Todo

**General**

* Rewrite this list. (After outlining a ‘definition of done’.)
* Todo list.
  + Consider porting to Google Code issue tracker.
* Documentation.
  + Improved and expanded documentation. Add notes, cautions, symbols, cross-refs, etc.
  + Threading guarantees.
  + Exception guarantees.
  + Document preconditions, postconditions, and invariants.
  + Better annotations (warnings, notes, etc).
  + Note where handles are inheritable.
  + Note what access control handles are granted.
  + Improved commenting of components.
* Unit tests.
  + Write a specially crafted process to use as a sample ‘target’ for unit tests and examples. This way known values can be checked for cases where they are typically unknown, even for our own process.
  + Clean up tests.
  + 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.)
  + More tests for cases that should throw (BOOST\_CHECK\_THROW).
* Examples
  + Demonstrate exception handling.
  + Demonstrate multi-threading.
  + Basic memory hacking app.
* Re-add all components from v1.0.0.
* Convert exceptions to assertions where appropriate.
* Convert exceptions to return values where appropriate.
* Code cleanup. Global namespace operator etc where appropriate.
* Commenting cleanup.
* Optimize all components, both in terms of interface and implementation.
  + Put locks on their own cache line?
* Set up Jenkins build bot.
* Rewrite tests to not Read/Write data on the stack.
* Rewrite tests to be less ‘hackish’ and ‘ugly’.
* Investigate whether it’s feasible and worthwhile to detect when HadesMem is being used ‘in-process’ and drop to lower level implementations of certain functions such as Read/Write (using SEH instead of RPM/WPM for example), and also using different APIs (such as using local threads rather than remote threads, etc).
* Review current code for compliance to relevant coding standards/guidelines (Google, Boost, etc).
* Clean up this list (again).
* Forward declaration header.
* Make better use of auto.
* Compile-fail tests for a Call.
* Update doc gen tools and improve doc gen process.
* Raise Intel warning level.
* Stack trace on error.
* Improve process enumeration (improve ProcessEntry type and merge with ProcessList header).
* Support default construction for more types where it makes sense (e.g. Process).
* Look into how Process handle is managed, and potential leaks (raised by Intel Inspector). Also look at how other handles are managed. Decide between BOOST\_VERIFY, and an optional ‘Cleanup’ function which then swallows the error in the destructor (again with a verify?).

**New Modules**

* Remote memory ‘pool’ to avoid allocating entire pages for only a few bytes of data.
* Python bindings.
  + Important! Ensure -fno-strict-aliasing is used under GCC as it seems Boost.Python has aliasing violations which cause spurious segfaults and other issues.
* Basic base hook.
* Helper service to run HadesMem tools as ‘SYSTEM’ for when manipulating certain protected/critical processes (running in separate desktops, sessions, etc.).
* Debugging APIs (sw bp, hw bp, conditional bp, single stepping, stack trace, etc).
  + Investigate the debugger engine API (<http://goo.gl/ZV6K7>).
* Thread information and enumeration.
* Remote code ‘emulator’.
* Custom GetModuleHandle, GetProcAddress, etc.

**MemoryMgr**

* Improve Read/Write tests by doing checks both before and after writes etc.

**ManualMap**

* Exception handling support under x86 SafeSEH and x64.
* Improved TLS support.
* CLR hosting support.

**Injector**

* .NET injection.
  + Without DLL dependency if possible.
* Cross-section injection.
* IAT injection.
* 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**

* Class function hooking (ecx preservation). (xchg ecx, [esp]; push ecx)
* Fastcall function hooking (ecx, edx preservation).
* 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).
* Uncopyable, so make moveable.
* VMT hooking.
* IAT/EAT hooking.
* Hotpatching method for Windows API. (For thread safety.)
* 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**

* Pattern generator.
* ‘Multi-pass’ support (e.g. search for pattern, apply for manipulators, use as starting point for second search).
* Arbitrary region support.

**PeLib**

* Investigate use of virtual functions for file vs memory access (RvaToVa).
  + Alternatively, investigate use of templates, which may ‘merge’ better with x86/x64 cross compatibility.
  + Note: May cause problems when copying ‘PeFile’ type.
* 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.
* 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, IsDotNet, GetPDB, etc.
* Test against pathological cases such as Corkami tests.

**Disassembler**

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

**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.