Author: Troy Shurtleff


---

**Reverse Engineering the Variable Message Sign**

- Objective #1 (Recover the Default Password)
- Objective #2 (Identify and Report a Security Vulnerability)

---



---

Approved for public release; unlimited distribution
Not export controlled per ES-FL-091619-0203

# Reverse Engineering the Variable Message Sign

## Objective #1 (Recover the Default Password)

***Exercise: Provide a screenshot showing the entry point of the 'check_password' function in the Listing view.***

***Exercise: Provide a screenshot showing the original decompilation of the 'check_password' function performed by Ghidra.***

***Exercise: Do you observe any global variable names in the 'check_password' function? If so, what are they called? What data do you think might be stored in these global variables (In other words, what are they used for)?***

***Exercise: Analyze the Ghidra's decompilation of the 'check_password' function and rename some of the local variables so the names reflect more accurately how they are used. Provide a screenshot of your updated function.***

***Exercise: Describe the algorithm used to decrypt the correct password for the `signApp`.***

***Exercise: What are the byte values, in hexadecimal, for each character in the ciphertext password?***

***Exercise: What is the key used for decrypting the ciphertext password?***

***Exercise: What is the ASCII character representation of the plaintext password?***

## Objective #2 (Identify and Report a Security Vulnerability)

***Exercise: Identify all the registers we control.***

***Exercise: For all of the registers we control, identify (1) the register identifier (e.g. `r0`, `pc`), (2) the sequence of bytes in that register, and (3) the offset into the input buffer where the value starts. Count the offsets using a 1-based index (e.g. in the following buffer the sequence of four 'B' characters starts at index 5: AAAABBBBCCCC).***

***Exercise: Construct a buffer that populates a four-byte sequence of ASCII 'B' characters (0x42) in the `pc` register and a four-byte sequence of ASCII 'C' characters in the `r0` register. Your buffer will cause the `signApp` process to crash and the register state will be captured in the resulting core file. Your buffer should contain only four ASCII 'B' characters and only four ASCII 'C' characters. All other characters in your sequence should be ASCII 'A' characters. Your buffer should end with the sequence of four ASCII 'B' characters. Provide the following three items as your response to this exercise:***

1. ***The text of the command you ran to send your buffer to the VMS Administrative Interface.***
2. ***A screenshot showing the full output you see in the terminal after running the command.***
3. ***A screenshot showing the state of the registers, as captured in the core file from the `signApp` crash, in GDB.***

***Exercise: Provide a screenshot of the VMS process listing in your environment.***

***Exercise: Provide a screenshot of the memory map for the `signApp` process in your environment (i.e. the contents of the `/proc/[pid]/maps` pseudo-file).***

***Exercise: What is the virtual memory address of where we can find the data stored in the `message_of_the_day` variable?***

***Exercise: What is the base address for the `libc.so.0` library in the context of the virtual memory address space for the `signApp` process?***

***Exercise: What is the virtual memory address of where we can find the `system` function in the `signApp` process?***

***Exercise: Provide the following items to demonstrate you were able to complete a working proof-of-concept exploit.***

1. ***The text of the command you ran to send your final proof-of-concept attack buffer to the VMS Administrative Interface.***
2. ***A screenshot showing the full output you see in the terminal after running the command.***
3. ***A screenshot showing the results of running the `id` and `uname -a` commands in your `netcat` shell connection to the VMS.***

---

Author: Troy Shurtleff