Todo

-

Testing

- test with xbot
- have the server connect back with option -a <addr> -p <port>
- have the client listen with option -p <port> -l
- the client must be listening prior to the server running.

Issues:

- The Bulldozer server was connecting to the bulldozer client intermittently. I traced the problem to the xConnect call in bulldozer. I never figured out what the problem was. If other networked apps have the same issue, this might be the problem.
- The memory allocated for a driver will essentially be a memory leak. I have no way to tell when a driver will be done executing, so I can't delete it.
- Drivers can't be unloaded because the driver object has no section handle associated with it. If this becomes a problem, I have code that's commented-out in CreateDriverObject that is broken, but should create a section handle that can be used.
- C:\Windows\system32\svchost.exe is hardcoded. If windows is installed on another partition (i.e. D:) then this would look strange in taskmanager.
- The processes used in launching an implant with MagicWand were chosen carefully. If the parent process goes away while an implant is running, the system could crash. We also wait for explorer.exe to launch before we launch any user processes. This has yielded the most reliable results and any modification to this policy should be tested thoroughly.

Future Mods

- Find a way to do exception handling.
- Rewrite MagicWand
- Provide a way to inject a dll into another process with Magic Wand
- Put the covert storage area in a deleted (still allocated) registry key/value. We could also put it in an NTFS attribute.

- Should we store a hash of files in the covert storage area? This would avoid corruption.

Debugging Instructions:

- Type this in Windbg: .reload /f wc.sys=0x<image_start_address>
- The image start address is printed out to the debugger, but you'll have to hit F5 once to get it to print. If you need to debug code prior to this, you are stuck looking at assembly.

Things to document:

- In XP, the process execution only works on PAE systems.

Things I would like in the next version of BadMFS

- Add directories
- Encryption on the whole file system
- Simplify it. I don’t think all of the locks are necessary.
- Make it easier to use different backing stores, such as registry, $boot, etc. This will require abstraction of the reads and writes.
- Make it more intuitive to use. Get rid of the requirement to call initialize every time.
- There should be an install function that creates the file system. If the file system hasn’t been installed, then none of the other file system functions should work.
- The users shouldn’t have to open a handle to the file or disk themselves.
- A function that sets the file pointer within a badmfs file. Currently there is only a set file pointer for the file handle to the backing store.
- Add a compression option to the files.
- Add metadata to the files. For example, it would be nice to have time created, modified, etc. – I’m not sure about this one.
- There needs to be an integrity check on the files.
- Get rid of the NDK.
- Add the ability to redirect Windows OS file system access (i.e. calls to NtCreateFile, NtReadFile, etc) into badmfs. This would allow implants to access badmfs without having to link the badmfs library into their code. This,

however, would require that Gametime is used to provide the hooks.   This is probably more of a wolfcreek modification, but changes to badmfs might be necessary.