



CROWDSTRIKE

Crashdmp-ster Diving the Windows 8 Crash Dump Stack

Overview

- Happy to be in Paris – thanks **NSC!**
- History of this research
 - Two prior techniques to leverage crash I/O path
- New crash path technique
 - Crashdmp.sys and crash filter drivers
 - Abusing crash dump stack logging
 - *Not* a “vulnerability”
- Code walk-through and demo
 - SOURCE Boston CTF challenge

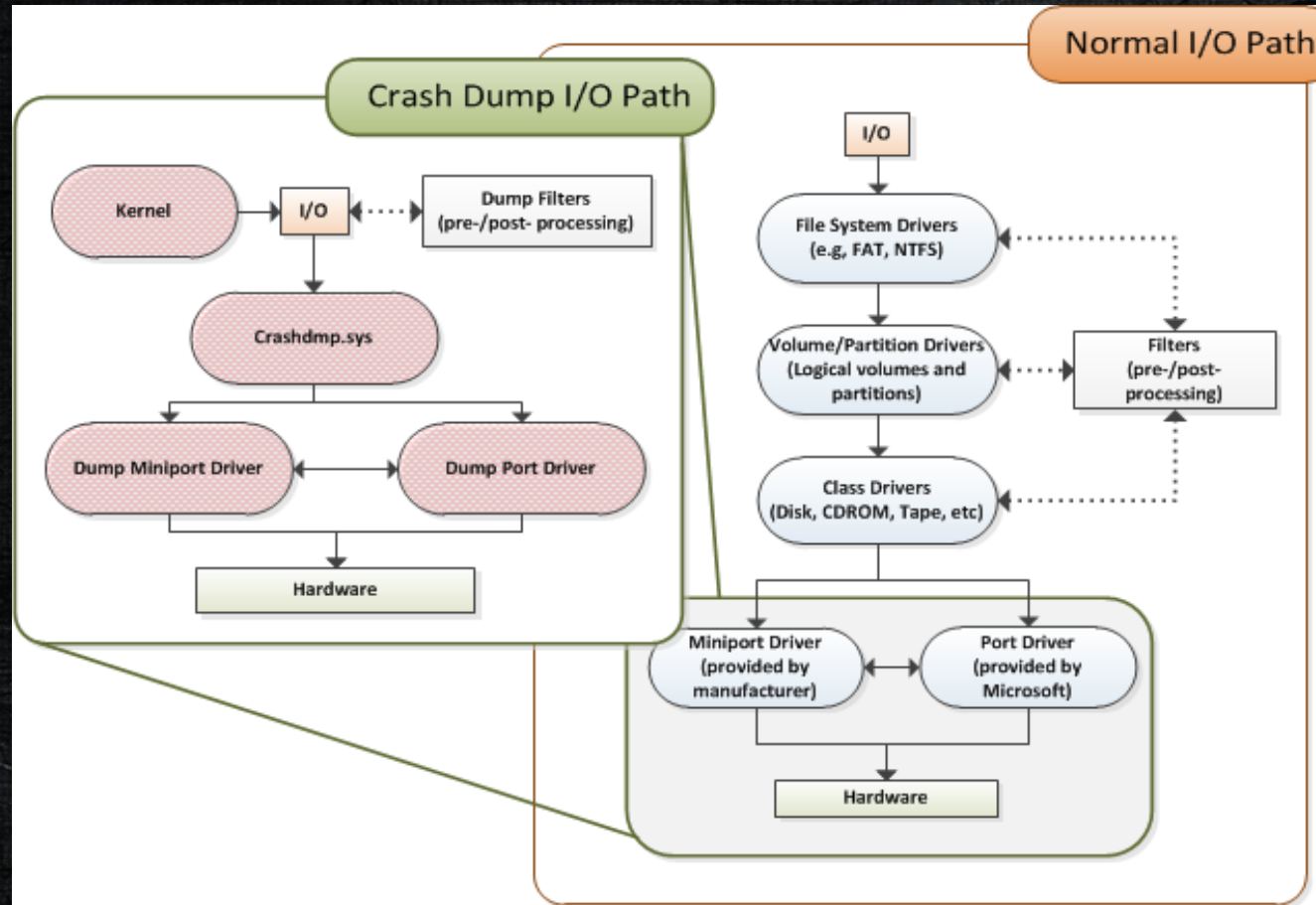


Research History Primer

The Crash Dump Stack

- A “stack” of drivers providing an “I/O path” to a mass storage device, consisting of:
 - A dump port driver
 - A dump miniport driver
 - One or more crash dump filter drivers
 - Crashdmp.sys
- Initialized in two phases:
 - System startup/page file creation (pre-initialization)
 - System crash (post-initialization)
- Used when:
 - A bug check occurs
 - The system is about to hibernate

Two Paths



Bypass Technique

- Platforms: Windows XP - 7
- Target: Dump port driver (eg, diskdump.sys)
- Goal: Use crash dump I/O path to defeat MBR rootkits
- Technique: Force a path that can only WRITE to disk to also READ from disk using SRB's and a few hacks to bypass normal OS usage of the stack and communicate directly with the port driver
- See [6] and [7]

Windows 8 Technique

- Platforms: Windows 8
- Target: Dump port driver
- Goal: Read or write any location on mass storage device
- Technique: Use new read functions added to crash dump port driver to read from disk without any “bypass”
- See [4] and [5]

Crash dump stack logging

- Platforms: Windows 8
- Target: crashdmp.sys
- Goal: Read or write any location on mass storage device
- Technique: As a crash filter driver, abuse new crash dump stack logging capabilities.



Crashdmp.sys and Crash Dump Filter Drivers

Crashdmp.sys

- Maintains state for crash and hibernate features
- Load and unload dump stack drivers
- Broker I/O requests between the kernel, crash filter drivers, and the dump port driver
- Various house-keeping duties:
 - Dump stack logging
 - Error simulation
 - User experience (dump progress notify)

Crash Filter Drivers

- Similar to file system filter drivers, crash filters can modify read/write requests through the crash I/O path
- Installed via `CrashControl\DumpFilters`
- Documented!! (sorta)
- Loaded into crash stack as `dump_<modname>.sys`
- Can't really do much with crash data
 - WDE crash filters encrypt/decrypt contents

Crash Filter Drivers (cont'd)

- **Non-standard DriverEntry:**

```
DriverEntry (  
    __in PFILTER_EXTENSION FilterExtension,  
    __inout PFILTER_INITIALIZATION_DATA FilterInitialization  
);
```

- **Populate FilterInitialization structure with callbacks:**

- Dump_Start, Dump_Write, Dump_Read,
Dump_Finish, Dump_Unload

- **Crashdmp.sys calls these as data is written (or read, in Win 8) through crash path**

Filter Context

- FilterInitialization pointer is actually the address of a field in a context structure allocated by crashdmp.sys:

```
typedef struct _FILTER_CONTEXT {  
    FILTER_INITIALIZATION_DATA FilterInitialization;  
    FILTER_EXTENSION FilterExtension;  
    LIST_ENTRY Next;  
    ULONG Unknown;  
    PVOID GlobalContext;  
    PVOID Image;  
} FILTER_CONTEXT, *PFILTER_CONTEXT;
```


Filter Context (cont'd)

- A linked list of these structures is maintained by `crashdmp.sys`
- Can walk/modify the list using the `Next` member
 - Disable or hook other crash filter drivers
- `GlobalContext` is a pointer to an undocumented book-keeping structure used by `crashdmp.sys`
 - No idea why this is exposed to filters!
 - Let's explore it...



Abusing Crash Dump Stack Logging

What is dump stack logging?

- Undocumented; new in Windows 8
- Enabled via registry:
 - `CrashControl\EnableLogFile = 1`
 - `CrashControl\DumpLogLevel = 0xffffffff`
- Writes basic diagnostic information to `C:\DumpStack.log.tmp`
 - Created when dump stack is initialized, just after filters are loaded
 - Keeps exclusive handle opened, stored in `GlobalContext`
 - Stores the file's disk runs/mappings in `GlobalContext`

Disk Run/Mapping Pair

- A series of volume-relative {size,offset} pairs that describe the location of a file
 - Maintained by the file system (e.g., NTFS)

- Documentation of

FSCTL_QUERY_RETRIEVAL_POINTERS:

```
struct {  
    LONGLONG    SectorLengthInBytes;  
    LONGLONG    StartingLogicalOffsetInBytes;  
} MappingPair;
```


Disk Run/Mapping Pair (cont'd)

- Since there can be multiple runs for a file, GlobalContext stores them in a structure :

```
typedef struct _LOG_DISK_RUNS {  
    ULONG Count;  
    PMAPPING_PAIR Array;  
} LOG_DISK_RUNS, *PLOG_DISK_RUNS;
```
- Disk runs are used to build SRBs in internal I/O routines
- *NB: The disk runs are retrieved through the normal I/O path.*

Simple Abuse Example

- As a crash dump filter driver, we have indirect access to `GlobalContext` through our `FilterInitialization` pointer
- Simply retrieve the disk runs of some other file, say, `ntoskrnl.exe`
 - Might need `FSCTL_GET*` instead of `FSCTL_QUERY*`
- Overwrite `GlobalContext->LogFileDiskRuns`
- Cause system hibernation or crash
 - As `dump/hiberfil.sys` is written and `crashdmp.sys` logs its actions, it's actually writing to `ntoskrnl.exe` not `dumpstack.log.tmp`
- Your kernel gets trashed!

But we can do better...

- To achieve arbitrary read/write, locate and use crashdmp.sys internal logging functions:
 - `WriteLogDataToDisk()` – Iterates over disk runs stored in global context structure, building an MDL for each request and using internal I/O functions (which call into dump port driver) to write the data to disk using the crash I/O path
 - `ReadLogDataFromDisk()` – Same as `WriteLogDataToDisk()`, except it uses read functions to read the log data from disk
- *For function prototypes, see whitepaper or released source code*

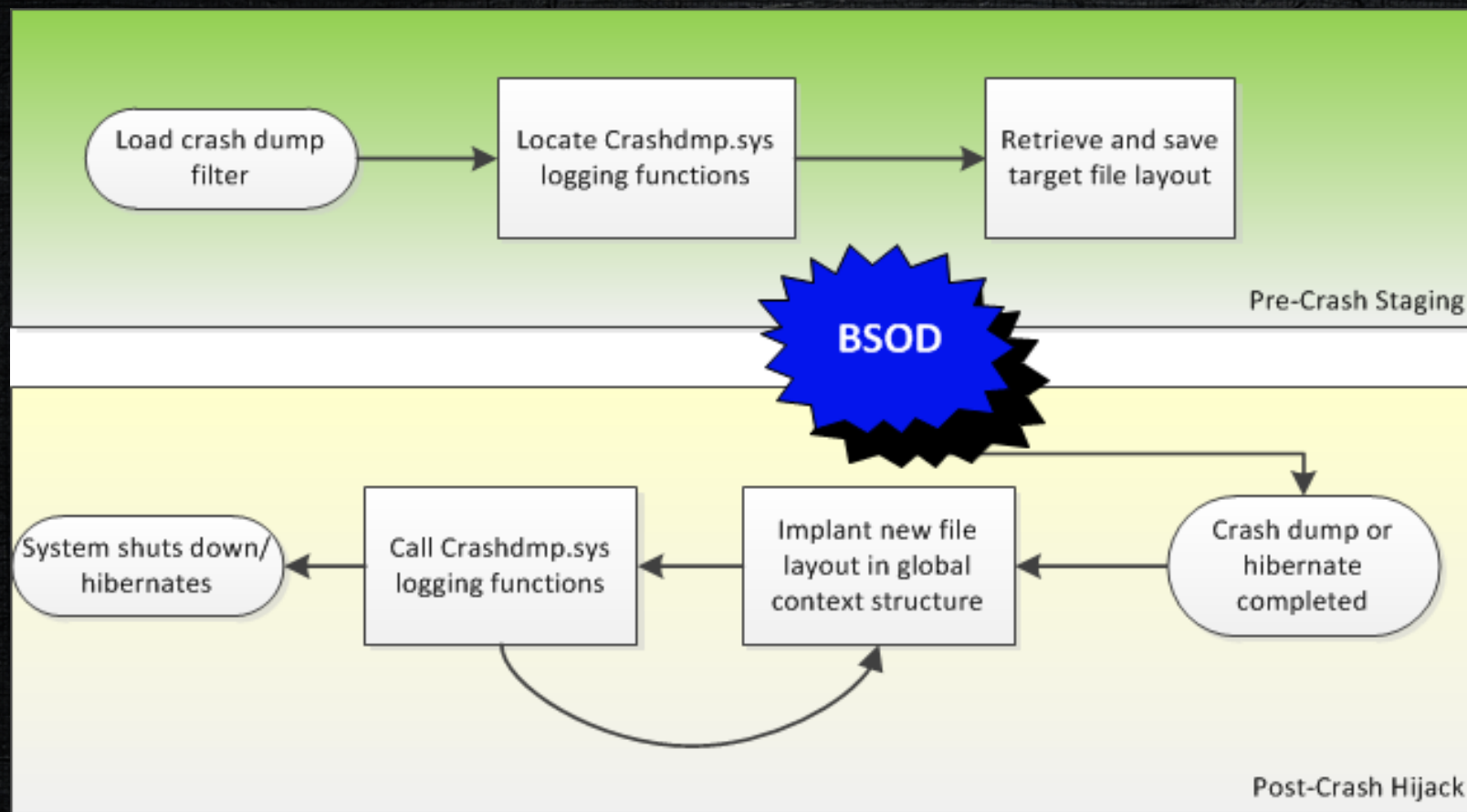
Restrictions

- ***These functions can only be called during crash or hibernation, because:***
 1. They implicitly assume the restrictions of a crash dump environment – most notably that the normal I/O path is disabled
 2. Required crashdmp.sys internal functions are not initialized until they are needed (at crash or hibernation time).
 - Calling either of the logging functions outside of a crash/hibernate context will result in a null pointer dereference.

Two-stage Hijack

- In addition to those restrictions, because the normal I/O path is needed to retrieve the disk runs, the dump logging feature must be hijacked in two stages:
 - Pre-crash/hibernate: Retrieve target file disk runs
 - Post-crash/hibernate: Implant disk runs, call read/write functions

Two-stage Hijack (cont'd)



Advanced Abuse Example: Patch a driver

1. Retrieve disk runs of the driver
2. (Hibernation or crash occurs)
3. Implant the file layout in the global context structure
4. Call `ReadLogDataFromDisk()` to retrieve the driver's contents
5. Modify the contents held in the resulting buffer in memory
6. Call `WriteLogDataToDisk()` to overwrite the driver with the modified contents
7. Restore original log file layout so that `crashdmp.sys` does not overwrite target file when it finalizes the log file

Implementation Protips

- Pre-crash/hibernate staging can be done any time after filter driver is loaded
- Post-crash/hibernate work is best done in `Dump_Finish` callback
 - At this point, `crashdmp.sys` has finished any logging and no longer using internal log file position info
- `FILTER_CONTEXT` and `GlobalContext` can be modified without concern for synchronization
 - They are allocated at kernel initialization and left alone until they are needed during a crash/hibernation



SOURCE Boston CTF

Challenge Overview

- Uses the technique described in this presentation
- Contestant must cause a read and write through crash I/O path using `DumpStack.log.tmp` as a control file
 - Challenge key written to corrupt dump file
- Automatic bugcheck every 4 minutes
 - Clues given in bugcheck messages
- Key stored in pre-determined disk run on disk

Challenge Stage 1

- The user must enable dump stack logging
- Bugcheck message: “U MAD YET BRAH? Dump stack logging is disabled”

Challenge Stage 2

- On startup, crash filter driver adjusts file permissions on dump stack log file
- The user must cause a file to be read through crash I/O path by modifying dumpstack.log.tmp:
 - Type any valid path in the file “\\??\C:\<pathtofile>” and save it -> bugcheck
- Specifying the fake key file name initiates Stage 3

Challenge Stage 2 (cont'd)

- Three possible bugcheck messages:
 1. No file specified – “No file specified yet!!”
 2. Valid file – its contents up to 1mb are copied into `dumpstack.log.tmp`: “File copied. Cool story, bro”
 3. Fake key file (on desktop) is specified – “Key file is empty, nice try! `[path][[0xoffset][[0xbyte1],[0xbyte2]...`”

Challenge Stage 3

- The user must cause a file write through the crash path
 - Use format specified in bugcheck message

Challenge Stage 4 – End Game

- Upon detecting this write, the crash filter driver:
 - Appends the CTF key to the dump file being generated by the OS (inside DumpWrite callback)
 - Disables itself by moving its image file
 - Displays bugcheck message “Troll complete. We got a badass here.”
- Upon reboot, key is written dozens of times to the generated minidump (Windows\minidump), most likely corrupting it.



Source Code Walk-Through



Demo

Resources

1. Whitepaper accompanying this presentation
2. Source Boston CTF Challenge write-up and source code:
<https://code.google.com/p/dmpflt/>
3. All kinds of relevant information: <http://www.crashd.mp>
4. [I/O You Own: Windows 8 Update](#) (1/9/2013) – A blog post discussing the new features in the Windows 8 crash dump stack, as well as an overview of a new technique to use the stack outside the operating system.
5. [BSides Jackson: I/O You Own: Regaining Control of Your Disk in the Presence of Bootkits](#)(11/10/2012) – Slide deck of my updated presentation for BSides Jackson. Covered previously-published but revised material that included an overview of Windows 8 crash dump stack changes.
6. [SyScan Singapore 2012: I/O You Own: Regaining Control of Your Disk in the Presence of Bootkits](#)(4/26/2012) – Program overview and link to original slide deck for my presentation at SyScan 2012.
7. [SyScan 2012 Preview – I/O You Own: Regaining Control of Your Disk in the Presence of Bootkits](#)(4/23/2012) – A blog post introducing the research and upcoming SyScan presentation.

Thank You!

@lilhoser

lilhoser@gmail.com

<http://crashd.mp>

Greetz to Alex Ionescu



CROWDSTRIKE