A Chat Application on a Bare Internet

Fahad S. Alotaibi
Dept. of Computer and Information Sciences
Towson University
Towson, MD, US
falota3@students.towson.edu

Ramesh K. Karne
Dept. of Computer and Information Sciences
Towson University
Towson, MD, US
rkarne@towson.edu

Alexander L. Wijesinha
Dept. of Computer and Information Sciences
Towson University
Towson, MD, US
awijesinha@towson.edu

Nirmala Soundararajan
Dept.of Chemistry, Computer and Physical Sciences
Southeastern Oklahoma State University
Durant, OK, US
nsoundararajan@se.edu

Abhishek Rangi
Dept. of Computer and Information Sciences
Towson University
Towson, MD, US
arangi1@students.towson.edu

***Abstract*—Chat applications are available on many computer platforms. We present a novel chat application on a bare Internet using bare PCs. In a bare Internet, which is overlaid on and coexists with the Internet, all computing devices are bare, meaning they have no operating system and no persistent storage. We describe the design and implementation of the chat application and use it to conduct preliminary tests on the Internet. The results show the feasibility of a bare Internet. Our contributions include a simple chat design, a closed system approach, a bare Internet architecture, context-based user authentication, security by design, server-controlled chat sessions, and extensibility to other application domains. This work lays a foundation to build other domain-specific applications on a bare Internet.**

Keywords—bare machine computing, bare Internet, chat application, context-based authentication

I. INTRODUCTION

Today's Internet and its applications serve the needs of a growing number of users This number is currently in the billions [1] and continues to grow. It is difficult to control and manage Internet use due to this growth. Furthermore, attackers exploit multi-dimensional security vulnerabilities [2] or use social engineering to learn passwords [3] and other secret information. Physical security, DDoS attacks, and quantum-safe computing are additional challenges when designing secure networks. Stronger defense mechanisms can help to counter some threats and protect the privacy of communications. Many organizations utilize a Zero Trust model that includes multi-factor authentication and micro-segmentation. Also, cryptographic mechanisms include forward secrecy as a basic requirement to minimize the impact of an attack. Unfortunately, attackers often only need to find a single flaw in an application, protocol, or security mechanism to gain unauthorized access to a network. In addition, the ubiquity of IoT devices [4] and wireless networks provide a larger attack surface for exploitation. NSA gives an overview of measures that can help balance convenience and security for mobile devices [5].

In this paper, we propose the use of an alternate approach, bare machine computing (BMC) [6, 7] and an associated novel bare Internet architecture that may make it easier to address some of the security concerns noted above. In essence, BMC is application-centric and adopts a minimalist approach that eliminates the operating system (OS) and its vulnerabilities. BMC originated over 20 years ago with the application-oriented object architecture [8] and dispersed operating system computing [9]. Several BMC network applications including web servers, email servers, and VoIP systems have been implemented and tested (see Section II). Standard protocols such as TLS, IPsec or SRTP were used to secure these BMC applications as needed. However, BMC and its associated paradigm has not been investigated as a possible means to achieve security by design for applications. A detailed investigation of the security tradeoffs relevant to BMC is not a trivial undertaking and is not possible given the space limitations of this paper. In lieu, we design, implement, and test a new BMC Internet application, chat (aka instant messaging), that serves to demonstrate the feasibility of a secure bare Internet wherein users can communicate privately in real-time without being overseen by an OS. In addition to reducing the attack surface, the absence of an OS results in a smaller code size and less complex code making it easier to detect and fix security flaws. This paper focuses on describing the internals of the bare chat application and does not discuss the relatively straightforward addition of the existing bare implementations of TLS and/or IPsec to secure chat sessions. The main contributions of this work are summarized below.

1. Design and implement a novel and reliable UDP-based bare chat application independent of any OS or third-party software.

2836-3795/24/$31.00 ©2024 IEEE
DOI 10.1109/COMPSAC61105.2024.00386

2. Demonstrate the feasibility of private real-time communication without typical OS vulnerabilities.
3. Show how a bare Internet can be overlaid on the current Internet.
4. Test the bare Internet chat application with multiple clients and a single server.
5. Illustrate the use of special commands at run-time to control, monitor, and authenticate chat sessions.

The remainder of the paper is organized as follows. Section II gives an overview of the BMC paradigm and bare Internet concept. It also briefly reviews related work. Section III introduces the chat application. Section IV presents an architecture for a bare Internet. Section V provides design and implementation details of the chat application. Section VI describes testing the chat application on a bare Internet and discusses preliminary performance results. Section VII contains the conclusion.

II. Bare Machine Computing and a Bare Internet

A bare machine is simply a physical computing device with no persistent storage and no operating system. It is for general-purpose computing and is thus not an embedded system. A bare machine has no value-added resources in it and does not run any code until a program is loaded from external media such as a flash drive. Bare applications run on bare machines and are based on the BMC paradigm, which uses a single programming language without the support of other code interfaces. This enables applications to directly access the hardware. The necessary hardware drivers and network protocols are part of the application. It is not possible to open new ports or execute unintended operations on the hardware at run time. Bare machines only support wired Ethernet connections.

The bare application code consists of a self-contained single monolithic executable that can bundle a given suite of applications. The application-centric BMC paradigm can be customized if needed as a closed user-controlled approach. Bare machines running BMC clients and servers can be connected via Ethernet switches and ISP gateways forming a virtual bare Internet that is overlaid on the existing Internet. In future, a bare Internet can be used for a controlled environment, where domain-specific applications including chat, email, banking, and voice communication between private groups can be securely conducted. It is possible for BMC applications in a bare Internet to interoperate with conventional OS-based Internet applications, but this increases the attack surface.

The bare chat application and bare Internet concept is based on BMC research done in prior studies. Existing bare applications include multicore Web servers [10-14], a mail server [15], a bare file system [16], a bare SQLite database [17], a bare USB driver [18], a text-only browser [19], and a split server [20]. These applications run as multi-threaded programs with hundreds of threads and yield high performance with built-in security as there is no centralized OS or kernel. The applications inherit a very small image with limited functionality according to needs, and a single programming environment with no outside dependencies and controls. It is essentially a closed end-user system with full control given to the domain-specific application users.

Approaches such as Exokernel [21], which gives applications more direct access to hardware, Kitten [22], which is a lightweight kernel (LWK) OS with a virtual machine monitor (VMM) called Palacios for loading guest OSs, RIOT for IoT [23], and microkernels [24] are like BMC in that they reduce the OS footprint, but none eliminate the kernel.

While OS-specific security issues are well-known, most do not directly apply to BMC since there is no OS or kernel. For example, it is not possible to run scripts, use DLLs, modify the statically compiled bare code and/or execute malicious/remote code, reassemble fragmented packets, or escalate privileges in a bare system. OS attacks targeting hardware and software vulnerabilities are discussed in [25]. The minimal kernel OS architecture in [26] attempts to prevent physical attacks.

III. A Novel Chat Protocol

Some conventional chat applications follow the XMPP standard (RFC 6120). Popular OS-based chat applications such as WhatsApp, WeChat, Facebook Messenger, Telegram, and Snapchat [27] require a server that does not usually monitor chat sessions. The bare chat application is different from conventional chat applications in that it is server-controlled. We describe its novel characteristics below.

In a one-to-one bare chat protocol, a pair of clients can communicate with each other. The bare server relays packets between the clients and provides all communication and control. Server involvement ensures the reliability of chat sessions. The server can also be used for user authentication, and to guarantee the privacy, integrity, and authentication of data exchanged during chat. This approach reduces the ability of attackers to compromise chat sessions, but only introduces a modest overhead on the server with respect to delays as shown later. Each user gets a username from the server administrator, which is unique among the domain-specific users. The message exchange for the one-to-one protocol is shown in Fig. 1. A client does registration (REG) to inform the server that its user is online. All information for a registered user is stored in the server. Each user must issue a chat command (CCMD) with the other user's ID, which will be forwarded to that user's client. The clients also exchange their respective acceptances. All interfaces go through the server as shown in the dotted lines in the figure. Each client waits until it gets a "GO" command from the server, which is sent after receiving accepts from both clients. The clients can then send and receive data. After sending data, a client waits for an acknowledgment. As shown in the figure, a server command (SCMD) can also originate from a client and it will receive a response from the server. The server can likewise send a command to manage and control a client.

In a one-to-many bare chat protocol, a given client can communicate to two or more clients at the same time. We implemented a one-to-two client protocol as shown in Fig. 2 to demonstrate the feasibility of this mode. In this mode, a client sends two separate chat commands to two other clients. The other clients do not send any chat commands. The server handles accept responses from the other clients and sends the "GO" command to all clients to move to the data stage. All other aspects of this protocol are the same as for the one-to-one protocol described above. In addition, a chat message from a client can also be broadcast to all other clients currently

registered with the server. This design can be modified to broadcast to a select group of clients if needed.

Fig. 3 shows a chat protocol feature referred to as context-based authentication. It can be used when two communicating parties known to each other have some prior communication before they begin a chat session. They can then agree out-of-band on some conversation context before their chat session. The server maintains a database with user contexts and users can dynamically add/delete/modify this context. This feature can complement or eliminate passwords and passphrases between known users in a domain-specific application. The server then authenticates clients and establishes keys based on previously saved contexts. Client contexts can also be accumulated and validated dynamically at the server if desired. Notice that the server is involved in every step during the chat session. Reliability, authentication, security, privacy, and groups are implemented at the server. The server-based approach also avoids synchronization problems. The tradeoffs in using context-based authentication with a server are increased overhead and complexity. Context-based authentication can also be used in conventional systems.

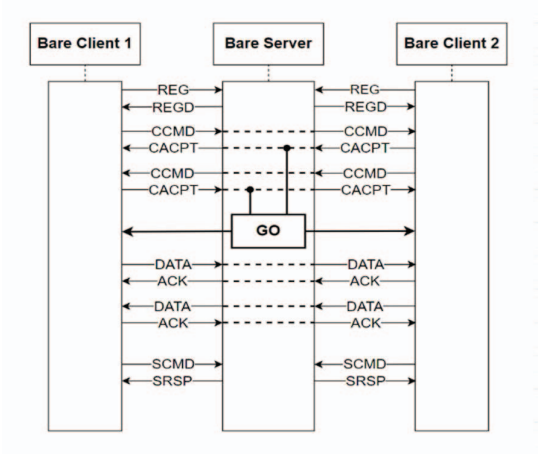

Fig. 1. One-to-one protocol.

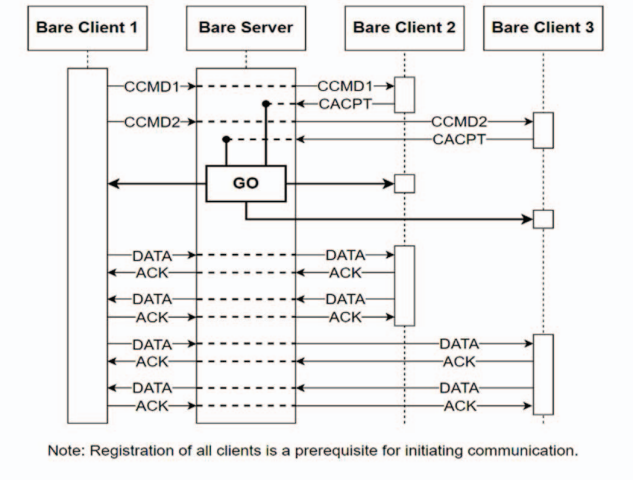

Fig. 2. One-to-many protocol.

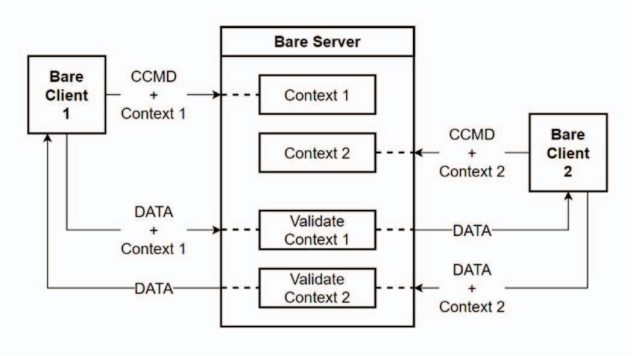

Fig. 3. Context-based authentication mechanism.

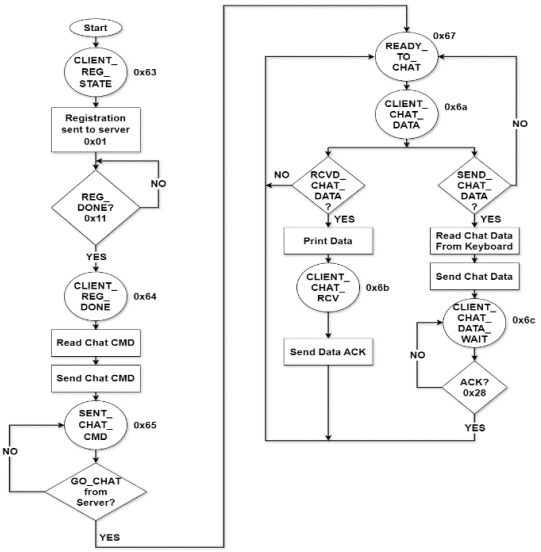

Fig. 4. Client control flow for registration, chat commands, and data.

IV. Bare Internet Architecture

A bare Internet uses the underlying Internet infrastructure, but not its services. In this architecture, a bare server and clients follow the BMC paradigm. In an ideal bare Internet, all nodes including switches, ISP gateways, and routers will be bare. A bare server has a domain name and a gateway with a static public WAN IP. This server is usually connected to a switch, which in turn is connected to an Internet gateway at home or at an enterprise. Similarly, the bare clients are connected to switches and use gateways with public WAN IP addresses. A domain-specific application or enterprise can also be part of a bare Internet. For example, the chat application has its own bare server and bare client groups, which are visible to other domain-specific applications. Each domain-specific application, its users and security policies can be independently managed. An enterprise can also use its own security protocols and mechanisms, user authentication methods, and any application-to-application API. However, using accepted security standards enables interoperability and reduces the chances of undetected flaws that can be exploited by attackers. If there is a need to communicate between two disparate domains, they will connect to another cross-domain bare server. This isolation between

application domains sacrifices convenience but simplifies the provision of security and privacy for clients. Domain-specific applications run on bare Internets, forming secure islands overlaid on the Internet.

V. Design and Implementation of the Chat Application

The chat application protocols described in Section III are designed and implemented using state transition diagrams that capture the control flow of the server and clients. The server and client code design follow these control flows to guarantee the proper operation of the chat application. Control flows are also used in actual code implementation with switch statements. Fig. 4 illustrates client control flow during registration, and when sending and receiving chat commands or responses, and chat data. Each transaction between a client and the server also includes an 8-byte control field. This 8-byte control field included in commands, responses, and data helps simplify the BMC code design and implementation. The codes specified in the control field are used at the sending and receiving sides by clients and the server.

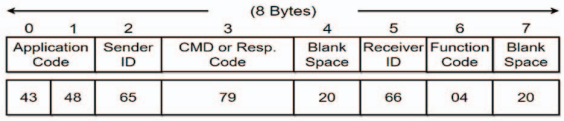

Fig. 5. Control field and an example of command/response codes

An example control field is shown in Fig. 5. Here, Bytes 0, 1 are fixed characters 'C' and 'H', which signifies the start of the control field. Bytes 4 and 7 are space characters. Byte 2 is the client 1 ID in hex (0-255 clients), and byte 5 is a client 2 ID in hex. Byte 3 field indicates either a command or response. In this case, 0x79 indicates it is a server command. There are many command and response codes including those used during registration and chat. Byte 6 is a function code for server commands (FN). The FN code 3 is used for timeouts and 4 is used for a client to end the chat.

Fig. 6 shows the client's state transitions during registration and when setting up chat sessions or transferring chat data. These state transitions help to determine a client's state and are useful for debugging. The interactions between clients and the server were described in Section III. Once registration is done and the chat session is set up, a client will stay in READY_TO_CHAT (0x6a) state to send and receive data as shown in the figure. After receiving data, a client sends an ACK to the sender. Timeouts and other mechanisms are used to handle the asynchronous nature of clients and the possibility of UDP packet loss.

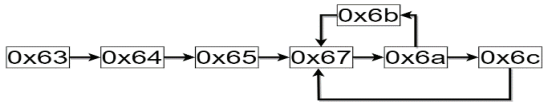

Fig. 6. Client state transitions.

The server and client code are written in C++/C and ASM. All direct hardware interfaces are implemented in C++, which calls C functions, which in turn calls ASM functions if needed. The only way to reach hardware at the end is using an assembly call. A direct hardware interface in BMC is also referred to as a hardware API (HAPI). This is not the same as a system call to an OS or kernel as there is no centralized kernel or OS running in a bare machine. A BMC programmer uses HAPIs based on events that occur in the application. The event driven approach in BMC eliminates a centralized OS or kernel calls. Fig. 7 shows some sample code. Code BLOCK 1 shows a client sending data to another client. It reads data from the user and sends it to the server. The server in turn relays the data to the other client. The sendChatData() function invoked by the application programmer is a HAPI used to directly send data from the Ethernet buffer. Code BLOCK 2 shows a client receiving data from another client and printing the data on the screen. The AOAPrintTextChat() function prints the data on the user screen using HAPIs. Code BLOCK 3 shows the server receiving data from Client 1 and sending it to Client 2, using the HAPI sendDataNew(). It validates the data from client 1 and removes context data before sending it to Client 2. Client 1 and Client 2 details are stored in a linear list at the server and updated for each interaction. This list is indexed with the client user ID to avoid search. Code sizes in BMC are small.

A significant part of the implementation involves reading user data asynchronously while the client program is running. To avoid too many interrupts for I/O, as it is a text-only based chat, we defined some special keys to begin and end input commands from the user. For example, the user enters a special first character "%" to begin a commnd and a second character after the "%" character to specify more details on the command. If the second character is "r", it signals the code to start reading user data. Thus, "%r" means read data from the user. Similarly, the second character "c" means send a chat command, "d" means send chat data, "x" means change context, "u" means disconnect from the current chat session, and "q" means the client program should quit and return to the main menu.

VI. Testing Chat on a Bare Internet

Fig. 8 illustrates a bare Internet chat testbed with eight clients: three at Towson University (TU) in MD, two in PA, two in MD (Maryland), and one in OK (Oklahoma). The server in PA (Pennsylvania) used an Optiplex 9010 (3.4 GHz, 4GB memory), while clients used an Optiplex 960 (2.9 GHz, 2 GB memory) or an Optiplex 260 (2.4 GHz, 1 GB memory). Note that the server and clients run on old models with a very small amount of memory. Applications directly communicate to hardware, and they have no platform dependencies except for the instruction set architecture.

Three operating modes were tested. In the first mode called 1:1, there are 4 pairs of users with communication between each user pair, but not between different user pairs. Each user has a unique ID. The user pairs tested were (101, 107), (102, 105), (103,106) and (104, 108). In the second mode called broadcast, any client can broadcast to all other clients through the server. In the tests, users 101, 102, 106 and 107 did broadcast. In the third mode called 1:M, one client communicates with multiple clients. In the test, user 101 chatted with users 102 and 103. In all modes, each client first registered with the server. The server monitored and controlled all chat commands and responses, and generated traces as needed.

Some results from a 45-minute chat session using the testbed are discussed below. Note that the bare screens shown only display text since there is no bare graphics adaptor. Fig. 9 shows sample data on the server screen in 1:1 mode during the session. The top part of the screen shows some debugging and trace information for a multicore server, which was not used for chat. The middle part shows data for all eight clients during chat, where each line is for one client and the columns identify the data. The bottom part shows client IDs and their WAN IP addresses. Fig. 10 shows the Pennsylvania user 104's screen during chat with user 108 in 1:1 mode. The status line shows state transitions in the client code during the chat session. Fig. 11 shows the Maryland user 101's screen with messages exchanged during chat with users 102 and 103 in 1:M mode. In this mode, users 102 and 103 do not do any chat requests.

During the chat sessions using the testbed, packets were captured using Wireshark. For example, Fig. 12 shows a server-side Wireshark capture with broadcast messages in 1:M mode. Similarly, Fig. 13 shows server-side Wireshark data in the 8-byte control header when the server is sending data from user 101 to users 102 and 103. Wireshark data in 1:1 mode shows that a typical registration period took 5.423 milliseconds, a typical client chat command took 5.147 milliseconds, and a typical "GO" response from the server took 47.587 milliseconds. The last delay is large because the server waits until both clients send chat commands and responses. It also includes user input time to type the chat commands. However, when broadcast data is sent to the server, the delay to resend it to all clients is small. The client users also did not encounter any noticeable delay in sending or receiving chat commands and data irrespective of location.

VII. Conclusion

Most chat applications require OS support and run over TCP. We described the design and implementation of a novel UDP-based bare chat application that runs on bare PCs. The chat application illustrates use of a bare Internet, which is overlaid on the existing Internet, and in which all computing devices are bare. We also introduced context-based user authentication for chat security. Three possible modes of bare chat including one-to-one, one-to-many and broadcast were tested. The one-to-two mode shows the feasibility of a server-controlled approach for a chat application. This approach can also be used in other domain-specific applications. Benefits include design simplicity, closed system operation, ease of client control and system management, and server-based chat security. The proposed architecture allows domain specific users to communicate on a bare Internet isolated from other Internet users. Domain-specific applications can customize protocols, security mechanisms, and operations to suit their own needs. This improves security since it is harder for attackers to master disparate domains with heterogeneous protocols and implementations. When all nodes on a bare Internet are bare, many current security vulnerabilities will be eliminated. The proposed approach facilitates security by design. More comprehensive studies using a variety of domain-specific applications on a bare Internet are needed to evaluate the security strengths and weaknesses of this approach.

```
/***************************** START BLOCK 1 *****************************/
// The client state 0x6a
// Client sends data to another client
case CLIENT_CHAT_DATA_SENT: // 0x6a
{
    // When user finishes typing data and wants to send it
    if(io.kbdReadDone == 1)
    {
        // Function to send chat data
        retcode = sendChatData();
        // ..................
        // Some code details are not shown
        // Client sets timer and waits for ACK
        // ..................
        io.kbdReadDone = 0; // To enable reading new input from user
    }
}
break;
/***************************** End BLOCK 1 *****************************/

/***************************** START BLOCK 2 *****************************/
// Receiving data from another client: 0x13
if(UDPPack[0] == 'C' && UDPPack[1] == 'M' && UDPPack[3] == CHAT_DATA_1)
{
    // Print data on the screen
    io.ASAPrintTextChat(&UDPPack[6], (psize - 6));
}
/***************************** End BLOCK 2 *****************************/

/***************************** START BLOCK 3 *****************************/
// Receive data from the sender and forward it to the recipient: 0x13
if(udpdata[0] == 'C' && udpdata[1] == 'M' && uc == CLIENT_CHAT_DATA_1)
{
    client1 = udpdata[2]; // Capture the client 1 ID (sendr ID)
    client2 = udpdata[5]; // Capture the client 2 ID (recipient ID)
    uc1 = (unsigned char) udpdata[6]; // The size of contextual data
    // ..................
    // Some code details are not shown
    // Extract the contextual data from client 1 (sender) packet
    // ..................
    // Send data to client 2
    retcode = sendDataNow (dip1, dport1, ptr, (psize - uc1 -1), dmc1, client2, 0, 0);
}
/***************************** End BLOCK 3 *****************************/
```

Fig. 7. Examples of client and server code.

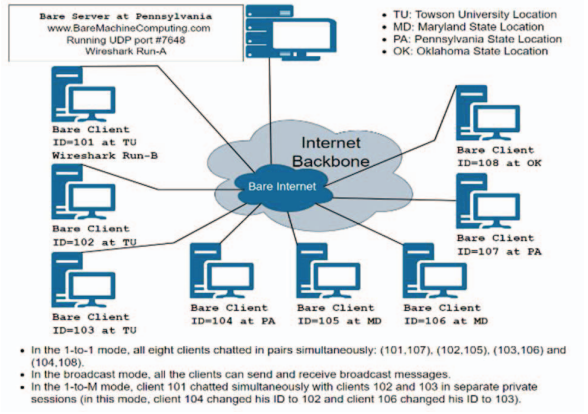

Fig. 8. Testbed configuration.

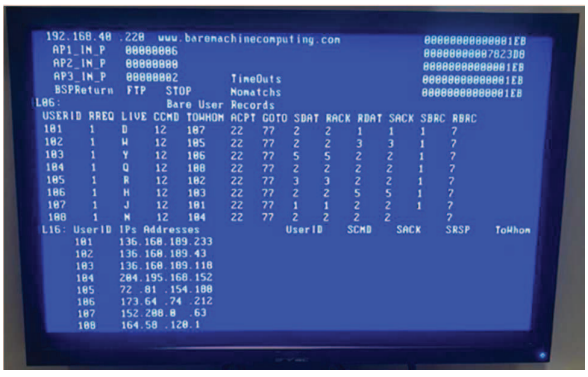

Fig. 9. Server screen for 1:1 mode.

Fig. 10. Pennsylvania client screen for 1:1 mode.

Fig. 11. Maryland client screen for 1:M mode.

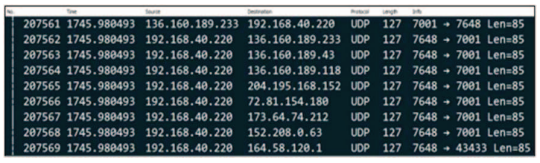

Fig. 12. Server-side Wireshark capture for broadcast messages.

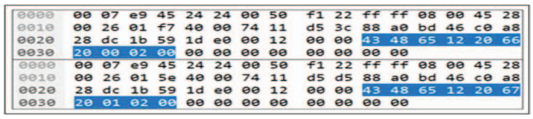

Fig. 13. Server-side Wireshark capture for one-to-many chat requests sent.

REFERENCES

[1] R. Shewale. How many use the Internet in 2024 (global data). www.demandsage.com/internet-user. Accessed: May. 28, 2024.

[2] Ö. Aslan et al., "A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions," MDPI Electronics, Vol. 12, Issue 6, March-2 2023.

[3] C. Gersch and J. Birkett. What makes a secure password. https://cyberhound.com/wp-content/uploads/CH-Research-Paper-Password-Security-LR-.pdf. Accessed: May 28, 2024.

[4] IoT Business News. State of IoT 2023: Number of connected IoT devices growing 16% to 16.0 billion globally – Wi-Fi, Bluetooth, and Cellular driving the market. https://iotbusinessnews.com/2023/05/25. Accessed: May 28, 2024.

[5] NSA Mobile device best practices. https://media.defense.gov/2021/Sep/16/2002855921/-1/-1/0/MOBILE_DEVICE_BEST_PRACTICES_FINAL_V3%20-%20COPY.PD. Accessed May 28, 2024.

[6] Bare machine computing. en.wikipedia.org/wiki/Bare_machine_computing. Accessed: May 28, 2024.

[7] U. Okafor, R. Karne, A. Wijesinha, and P. Appiah-Kubi, "Eliminating the operating system via the bare machine computing paradigm," 5th International Conference on Future Computational Technologies and Applications (Future Computing), 2013.

[8] R. K. Karne, "Object-oriented computer architectures for new generation of applications," Computer Architecture News, Vol. 23, No. 5, December 1995.

[9] R. K. Karne, K.V. Jaganathan, T. Ahmed, and N. Rosa, "DOSC: dispersed operating system computing," OOPSLA'05: Companion to the 20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2005.

[10] F. Alotaibi, R. Karne, and A. Wijesinha, "A stateless bare PC web server," 19th International Conference on Web Information Systems and Technologies (WEBIST) , 2023.

[11] N. Ordouie, R. Karne, A. Wijesinha and N. Soundararajan, "A simple UDP-based web server on a bare PC with 64-bit multicore processors: design and implementation," International Conference on Computing, Networking and Communications (ICNC), 2023.

[12] N. Ordouie, N. Sounderarajan, R. K. Karne, and A. L. Wijesinha, "Developing computer applications without any OS or kernel in a multi-core architecture," International Symposium on Networks, Computers and Communications (ISNCC), 2021.

[13] N. Soundararajan, R. K. Karne, A. L. Wijesinha, N. Ordouie, and B. S. Rawal, "A novel client/server protocol for web-based communication over UDP on a bare machine," 18th Student Conference on Research and Development (SCOReD), 2020.

[14] N. Soundararajan, R. Karne, A. Wijesinha, N. Ordouie, H. Chang, "Design issues in running a web server on bare PC multi-core architecture," IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), 2020.

[15] H. Alabsi et al., "A novel SQLite-based bare PC email server," 15th International Conference: Beyond Database Architectures and Structures (BDAS), 2019.

[16] W. Thompson, R. Karne, A. Wijesinha, H. Alabsi, and H. Chang, "Implementing a USB file system for bare PC applications," 11th International Conference on Internet and Web Applications and Services, (ICIW), 2016.

[17] W. Thompson, R. K. Karne and A.L. Wijesinha, "Interoperable SQLite for a bare PC," 13th International Conference: Beyond Database Architectures and Structures (BDAS), 2017.

[18] R. K. Karne, A. L. Wijesinha, and S. Liang, "A bare PC mass storage USB driver," International Journal of Computers and Their Applications, Vol. 20, No. 1, March 2013.

[19] S.Almutairi, R. K. Karne and A.L. Wijesinha, "A bare PC text based browser," International Conference on Computing, Networking and Communications (ICNC), 2019.

[20] B. Rawal, R. K. Karne, and A. L. Wijesinha, "Splitting HTTP requests on two servers," 3rd International Conference on Communication Systems and Networks (COMSNETS), 2011.

[21] D. R. Engler, "The exokernel operating system architecture," PhD thesis, MIT, 1998.

[22] J. Lange et al., "Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing," International Symposium on Parallel & Distributed Processing (IPDPS), 2010.

[23] RIOT – The friendly operating system for the Internet of Things. www. riot-os.org, Accessed May 28, 2024.

[24] I. Odun-Ayo et al., "An overview of microkernel based operating systems," IOP Conference Series: Materials Science and Engineering, 1107 012052, 2021.

[25] D. Gens, "OS-level attacks and defenses: from software to hardware-based exploits," PhD thesis, Technische Universität Darmstadt, 2018.

[26] S. Zhao et al., "Minimal kernel: An operating system architecture for TEE to resist board level physical attacks," 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID) 2019.

[27] S. J. Dixon, Most popular messaging apps 2024. www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps. Accessed May 28, 2024.