

Notes on Windows Error Reporting

@0xdabbad00 (Dabbadoo)



0xdabbad00.com

2014-01-08

Introduction	2
Key Points	2
About me	2
Digging deep	2
What is Windows Error Reporting?	3
Sample Crash Report	3
Event logs	4
Sample Crash Upload	4
Stage One	5
Stage Two	5
Stage Three	5
Stage Four	6
Stage Five?	6
CEIP	6
Dangers of Error Reporting	7
Passive only	7
If you could MiTM SSL	7
Disabling WER	7
Disabling WER for home users	8
Disabling WER via GPO	8
Conclusion	8
Recommendations to Microsoft	8
Recommendations to Software Companies	8
Recommendations to Windows Users and Admins	8
Appendix	9
windbg output from calc.exe crash	9
Event Logs	10
Testing on your own	11
Data sent in the .cab	12

Introduction

Recent [news](#)¹ discussed the use of data leaked through Windows Error Reporting (WER). These notes attempt to better explain what is and is not possible and to generalize the attack classes for all error reporting. It's important to note where these problems apply to any error reporting (not just Microsoft's WER) and where any entity (not just the US gov) could leverage this knowledge.

Key Points

1. Only the initial beacon for WER is unencrypted. All follow-on communication, including any crash dump is sent encrypted over HTTPS.
2. The initial, unencrypted beacon, does not provide any memory offsets, and can not be used as an ASLR bypass/infoleak.
3. The [US Configuration Baseline for Windows](#)², provided by NIST, has always recommended that Windows Error Reporting be disabled and documents how to do so.

About me

I have been developing a product for [Parsons](#)³ for the past few months that leverages Microsoft's built in error reporting, namely it's Corporate Error Reporting, which enterprises can use to collect and view the errors occurring on their networks instead of these reports being sent to Microsoft. My product analyzes the crash dumps it collects to identify exploit attempts against the systems on the network. Check out our booth at ShmooCon and ask about the product called "Cran: Crash Analyzer".</Advertising>

Through this work, I have a decent understanding of WER.

Digging deep

If you want to do something cool in infosec for Windows, but aren't sure where to start, I recommend picking a random document off Microsoft's [Technical Documents](#)⁴ page, reading it thoroughly and coming up with cool ideas on how to use it. Examples:

- [Responder](#)⁵ from SpiderLabs uses knowledge from [\[MS-LLMNR\]: Link Local Multicast Name Resolution \(LLMNR\) Profile](#)⁶
- Stuxnet's .LNK exploit ([CVE-2010-2568](#)⁷) could be discovered through a close read of [\[MS-SHLLINK\]: Shell Link \(.LNK\) Binary File Format](#)⁸

Dave Aitel explained this concept in his post [The Squeeze](#)⁹ where he states:

So in general my feeling on 0days is that they come from new attack surfaces. Finding those new attack surfaces takes a lot of initial time [...] At some point the team crosses a threshold and then the cracks start forming and [...] you're basically drowning in 0day at that point, and it's just a matter of picking up the pieces you want to use to construct your exploit.

¹<http://www.spiegel.de/international/world/the-nsa-uses-powerful-toolbox-in-effort-to-spy-on-global-networks-a-940969-2.html>

²usgcb.nist.gov/usgcb/documentation/USGCB-Windows-Settings.xls

³<http://www.parsons.com/>

⁴<http://msdn.microsoft.com/en-us/library/jj712081.aspx>

⁵<https://github.com/SpiderLabs/Responder>

⁶<http://msdn.microsoft.com/en-us/library/dd240328.aspx>

⁷<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568>

⁸<http://msdn.microsoft.com/en-us/library/dd871305.aspx>

⁹<https://lists.immunityinc.com/pipermail/dailydave/2013-October/000504.html>

In this case, you could have read

1. [\[MS-CER\]: Corporate Error Reporting Version 1.0 Protocol](#)¹⁰
2. [\[MS-CER2\]: Corporate Error Reporting V.2 Protocol](#)¹¹

Corporate Error Reporting (CER, instead of WER) just means that crash dumps are sent to a local WER server, instead of Microsoft's, if you buy (or build) a product that can receive those. It does work slightly differently and some aspects are OS dependent.

What is Windows Error Reporting?

Windows Error Reporting was introduced in Windows XP. When an application crashes, the user is prompted to send their crash dump to Microsoft. Microsoft uses these reports to identify and fix bugs. This information could help prioritize bugs (largest number of users affected) or help reproduce the bugs (crash only occurs when a third-party toolbar has been installed).

Since those early days the WER protocol has become a catch-all for any sort of telemetry data to be sent to Microsoft. The largest non-crash source of this data is Microsoft's CEIP (Customer Experience Improvement Program), which is what was informing Microsoft of USB's being plugged in. Websense noted this USB info leakage in [their report](#)¹².

WER is the default crash handler for Windows, so most applications, such as Internet Explorer, Adobe Reader, Skype, and anything else that hasn't taken special care to avoid it, will have their crash reports sent to Microsoft. Microsoft does provide access (for a price) to vendors that wish to see the crash reports for their applications. Some applications do not use WER. Notably, Google Chrome, Mozilla Firefox, and OpenOffice have their own crash reporting.

The protocol is a conversation, that starts with the client (your computer) sending an HTTP (unencrypted) message to the server. This is done through `werfault.exe`, not the crashed application, because the crashed application is not trust-worthy because it has crashed. If the server is interested, it asks for more info, and the rest of the conversation continues over HTTPS (encrypted). Throughout this conversation you should be asked to consent to sending different information.

Nothing is sent ever for this crash if you don't click "Check online for a solution and close the program". If you did consent and Microsoft wanted more info, you'd get another message asking for consent again to the next set of info to send.

This is the process in theory, but some systems end up having default consent.

Sample Crash Report

Let's see a quick example of these crash reports. Let's crash `calc.exe` and see what happens. This trick works on Windows 7 and 8. Bring up Fiddler or Wireshark to watch your network traffic (Fiddler is preferred so we can watch the HTTPS traffic later). Follow the [instructions](#)¹³ from Kris Kaspersky on crashing calc by switching to scientific mode in calc.exe, typing "1/255", enter, and click the "F-E" button.

Consent to any WER pop-up, and you should then see an HTTP request to `http://watson.microsoft.com`. Mine looks like:

¹⁰<http://msdn.microsoft.com/en-us/library/cc224574.aspx>

¹¹<http://msdn.microsoft.com/en-us/library/dd942170.aspx>

¹²<http://community.websense.com/blogs/securitylabs/archive/2013/12/29/dr-watson.aspx>

¹³<http://nezumi-lab.org/blog/?p=239>

```
GET http://watson.microsoft.com/StageOne/calc_exe/6_1_7601_17514/4ce7979d/
ntdll_dll/6_1_7601_18247/521ea91c/c00000fd/00052c26.htm
?LCID=1033&OS=6.1.7601.2.00010100.1.0.1.17514&SM=innotek%20GmbH&SPN=VirtualBox
&BV=VirtualBox&MID=EADE9A2C-C206-40B9-9986-13D4B1449226
```

In order to break this down, you can run `calc.exe` from windbg and then run `!analyze -v` on the crash. This output is included in the [Appendix](#). This analysis shows the following line:

```
WATSON_STAGEONE_URL: http://watson.microsoft.com/StageOne/calc_exe/6_1_7601_17514/4ce7979d/
ntdll_dll/6_1_7601_18247/521ea91c/c00000fd/00089ecf.htm?Retriage=1
```

This matches the URL that was contacted, except the final “file” (00089ecf). This can be broken down as follows:

- GET request made to `http://watson.microsoft.com/StageOne/`
- `calc_exe` is the name of the process (all strings are scrubbed).
- `6_1_7601_17514` Version as identified in the PE resources of the process file (`calc.exe`).
- `4ce7979d` Link time stamp from the PE header. Note that the Websense report misidentified this as a “Crash location”. This correction is important as a leak of a memory address would be bad for ASLR.
- `ntdll_dll/6_1_7601_18247/521ea91c` Same info for module believed to be responsible
- `c00000fd` Exception code, in this case “Stack overflow”.
- `00052c26` Exception offset. In this crash, EIP is at `77f49ecf` (`ntdll!RtlpCreateSplitBlock+0x4f2`), and the base of `ntdll` was `77ec0000`, which means that `77f49ecf - 77ec0000 = 00052c26`.
- `LCID=1033&OS=6.1.7601.2.00010100.1.0.1.17514` My language setting and OS.
- `&SM=innotek%20GmbH&SPN=VirtualBox&BV=VirtualBox&`: The system manufacturer, system product name, and bios version for my system, as found in the values in `HKLM\SYSTEM\ControlSet001\Control\SystemInformation`.
- `MID=EADE9A2C-C206-40B9-9986-13D4B1449226` MachineID as found in `HKLM\Software\Microsoft\Windows\Windows Error Reporting\MachineID`. This value does not exist anywhere else in the registry, and as best as I can tell is an ID that is only used by Windows Error Reporting, so Microsoft knows crashes are coming from the same computer, but can’t associate this with anything else.

The response is:

```
Bucket=-406313712
BucketTable=1
Response=1
```

A negative bucket number is the server’s way of saying “Go away, I don’t care.”

Event logs

Much of this same information can be seen in the Windows Event Log. My two event logs are shown in the [Appendix](#).

Sample Crash Upload

Let’s see an example of what happens when Microsoft does care about a crash. Oddly, at one point in my testing, Microsoft suddenly became interested in my `calc` crash on a Windows 7 x64 system and the following conversation ensued:

Stage One

Unencrypted GET. The wer client uses the User-Agent “MSDW”.

```
GET http://watson.microsoft.com/StageOne/calc_exe/6_1_7600_16385/4a5bc9d4/ntdll_dll/
6_1_7601_18247/521eaf24/c000041d/0000000000054f01.htm?LCID=1033&OS=
6.1.7601.2.00010100.1.0.48.17514&SM=innotek%20GmbH&SPN=VirtualBox&BV=VirtualBox
```

Response

```
HTTP/1.1 404 Not Found
Cache-Control: private
Content-Type: text/html
Server: Microsoft-IIS/8.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
```

404 response html followed. Previously the client received a 200 with “Bucket=-406313712” which meant it should stop, but because we received a 404 this time, the client seems to have broken logic that makes it think the server must be interested in more information.

Stage Two

Encrypted GET.

```
GET https://watson.microsoft.com/dw/stagetwo64.asp?
szAppName=calc.exe&szAppVer=6.1.7600.16385&szAppStamp=4a5bc9d4&szModName=ntdll.dll&szModVer=
6.1.7601.18247&szModStamp=521eaf24&ExceptionCode=c000041d&offset=0000000000054f01&LCID=1033&
OS=6.1.7601.2.00010100.1.0.48.17514&SM=innotek%20GmbH&SPN=VirtualBox&BV=VirtualBox
```

Response

```
iData=1
DumpFile=/Upload/iCab/1fc1d68925e84665a94470d7ef4891a9-caed5b4c81fa8c999d50011c597f5921-4-
134701747-AppCrash64-6-1-7601-2.cab
DumpServer=watson.microsoft.com
ResponseServer=watson.microsoft.com
ResponseURL=/dw/StageFour64.asp?iBucket=134701747&szCab=1fc1d68925e84665a94470d7ef4891a9.cab
&EventType=AppCrash64&BucketHash=caed5b4c81fa8c999d50011c597f5921
Bucket=134701747
BucketTable=4
Response=1
```

Stage Three

Encrypted upload of the .cab file.

```
PUT https://watson.microsoft.com/Upload/iCab/1fc1d68925e84665a94470d7ef4891a9-
caed5b4c81fa8c999d50011c597f5921-4-134701747-AppCrash64-6-1-7601-2.cab
```

The .cab file can be extracted with 7-zip and in my case, contained 3 files:

1. **AppCompat.txt** Information about some of the PE files (See the [Appendix](#)).
2. **WERInternalMetadata.xml** Mostly the same data in the StageOne GET request, but formatted better.
3. **minidump.mdmp** A minidump for the process. These contain some data from the stack, information about the loaded modules, register data for each thread, and a bit of the memory relevant to the instruction pointers for the threads.

The cab file is created before the StageOne GET is even sent.

Response

Responds with a 200

Stage Four

```
GET https://watson.microsoft.com/dw/StageFour64.asp?iBucket=134701747&szCab=
1fc1d68925e84665a94470d7ef4891a9.cab&EventType=AppCrash64&BucketHash
=caed5b4c81fa8c999d50011c597f5921
```

Response

```
iCab=-164892517
```

Stage Five?

There is a stage 5 possible, as evidenced by the undocumented registry setting possible called “DoNotUseStage5”. Stage 5 I believe should allow for additional data to be requested from the system.

CEIP

The Customer Experience Improvement Program (CEIP) is responsible for a range of telemetry data, such as information about USB’s being plugged in, applications being installed, windows updates failing, and many other events. This telemetry data is passed back to Microsoft through the same error reporting channel.

You can get a feel for what CEIP does by looking at the tasks in the Task Scheduler by running “taskschd.msc”. Expand the trees for

- “Task Scheduler Library”
- “Microsoft”
- “Windows”

The tasks for “Application Experience” and “Customer Experience Improvement Program” are relevant to CEIP. I haven’t fully understood what all of these do.

If you’ve never seen the task scheduler before, it’s a treasure trove of weird tasks your system runs at odd times, and is probably one of the reasons why suddenly your system’s disk start spinning late at night: Either the defrag task kicked off or you’ve got BadBios.

Dangers of Error Reporting

Passive only

As described in the news story, it seems that traffic bound for `watson.microsoft.com` was just passively watched. This allows you to identify the applications in use on the network and version numbers for possible later exploitation. The other big issue is all the other non-crash data that gets sent to the WER server via CEIP.

If you could MiTM SSL

As we've seen with the various CA issues that Google has exposed, groups have been able to MiTM SSL traffic, and if done intelligently, so you didn't MiTM `google.com`, you could probably get away with doing MiTM on `watson.microsoft.com`. Being able to sniff MiTM is a very difficult requirement though, and honestly if you can do it, then there are smarter things you could do, but let's explore some ideas here as perhaps there is some flaw in WER that I have not uncovered.

The Alley Oop

One of the big difficulties with exploiting software nowadays is ASLR. If you're trying to exploit something that doesn't have a Turing complete language in it, like JavaScript, you're going to have problems due to ASLR. So what you could do is:

1. Throw an exploit, or just a bug, at a process to crash it.
2. Cause the client to send a crash dump
3. Analyze the crash dump to determine the memory offsets you care about and modify your exploit to use those.
4. Throw your modified exploit at the victim with the known memory offsets.

A common confusion about ASLR is that some of the "randomized" items, are only randomized on reboots. So the same process started twice will have system DLL's at the same locations, even if they "ASLR-enabled" (dynamic based).

Greedy debugging

Corporate Error Reporting is capable of much more than simply retrieving a minidump, and I assume WER must work the same. These additional capabilities are:

- You can retrieve a full crash dump (all the memory for that process) instead of just a minidump. This would allow you to see passwords, and potentially browser history, emails, or whatever other data exists in memory at the time of the crash.
- You can retrieve arbitrary files and registry keys. According to the CER 2.0 spec, you can pull back any file or registry key. However, if you're Microsoft, or have enough control over the network, including being able to MiTM SSL, to act as Microsoft, you could just send down an arbitrary Windows Update to install your malware to collect arbitrary files.
- You can run arbitrary WQL queries. This lets you retrieve other data about the system. As an example, check out this [page](#)¹⁴.

Disabling WER

The [United States Government Configuration Baseline \(USGCB\)](#)¹⁵ has always recommended disabling WER. This is seen in [CCE-10441-4](#)¹⁶ with the reason "To lower the risk of a user unknowingly exposing sensitive

¹⁴<http://www.codeproject.com/Articles/46390/WMI-Query-Language-by-Example>

¹⁵<http://usgcb.nist.gov/>

¹⁶usgcb.nist.gov/usgcb/documentation/USGCB-Windows-Settings.xls

data.” If you admin Windows systems, you should be following these guidelines.

Disabling WER for home users

Add a DWORD registry value called “Disabled” with a value of “1” to

```
HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Disabled
```

Disabling WER via GPO

Create a new Group Policy Object (GPO) and expand the following branches:

1. Policies
2. Administrative Templates
3. Windows Components
4. Windows Error Reporting

For Windows XP and 2003: Double-click “Configure Error Reporting”, select “Disable”.

For Vista and up: Double-click “Disable Windows Error Reporting”, select “Enable”.

Conclusion

Recommendations to Microsoft

Microsoft did a good job of ensuring the most valuable data is sent encrypted. They also provide a large assortment of configurable options, many of which are built into Group Policy for easy network configuration.

The two biggest problems with WER are:

- The first message is sent unencrypted.
- Data that wasn’t relevant to error reporting was show-horned into Windows Error Reporting.

Windows Error Reporting should be modified so that all beacons are sent encrypted.

Recommendations to Software Developers

Any telemetry data, including error reporting, should be sent encrypted. Ideally, all network communications should be encrypted. Be cautious what capabilities you build into error reporting that could potentially be abused. The more dangerous capabilities of Windows Error Reporting were protected via encrypted communication.

Recommendations to Windows Users and Admins

Disable Windows Error Reporting until a more secure communication is built. Follow the the United States Government Configuration Baseline (USGCB) for locking down your systems.

Appendix

windbg output from calc.exe crash

Output from windbg when !analyze -v is run on a crash of calc.exe.

```

0:000> !analyze -v
*****
*                               Exception Analysis                               *
*****
Failed calling InternetOpenUrl, GLE=12029

FAULTING_IP:
calc!putnum+1a7
010070e3 e993e8ffff      jmp     calc!putnum+0x32e (0100597b)

EXCEPTION_RECORD: ffffffff -- (.exr 0xffffffffffffffff)
ExceptionAddress: 77f49ecf (ntdll!RtlpCreateSplitBlock+0x000004f2)
ExceptionCode: c00000fd (Stack overflow)
ExceptionFlags: 00000000
NumberParameters: 2
  Parameter[0]: 00000001
  Parameter[1]: 00032ffc
FAULTING_THREAD: 00001764
DEFAULT_BUCKET_ID: STACK_OVERFLOW
PROCESS_NAME: calc.exe
ERROR_CODE: (NTSTATUS) 0xc00000fd - A new guard page for the stack cannot be created.
EXCEPTION_CODE: (NTSTATUS) 0xc00000fd - A new guard page for the stack cannot be created.
EXCEPTION_PARAMETER1: 00000001
EXCEPTION_PARAMETER2: 00032ffc
RECURRING_STACK: From frames 0xb to 0xb
MOD_LIST: <ANALYSIS/>
NTGLOBALFLAG: 70
APPLICATION_VERIFIER_FLAGS: 0
PRIMARY_PROBLEM_CLASS: STACK_OVERFLOW
BUGCHECK_STR: APPLICATION_FAULT_STACK_OVERFLOW
LAST_CONTROL_TRANSFER: from 77f15d5a to 77f49ecf

STACK_TEXT:
00033028 77f15d5a 00206400 77f15d04 00000001 ntdll!RtlpCreateSplitBlock+0x4f2
00033114 77f15ae0 0000009c 000000b8 0020634a ntdll!RtlpAllocateHeap+0xb5d
00033198 77f85f63 00130000 5014016b 0000009c ntdll!RtlAllocateHeap+0x23a
000331e4 77f4a40a 00130000 5014016b 0000009c ntdll!RtlDebugAllocateHeap+0xb5
000332c8 77f15ae0 0000009c 00000000 00000000 ntdll!RtlpAllocateHeap+0xc4
0003334c 0dce7751 00130000 40140068 0000009c ntdll!RtlAllocateHeap+0x23a
00033398 0100213f 00000040 0000009c 000333d4 KERNELBASE!LocalAlloc+0x5f
000333a8 0100715c 0000009c 00033448 ffffffff calc!_createnum+0x4b
000333d4 01007112 00033418 0013bb00 0000000a calc!_divnum+0x3e
000333e8 010252a0 00033418 0013bb00 0000000a calc!divnum+0x34
0003341c 010070e3 0006e3c0 0000001f 00000001 calc!putnum+0xd4
00033450 010070e3 0006e3c0 0000001f 00000001 calc!putnum+0x1a7
00033484 010070e3 0006e3c0 0000001f 00000001 calc!putnum+0x1a7
000334b8 010070e3 0006e3c0 0000001f 00000001 calc!putnum+0x1a7
000334ec 010070e3 0006e3c0 0000001f 00000001 calc!putnum+0x1a7
...

```

```
FOLLOWUP_IP:
calc!putnum+1a7
010070e3 e993e8ffff      jmp      calc!putnum+0x32e (0100597b)

SYMBOL_STACK_INDEX:  b
SYMBOL_NAME:  calc!putnum+1a7
FOLLOWUP_NAME:  MachineOwner
MODULE_NAME:  calc
IMAGE_NAME:  calc.exe
DEBUG_FLR_IMAGE_TIMESTAMP:  4ce7979d
STACK_COMMAND:  ~0s ; kb
FAILURE_BUCKET_ID:  STACK_OVERFLOW_c00000fd_calc.exe!putnum
BUCKET_ID:  APPLICATION_FAULT_STACK_OVERFLOW_calc!putnum+1a7
WATSON_STAGEONE_URL:  http://watson.microsoft.com/StageOne/calc_exe/6_1_7601_17514/4ce7979d/
ntdll_dll/6_1_7601_18247/521ea91c/c00000fd/00089ecf.htm?Retriage=1

Followup: MachineOwner
-----
```

Event Logs

After crashing calc, two event logs showed up:

```
Faulting application name: calc.exe, version: 6.1.7601.17514, time stamp: 0x4ce7979d
Faulting module name: ntdll.dll, version: 6.1.7601.18247, time stamp: 0x521ea91c
Exception code: 0xc00000fd
Fault offset: 0x00052c26
Faulting process id: 0x14c8
Faulting application start time: 0x01cf0c18be05df7a
Faulting application path: C:\Windows\system32\calc.exe
Faulting module path: C:\Windows\SYSTEM32\ntdll.dll
Report Id: fd9004d1-780b-11e3-a4d4-08002772e4f5
```

```
Fault bucket 0, type 0
Event Name: APPCRASH
Response: Not available
Cab Id: 0
```

Problem signature:

```
P1: calc.exe
P2: 6.1.7601.17514
P3: 4ce7979d
P4: ntdll.dll
P5: 6.1.7601.18247
P6: 521ea91c
P7: c00000fd
P8: 00052c26
P9:
P10:
```

Attached files:

```
C:\Users\user\AppData\Local\Temp\WERF154.tmp.WERInternalMetadata.xml
```

These files may be available here:

```
C:\Users\user\AppData\Local\Microsoft\Windows\WER\ReportArchive
\AppCrash_calc.exe_f3f7aab2e17f591e341b7f5421d4b9a9941592_0608fd1c
```

Analysis symbol:

```
Rechecking for solution: 0
Report Id: fd9004d1-780b-11e3-a4d4-08002772e4f5
Report Status: 0
```

Testing on your own

Assuming you're not lucky enough to be one of the chosen crashes, you can do some minimal testing by spoofing the watson server to your system. These results are not very good, I apologize, but may help someone else figure out what should happen.

Directions from the CER spec

I edit C:\windows\system32\drivers\etc\hosts and add the following line:

```
127.0.0.1      watson.microsoft.com
```

Then create a file with the following contents (this comes from the CER 2.0 document):

```
Response=http://oca.microsoft.com/resredirect.aspx?SID=32
Bucket=500
BucketTable=5
DumpFile=1.cab
```

I then use cygwin to run netcat with the following:

```
$ cat wer.txt | nc -l 80
```

Now crash calc again and your client will make an HTTPS connection to `wer.microsoft.com`. That line with `oca.microsoft.com` in it is ignored, but required.

My system makes a call to

```
POST https://wer.microsoft.com/Responses/v1.0/32/1033.9/6.1.7601.2.00010100.1.0/16777217/innotek%20GmbH
```

The contents of the POST are:

```
Bucket=500&BucketTable=5&Response=http://oca.microsoft.com/resredirect.aspx?SID=32&DisplayType=0&SystemManufacturer=innotek GmbH&SystemProductName=VirtualBox
```

The server responds with just an error message saying amongst other things “Microsoft is currently researching the cause of this problem. Please continue to submit all Windows problem reports.” However, the user does not see this.

Disabling HTTPS

An alternative result can be seen by creating a DWORD registry value, set to 1, named:

```
HKLM\Software\Microsoft\Windows\Windows Error Reporting\Debug\DoNotUseHttps
```

Have the initial message 404, and this will cause the client to then attempt

```
GET http://watson.microsoft.com/dw/stagetwo.asp?szAppName=calc.exe&szAppVer=6.1.7601.17514&szAppStamp=4ce7979d&szModName=ntdll.dll&szModVer=6.1.7601.18247&szModStamp=521ea91c&ExceptionCode=c00000fd&offset=00052c26&LCID=1033&OS=6.1.7601.2.00010100.1.0.1.17514&SM=innotek%20GmbH&SPN=VirtualBox&BV=VirtualBox&MID=EADE9A2C-C206-40B9-9986-13D4B1449226
```

This is similar to the successful `.cab` upload seen earlier, but I have forced the server to use HTTP. You could set up a fake HTTPS server and spoof things completely with the results seen earlier, but that requires forcing your system to trust the fake certificate you create for `watson.microsoft.com`.

Data sent in the `.cab`

In addition to the `.mdmp` (minidump) file, there is also an `AppCompat.txt` and `WERInternalMetadata.xml` file.

`AppCompat.txt`

```
<?xml version="1.0" encoding="UTF-16"?>
<DATABASE>
<EXE NAME="SYSTEM INFO" FILTER="CMI_FILTER_SYSTEM">
  <MATCHING_FILE NAME="kernel32.dll" SIZE="1161216" CHECKSUM="0xED2A37B2"
  BIN_FILE_VERSION="6.1.7601.18229" BIN_PRODUCT_VERSION="6.1.7601.18229"
  PRODUCT_VERSION="6.1.7601.18015" FILE_DESCRIPTION="Windows NT BASE API Client DLL"
  COMPANY_NAME="Microsoft Corporation" PRODUCT_NAME="Microsoft Windows Operating System"
  FILE_VERSION="6.1.7601.18015 (win7sp1_gdr.121129-1432)" ORIGINAL_FILENAME="kernel32"
  INTERNAL_NAME="kernel32" LEGAL_COPYRIGHT="Microsoft Corporation. All rights reserved."
  VERDATEHI="0x0" VERDATELO="0x0" VERFILEOS="0x40004" VERFILETYPE="0x2" MODULE_TYPE="WIN32"
  PE_CHECKSUM="0x11EB53" LINKER_VERSION="0x60001" UPTO_BIN_FILE_VERSION="6.1.7601.18229"
  UPTO_BIN_PRODUCT_VERSION="6.1.7601.18229" LINK_DATE="08/02/2013 02:16:22"
  UPTO_LINK_DATE="08/02/2013 02:16:22" EXPORT_NAME="KERNEL32.dll"
  VER_LANGUAGE="English (United States) [0x409]" EXE_WRAPPER="0x0"
  FILE_ID="00007244ae695f8e5a730857781635acb2969f15c594"
  PROGRAM_ID="0000f519feec486de87ed73cb92d3cac802400000000" />

  <MATCHING_FILE NAME="ntdll.dll" SIZE="1732032" CHECKSUM="0x7EC8079C"
  BIN_FILE_VERSION="6.1.7601.18247" BIN_PRODUCT_VERSION="6.1.7601.18247"
  PRODUCT_VERSION="6.1.7600.16385" FILE_DESCRIPTION="NT Layer DLL"
  COMPANY_NAME="Microsoft Corporation" PRODUCT_NAME="Microsoft Windows Operating System"
  FILE_VERSION="6.1.7600.16385 (win7_rtm.090713-1255)" ORIGINAL_FILENAME="ntdll.dll.mui"
  INTERNAL_NAME="ntdll.dll" LEGAL_COPYRIGHT="Microsoft Corporation. All rights reserved."
  VERDATEHI="0x0" VERDATELO="0x0" VERFILEOS="0x40004" VERFILETYPE="0x2" MODULE_TYPE="WIN32"
  PE_CHECKSUM="0x1A875F" LINKER_VERSION="0x60001" UPTO_BIN_FILE_VERSION="6.1.7601.18247"
  UPTO_BIN_PRODUCT_VERSION="6.1.7601.18247" LINK_DATE="08/29/2013 02:17:08"
  UPTO_LINK_DATE="08/29/2013 02:17:08" EXPORT_NAME="ntdll.dll"
  VER_LANGUAGE="English (United States) [0x409]" EXE_WRAPPER="0x0"
  FILE_ID="00002b1dc5de7a39b95a6c4c2da4645ca47597b16ab5"
  PROGRAM_ID="0000f519feec486de87ed73cb92d3cac802400000000" />
</EXE>
</DATABASE>
<EXE NAME="ntdll.dll" FILTER="CMI_FILTER_THISFILEONLY">
  <MATCHING_FILE NAME="ntdll.dll" SIZE="1732032" CHECKSUM="0x7EC8079C"
  BIN_FILE_VERSION="6.1.7601.18247" BIN_PRODUCT_VERSION="6.1.7601.18247"
  PRODUCT_VERSION="6.1.7600.16385" FILE_DESCRIPTION="NT Layer DLL"
  COMPANY_NAME="Microsoft Corporation" PRODUCT_NAME="Microsoft Windows Operating System"
  FILE_VERSION="6.1.7600.16385 (win7_rtm.090713-1255)" ORIGINAL_FILENAME="ntdll.dll.mui"
  INTERNAL_NAME="ntdll.dll" LEGAL_COPYRIGHT="Microsoft Corporation. All rights reserved."
  VERDATEHI="0x0" VERDATELO="0x0" VERFILEOS="0x40004" VERFILETYPE="0x2" MODULE_TYPE="WIN32"
  PE_CHECKSUM="0x1A875F" LINKER_VERSION="0x60001" UPTO_BIN_FILE_VERSION="6.1.7601.18247"
  UPTO_BIN_PRODUCT_VERSION="6.1.7601.18247" LINK_DATE="08/29/2013 02:17:08"
  UPTO_LINK_DATE="08/29/2013 02:17:08" EXPORT_NAME="ntdll.dll"
  VER_LANGUAGE="English (United States) [0x409]" EXE_WRAPPER="0x0"
  FILE_ID="00002b1dc5de7a39b95a6c4c2da4645ca47597b16ab5"
  PROGRAM_ID="0000f519feec486de87ed73cb92d3cac802400000000" />
</EXE>
</DATABASE>
```

WERInternalMetadata.xml

```
<?xml version="1.0" encoding="UTF-16"?>
<WERReportMetadata>
  <OSVersionInformation>
    <WindowsNTVersion>6.1</WindowsNTVersion>
    <Build>7601 Service Pack 1</Build>
    <Product>(Ox30): Windows 7 Professional</Product>
    <Edition>Professional</Edition>
    <BuildString>7601.18247.amd64fre.win7sp1_gdr.130828-1532</BuildString>
    <Revision>1130</Revision>
    <Flavor>Multiprocessor Free</Flavor>
    <Architecture>X64</Architecture>
    <LCID>1033</LCID>
  </OSVersionInformation>
  <ParentProcessInformation>
    <ParentProcessId>1980</ParentProcessId>
    <ParentProcessPath>C:\Windows\explorer.exe</ParentProcessPath>
    <ParentProcessCmdLine>C:\Windows\Explorer.EXE</ParentProcessCmdLine>
  </ParentProcessInformation>
  <ProblemSignatures>
    <EventType>APPCRASH</EventType>
    <Parameter0>calc.exe</Parameter0>
    <Parameter1>6.1.7600.16385</Parameter1>
    <Parameter2>4a5bc9d4</Parameter2>
    <Parameter3>ntdll.dll</Parameter3>
    <Parameter4>6.1.7601.18247</Parameter4>
    <Parameter5>521eaf24</Parameter5>
    <Parameter6>c000041d</Parameter6>
    <Parameter7>0000000000054f01</Parameter7>
  </ProblemSignatures>
  <DynamicSignatures>
    <Parameter1>6.1.7601.2.1.0.256.48</Parameter1>
    <Parameter2>1033</Parameter2>
    <Parameter22>7867</Parameter22>
    <Parameter23>78676ffd9df09faf1c4e509a092b8c6a</Parameter23>
    <Parameter24>2da7</Parameter24>
    <Parameter25>2da7dccfc57b53ecf85b7e1ba5cbfb6d</Parameter25>
  </DynamicSignatures>
  <SystemInformation>
    <SystemManufacturer>=innotek GmbH</SystemManufacturer>
    <SystemProductName>VirtualBox</SystemProductName>
    <BIOSVersion>VirtualBox</BIOSVersion>
  </SystemInformation>
</WERReportMetadata>
```