Analysis Report for: 905acadd3d9e056d5342d2356dc6f9d0.exe.c

Overall Functionality

This C code appears to be a Windows-based program that acts as a bridge or intermediary between a Lisp application (likely AutoCAD Lisp) and a Windows application. It handles communication between these two environments, managing arguments, data transfer, and error handling. The program uses inter-process communication (IPC) mechanisms, possibly involving shared memory or named pipes, to exchange data with the target application. The extensive error handling suggests robustness is prioritized. The code uses a custom communication protocol (indicated by the frequent use of codes like 100, 102, 105, etc.) with the target application.

Function Summaries

* **`sub_401000()`**: Main function; initializes the system, prints a copyright message, and enters a main loop handling commands from the Lisp application. Returns an error code.
* **`sub_401090()`**: Sends initialization data (likely keys or settings) to the target application. Returns 1.
* **`sub_4010C0()`**: Sends a plot command to the target application. Returns a result from a subsequent function call.
* **`sub_401110()`**: Handles a specific command from the Lisp application (appears to involve processing a string argument). Returns an error code or 0 on success.
* **`sub_4011D0(__int16 a1)`**: Interacts with the target application. Handles various return codes (4, 13, 26, etc.) from the application, maps them to internal codes (100, -2, -6, -3, etc.) and performs error handling.
* **`sub_401240()`**: Initializes internal data structures, likely related to command buffers or communication status. Returns 2.
* **`sub_401260()`**: Handles application exit. Cleans up allocated memory and displays error messages. Exits the program.
* **`sub_401320(const char *a1)`**: Formats and sends an error message to the target application. Returns a result from a subsequent function call.
* **`sub_401350(char **a1)`**: Attempts to retrieve data from the target application. Handles various status codes and sets output. Returns -1 on error, 100 otherwise.
* **`sub_401390(char **a1, __int16 a2, int a3)`**: A complex function; appears to retrieve and process multiple data structures (lists or similar) from the target application, converting them to an internal representation.
* **`sub_401650()`**: Handles a return from a communication function and performs additional handling or data conversion.
* **`sub_401690(int a1, int a2, int a3)`**: Recurses through the communication buffer to build the response; handles various data types and their conversion between the Lisp and internal representations.
* **`sub_401A80(__int16 a1)`**: Calculates a buffer size based on input and other data.
* **`sub_401AB0(__int16 a1)`**: Manages data transfer to the target application, splitting data into chunks if necessary.
* **`sub_401C70(char *Format, ...)`**: A wrapper for `vsprintf`; formats and prints a message to the application or console.
* **`sub_401CB0(_BYTE *a1, _BYTE *a2, int a3)`**: Copies a portion of a byte array. Handles buffer limits.
* **`sub_401CF0()`**: A critical function that manages the communication buffer, handles incoming commands, and determines the number of next elements.
* **`sub_401F80(int a1)`**: Allocates memory using `malloc`. Displays an error message if allocation fails.
* **`sub_401FB0(const char *a1)`**: Creates a copy of a string in memory using `malloc` and `strcpy`.
* **`sub_402010(const char *a1, int a2)`**: Sends a string to the target application, handling various error conditions and splitting larger strings.
* **`sub_402100(const char *a1, __int16 a2)`**: Sends a string command to the target application along with additional data (a2). Handles errors.
* **`sub_4021D0(unsigned __int8 a1, __int16 a2, char a3, unsigned __int8 a4)`**: Appears to store data in a simple custom structure.
* **`sub_402210(_DWORD *a1)`**: Initializes a DWORD array (likely a data structure).
* **`sub_402230(void **a1)`**: Frees a set of allocated memory blocks.
* **`sub_402260(int a1, __int16 a2)`**: Checks if a buffer has enough space for incoming data.
* **`sub_402280()`**: Initializes the communication channel with the target application and handles the initial handshake.
* **`sub_402370()`**: Sends a message (likely an error message) to the target application.
* **`sub_4023B0(int a1)`**: Handles commands from the main loop.
* **`sub_402400(__int16 a1)`**: A crucial function that interacts with the target application, sending and receiving data. It handles a variety of application responses.
* **`sub_4026E0(int a1)`**: Finds a specific entry in a linked list (likely created by `sub_401690`).
* **`sub_402720(_DWORD *Block)`**: Frees a linked list of data structures.
* **`sub_402740(__int16 a1)`**: Allocates a small structure; likely a node for a linked list.
* **`sub_402780(char *Block)`**: Frees memory allocated for a data structure, handling different data types within the structure.
* **`sub_4028B0()`**: Returns an integer value (possibly an index or flag).
* **`sub_4028C0()`**: Returns a pointer to a data structure (possibly a buffer).
* **`sub_4028D0()`**: Performs a check or operation with the target application.
* **`sub_402B10(HINSTANCE hInstance, const char *a2)`**: Creates and manages the Windows message loop, handling messages from the target application and the main application thread.
* **`sub_402C70(int a1)`**: Performs actions based on flags (3 or 5), involving setting events or other inter-process communication actions.
* **`sub_402CD0(int a1)`**: Manages the main event loop, handling messages from the target application and performing a wait operation (possibly using shared memory).
* **`sub_402E20(int a1)`**: Performs actions based on a flag (3 or 5), using the target application's interface functions.
* **`sub_402E50(UINT uExitCode)`**: Exits the application cleanly, potentially performing cleanup actions.
* **`sub_402E70(int a1)`**: Performs a wait for response from the target application, managing data exchange and error conditions.
* **`sub_402F20(int a1, UINT uExitCode)`**: Performs a clean exit of the application, closing connections with the target application.
* **`sub_402F90(int a1, int *a2, __int16 a3)`**: Initializes the communication buffer, writing initial data to the buffer.
* **`sub_402FD0(UINT uExitCode)`**: A wrapper for `sub_402E50`.
* **`sub_402FE0()`**: Initializes internal data structures for the communication process.
* **`sub_403000(int a1)`**: Handles data structure conversion and communication.
* **`sub_403090(_BYTE *a1, int a2, int a3)`**: Copies a byte array.
* **`sub_4030E0(const char *a1)`**: Registers a window message.
* **`sub_403190(int a1)`**: Formats a name for shared memory (likely for inter-process communication).

* **`sub_4031C0(HANDLE hFileMappingObject)`**: Maps a shared memory object to the application's address space.
* **`sub_4031E0(int a1)`**: Opens and maps a named shared memory object for inter-process communication.
* **`sub_403220(int a1)`**: Unmaps and closes a shared memory object.
* **`sub_403280(int a1, int a2)`**: Sets a flag in shared memory for inter-process communication.
* **`sub_4032D0(int a1, int a2)`**: Clears a flag in shared memory for inter-process communication.
* **`sub_403320(int a1, int a2)`**: Checks a flag in shared memory for inter-process communication.
* **`sub_4074E3(int a1, int a2)`, `sub_4074F9(int a1, int a2)`, `sub_40750F(int a1, int a2)`, `sub_407540(int a1, int a2)`**: These functions likely handle data conversion between different numeric types and representations.
* **`sub_407B7E()`**: Calls an external function `flsall`, likely related to floating point handling.
* **`WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)`**: Entry point of the program.

Control Flow

The control flow is complex and heavily relies on the return codes from `sub_402400` which interfaces with the external process. The main loop in `sub_401000` continues until an error is encountered in `sub_4023B0`. Functions like `sub_401390` and `sub_401690` have nested loops to process complex data structures received from the target application. Error handling paths frequently lead to `sub_401320` to send error messages and then to `sub_401260` for graceful shutdown.

Data Structures

The code uses several key data structures:

* **Linked lists:** `sub_401690` and `sub_402740` suggest the use of linked lists to store data received from the target application. These lists are likely used to represent lists or trees from the Lisp environment.
* **Custom structures:** Several functions seem to operate on structures of unknown type (marked as `_UNKNOWN`). These might represent different data types or command responses from the Lisp application.
* **Arrays:** `dword_40A0D0` is a large array, possibly used as a buffer or to store internal state.
* **Communication Buffer:** `dword_40D4C0` appears to be a shared memory buffer used to manage communication with the target application. It is structured to hold multiple data packets with type and value fields.

Malware Family Suggestion

Given the code's functionality—acting as a backdoor or intermediary between a Lisp application (potentially AutoCAD) and a separate, possibly malicious, process, employing sophisticated inter-process communication and error handling—it strongly suggests a **backdoor** or a **remote access trojan (RAT)** designed to interact with and potentially control the host system indirectly through the AutoCAD interface. The use of shared memory and named pipes is consistent with many RAT techniques, allowing silent and persistent communication with a command-and-control server. The extensive error handling is a characteristic often found in more sophisticated malware to increase stealth and resilience. The complex data structure manipulation and protocol suggest a degree of sophistication. The function names, use of weak symbols (declared as `weak`), and the comment about being generated by a decompiler are typical artifacts from reverse engineering, so it is likely an obfuscated sample.