Design Document: Matchmaker Subcontext for Virtual Memory

Maria Fay Garcia

1 Overview

In implementing virtual memory subcontexts, we have thus far implemented a client-side library and a server-side library. Processes wishing to snapshot their memory pick up the server-side library, map their memory into an image file, and die. Processes wishing to map a specific subcontext into their memory pick up the client-side library and select an image file to map into their memory. Client processes can then call functions that reside within the memory of the server process.

2 Motivation and Problem Statement

Clients want to map image files of server processes' memory in order to call functions that reside within the server processes' virtual memory. To ensure that this entry point is well-defined and safe, we will create a "Matchmaker" subcontext which we will map into every client processes' memory before they can map any server image files.

3 Aims

The Matchmaker subcontext will serve two primary purposes:

(1) Act as an observer and/or mediator in the "handshake" between the client process and the memory of the server process.

(2) Provide a well-defined entry point into the server processes' memory when the client process wishes to make function calls.

4 Client-Server Interaction

After initialization (i.e., mapping the Matchmaker) a client process will request that the Matchmaker map image files into its memory. The Matchmaker, in turn, will perform the necessary checks (if the image file exists, if the client has permission to map the image file, etc.) before mapping the image file into the client processes' memory.

After mapping a given server image process into the memory of a client process, the Matchmaker will provide an interface through which the client process can make function calls into the memory of the server process. The Matchmaker will maintain a list of the server processes' function pointers and will provide this list (or perhaps an abstraction of this list?) to the client process upon mapping. When the client process requests to call a server process' function, it again checks permissions and facilitates or rejects the call. In this way, the Matchmaker enforces protections, both over the mapping of new server image files and the calling of functions within already-mapped server image files.

5 A Tentative API

(1) `init()`: Maps the Matchmaker into the client process' memory.

(2) `request_map(image_filename)`: Asks the Matchmaker to map a server image of the given image file name into its memory.

(3) `request_call(server, func_ptr)`: Asks the Matchmaker to call the function with the given pointer in the server process' memory.

(4) `finalize()`: Asks the Matchmaker to unmap all mapped server images from the client process' memory. The Matchmaker does this and then unmaps itself. This function should be called upon finalization of the client program.

6 Requirements

(1) If a process picks up the client-side library, it should connect to the Matchmaker automatically. In other words, Once a client picks up the client-side library, it should not be allowed to map any image files into its memory before it has mapped the Matchmaker.

(2) A client process should not be able to make raw function calls directly into the memory of a server process. Function calls should either enter at a well-defined point through the Matchmaker or be mediate by the Matchmaker in such a way that they are definitely safe to complete.

7 Security and Isolation

The existence of the Matchmaker will ensure that client processes are not granted god-like power over the memory of server processes which they map. Interaction with server processes happens via the Matchmaker, which will have unrestricted permissions and will be able to modify the permissions of the client process as it sees fit.

Ideally, a function call from a client process into a server process will cause a segmenation fault, which we will then allow the Matchmaker to handle by registering the SEGV signal handler. The Matchmaker will check the permissions of the client and turn on/off permissions as necessary to allow the client to make the desired function call or to reject the client's access to that part of the server's memory.

8 Concurrenchy and Synchronization

Multiple clients should be able to map the same server image at the same time. Clients should never modify the memory belonging to a server image. To keep track of which clients are mapping which images, the client-side library should maintain a global list of mapped subcontexts.

9 Questions about Failure Handling

(1) What if an image file is missing, or the mapping of an image file overlaps with the memory of the client process? For now, we bail out, but we should consider taking a different action in the future.

(2) Should we provide cleanup routines for zombie clients?

10 Questions about Constraints and Assumptions

What limitations, if any, should we enforce on:

(1) The number of subcontexts a single client is allowed to map

(2) The maximum size of an image file that a server is allowed to create

(3) The number of function pointers a server process is allowed to pass to the Matchmaker

11 Future Work

(1) Enforce permissions restrictions by using mprotect and registering the SEGV signal handler.

(2) Transition from a shared library to a static library by building an ELF executable by hand.