, the value name looked self describing, I had to change that value to *"1"* and look if something happened, I did it but nothing happened at that time.

I thought, let's try restarting the service, I stopped it and suddenly a file named tapisrv.log was created in *"c:\windows\tracing"* folder. Looking at ProcMon I could see that TapiSrv was accessing (writing) that file after reading the folder name from the registry key. I had to be sure, that the folder value was indeed read from that registry key so I changed *"FileDirectory"* value to a new value and then started the service. Finally I was right, TapiSrv tried to access a folder named as the new registry value I had set.

Those who know about Windows local exploitation should already be familiar on how to exploit this issue. It's pretty simple but we will need a special privilege in order to exploit this issue, we will need to be able to impersonate. Impersonation privilege is held by most Windows services and some regular processes. Some popular accounts that have this privilege by default are IIS application pool accounts used to run IIS worker processes. These processes are used to run ASP .NET or classic ASP applications, they are a good target for attacks, if you can upload web pages then you can compromise the server.

The attack is simple, it consists of using impersonation over named pipes [2]. Any user with impersonation privileges can build an exploit that will create and listen on a named pipe waiting for a connection (for TapiSrv the named pipe would be \\.\pipe\x\tapisrv.log). Then set the *"FileDirectory"* registry value to the name of the already created named pipe without the file name and using a UNC path (\\localhost\pipe\x), finally setting the *"EnableFileTracing"* value to *"1"*. After that, the service will read the registry values set by the exploit and connect to the named pipe allowing the exploit to impersonate the user that the exploited service is running under. If the impersonated user has more privileges than the user running the exploit then elevation of privileges is successful.
As detailed before, services using tracing functionality monitor for changes to their associated registry subkey under *"HKLM\Software\Microsoft\Tracing"* key, the service process will immediately read the subkey values after the exploit changes them.

While exploiting TapiSrv to elevate privileges was possible, the exploit would have to do many steps in order to get a Local System impersonation token to fully compromise the system. I preferred to find a service that ran under Local System account so the exploit could directly impersonate this account.
Looking at the names of the subkeys under *"HKLM\Software\Microsoft\Tracing"* registry key I found a subkey named IpHlpSvc, this name seemed to reference IP Helper service. Using ProcExp I looked at the properties of the processes in the *"Services"* tab until I found the IP Helper service running under a svchost.exe process. This process was running as Local System account being the perfect candidate for exploitation.

After doing some tests I could come up with a reliable exploit that works really well and can be used on different Windows services including IIS 7 & 7.5, SQL Server, etc. running on Win2k8 R1 and R2, Vista and Win7.

I continued researching TapiSrv just in case there were more issues. I found that dialer.exe tool interacted with TapiSrv, some actions were recorded by ProcMon when running dialer.exe. TapiSrv was accessing *"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Telephony"* registry key, again after a quick look at the permissions I found a new issue, Network Service account had full control over that key. That was clearly an issue since it broke per process SID service protection allowing any process running under Network Service account to perform any actions on that registry key. But an issue is no more than a simple bug if you can't exploit it. Looking at the subkeys I found an interesting one,

"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Telephony\Providers" and under this subkey some interesting values named *"ProviderFileNameX"* (where the "X" was a number starting at 0) that looked as file names.

I looked at ProcMon and I saw that after Tapisrv read these subkey values it loaded files with those names as Windows DLLs, this was really interesting, the issue just went from simple bug to a vulnerability.

Since the key allowed any process running under Network Service account to perform any action, it was possible to set one of these registry values to an arbitrary DLL to be loaded by TapiSrv in order to run arbitrary code. TapiSrv runs under Network Service account but in case you don't remember, it had a process handle with DuplicateHandle privileges of a process running as Local System account. Once you get an arbitrary DLL inside TapiSrv process, the DLL code can get that handle and duplicate a Local System impersonation token handle from the privileged process and fully compromise the system.

I started to think how to build an exploit for that vulnerability. It wasn't straight forward this time since it was needed to introduce a DLL in the system, then change the registry values and finally trigger the functionality that will load the DLL.

I went to MSDN to check for TAPI APIs because I needed to find a way to interact with TapiSrv to get an arbitrary DLL loaded. I remembered that TapiSrv read the registry values and loaded the DLLs when dialer.exe was ran, probably dialer.exe was calling some TAPI APIs. Looking at MSDN an API got my attention, lineAddProvider(), the first parameter was named lpszProviderFilename. The parameter name looked familiar and it was similar to the subkey values name, I really had to try that API.

ProcMon was still monitoring Tapisrv, I built a simple program that called the API and ran it. I could see in ProcMon that Tapisrv was trying to access a file named the same as the first API parameter I used in the API call, that was amazing. Building an exploit for this issue was pretty simple, just creating a DLL that when loaded it will duplicate a Local System impersonation token handle obtained from the privileged process that TapiSrv had its handle.

When testing lineAddProvider() API, first I ran the test program as a low privileged user and it didn't work, it would have been nice if it had worked but it didn't. Then I ran it as an administrator and it worked, but I needed to try to run it under Network Service account. I tried that and it worked which made sense since that account could modify the registry key were the DLL information was saved. But then I tried to run it under Local Service account and it worked too, that was excellent, that meant it was possible to elevate privileges from any process running under Local Service account too. There was still something more I had to try, the same tests on Win2k3.

I checked for the issue in Win2k3 but the registry key had proper permissions, not Network Service access since Tapisrv runs as Local System on Win2k3. But on WinXP there was an issue, Network Service and Local Service accounts had full control on the registry key, any process running under those accounts could easily elevates privileges since TapiSrv runs as Local System in WinXP.

When testing lineAddProvider() on Win2k3, I got exactly the same results as in Win7. Exploitation was easier on Win2k3 since TapiSrv runs under Local System. Just calling lineAddProvider() passing a DLL and the DLL will be loaded and ran under Local System account by TapiSrv. Finally in WinXP I got the same results as in Win2k3.

It's worth to mention that I also ran tests on Win2k8 R1 & R2 with same results as Win7, which makes sense since Win2k8 R2 has the same technology as Win7 and it was based on Win2k8 R1, these operating systems are similar.

When I was running dialer.exe I noticed on ProcExp that Tapisrv process got a dialer.exe process handle with DuplicateHandle privileges and the same happened when running my test program that called lineAddProvider(). I realized that Tapisrv always got a process handle from the processes that interacted with it, calling the APIs, etc.

Since in order to be able to elevate privileges on Windows Vista and newer versions it's needed the privileged process handle inside Tapisrv process I started to look for the service name from which TapiSrv got the privileged process handle. I knew the process was on one of the svchost.exe processes running as Local System account but that process had running many services inside. I had to find the right service that interacted with Tapisrv. I searched for the svchost.exe process with ProcExp which I could easily identify it since I got the process id from the information displayed by ProcExp about the process handle inside Tapisrv process.

In the "Services" tab in the svchost.exe process properties a lot of services were displayed. There were two options for finding the right service, stopping the services one by one until the process handle would disappear from TapiSrv or try to guess which service could be the one based on the name, intuition, etc. Looking at the services names there was one called Remote Access Connection Manager (RasMan), let's see: remote, connection, access? That should ring some bells. This could be the one I'm looking for, I thought, so I stopped it and the svchost.exe process handle disappeared from TapiSrv process, it was the one.

I checked RasMan service details with Windows Services tool (Administrative Tools->Services), I saw that it has dependencies with Telephony service (TapiSrv) and that the service startup type was manual. This last option was the cause that sometimes the svchost.exe process handle wasn't present in TapiSrv in some of my tests. If RasMas was not started then TapiSrv didn't have the process handle. Luckily any user can start RasMan service if it isn't running, allowing elevation of privileges to be always successful since we can force the process handle of RasMan to be always present inside TapiSrv.

By just looking at Telephony service I had found so far:
1– Win2k8 R1 & R2, Win7 and Vista Elevation of privileges by any user with impersonation rights by exploiting weak permissions on Service Tracing functionality registry keys.
2– Win2k3, WinXP, Win2k8 R1 & R2, Win7 and Vista Elevation of privileges by Network and Local Service accounts by calling lineAddProvider() API.

These issues can be exploited on Windows services such as IIS 6, 7 & 7.5 and SQL Server. On IIS an attacker only needs to upload a .NET web page with exploit code and then run it to complete compromise the server. On SQL Server an attacker will need database administrative permissions and run the exploit by executing xp_cmdshell or sp_addextendedproc stored procedures allowing the attacker to elevate privileges and run code under Local System account.

It's important to note that these issues can also be used on post exploitation scenarios, where an attacker is exploiting a Windows service that has impersonation privileges but it's not running under Local System account. In this case exploitation is limited since attacker will be trapped in that service not being able to access other processes, resources, etc. due to new Windows protections. Abusing the issues detailed in this paper will allow the attacker to elevate privileges and run code under Local System account bypassing all the new Windows protections.

*See Chimichurri exploit available with this paper.

Bypassing Microsoft fix for Token Kidnapping on Windows 2003 and XP

On my previous Token Kidnapping research I had found how to get a Local System impersonation token from WMI processes running under Network or Local Service accounts. These WMI processes didn't have any protection in place to prevent other processes running under the same accounts to access them. This allowed any process running under Network or Local Service accounts to get a Local System impersonation token and elevate privileges. Microsoft fixed this issue by properly protecting WMI processes don't allowing other process running under the same account to access them.

While researching the TapiSrv issues with ProcMon on Win2k3 I noticed some strange behavior. There were some processes accessing or trying to access the same subkeys and values under HKEY_CLASSES_ROOT and HKEY_USERS\<UserSID>_Classes registry keys, getting sometimes "NAME NOT FOUND" when trying to access subkyes and values under this last key. These subkeys and values were always found under HKEY_CLASSES_ROOT key but not always under HKEY_USERS\<UserSID>_Classes key. I though that this behavior was because sometimes an application can be installed for only one user and not for all users, if this was the case then the information would be on HKEY_USERS\<UserSID>_Classes key and not on HKEY_CLASSES_ROOT key. But that was just a guess.

I knew that HKEY_CLASSES_ROOT key is mostly used to save information about OLE/COM/DCOM/ActiveX objects, processes read from that key information needed to instantiate objects. I looked at the available HKEY_USERS\<UserSID>_Classes keys and I found that the keys for Network and Local Service accounts, HKEY_USERS\S-1-5-20_Classes and HKEY_USERS\S-1-5-19_Classes respectively, didn't have any subkeys nor values. This was weird.

One of the process that tried to read values from those registry keys, was svchost.exe running DCOM Server Process Launcher service (DcomLaunch), this process runs under Local System account. I identified that before a WMI process was ran by DcomLaunch service, this service tried to read those registry keys. I saw a possible issue there since HKEY_CLASSES_ROOT key can only be modified by highly privileged accounts such Administrators and Local System, but HKEY_USERS\<UserSID>_Classes key can also be modified by the account to which the key belongs to. It's worth to mention that HKEY_USERS\S-1-5-20_Classes and HKEY_USERS\S-1-5-19_Classes can be modified by Network and Local Service accounts respectively. These less privileged accounts, Network and Local Service, can modify values that then could be read from a high privileged process, in this case DcomLaunch. If DcomLaunch use those values read from the mentioned registry keys to perform some actions then that could lead to privilege elevation, so there was a possible issue.

I thought that in order to confirm and exploit this possible issue I would have to create the same subkeys and values that were read from HKEY_CLASSES_ROOT key under HKEY_USERS\S-1-5-20_Classes key or HKEY_USERS\S-1-5-19_Classes key, depending which of them was being read by DcomLaunch process. Then if these values were used instead of the ones read from HKEY_CLASSES_ROOT key I could be able to confirm the issue and exploit it. I had a test program that launched a WMI process under Network Service account so I started to run tests creating subkeys and values under HKEY_USERS\S-1-5-20_Classes key.

I researched what values were read by DcomLaunch from HKEY_CLASSES_ROOT key and what those values were used for. I found that one of the values read was the default value under HKEY_CLASSES_ROOT\CLSID\{1F87137D-0E7C-44d5-8C73-4EFFB68962F2}\LocalServer32 subkey, which was "*%systemroot%\system32\wbem\wmiprvse.exe –secured*". wmiprvse.exe is the WMI executable file name, I supposed that this value was used to determine what to run when WMI was invoked. I thought if I remove the *–secured* argument then maybe WMI process won't be ran protected and I will be able to exploit it again as I did in my previous Token Kidnapping research. I created HKEY_USERS\S-1-5-20_Classes\CLSID\{1F87137D-0E7C-44d5-8C73-4EFFB68962F2}\LocalServer32 subkey and set the default value removing "-secured" argument. ProcMon showed that the new created value was read by DcomLaunch but no luck, after removing that argument the WMI process was also ran protected.
Another value that was accessed was AppIDFlags under HKEY_CLASSES_ROOT\AppID\{1F87137D-0E7C-44d5-8C73-4EFFB68962F2} subkey, the value was 0x2. I wondered what that value could be used for, searching on MSDN I found some information [3]. 0x2 value is used to secure COM servers, this started to look interesting. I though let's set this value to 0x0 to see what happens. I created AppIDFlags value under HKEY_USERS\S-1-5-20_Classes\AppID\{1F87137D-0E7C-44d5-8C73-4EFFB68962F2} subkey and set it to 0x0.
After setting the value to 0x0 I ran some tests and my initial thoughts were confirmed, it was an issue, I could ran the WMI process unprotected and exploit it as before the Microsoft patch.

Finally I realized that adding AppIDFlags value and setting it to 0x2 was the fix Microsoft introduced to patch the old issue on WMI processes [4] and that the new protection could be bypassed by exploiting this new issue.

The described issue only affects Windows 2003 and XP since HKEY_USERS\S-1-5-20_Classes and HKEY_USERS\S-1-5-19_Classes keys don't exist anymore on newer Windows versions.

*See Churraskito exploit available with this paper.

Finding more issues

While I was researching the already described issues, I saw and realized about some interesting things.
There is clearly a design mistake sharing HKEY_USERS\S-1-5-20 and HKEY_USERS\S-1-5-19 keys between all the processes that run under Network and Local Service accounts. Those registry keys have full control permissions for Network and Local Service accounts respectively, allowing any process running under those accounts to modify those registry keys values at will.
For instance a process "X" running under Network Service account can modify some file path value with a named pipe while listen on it. Then the process "X" can get an impersonation token when another process "Y" running under Network Service account reads that registry value and then tries to access the named pipe.
This completely breaks almost all new services protections. It allows access from process "X" to process "Y" if both run under the same account because once process "X" gets the impersonation token it can be used to access process "Y".

I also found a couple of minor issues related to high privileged process not dropping privileges before trying to access files. HKEY_USERS\UserSID registry key has full control permissions for the user to which the key belongs to, meaning that the user can set arbitrary registry values.
Consent.exe (Consent UI for administrative applications) is the program that shows the dialog window when you choose to run a program as Administrator in newer Windows versions. This program runs under Local System account and it reads HKEY_USERS\UserSID\AppEvents\Schemes\Apps\.Default\WindowsUAC registry key values. These values consist of .WAV file paths that the program uses for playing the sound specified by the user for UAC events. Consent.exe doesn't drop privileges when accessing the .wav file allowing any user with impersonation privileges to impersonate Local System account by using the already described named pipe trick.

Windows Defender service has a similar problem as Consent.exe. Windows Defender process reads *HKEY_USERS\UserSID\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folder* key to get information about different user related folders such as Documents and Internet related folders. Windows Defender service runs under Local System account and it also doesn't drop privileges allowing any user with impersonation privileges to impersonate Local System account by using the already described named pipe trick.

Preventing exploitation

You must avoid running processes under Network Service and Local Service account when possible, try running them as a regular user with the required privileges. Examples of processes that commonly run under those accounts and are exposed to attacks are IIS worker processes and SQL Server service process.

On IIS don't run ASP .NET web application in full trust, this won't allow web applications to impersonate.

On Windows 7, Vista and 2008 R1 & R2 remove Users group from HKLM\Software\Microsoft\Tracing registry key permissions. This will only prevent exploitation from regular users with impersonation privileges but won't protect against elevation of privileges from Network and Local Service accounts. If you already configured IIS worker processes and SQL Server service process to run under regular user accounts then you will be safer.

You must disable Telephony service (TapiSrv) if not used, this will prevent elevation of privileges by loading an arbitrary Dll using lineAddProvider() API on all Windows versions or by editing *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Telephony\Providers* registry key on Windows 7, Vista, 2008 R1 & R2.

Conclusion

Little and insignificant issues can lead to find more interesting issues. While Windows operating systems are becoming the most secure operating systems on earth there are still some issues that need attention.

It's possible to elevate privileges on all Microsoft Windows versions, the only requirement is to be able to impersonate by the user running the exploit.

On Windows Vista, Windows 2008 R1 & R2 and Windows 7 any user having impersonation privileges can elevate privileges and completely compromise the systems bypassing almost all new Windows protections. On Windows XP and Windows 2003, Network and Local Service accounts can elevate privileges and completely compromise the systems.

Some of the applications that are more susceptible to be exploited are Microsoft Internet Information Services 6, 7 & 7.5 and Microsoft SQL Server.

Special Thanks

To Mark Russinovich author of Process Explorer and other great Sysinternals tools, without his great tools I wouldn't have been able to find most of the vulnerabilities I have found on Windows and other software.

About the author

Cesar Cerrudo is the founder and CEO of Argeniss, a security consultancy and software firm based in Argentina. He is a security researcher and consultant specializing in application security. Regarded as a leading application security researcher, Cesar is credited with discovering and helping to eliminate dozens of vulnerabilities in leading applications including Microsoft SQL Server, Oracle Database Server, IBM DB2, Microsoft BizTalk Server, Microsoft Commerce Server, Microsoft Windows, Yahoo! Messenger, etc.

Cesar has authored several white papers on database, application security, attacks and exploitation techniques and he has been invited to present at a variety of companies and conferences including Microsoft, Black Hat, Bellua, CanSecWest, EuSecWest, WebSec, HITB, EkoParty, H2HC, FRHACK, Microsoft BlueHat, etc. Cesar collaborates with and is regularly quoted in print and online publications such as eWeek, ComputerWorld and other leading journals.

References

[1] Token Kidnapping
http://www.argeniss.com/research/TokenKidnapping.pdf
[2] Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit
http://www.blakewatts.com/namedpipepaper.html
[3] AppIDFlags
http://msdn.microsoft.com/en-us/library/bb427411(VS.85).aspx
[4] Vulnerabilities in Windows Could Allow Elevation of Privilege
http://www.microsoft.com/technet/security/bulletin/MS09-012.mspx
[5] Exploiting design flaws in the Win32 API for privilege escalation
http://web.archive.org/web/20060904080018/http://security.tombom.co.uk/shatter.html

About Argeniss

Argeniss is a small but very dynamic and creative company created in 2005. Argeniss offers information security consulting and software development services in an outsourcing model. More than 5 years of experience and satisfied customers prove Argeniss success.

Contact us

Velez Sarsfield 736 PA
Parana, Entre Rios
Argentina

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

Tel/Fax: +54 343 4316113