ECE 531: Introduction to Internet of Things

Homework #4: Attack Surface of Daemon

Joseph Hilland – 101104172 – jhilland@unm.edu

Summer 2022

Contents

List of Figures

2 Abstract

This document outlines the attack surface for the current ARM image and daemon application written in the C programming language. This document will provide a guide to further harden the C daemon program from any attack vectors that are found. Hardening the ARM image is out of scope of this assignment, however it will be helpful to identify as many of these vectors for possible hardening at a later date.

The C daemon application prints the current system time to syslog every second. The application does not require any command line arguments to start. The original application structure was given by Professor Lamb in the Module 4 slides.

3 Attack Surface

The attack surface is the number of possible attack vectors where unauthorized users attempt to access and/or extract data. The attack surface can encompass the entire ARM image, kernel, network and programs running. For this assignment, the scope of the attack surface will be limited to the startup script and the daemon application written in C. Based on the analysis of the attack surface, the C daemon application and startup script will be hardened. Securing the ARM image and kernel will be out of scope for this assignment. However, attack vectors pertaining to these components may still be listed.

3.1 Attack Vector: Harden Variables

Fault injection attacks (FIA) is one method of causing adverse side effects within a program. With FIA, the attacker can attempt to swap bits on the stack or via registers to cause simple conditional checks within the code to occur. Historically in C code, false values equate to 0 where as true values are anything other than zero. One method for hardening against these attacks would be to define an enumerate type.

3.2 Attack Vector: Argument and Input Validation

Input validation involves filtering or rejecting data that might be malicious coming into the application. Argument validation can occur further within the application, but at minimum the input should be validated. In the daemon application program, there is no user input expected.

3.3 Attack Vector: Attacks on Formatting Functions (printf(), syslog())

The printf() ad syslog() functions both require an argument that specifies a format, as well as other variables to produce the formatted output. So, the program needs a way to protect a function that accepts a variable number of arguments from reading more arguments than were passed to the function. The application should have hardcoded strings rather than allowing from input. This ensures that user input cannot utilize the '%s' as input to print out string contents from the program.

3.4 Attack Vector: Buffer Overflows and Integer Coercion

C does not perform array bounds checking. This turns out to be a critical security issue especially when handling strings. The risk is increased with local variables as they are on the program stack. There are many options to harden against these types of attacks such as using 3rd party software for string handling, asserting that buffer lengths are correct length within the program, or even move to another language that does not pose these risks. The SafeStr library is another tool that can be used for hardening against buffer overflows. This library provides implementation of dynamically sizable strings in C.

Integer values can go out of range that may not be obvious. The programmer must be diligent in ensuring these types of values stay within range. Unit testing can be a great help with testing code sections that may contain integer coercion or wrap around issues.

3.5 Attack Vector: SSH Access

SSH keys can be used for accessing a system remotely. Often times these keys do not have expiration dates and are valid until someone removes them from the server. There are some configuration settings that reduce security risks around SSH.

- Disabling password-based authentication.
- Disabling root account remote login.
- Allow or deny access based on user or IP address.
- Port forwarding.

3.6 Attack Vector: Coding Best Practices

There are many resources regarding coding best practices. When dealing with user input it is best to follow the below guidelines.

- Prefer rejecting data to filtering data.
- Perform data validation both at input points and at the component level.
- Do not accept commands from the user unless they are parsed by the program.
- Beware of special commands, characters and quoting.

Programmers should also follow coding best practices when working within a team environment.

- Use version control and peer review processes.
- Conduct unit testing at all levels of code.
- Use email lists, NVD, NIST resources and continuous integration for vulnerability checks.

Best practices for coders also include:

- Scope variables to only appropriate level of usage, class, method, etc level.
- Handling variable or function accessor levels, private, protected or public.

3.7 Scripting vulnerabilities

Don't use variables in the printf format string within the script. Use printf '..%s..' "$foo". Always utilized the least amount of privileges as possible. Try not to use root user for the usage of scripts. This will ensure that if the program is not running as root, then an attacker cannot utilized root access from gaining entry into the program. It is however recommended to use root for running scripts or any programs within /etc directory.

4 Conclusion

Most of these vulnerabilities do not pertain to the simple daemon application that prints time. Proper strings are utilized within functions, user input is not required therefore validation is not needed. The program does not use any buffers itself (except within function calls to other libraries). However, since the program is not using user input, it is not expected that buffer overflows will occur.

5 References

`https://h2lab.org/blogposts/dev_c_fia/`
`https://edu.anarcho-copy.org/Against%20Security%20-%20Self%20Security/secure-programmin`
`https://security.web.cern.ch/recommendations/en/codetools/c.shtml`
`https://cheatsheetseries.owasp.org/cheatsheets/C-Based_Toolchain_Hardening_Cheat_Sheet.`
`https://www.shellcheck.net/`