REVERSE ENGINEERING - CLIENT-SERVER LIVE CHAT SERVICE

Akshay Kaikottil
Johns Hopkins University, Information security Institue
Baltimore, MD
akaikot1@jh.edu

***Abstract*— The task was to reverse engineer an open-source software and a custom binary using various diagnostic tools and security assessment techniques. The open-source software we have is chatroom application – Server-Client Live Chat Service, it's a Chatroom service that connects to a web server using web socket which runs in Linux. We were using tools like obj dump, ldd, nm, strace and VM table dump to analyze the binary we build. We also utilized the Scitools, Kaitai struct tool, arch studio to better visualize the binary. We initially fixed a single byte error in the custom binary and visualized it using tools like binvis.io and veles tools. Lastly, we have attached key screenshots to our task.**

Keywords—chatroom, sockets, open-source

I. PROJECT OVERVIEW

The project included performing various reverse engineering tasks on 2 separate binary files. The first binary file is the open-source application we selected, Server-Client Live Chat Service, which must be compiled and run in Linux. We were able to accomplish this by doing the following:

- Dumping the headers
- Listing the shared library dependencies
- Listing various symbols in the binary
- Dumping VM tables
- Diagnosing with Strace tool
- Constructing a top-level system architectural diagram
- Plotting various views like cluster call butterfly graph and control flow graph
- Analyzing the software for new vulnerabilities and the approach for discovering the same

The second binary file, that was provided, had to be initially corrected and it was reverse-engineered through visualizing tools like:

- Binvis
- Veles

We were able to visualize the reverse engineer both the binaries and the results from the various tools provided a wholesome picture of both the software in a very different light. This helped us in understanding how to discover the vulnerabilities using reverse engineering methodology.

II. SERVER-CLIENT LIVE CHAT SERVICE OVERVIEW

Server-Client Live Chat Service makes use of sockets to send and receive data over a network. The application has two major components; a sever and a client. The chatroom is initiated by the admin by calling the server function. The users then join this server as clients by providing the server address, username and password provided by the admin.

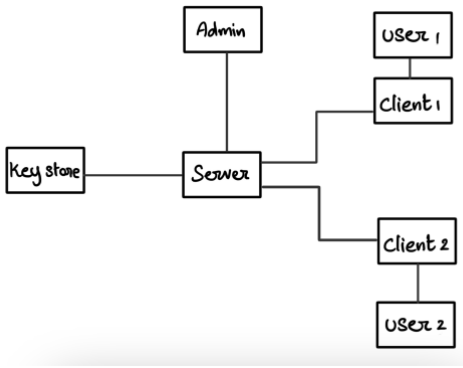

Figure 2.1: Architecture diagram Server-Client Live Chat Service

III. PROJECT EXECUTION

We started initially by going through the documentation regarding Server-Client Live Chat Service that is available on GitLab [https://github.com/andrea-covre/client-server-live-chat-service]. We git cloned the code to our Linux system. We installed the pre- required libraries provided by the documentation. This was followed by configuring, installing, and compiling we obtained the binary. We used the Kaitai tool for decompiling the binary and was able to parse through the file which made it comprehensible.

We were able to obtain the relevant headers using the command $ objdump -h server, which displays summary information from the section headers of the object files. Then the shared linked libraries of the binary are obtained through the command $ ldd server.

The symbols from the binary executable are obtained through the command $ nm server. We were able to further explore the functionality of the service by running the binary. We dumped the virtual memory tables associated with the binary using the command $ cat /proc/<pid>/maps where pid corresponds to process id of Server-Client Live Chat Service. We also were able to diagnose the Server-Client Live Chat Service using strace, $ strace - p <pid>, which intercepts and records all the system calls and signals which

are called and received by the Server-Client Live Chat Service respectively.

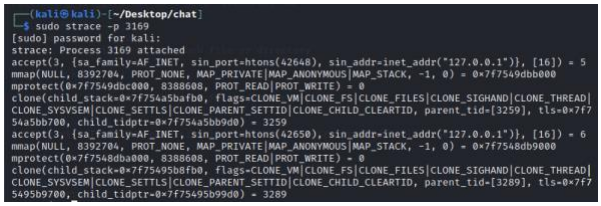

Figure 3.1: Strace of the Server

Hence, we were able to document the key functionality of the Server-Client Live Chat Service and the various interaction with the other system process. By these observations, we were able to also construct a top-level architecture diagram using Arch Studio, which clarified different parts of the software structure.

We were also able to plot various metrics associated with the binary by using the SciTool Understand.

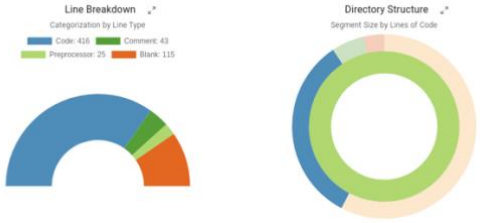

Figure 3.2: Metrics overview of Server-Client Live Chat Service

With the same tool, we were also able to get a visual representation of the binary file with plots like cluster call butter-fly graph and control flow graph views that represent pictorial depiction between components and the logic flow.

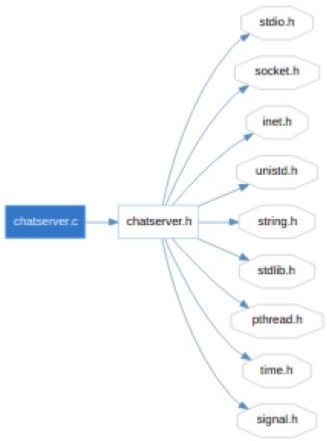

Figure 3.3: Cluster call butterfly graph form Server-Client Live Chat Service

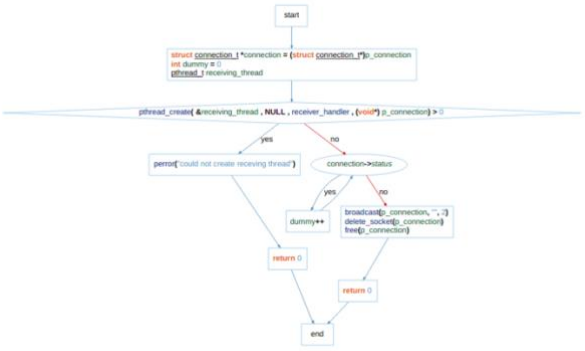

Figure 3.4: Control flow graph form Server-Client Live Chat Service

We were able to identify functions that handled the sanctifying of user input and possible vulnerabilities present in the software.

IV. Costum Binary Executable

We started reverse engineering of the binary file using various visualization tools like binvis.io and veles tool.

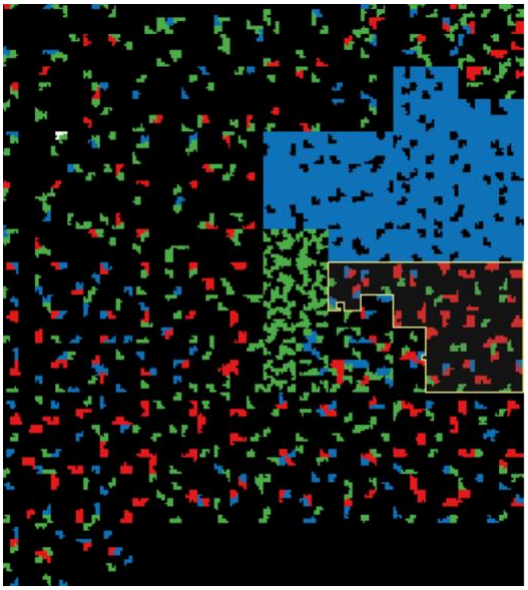

Figure 4.1: Binvis Entropy Image before correcting byte

We initially fixed the single-byte error in the binary file by going through the headers.

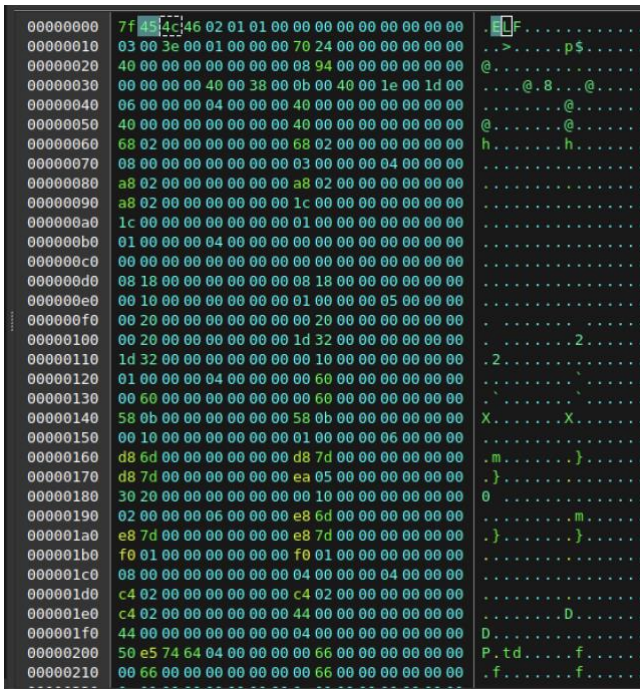

Figure 4.2: Modified 2nd bit (E→A)

We were able to compare the entropy and file structure of both the erroneous and fixed binary files using binvis.

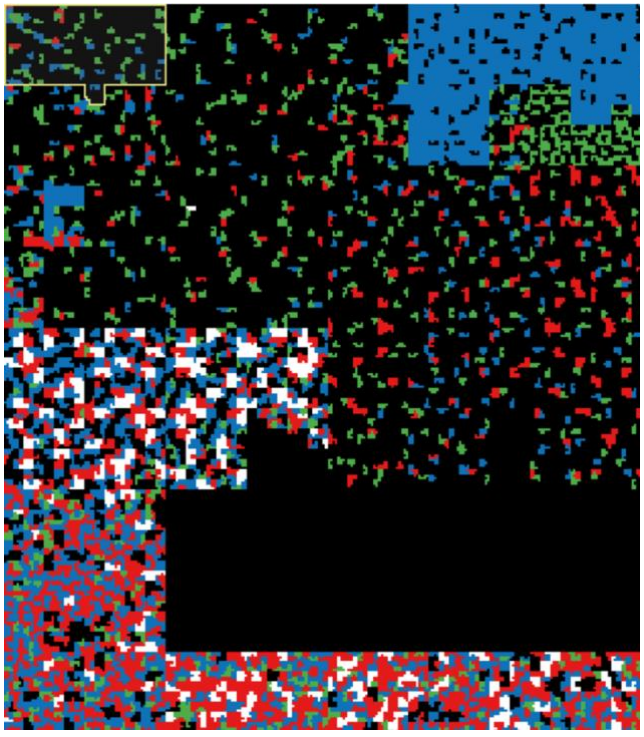

Figure 4.3: Binvis Entropy Image after correcting modified byte

We also used the veles to visualize and observe the layered and trigram view of both the binaries and noted the difference.

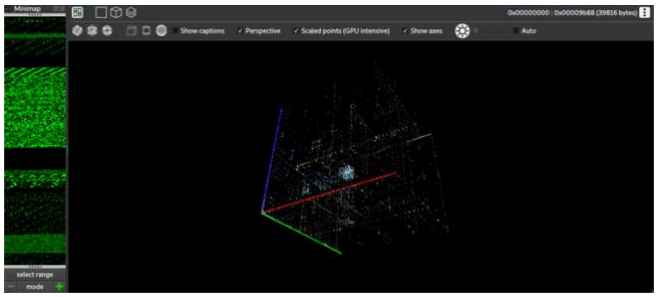

Figure 4.4: 3D visualization using Veles

We also used the tools from previous tasks to identify the behavior, headers, and executables of both binary files.

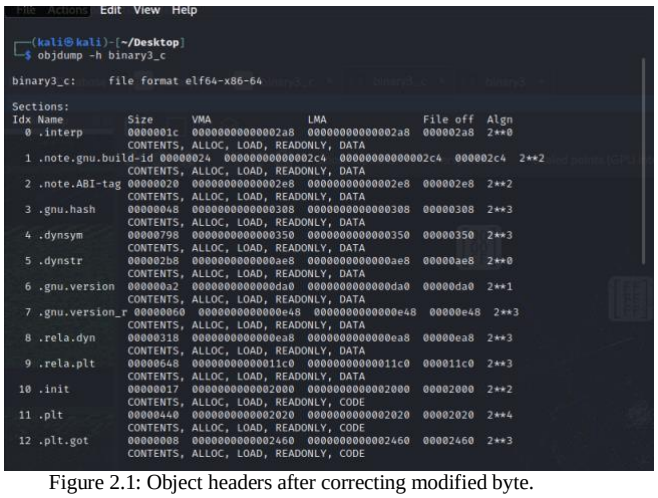

Figure 2.1: Object headers after correcting modified byte.

V. ACCOMPLISHMENTS

We were able to understand the binary in more details after reading the man pages for several Linux commands like ldd, objdump and dumping the process VM Tables. Also, we were able to successfully compile the program using the steps outlined in the project website and in the process, we encountered several issues, and after referring online we solved the issues after installing build tools in the Linux system and able to make the build file and install it on the Linux system.

We were able to launch the process and able to extract the Process id and use cat/proc/<PID>/maps to obtain the VM Tables which are very useful resource to understand the dynamic linking inside the binary and, we used strace system command to see how the binary execution takes place internally and able to read the strace command output and identified several sys calls and interrupts. Several online resources and man pages gave us valuable information on how to use these tools to successfully analyze the binary.

We were able to successfully build a binary and reverse engineer it with the help of several industry grade tools. While performing the reverse engineering, we were able to think like an attacker and execute several commands just like how an attacker tries to obtain information from a given binary including headers dump which gave us valuable information about various sections of the binary including heap, stack, code and text, this information can be used to launch attacks such as Buffer Overflow and dirty cow vulnerabilities.

Also, we analyzed the binary source code of the Server-Client Live Chat Service and identified a top-level system architecture and identified main system functions and various

threat levels associated with them. While using the Understand tool, we learned about the code level metrics like no of executable lines, no of functions, etc., and able to successfully identify the dependency graph and call flow graph of the Server-Client Live Chat Service.

ACKNOWLEDGMENT

The author wishes to thank Reuben Johnston for his support for the successful completion of Reverse Engineering the Server-Client Live Chat Service.

REFERENCES

[1] Server-Client Live Chat Service, https://github.com/andrea-covre/client-server-live-chat-service

[2] Cybersecurity Chat: The Security Risks Of Popular Collaboration Apps, https://www.forbes.com/sites/forbestechcouncil/2019/04/03/cybersecurity-chat-the-security-risks-of-popular-collaboration-apps/?sh=506199493e1d

[3] The Instant Messaging Menace: Security Problems in the Enterprise and Some Solutions, https://www.sans.org/white-papers/479/

KEY TERMS

Threat: Threat is a possible risk that might exploit a vulnerability to breach security.

Vulnerability: A flaw in a system that leaves information expose to a threat.

Attack or Exploit: Code or data which takes advantage of a vulnerability in a system for malicious intent.