The Inside Story Behind MS08-067

John Lambert (MSTIC) 26 Sep 2015 11:10 AM 12

Seven years ago a small set of targeted attacks began. In 2008 an unknown set of attackers had a zero day vulnerability that would soon have worldwide attention. They were patient and used it quietly in several countries in Asia. The vulnerability was not just good--it was the kind of vulnerability that offensive teams and bug hunters dream about. It was, as we say in the business, “wormable”. That word sends chills down any defender’s spine. In short, the attackers had a remote code execution (RCE) vulnerability that affected every version of Windows, gave them full control at SYSTEM level rights, left almost no forensic footprint, and could be used anonymously from anywhere on the Internet. Their exploit was 95% reliable. Almost perfect. Almost.

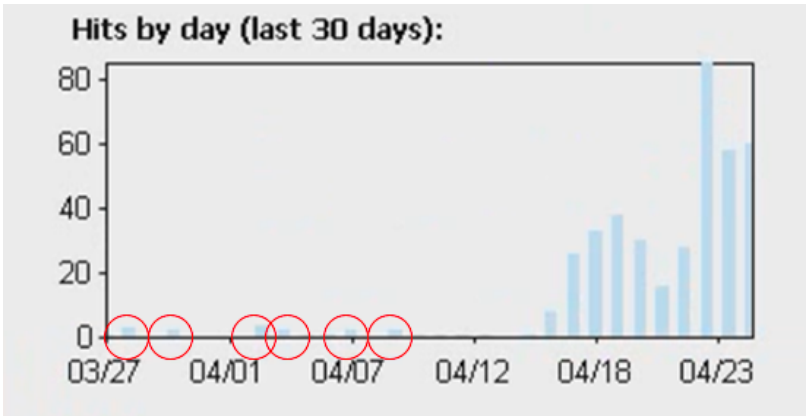
Exploit Mitigations

The story starts a year earlier. If you meet anyone that works in the Trustworthy Computing group at Microsoft, one thing you notice is that many of us chart our career milestones by Microsoft security bulletins. “MS03-007? That was a doozy.” “Sure, but MS03-026 was crazy.” “You’re right, that was crazy.” To understand MS08-067 you need to understand MS07-029, an RCE vulnerability in Windows DNS. MS07-029 was one of a series of Remote Procedure Call (RPC) server vulnerabilities that were steadily being ferreted out by Microsoft, attackers, and security researchers alike. There was one difference. MS07-029 was the first RCE that where we had our Visual Studio return address protection (/GS) and Windows Data Execute Prevention (DEP) in effect. We refer to these defenses as exploit mitigations and we had been steadily adding them since XP SP2. It was one of the ways we were using *security engineering* to combat security issues in engineering. Once an exploit has trashed the internal memory of a process, there is no recovery and the only option is to force a crash—a terrible user experience for sure, but better than resulting in a compromised machine. Exploit writers know we have been adding a steady stream of mitigations: SafeSEH, SEHOP, heap hardening, ASLR, MemProtector, IsoHeap, and Control Flow Guard. These mitigations not only close off avenues for exploitation, but they also give us the ultimate two-fer. They feed telemetry.

Exploits feed Telemetry

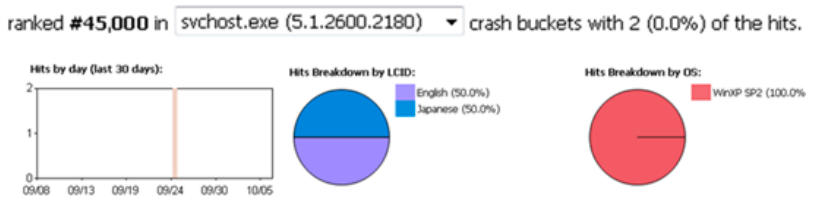
Crashes are a terrible experience for users so Windows has a facility called Windows Error Reporting (WER) that can, if the user consents, send the crash data to Microsoft. WER receives hundreds of millions of crashes from poorly behaving apps and buggy hardware running on over one billion PCs. Engineers “bucket” them by the underlying root cause and work their way down the priority list—first fixing bugs affecting the greatest number of customers. Remember that the next time a Windows Error Reporting dialog appears—you’re voting for your bug.

The crash buckets for the bug in MS07-029 were revealing. The figure shows a significant increase in crashes in Windows DNS after the issue became public (early April 2007). Once an issue is public, security researchers and attackers alike race to re-discover the vulnerability and move from proof-of-concepts to working exploits. The crash buckets for Windows DNS suddenly became very active. Every idea has its moment of inspiration. For me, it was looking at that chart: just before we learned of the attack, every few days there was a crash--a blip in the WER universe. I looked at each of them and saw that all of them were exploits. If we had the ability to determine when a crash was an exploit, we could respond much earlier in the attack lifecycle and protect customers sooner. To find 0-days, you won’t find them in the big noisy buckets generating hundreds of thousands of hits per day. You need to live in the long tail of small data. Determined to make security crashes “loud”, I went into the strange and bizarre world of bit flips, one hit wonders, unreproducible race conditions, and 0-days.



Living in the Tail

By September 2008 we had built a system that screened millions of crashes for security exploits. Along the way I felt like I joined the world’s smallest profession—that of an exploit failure engineer. On September 25th a crash came in that got my attention--an exploit in netapi32.dll. This alone was not unusual. With MS06-040 we fixed a buffer overrun in netapi32!NetpwPathCanonicalize that received millions of crashes every day from botnets attempting (and failing because of /GS) to infect vulnerable PCs. This new crash was in very similar code, but in a different WER bucket. It was not in the top 100 or top 1,000 issues. It was bucket #45,000 with exactly 2 hits ever. This was living in the tail.



Enter !pwnd

What made this tiny bucket stand out? First, there was an exploit. We had written a custom debugger extension, colorfully called !pwnd (Windows debugger extensions are invoked by prefixing it with the 'bang' symbol). It found shellcode in the crash dump. I reviewed the shellcode and saw that it used an egghunt to find the payload. An egghunt in an exploit engineering technique used when a buffer overrun is constrained in terms of how much payload can be sent. The solution is to split the attack into two parts. In the first part, you interact with the target in a benign way, sending over data that contains your real payload preceded by a unique sequence of bytes (the "egg"), but without triggering the vulnerability. If you do this the right way, your payload sticks around in memory somewhere. Then you trigger the vulnerability and your constrained shellcode searches around in memory for the egg and jumps to it. This was the first exploit for netapi32.dll that used an egghunt. An eyebrow was raised.

The second thing unusual about this crash dump was not just the way it failed. It was the way it was succeeding before it crashed. I looked beyond the crashing thread to the other threads in the process. One of them revealed the attacker had already exploited the process and the shellcode was in the middle of downloading a payload using URLDownloadToFileA!

```
0:066> kPn99
 # ChildEBP RetAddr
00 0150f048 7c90e9ab ntdll!KiFastSystemCallRet(void)
01 0150f04c 7c8094e2 ntdll!ZwWaitForMultipleObjects(void)+0xc
...
08 0150f2b4 7813b13a urlmon!CUrlMon::StartBinding(
                int fBindToObject = 0,
                struct IBindCtx * pbc = 0x00185540,
                struct IMoniker * pmkToLeft = 0x00000000,
                struct _GUID * riid = 0x7813b394 {0000000c-0000-0000-c000-
000000000046},
                void ** ppvObj = 0x0150f2fc)+0x169
09 0150f2d8 7815bcd2 urlmon!CUrlMon::BindToStorage(
                struct IBindCtx * pbc = 0x00185540,
                struct IMoniker * pmkToLeft = 0x00000000,
                struct _GUID * riid = 0x7813b394 {0000000c-0000-0000-c000-
000000000046},
                void ** ppvObj = 0x0150f2fc)+0x49
0a 0150f31c 781b73d2 urlmon!CBaseBSCB::KickOffDownload(
```

```
                wchar_t * szURL = 0x0150f400 "hxxp://59.106.145.58/n2.exe")+0x193
0b 0150f330 781b7847 urlmon!CFileDownload::KickOffDownload(
                wchar_t * szURL = 0x0150f400 "hxxp://59.106.145.58/n2.exe")+0x2d
0c 0150f348 781b7a4a urlmon!URLDownloadToFileW(
                struct IUnknown * caller = 0x00000000,
                wchar_t * szURL = 0x0150f400 "hxxp://59.106.145.58/n2.exe",
                wchar_t * szFileName = 0x0150f380 "update.exe",
                unsigned long dwReserved = 0,
                struct IBindStatusCallback * callback = 0x00000000)+0x51
0d 0150f480 0150f5b8 urlmon!URLDownloadToFileA(
                struct IUnknown * caller = 0x00000000,
                char * szURL = 0x0150f6a1 "hxxp://59.106.145.58/n2.exe",
                char * szFileName = 0x0150f6d1 "update.exe",
                unsigned long dwReserved = 0,
                struct IBindStatusCallback * callback = 0x00000000)+0x109
```

While egghunts weren't new, this was a new flavor of shellcode for netapi32 exploits and clear evidence of a successful exploit. The final nail in the coffin was the version information in the crash dump. Netapi32.dll was fully patched! There seemed to be only one explanation for this: a new 0-day in the wild.

```
0:088> lmvm netapi32
    Image path: C:\WINDOWS\system32\netapi32.dll
    Image name: netapi32.dll
    Timestamp:        Thu Aug 17 05:28:27 2006 (44E460EB)
    File version:     5.1.2600.2976
```

Going in Reverse

Most of the time security researchers find a vulneraility then work to write an exploit. I was going in reverse: examining an exploit to determine the vulnerability, armed with only a forensic crash and no way to reproduce it. Had the exploit blown away the crucial clues in the buffer overrun itself? I studied the crash over and over. I looked at the source code for netapi32. Vulnerabilities are often obvious in hindsight but stubborn to reveal themselves at first. Here was my dilemma: if I could not find the vulnerability, despite having a clear exploit, we could not act. Engineers first need to understand the bug to fix it. If I could not find the bug, this opportunity would pass us by. I was stuck and the clock was ticking, so what did I do? I asked for help.

I brought the case to the manager of the MSRC security engineers, Andrew Roths. Andrew is a brilliant security engineer. The only problem was Andrew had, like me, looked at many crash dumps before and knew all too well the elusive nature of finding new issues in the tail of data. There are many dead ends, false starts, and fool's gold. I remember this time vividly. There was some tension in the air because I was pressing for help from a busy team on something the odds said were unlikely. After all, this code had been pored over by Microsoft and security researchers intensely already. Would a latent buffer overrun have escaped all those eyeballs? There are often confounding factors in a crash. You can debug a crash for hours confident you have a vulnerability dead to rights only to find out the machine had a buggy driver randomly flipping bits in memory. Life in the tail is harsh.

I remember the moment Andrew stopped by my office. I was ready for a debate about whether we should pull security engineers off of already confirmed vulnerability cases, potentially delaying them, to work on this elusive vulnerability. The look on his face told me something had changed. He said, "I found a vulnerabilty." What Andrew had done said a lot about himself. Like any good manager, Andrew wanted to shield his people from randomization. He took the case on himself. He thought this might be another dead end and wanted to prove it to bring the issue to closure. MSRC engineers care deeply about their analysis of vulnerabilities and his doggedness broke the case. He walked me through his analysis, and once we were sure, there was only one thing to do next.

"Wormable"

We walked down the hallway to the office of the crisis manager, Phillip. He was in the middle of a meeting with someone in his office. There must have been something about the expression on our faces because he turned to his visitor and abruptly said, "I'll talk with you later". We entered and I said, "we have a zero day." We explained the basic facts. We had a vulnerabilty, that could be exploited remotely, anonymously, that affected all versions of Windows. It was wormable and someone was already exploiting it. When you say the word 'wormable' to a crisis manager, it activates some latent response DNA. In his quiet way he went from 1 to 11 and immediately got to work mobilizing everyone. Scarred by Code Red and Blaster, when an issue is wormable, at Microsoft everyone shows up and works it as job #1.

At this point, I let the vulnerability response process drive the engineering fix and went back to my crashes. What else could telemetry tell us about activity and spread? Could I tell if this was targeted or already being used in mass infections? I used one of my exploit failure engineering skills of "dialing into the buckets." That is, finding all the

relevant crashes that represent this issue on every version of Windows, every SKU, every possible way the exploit could fail. On Windows Vista and Windows Server 2008 it always failed. The Security Development Lifecycle (SDL) process at Microsoft made sure those OS editions had full ASLR and DEP for the svchost.exe process and the RPC interface had been upgraded to require authentication—a classic example of a win from defense in depth, now fueling my radar.

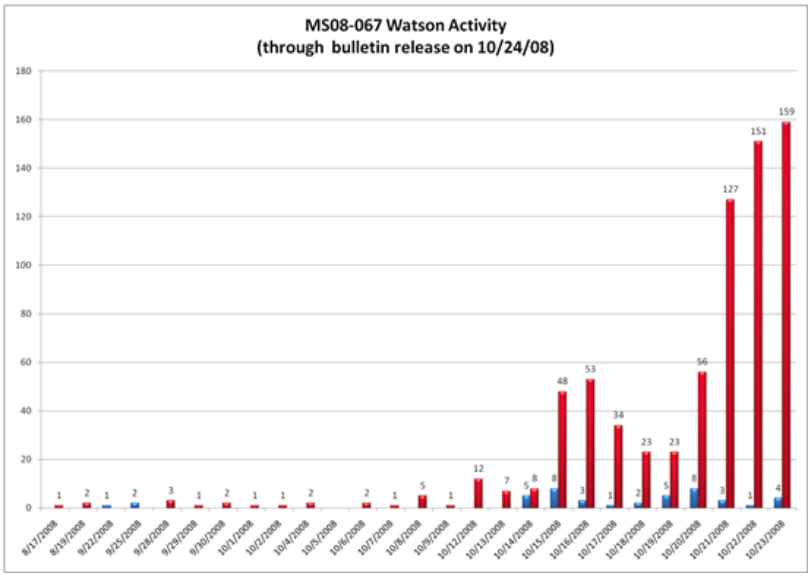
Attacker Impatience is the Defender's Reward

Why did the original exploit fail? The short answer is that the attacker made a mistake. The vulnerability was in canonicalizing the path component of a filename (e.g. turning \foo\..\bar\filename.ext into \bar\filename.ext). An attacker could trigger conditions that confused the way the code indexed the string and a buffer would get copied somewhere on the stack it should not. Normally this invalid input would cause the buggy code to crash, but the exploit authors figured out a way around this problem. Triggering the vulnerability would cause the code to sweep and search for a slash character ('\' or 0x5c) that was outside the bounds of the buffer. All that normally sits outside of the buffer are thread stack contents: local variables, return addresses, and pointers. The attacker needed to place a 0x5c byte nearby otherwise the code would sweep through the stack and crash the second it touched the guard page that abuts every thread stack. Their solution for this was to first call the vulnerable function with a benign input that had the slash character but would not trigger the vulnerability. This data would stay latent on the stack, like a ghost, the next time the function was called. This technique was perfectly reliable if Windows used the same thread for both requests. This happened nearly all the time. Nearly. In a quirk of fate, the Windows RPC thread pool handed the second request containing the exploit to a different thread—one that did not have the carefully placed slash character. The netapi32 code kept searching for it, eventually running off the end of the thread stack, hitting the guard page, and crashing the process with a stack overflow error (0xC00000fd). My interpretation is that the attack had succeeded and was downloading the payload, but the attacker got impatient or goofed and ran the exploit a second time. Patience is a virtue after all!

```
0:088> .ecxr
eax=064c2ffe ebx=064f005c ecx=064ff49a edx=064ff544 esi=064ff464 edi=064ff464
eip=5b878809 esp=064ff464 ebp=064ff474 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010203
netapi32!ConvertPathMacros+0x101:
5b878809 6683385c       cmp     word ptr [eax],5Ch      ds:0023:064c2ffe=????
0:088> .exr -1
ExceptionAddress: 5b878809 (netapi32!ConvertPathMacros+0x00000101)
  ExceptionCode: c00000fd (Stack overflow)
 ExceptionFlags: 00000000
NumberParameters: 2
  Parameter[0]: 00000000
  Parameter[1]: 064c2ffe
```

Crashes as Situational Awareness

After "dialing into the buckets" I provided situational awareness to the response process. By examining the locale information in the crashes we had some sense of geographic spread. It went from Japan to Malaysia, Philippines, Vietnam, and beyond. Below is the chart of activity. The barely visible blue bars are the crash activity. The red bars are the success activity.



Once MSRC was ready with the patch, we made the decision to ship it as an out-of-band update. Every patch release starts the clock in terms of copycat exploits. This is the one of those dilemmas in the MSRC business. Naturally you want to ship an update as soon as it's ready. But when you ship an out-of-band update, many IT teams aren't ready and this slows down how quickly systems are updated. Attackers don't hesitate to download the patch, diff it, and start building exploits, and defenders caught on their back foot may be at a disadvantage as they scramble to rearrange their schedule to deploy the update. We considered. Can you hold until Patch Tuesday when IT teams around the world are ready to receive and act? Or do you ship early and disrupt customers? The answer was clear. We had a critical vulnerability. We saw an uptick in activity. The patch was ready. We went out-of-band.

Windows Update

We were rushing to extinguish a fire before it turned into an inferno and Windows Update was our fire hose. The engineers who built Windows Update made a phenomenal inoculation system. While all we observed were targeted attacks, we knew a worm could be released at any moment with hundreds of millions of venerable systems to use as oxygen. MSRC used every megaphone it could to tell customers to patch. In a week, Windows Update patched 400 million PCs and untold millions more behind corporate firewalls with WSUS. I can't think of another system that can update 400 million of anything at a similar pace. Ask anyone about MS08-067 and most will mention Conficker. At this point in October, Conficker did not even exist. Conficker, as disruptive as it was, affected only the tail of computers that had not patched. Imagine what would have happened if Conficker had half a billion more systems to infect.

Exploit Reliability

The moment we went public, the attackers disappeared and I never saw them again. I soon saw crashes from people rediscovering the vulnerability, from vulnerability scanners probing for vulnerable systems, and soon enough, from botnet exploits. In the chart above, I show successes. WER only sees failures so how did we know about successes? When the exploit was successful it would post a reply to a webserver. Once the attack became public, security researchers from around the world wanted those logs and soon enough the logs were publicly available on the Internet, revealing the successful attacks. Correlating the successes with the failures we were able to conclude that we had found the exploit within its first 6 uses. The exploit was 95% reliable and our assumption that we were only seeing a shadow of real activity was correct. The decision to go out-of-band was the right one.

Final Thoughts

For me, MS08-067 will always be a special memory. Security is often a slog and offense gets the headlines. This story shows that defense can be just as creative and exciting. The ingenuity in the hunt, the thrill of discovery, and the desire to protect customers gets in people's blood and it's the reason security is more often a tribe than a job. That's the Defender's Mindset.

While I describe these events as a personal anecdote, pioneering exploit crash dump analysis would not have been possible without many incredible people, including: Fred Aaron, Jinwook Shin, Darren Canavor, Andrew Roths, Matt Thomlinson, David Grant, Nicola Cowie, Mark Wodrich, Damian Hasse, Roberto Bamberger, Kinshuman Kinshumann, Steven Wort, Vince Orgovan, Matt Miller, Ken Johnson, Adam Zabrocki, and, Kirk Glerum, who is widely considered the father of "Watson". Many others were involved in the response and I apologize for not naming all of you. To all our security partners and the security professionals at customers who worked this incident, you protected the world!

References

Debugging in the Very Large by Kirk Glerum et al., http://research.microsoft.com/apps/pubs/default.aspx?id=81176

Security Bulletin for MS08-067, https://technet.microsoft.com/en-us/library/security/ms08-067.aspx

Comments

Jake Williams (@malwarejake) 26 Sep 2015 8:12 PM

You mention you caught the exploit in it's first six uses. That seems to imply that you were able to see the web server (or at least server logs) to know when the exploit was successful. Also, that assumes that the original exploit always used the same payload, which seems a dangerous assumption.

That being said, this is a great read. Thanks for posting!

Ray Lewis (@lewiray) 26 Sep 2015 9:39 PM

I agree with @malwarejake. Great read! Thanks for sharing.

John Lambert (MSTIC) 26 Sep 2015 10:03 PM

@Jake, You're right that we'll never know for sure. WER has a record of every bucket it has has ever seen (going back 15 years now). There were no buckets for this issue before this time. So, either they used it and no one clicked Send Error Report, things worked perfectly for some set of attempts, or suddenly it got unreliable. Unless the attackers reply on twitter or on this blog :), we'll won't know. In any case, we took a favorite toy away from them.

Reuben Scratton 27 Sep 2015 12:06 AM

Well that's far and away the most incredible piece of debugging I've ever read about! Thanks for sharing it.

uh? 27 Sep 2015 1:55 AM

why does this look like a "w10 telemetry is gud" advertisement to me?

Yuhong Bao 27 Sep 2015 2:09 AM

To be honest, SQM and crash reporting are very different. They both dates before Win10 though.

Old Timer 27 Sep 2015 2:47 AM

Fantastic read. Kept the wife waiting while I read it, she's upset now!

As someone who used to likely gave your team hell (lived and breathed win exploits 95-2007) I have to say writing and finding exploits seems just as interesting to me as what you were/are doing. Given different circumstance I could have easily been in your shoes. Also have to commend Microsoft and you for this find, as I know when I was involved in the community the consensus was Microsoft was not actively trying to find exploits (only react), or monitoring watson reports. Some were extra careful and used completely isolated VM's and even refused to use Visual Studio for fear it was sending code samples back, but they were mostly thought of as paranoid. To think you actually were not only monitoring logs for exploits, but had the skills to spot an outlier/out-of-band exploit is extremely impressive. Anyways I threw in the towel once ASLR had most of the kinks worked out and it was clear 64bit was the future.

j 27 Sep 2015 4:44 AM

Thanks for a nice read. The description of the process of how the crash reports are used was very enlightening. There is one issue that may bring this well-designed mechanism to grind: I do not trust Microsoft. Many people do not trust Microsoft and happily hit "Cancel" whenever they are asked to send any information back to Mother Ship.
I've been a long-time Windows user since the times of Win 3.0, however MS' disregard for security (in the earlier times), and then for users' privacy made me look for alternatives. Since that time MS did a tremendous work to regain trust, however it was destroyed again by introduction of mandatory user surveillance mechanisms in Windows 10. I mistrust MS again to the point, that I will rather risk running unsupported version of Windows on an isolated VM, than installing a spy tool on my computer. Does anybody in MS learn from past mistakes? Why do you insist on destroying foundations of trust, on building which you spent so much time and money?

AO 27 Sep 2015 2:36 PM

great writeup.

It curious though how Microsoft and other security vendors know if such RCE's are in the "wild"? state-sponsored and other hackers with unlimited resources probably have such "weapon" for their APT's, so what's the solution for the long-term??? just thinking out loud...

Old Hack 27 Sep 2015 4:04 PM

Thank you. I saw this come up on my radar, as an out of band, and rallied the troops. APB. I ran around patching all the PCs I could touch. I recently turned off Telemetry data, as I can see what is sent, but now I see exactly how helpful it is. I am turning telemetry data back on for good, and sending crash reports. Thanks again, really.
(started with DOS 1.1, I have seen and used EVERY version of windows...)

Josh Phillips 28 Sep 2015 7:23 AM

Great story John. I was one of the first responders on MMPC when Conficker came out (In fact I coined the term). I have one question though... There was an internal presentation on the story you just told, so why write about it publicly now?

John Lambert (MSTIC) 28 Sep 2015 1:40 PM

@OldTimer, thanks for your personal story. There are a lot of people working hard behind the scenes and it's not always possible to broadcast their work. The Visual Studio idea you mention is funny, but no we don't do that.

@OldHack, appreciate it!

@Josh, Sept 25th is the anniversary of when I looked at the dump. Every now and then I see people tweet about Conficker or MS08-067 and thought I may as well write down the story. Though I wrote it from my point of view, a team worked hard on building that system and it was of my most enjoyable projects. One day maybe I'll write about the other 0-days we snatched with it. Besides, we can always use more stories on defense for a change :)