Todo

**General**

* Documentation.
  + QuickBook generated.
  + Write full in-source docs simultaneously.
  + Write threading docs simultaneously.
  + Write exception guarantee docs simultaneously.
* Unit tests.
  + All components.
  + Cleanup and in-source documentation.
  + Extend and improve.
  + Pass application verifier.
  + Leak testing.
  + In tests such as PeLib etc ensure at least one module is processed per test, in case the validity check function is the one with a bug.
  + Extra sanity checks to ensure not only that functions return without exception, but also that the returned data is valid. (e.g. GetName in ImpThunk.)
* Examples.
  + All components.
  + Cleanup and in-source documentation.
  + Comprehensive and aimed at providing useful real-world utilities for remote targets out of the box.
  + Pass application verifier.
  + Leak testing.
* Improve quality of existing code.
  + Improve exception-safety guarantees. (Rollback support etc)
  + Improve thread-safety guarantees and threading model.
  + Proper localization and internationalization support/framework.
  + Consider moving from UTF-16 internally to UTF-8, and only using UTF-16 at API boundaries. If implemented, all files and other IO should be in UTF-8.
  + Template concept checking and improved error messages.
  + Compile time checks to ensure MSVC, Boost, etc meet the minimum requirements.
  + Assertions where appropriate.
  + Config header to account for differences in compilers (or compiler versions). Consider Boost.Config.
  + Improved error reporting via exceptions.
  + Support move semantics where appropriate.
  + Support swapping where appropriate (and use to simplify operator= if possible).
  + Support perfect forwarding where appropriate.
  + Everywhere PeFile is used and assumed to be valid, DosHeader and NtHeaders should also be used to ensure validity.
  + Consider using Boost.Iterator to implement iterators.
  + Clean up iterators to be more lightweight, and shift most of the processing to the parent object.
* Improved binary generation and distribution.
  + Binaries with update server.
  + Automatic binary versioning using SVN revision.
  + Automatic build bot.
  + Digital signature (for build bot, WER, etc).
* Shared library build option.
  + Including building dependencies as shared libraries.
* Improve API.
  + Clean up APIs returning pairs/tuples/etc to return objects with named fields and/or getters.
  + Investigate support for optional error reporting via error codes to avoid littering code with try-catch blocks in certain scenarios.
  + Improve API genericity.
    - Instead of taking or returning a vector<T>, instead use a C<T> where ‘C’ is a template parameter representing an arbitrary container.
    - Use ‘ranges’ over direct containers where possible.
* Reduce compile time.
  + Reduce include dependencies.
    - Pointer-to-impl idiom.
    - Forward declarations where possible. (Boost? STL?)
    - Update forward declaration header.
  + PCH support using Boost.Build.
  + Decouple components if possible.
  + Ensure all headers are self-sufficient.
* Todo list.
  + Consider porting to Google Code issue tracker.
* Maintain a changelog.
* Performance improvements in potential bottlenecks (e.g. Iterators, Scanner, PeLib, FindPattern, etc).

**New Modules**

* Helper service to run HadesMem tools as ‘SYSTEM’ for when manipulation certain protected/critical processes (running in separate desktops, sessions, etc.).
  + Ensure service is secure from abuse.
* Debugging.
  + Software breakpoint.
  + Hardware breakpoint (including memory read/write detection).
  + Conditional breakpoints.
  + Single stepping.
  + Stack trace.
* Anti-anti-cheat.
* Thread/process information and enumeration.
* Manual mapper.
  + Windows PE loader reversing.
  + TLS support.
  + Exception handling support. (Safe, rather than catch-all.)
  + No ‘external’ (read: MMHelper) dependencies.
  + Extensive sanity checking on PE file.
  + Manually map dependencies and ‘link’.
* Remote code ‘emulator’.
* Custom GetModuleHandle, GetProcAddress, etc.
* Cheat Engine ‘replacement’.
* C++ WinAPI wrapper.

**Process**

* Constructors that take window or process name should throw if multiple matches are found. In this case, the PID should be used.

**MemoryMgr**

* Clean up memory reading/writing API to reduce use of template metaprogramming in public interface.
* Improve ‘safety’ of remote function caller via EH to minimize risk of crashing the target.
* Improve genericity of parameter passing in remote function caller.
* Memory reading via expression templates.
* Improved type genericity via type decay to ensure templates function correctly in the face of const, volatile, references, etc. (Perfect forwarding?)
* Fix type traits in Read/Write APIs to support std::vector< const T> and other ‘mutations’ that are not currently handled.

**Module**

* Remote GetModuleHandle implementation with same path semantics etc.

**Disassembler**

* Decode calls/jumps to function names if possible.
* NOP/UnNOP support.
* ASM searching API with ‘wildcards’. (MetaASM?)
* String based assembler with x64 support.

**Scanner**

* Rewrite to be more reliable and robust.
* Refactor to reduce code duplication.
* Use a file view with a small memory cache rather than consuming large amounts of RAM.
* Multi-threaded scanning options.
* Wildcard support for vector/string scanning.
* Regex support for string scanning.
* Memory protection filters (read, write, exec).
* Memory type filters (private, mapped, image).
* Support pausing target while scanning.
* Support injected scanning.
* Configurable scan buffer size.
* Pointer scanner.
* Unknown value scan.
* Progressive scan filtering based on either value or criteria.
* Scan history and undo.
* Support case insensitive string scanning.
* Binary scanning.
* Custom scanning via user-supplied predicate.
* Improved floating point support (configurable or ‘smart’ epsilon).
* Group search support.

**Injector**

* .NET injection.
  + Without DLL dependency if possible.
* Cross-section injection.
* IAT injection.
* Varargs/generic export calling.
* Consider NT API based injection (LdrLoadDll).
* Add ‘FreeDll’ API.
* Get address of Kernel32!LoadLibrary ‘manually’ rather than using a local GetProcAddress and pointer arithmetic.
  + Whilst this works in all normal cases, it doesn’t work when the target has shims enabled which hook LoadLibrary.

**Patcher**

* VEH hooking (both INT3 and DR).
* Transactional hooking.
* Improved relative instruction rebuilding (including conditionals). x64 has far more IP relative instructions than x86.
* Freeze target when hooking (except calling thread if applicable – e.g. in injected code).
* When hooking on x64 try to find a free memory block for the trampoline that is within RIP-relative range of the detour. Only if one cannot be acquired should we fall back to a system-provided address and an absolute jump.
* Hook on x64 in a way that does not mix code and data.
* Uncopyable, so make moveable.
* VMT hooking.
* IAT/EAT hooking.
* Explicitly support hook chains (and write test).
* Use relative jumps where possible (detect delta at runtime).
* Detect cases where hooking may overflow past the end of a function, and fail. (Provide policy or flag to allow overriding this behaviour.) Examples may be instructions such as ‘int 3’, ‘ret’, ‘jmp’, etc.

**FindPattern**

* Use a file view with a small memory cache rather than consuming large amounts of RAM.
* Pattern generator.
* Multi-threaded scanning.
* ‘Start’ attribute support.
* Data/Rdata scanning support (for ‘start’ attribute and constant scans).
* Support for loading XML file from resource or memory.
* Dump results to file.
* Change parser library from RapidXML to Boot.Spirit (may require change to custom format).
* Object or stream-based implementation that allows operation chaining.
* Improve constructors.
  + I forget what this means exactly… Maybe reduce code duplication? Or provide constructor for custom start/end?
* Improve tests (relative vs absolute regression was missed).
* Optional relative address support (for dumping address where the game has ASLR enabled).

**PeLib**

* Support for working on x86 PE files from x64 and vice versa.
* Finish implementing matching ‘setters’ for existing ‘getters’.
* Investigate use of virtual functions for file vs memory access (RvaToVa).
* Extra sanity checking in all components.
  + E.g. Check NumberOfRvaAndSizes in NtHeaders before attempting to retrieve a data dir.
* Cache base pointers etc rather than retrieving it manually in every getter/setter. Slightly less ‘robust’, but due to the typically ‘read-only’ nature of the data this is the expected behaviour in all known cases anyway.
* Consistent API for GetBase. (e.g. PVOID vs PBYTE vs T\*)
* Support more of the PE file format.
  + Overlay data.
  + Resource directory.
  + Exception directory.
  + Relocation directory.
  + Security directory.
  + Debug directory.
  + Load config directory.
  + Delay import directory.
  + Bound import directory.
  + IAT (as opposed to Import) directory.
  + CLR runtime directory support.
* Full support for writing back to PE file, including automatically performing adjustments where required to fit in new data or remove unnecessary space.
* Improve export forwarding code to detect and handle forward-by-ordinal explicitly rather than forcing the user to detect it and do string manipulation and conversion.
* Helper functions such as FindExport, FindImport, HasDataDir, GetArchitecture, etc.