It rather involved being on the other side of this airtight hatchway: Invalid parameters from one security level crashing code at the same security level

devblogs.microsoft.com/oldnewthing/20101208-00

December 8, 2010

Raymond Chen

In the category of dubious security vulnerability, I submit the following (paraphrased) report:

> I have discovered that if you call the `XYZ` function (whose first parameter is supposed to be a pointer to a *IUnknown*), and instead of passing a valid COM object pointer, you pass a pointer to a random hunk of data, you can trigger an access violation in the `XYZ` function which is exploitable by putting specially-crafted data in that memory blob. An attacker can exploit the `XYZ` function for remote execution and compromise the system, provided an application uses the `XYZ` function and passes a pointer to untrusted data as the first parameter instead of a valid `IUnknown` pointer. Although we have not found an application which uses the `XYZ` in this way, the function neverless contains the potential for exploit, and the bug should be fixed as soon as possible.

The person included a sample program which went something like this (except more complicated):

```
// We can control the behavior by tweaking the value
// of the Exploit array.
unsigned char Exploit[] = "\x01\x02\x03...";
void main()
{
   XYZ((IUnknown*)Exploit);
}
```

Well, yeah, but you're already on the other side of the airtight hatchway. Instead of building up a complicated blob of memory with exactly the right format, just write your bad `IUnknown`:

```
void Pwnz0r()
{
  ... whatever you want ...
}
class Exploit : public IUnknown
{
public:
  STDMETHODIMP QueryInterface(REFIID riid, void **ppv)
  { Pwnz0r(); return E_NOINTERFACE; }
  STDMETHODIMP_(ULONG) AddRef() { Pwnz0r(); return 2; }
  STDMETHODIMP_(ULONG) Release() { Pwnz0r(); return 1; }
};
void main()
{
   XYZ(&Exploit);
}
```

Wow, this new "exploit" is even portable to other architectures!

Actually, now that you're on the other side of the airtight hatchway, you may as well take `XYZ` out of the picture since it's just slowing you down:

```
void main()
{
   Pwnz0r();
}
```

You're already running code. It's not surprising that you can run code.

There's nothing subtle going on here. There is no elevation of privilege because the rogue activity happens in user-mode code, based on rogue code provided by an executable with trusted code execution privileges, at the same security level as the original executable.

The people reporting the alleged vulnerability do say that they haven't yet found any program that calls the `XYZ` function with untrusted data, but even if they did, that would be a data handling bug in the application itself: Data crossed a trust boundary without proper validation. It's like saying "There is a security vulnerability in the `DeleteFile` function because it is possible for an application to pass an untrusted file name and thereby result in an attacker deleting any file of his choosing." Even if such a vulnerability existed, the flaw is in the application for not validating its input, not in `DeleteFile` for, um, deleting the file it was told to delete.

The sad thing is that it took the security team five days to resolve this issue, because even though it looks like a slam dunk, the issue resolution process must be followed, just to be sure. Who knows, maybe there really is a bug in the `XYZ` function's use of the first parameter that would result in elevation of privilege. All supported versions of Windows need to be examined for the slim possibility that there's something behind this confused vulnerability report.

But there isn't. It's just another dubious security vulnerability report.

Exercise: Apply what you learned to this security vulnerability report. This is also paraphrased from an actual security report:

> There is a serious denial-of-service vulnerability in the function `XYZ`. This function takes a pointer to a buffer and a length. If the function is passed malformed parameters, it may encounter an access violation when it tries to read from an invalid buffer. Any application which calls this function with bad parameters will crash. Here is a sample program that illustrates the vulnerability:
>
> ```
> int main(int argc, char **argv)
> {
>  // crash inside XYZ attempting to read past end of buffer
>  XYZ("x", 9999999);
>  return 0;
> }
> ```
>
> Credit for discovering this vulnerability goes to ABC Security Research Company. Copyright© 20xx ABC Security Research Company. All Rights Reserved.

Raymond Chen

Follow