Project Zero

News and updates from the Project Zero team at Google

Wednesday, September 25, 2019

Windows Exploitation Tricks: Spoofing Named Pipe Client PID

Posted by James Forshaw, Project Zero

While researching the Access Mode Mismatch in IO Manager bug class I came across an interesting feature in named pipes which allows a server to query the connected clients PID. This feature was introduced in Vista and is exposed to servers through the GetNamedPipeClientProcessId API, pass the API a handle to the pipe server and you'll get back the PID of the connected client.

It was clear that there must be some applications which use the client PID for the purposes of security enforcement. However I couldn't find any first-party applications installed on Windows which used the PID for anything security related. Third-party applications are another matter and other researchers have found examples of using the PID to prevent untrusted callers from accessing privileged operations, a recent example was Check Point Anti-Virus. As relying on this PID is dangerous I decided I should highlight ways of spoofing the PID value so that developers can stop using it as an enforcement mechanism and demonstrate to researchers how to exploit such dangerous checks.

A simple example of a security check using the client PID written in C# is shown below. This code creates a named pipe server, waits for a new connection then calls the *GetNamedPipeClientProcessId* API. If the API call is successful then a call is made to *SecurityCheck* which performs some verification on the PID. Only if *SecurityCheck* (highlighted) returns true will the client's call be handled.

Search This Blog

Search

Pages

- Working at Project Zero
- 0day "In the Wild"
- Vulnerability Disclosure FAQ

Archives

2019

- Windows Exploitation Tricks: Spoofing Name... (Sep)
- A very deep dive into iOS Exploit chains found in ... (Aug)
- In-the-wild iOS Exploit Chain 1 (Aug)
- In-the-wild iOS Exploit Chain 2 (Aug)
- In-the-wild iOS Exploit Chain 3 (Aug)
- In-the-wild iOS Exploit Chain 4 (Aug)
- In-the-wild iOS Exploit Chain 5 (Aug)
- Implant Teardown (Aug)
- JSC Exploits (Aug)

```
using (var pipe = new NamedPipeServerStream("ABC"))
{
    pipe.WaitForConnection();

    if (!GetNamedPipeClientProcessId(pipe.SafePipeHandle, out int pid))
    {
        Console.WriteLine("Error getting PID");
        return;
    }
    else
    {
        Console.WriteLine("Connection from PID: {0}", pid);
        if (SecurityCheck(pid))
        {
            HandleClient(pipe);
        }
    }
}
```

What exactly *SecurityCheck* does is not really that important for this blog post. For example the server might open the process by its ID, query for the main executable file and then do a signature check on that file. All that matters is if a client could spoof the PID returned by *GetNamedPipeClientProcessId* to refer to a process which isn't the client the security check could be bypassed and the service exploited.

Where Does the PID Come From?

Before describing some of the techniques to spoof the PID it'd be useful to understand where the value of the PID comes from when calling *GetNamedPipeClientProcessId*. The PID is set by the named pipe file system driver (NPFS) when a new client connection is established. For Windows 10 this process is handled in the function *NpCreateClientEnd*. The implementation looks roughly like the following:

```
NTSTATUS NpCreateClientEnd(PFILE_OBJECT ServerPipe,
 KPROCESSOR_MODE AccessMode, PFILE_FULL_EA_INFORMATION EaBuffer) {
 // ...
 if (!EaBuffer) {
  DWORD value = PsGetThreadProcessId();
  NpSetAttributeInList(ServerPipe, PIPE_ATTRIBUTE_PID, &value);
  value = PsGetThreadSessionId();
  NpSetAttributeInList(ServerPipe, PIPE_ATTRIBUTE_SID, &value);
```


- The Many Possibilities of CVE-2019-8646 (Aug)
- Down the Rabbit-Hole... (Aug)
- The Fully Remote Attack Surface of the iPhone (Aug)
- Trashing the Flow of Data (May)
- Windows Exploitation Tricks: Abusing the User-Mode... (Apr)
- Virtually Unlimited Memory: Escaping the Chrome Sa... (Apr)
- Splitting atoms in XNU (Apr)
- Windows Kernel Logic Bug Class: Access Mode Mismat... (Mar)
- Android Messaging: A Few Bugs Short of a Chain (Mar)
- The Curious Case of Convexity Confusion (Feb)
- Examining Pointer Authentication on the iPhone XS (Feb)
- voucher_swap: Exploiting MIG reference counting in... (Jan)
- Taking a page from the kernel's book: A TLB issue ... (Jan)

2018

- On VBScript (Dec)
- Searching statically-linked vulnerable library fun... (Dec)
- Adventures in Video Conferencing Part 5: Where Do ... (Dec)
- Adventures in Video Conferencing Part 4: What Didn... (Dec)
- Adventures in Video Conferencing Part 3: The Even ... (Dec)
- Adventures in Video Conferencing Part 2: Fun with ... (Dec)
- Adventures in Video Conferencing Part 1: The Wild ... (Dec)

```
 } else {
  if (AccessMode != KernelMode)
   return STATUS_ACCESS_DENIED;
  LPWSTR computer_name;
  NpLocateEa(EaBuffer, "ClientComputerName", &computer_name);
  NpSetAttributeInList(ServerPipe, PIPE_ATTRIBUTE_NAME, computer_name);
  DWORD value;
  NpLocateEa(EaBuffer, "ClientProcessId", &value);
  NpSetAttributeInList(ServerPipe, PIPE_ATTRIBUTE_PID, &value);
  NpLocateEa(EaBuffer, "ClientSessionId", &value);
  NpSetAttributeInList(ServerPipe, PIPE_ATTRIBUTE_SID, &value);
 }
 // ...
}
```

The PID (and associated session ID and computer name) values are set using a generic attribute mechanism through the *NpSetAttributeInList* function. The value stored in the attribute list can be retrieved by issuing a File System Control request with the undocumented *FSCTL_PIPE_GET_CONNECTION_ATTRIBUTE* code to the server pipe.

When setting the attributes there are two options. Firstly, if no Extended Attribute (EA) buffer is provided in the file creation request, the PID and session ID are taken from the current process. This is the normal operation when creating a client connection in-process. The second option is used by the local SMB server, by specifying an EA buffer the driver allows the SMB server to specify connection information such as the client's computer name and additional PID and session ID. As a normal user-mode process can specify an arbitrary EA buffer the code also checks that the operation is coming from kernel mode. The mode check should prevent a normal user-mode application spoofing the values.

Spoofing Techniques

With knowledge of how the PID is set let's describe a few techniques of spoofing the value of the PID. Each technique has caveats which I'll explain as we go along. All techniques have been verified to run on Windows 10 1903, although unless otherwise noted they should work downlevel as well.

Opening Pipe Through Local SMB and a NTFS Mount Point

As I discussed in my IO Manager blog post, the check that the NPFS driver is making to prevent spoofing of connection attributes can be bypassed, if you can find a suitable initiator which will set the previous access mode to *KernelMode*. Prior to the fix for CVE-2018-0749 it was possible to set an arbitrary local NTFS mount

- Injecting Code into Windows Protected Processes us... (Nov)
- Heap Feng Shader: Exploiting SwiftShader in Chrome... (Oct)
- Deja-XNU (Oct)
- Injecting Code into Windows Protected Processes us... (Oct)
- 365 Days Later: Finding and Exploiting Safari Bugs... (Oct)
- A cache invalidation bug in Linux memory managemen... (Sep)
- OATmeal on the Universal Cereal Bus: Exploiting An... (Sep)
- The Problems and Promise of WebAssembly (Aug)
- Windows Exploitation Tricks: Exploiting Arbitrary ... (Aug)
- Adventures in vulnerability reporting (Aug)
- Drawing Outside the Box: Precision Issues in Graph... (Jul)
- Detecting Kernel Memory Disclosure – Whitepaper (Jun)
- Bypassing Mitigations by Attacking JIT Server in M... (May)
- Windows Exploitation Tricks: Exploiting Arbitrary ... (Apr)
- Reading privileged memory with a side-channel (Jan)

2017

- aPAColypse now: Exploiting Windows 10 in a Local N... (Dec)
- Over The Air - Vol. 2, Pt. 3: Exploiting The Wi-Fi... (Oct)
- Using Binary Diffing to Discover Windows Kernel Me... (Oct)
- Over The Air - Vol. 2, Pt. 2: Exploiting The Wi-Fi... (Oct)

point and redirect all local SMB requests to any device including NPFS, which normally wouldn't be possible if the file was opened directly as the kernel would refuse to link to a non-volume targets. As SMB file open requests can also specify an arbitrary EA buffer, this allowed a local client to open a named pipe connection with completely spoofed values, including the PID.

Once CVE-2018-0749 was fixed it was technically no longer exploitable. Unfortunately since Windows 10 1709 the kernel's handling of NTFS mount point targets was changed to allow reparsing to named pipe devices as well as more traditional file system volumes. Therefore it's still possible to spoof an arbitrary PID using the local SMB server, a mount point and a suitable EA buffer. The following C# example shows how you can do that to spoof the client PID as 1234 when opening the pipe named "ABC". You'll need to reference my NtApiDotNet library to use some of the types:

```
EaBuffer ea = new EaBuffer();
ea.AddEntry("ClientComputerName", "FAKE\0", EaBufferEntryFlags.None);
ea.AddEntry("ClientProcessId", 1234, EaBufferEntryFlags.None);
ea.AddEntry("ClientSessionId", new byte[8], EaBufferEntryFlags.None);
using (var m = NtFile.Create(@"\??\c:\pipes", null,
            FileAccessRights.GenericWrite | FileAccessRights.Delete,
            FileAttributes.Normal, FileShareMode.All,
            FileOpenOptions.DirectoryFile | FileOpenOptions.DeleteOnClose,
            FileDisposition.Create, null))
{
    m.SetMountPoint(@"\??\pipe", "");
    using (var p = NtFile.Create(@"\??\UNC\localhost\c$\pipes\ABC",
                        FileAccessRights.MaximumAllowed,
                        FileShareMode.None,
                        FileOpenOptions.None, FileDisposition.Open,
                        ea))
    {
        Console.WriteLine("Opened Pipe");
    }
}
```

Using this technique you can also follow the initial option for setting the PID in NPFS, specifically if no EA buffer is set then the current PID is used. As the SMB server runs in the System process this will result in setting the client PID to the value 4. This isn't really that useful when you can already specify an arbitrary value for the PID.

Pros:

- Potential to spoof an arbitrary PID (and session ID and computer name if desired).

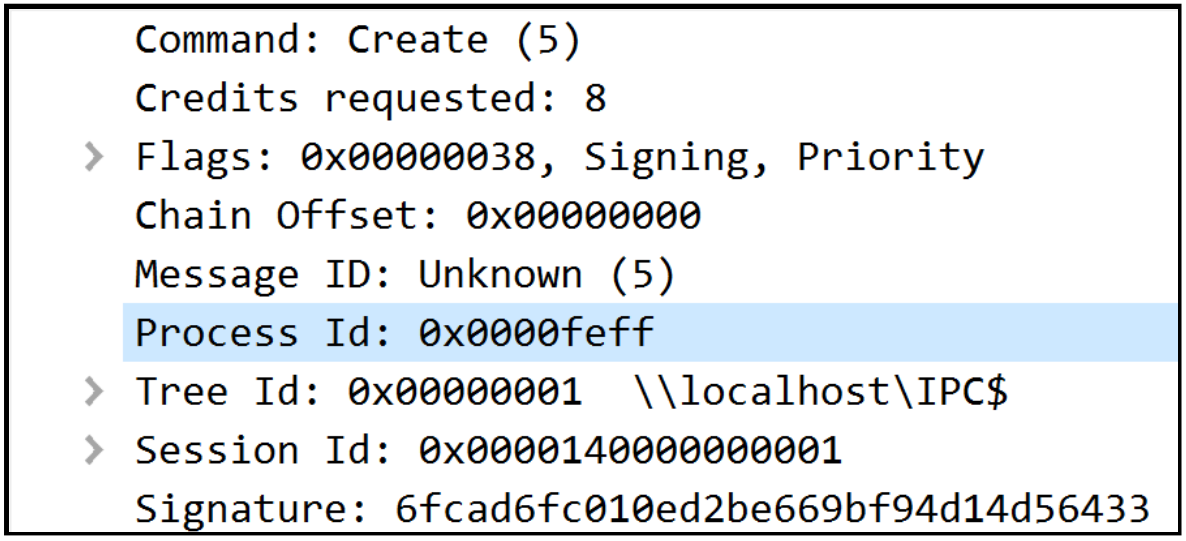
Cons:

- Requirement for a mount point and access to local SMB servers makes it impossible to exploit from a sandbox.
- Only works on Windows 10 1709 and above.

Opening Pipe Through Local SMB

If you're running on a version of Windows earlier than Windows 10 1709 all's not completely lost. You might assume that if you opened the named pipe using the local SMB server through the correct method i.e. open the path *\\localhost\pipe\ABC*, then the SMB server wouldn't set the PID attribute. A quick look at the server driver shows that it does indeed set it, specifically it sets it to a fixed value. On Windows 10 1903 that value is 65279/0xFEFF.

The fixed value comes from the SMB2 protocol header which is sent by the client. The header is documented by Microsoft. However the documentation reports the field containing the value used as the PID as "Reserved (4 bytes): The client SHOULD set this field to 0. The server MAY ignore this field on receipt.". Fortunately the Wireshark documentation is a bit more helpful, it points out it's a Process ID with a default of 0xFEFF. Capturing the SMB traffic in Wireshark when opening the named pipe shows the fixed value.



As the server doesn't seem to check the value you could set it to something arbitrary, however the in-built Windows client doesn't allow you to change the value from 0xFEFF. Can we exploit this without writing our own SMB2 client or using an existing one such as IMPacket? You can abuse the fact that Windows will re-use PID values and just create a suitable process which would meet the security check requirements until one of the processes has the correct PID.

Note that you can't actually create a process with ID 65279 as all current versions of Windows align PIDs to multiples of 4, however if the server calls OpenProcess on 65279 it will round down and open the PID 65276 which we can create. Also note that thread IDs are taken from the same pool as PIDs so you might be unlucky and create a thread with the ID you wanted. Cycling through PIDs could take a long time, especially with the semi-random allocation patterns of PIDs on modern versions of Windows, but it is possible to exploit.

A simple example of PID cycling is as follows:

```
while (true)
{
    using (var p = Process.Start("target.exe"))
    {
        if (p.Id == 65276)
        {
            break;
        }
        p.Kill();
    }
}
```

Once a suitable process has been created with ID 65276 you can then make a connection to the named pipe via the SMB server and if the server opens the PID it'll get the spoofed process.

Pros:

- Works on all versions of Windows.
- Can spoof the PID arbitrarily if willing to use a reimplementation of the SMB2 protocol.

Cons:

- Requirement for access to local SMB servers makes it impossible to exploit from a sandbox. Even if you reimplement the client it might not be possible to access localhost in an App Container

- Three bypasses and a fix for one of Flash's Vector... (Aug)
- Attacking ECMAScript Engines with Redefinition (Aug)
- One font vulnerability to rule them all #3: Window... (Aug)
- One font vulnerability to rule them all #2: Adobe ... (Aug)
- One font vulnerability to rule them all #1: Introd... (Jul)
- One Perfect Bug: Exploiting Type Confusion in Flas... (Jul)
- Significant Flash exploit mitigations are live in ... (Jul)
- From inter to intra: gaining reliability (Jul)
- When 'int' is the new 'short' (Jul)
- What is a "good" memory corruption vulnerability? (Jun)
- Analysis and Exploitation of an ESET Vulnerability... (Jun)
- Owning Internet Printing - A Case Study in Modern ... (Jun)
- Dude, where's my heap? (Jun)
- In-Console-Able (May)
- A Tale of Two Exploits (Apr)
- Taming the wild copy: Parallel Thread Corruption (Mar)
- Exploiting the DRAM rowhammer bug to gain kernel p... (Mar)
- Feedback and data-driven updates to Google's discl... (Feb)
- (^Exploiting)\s*(CVE-2015-0318)\s*(in)\s*(Flash$) (Feb)
- A Token's Tale (Feb)
- Exploiting NVMAP to escape the Chrome sandbox - CV... (Jan)
- Finding and exploiting ntpd vulnerabilities (Jan)

sandbox or get suitable authentication credentials.

- Only works if the server's security check uses the PID in *OpenProcess* and doesn't compare it directly to a running PID number.
- Getting a suitable process running with the correct ID to bypass the server security check might be very slow or difficult.

Opening Pipe in One Process. Using Pipe in Another.

This technique uses the fact that the PID is fixed once the client connection is opened, and the process which reads and writes to the pipe doesn't have to have the same PID. We can exploit this by creating the pipe client in one process, start a new sub-process and duplicate the handle to that sub-process. If the opening process now terminates the PID will be freed up and a PID cycling attack can again be performed. Once the PID is reused the sub-process can perform the pipe operations as required. The initial open looks like the following C# which uses handle inheritability over process creation to pass the pipe handle:

```
using (var pipe = new NamedPipeClientStream(".", "ABC",
             PipeAccessRights.ReadWrite,
             PipeOptions.None, TokenImpersonationLevel.Impersonation,
             HandleInheritability.Inheritable))
{
    int pid = Process.GetCurrentProcess().Id;
    IntPtr handle = pipe.SafePipeHandle.DangerousGetHandle();
    ProcessStartInfo start_info = new ProcessStartInfo();
    start_info.FileName = "program.exe";
    start_info.Arguments = $"{handle} {pid}";
    start_info.UseShellExecute = false;
    Process.Start(start_info);
}
```

Then in the sub-process the following code will wait for the parent to exit, recycle the PIDs until we get a match then use the pipe:

```
int ppid = int.Parse(args[1]);
Process.GetProcessById(ppid).WaitForExit();
RecycleProcessId(ppid);
var handle = new SafePipeHandle(new IntPtr(int.Parse(args[0])), true);
using (var pipe = new NamedPipeClientStream(PipeDirection.InOut,
                 false, true, handle))
```


2014

- Internet Explorer EPM Sandbox Escape CVE-2014-6350... (Dec)
- pwn4fun Spring 2014 - Safari - Part II (Nov)
- Project Zero Patch Tuesday roundup, November 2014 (Nov)
- Did the "Man With No Name" Feel Insecure? (Oct)
- More Mac OS X and iPhone sandbox escapes and kerne... (Oct)
- Exploiting CVE-2014-0556 in Flash (Sep)
- The poisoned NUL byte, 2014 edition (Aug)
- What does a pointer look like, anyway? (Aug)
- Mac OS X and iPhone sandbox escapes (Jul)
- pwn4fun Spring 2014 - Safari - Part I (Jul)
- Announcing Project Zero (Jul)

```
{
    pipe.WriteByte(0);
}
```

One big problem with this approach depends on where the service does the PID check. If the check is made immediately after connection then there's unlikely to be enough time to recycle the PID before the check is made. However, if the check is only made after a request has been made to the pipe (such as writing data to it) then the check can be put off until the PID is recycled.

Unlike the fixed value set by the SMB server it might be possible to create multiple separate connections with different PIDs to maximize the chances of hitting the correct recycled PID. How many connections can be made would depend on how many concurrent pipe instances the server supports.

Pros:

- Works on all versions of Windows.
- Should work in a sandbox if the named pipe can be opened.
- Possible to create multiple pipes with different PIDs to maximize the chances of PID recycling.

Cons:

- The check for PID can't immediately follow the connection, instead it must be after an initial read/write operation which limits the number of services which could be exploited.

Conclusions

If you're trying to exploit a service which is using the named pipe client PID as a security enforcement mechanism hopefully one of these techniques should suffice. Even in the absence of the ability to arbitrarily spoof the PID value it should be clear that this PID should not be relied upon to make security decisions as it doesn't necessarily reflect the actual client, just the process which opened the pipe.

Home Older Posts

Subscribe to: Posts (Atom)