Android Memory Modification Prototype

Introduction

The Android Operating System (OS) is plagued with issues relating to device fragmentation, OS upgrades and Malware. These issues primarily stem from a lack of user awareness as well as a general disinterest from original equipment manufacturers (OEM) to update devices. In order to mitigate some of these problems, a tool has been developed which can modify a currently executing process. Through this ability, device handsets can be patched or modified to reduce the impact of vulnerabilities and/or be made to comply with endpoint policy enforcement.

Prototype Usage

Installation

The prototype tool is currently installed by placing the patch (executable) and test.ini (configuration) files in the /data/local/tmp directory, and ensuring the proper file permissions are set. This can be accomplished with the standard Android Debug Bridge (adb) tool as seen in Figure 1.

Figure 1 – Installation

Execution

The tool is started by invoking the executable with root privileges. All information related to the operation of the tool is stored in the configuration file.

Figure 2 – Usage

Removal

Removal is accomplished by deleting the executable and configuration files. Also, the memory modifications are only present in the process while it is executing. To remove the modifications, the modified process must be restarted.

Figure 3 – Removal

Technical Walkthrough

Startup

Upon execution of the tool, a structure is initialized with the values present in the configuration file. If the configuration file cannot be found then the process will exit. Important values in the configuration are as follows:

Field Description

name The name of the process to inject into. This should match the output from ps.

image The name of the library to search. This should match the output from /proc/<PID>/maps.

symbol The exported symbol name of the function to hook.

pattern The base64 encoded byte stream of a function. To be used if the symbol cannot be resolved.

hook The base64 encoded byte stream of code which will be executed by the hooked function.

Process Search

Once the configuration file has been loaded, the tool uses the value in the name field to locate the desired process. This is accomplished by continuously walking the /proc directory and inspecting the cmdline file. The cmdline file holds the name of a currently executing process. A match is achieved when the cmdline file contains the same process name as specified in name.

Library Base Resolution

The memory map file /proc/<PID>/maps is parsed to uncover the base address of the library listed in image. Later, this memory region will be used to resolve the address of the desired function. Additionally, at this time the process is suspended with process trace (ptrace).

Symbol / Function Resolution

Before any modifications can occur, the address of the desired function must be resolved. This is done by symbol resolution and/or pattern matching. For symbol resolution, the dynamic symbol table of an Executable and Linkable Format (ELF) file is parsed to determine the address of the function matching the value in symbol. On symbol resolution failure, the memory region occupied by image is searched for the value in pattern.

Injection

With critical steps for injection accomplished, the value contained in hook is written to the processes memory space. Finally, the function specified via symbol or pattern is redirected to the injected code and the process is resumed. This is all accomplished using the ptrace application programming interface (API).

Summary of Files

Various components of the memory modification tool are broken out into the following files:

File Description

patch.c Contains the main loop logic of the tool.

parser.c/h Parses the configuration file.

poll.c/h Determines the process to inject into by walking /proc.

resolve.c/h Resolves the address of the function via symbol or pattern.

hooker.c/h Injects code into the target process.

myptrace.c/h Convenience functions for ptrace.

test.ini Contains the required configuration values for proper execution of the tool.

Compilation and Configuration

The Android Native Development Kit (NDK) is required to compile the tool. Compilation is accomplished by executing the NDK build script from the project directory

Figure 4 – Compilation

Configuration of the test.ini file requires several steps. First, the desired process must be identified by its image name. Next, the desired library name must be identified. Finally, the hook code needs to be generated. These steps are demonstrated in figures 5 and 6.

Figure 5 – Process and Library Name Identification

Figure 6 – Hook Code Generation

These values should be placed in their respective locations within test.ini. Also, generation of the symbol and pattern values should be accomplished.

Future Work

Areas for improvement within the scope of the current tool design include improving the process search, symbol resolution and process injection. Furthermore, adding dynamic library load detection and an in-memory assembler/disassembler will make the tool more robust and easier to maintain.