A Bare PC Text Based Browser

Saleh Almutairi
Department of Computer & Information Sciences
Towson University
Towson, Maryland, USA
salmut2@students.towson.edu

Ramesh K. Karne
Department of Computer & Information Sciences
Towson University
Towson, Maryland, USA
rkarne@towson.edu

Alexander L. Wijesinha
Department of Computer & Information Sciences
Towson University
Towson, Maryland, USA
awijesinha@towson.edu

***Abstract*—Most Web browsers require an OS or kernel based environment to run. In addition to being vulnerable to OS-based attacks, such browsers implement numerous features that can be exploited by attackers. We design and implement a text-based browser (TBB) that runs as a bare machine computing (BMC) application without the support of an operating system (OS) or kernel. We first describe the TBB architecture, task design and parser. We then demonstrate its basic functionality by accessing sample Web sites and showing the displayed content. Advantages of the TBB include reduced complexity, small code size, and intrinsic security due to BMC characteristics and the absence of an OS.**

Keywords-Bare Machine Computing (BMC); bare PC; bare PC Web browser; text based browser; Web client.

I. INTRODUCTION

Text-based browsers (TBBs) are useful in environments where the functionality and features of a conventional Web browser are not needed. TBBs used currently include Lynx [1], Links [2], W3m [3] and Browsh [4]. Links also supports graphics mode, while popular browsers such as Chrome and Mozilla support a text-only mode. Advantages of a TBB include enhanced performance and increased security. However, vulnerabilities in the Lynx TBB have been found [5]. Moreover, TBBs and Web browsers in text-only mode typically run with the support of an operating system (OS) or kernel, which makes them susceptible to attacks that target OS vulnerabilities and features [6]. A TBB that does not require an OS or kernel to run may provide an alternative to existing browsers with both security and performance benefits. We describe the design and architecture of a novel TBB that runs as a bare machine computing (BMC) application without requiring additional system software such as an OS or a kernel.

BMC applications run on a bare machine, which is just an ordinary laptop or desktop PC. Even if an OS or kernel is present in the machine, it is not used by the BMC application. In a BMC system, attackers can only exploit flaws in the application itself since there is no OS or kernel. BMC applications are also smaller and simpler than their OS counterparts, making it easier to analyze the code for security flaws and performance bottlenecks.

The bare PC TBB (referred to simply as the TBB from now on) is a self-contained, self-managed and self-controlled application. As with any BMC application, there is no software running in the background other than the TBB itself, and no permanent resident storage (such as a hard disk) is present in the bare machine. The TBB does memory management and uses its own API to communicate with the underlying hardware. It also creates and manages its own threads. All execution resides on a removable USB flash drive, which also includes the code for booting and loading the application. If high security is not a requirement, a USB could also be used for storing files. The TBB uses a customized TCP/IP stack and Ethernet NIC driver that only implement the necessary minimal functionality.

II. SYSTEM ARCHITECTURE

The TBB system architecture differs from that of a conventional Web browser [7] as there is no OS or kernel. In particular, the TBB has direct control of the underlying hardware resources via a BMC API that is customized for the application. The TBB application may be viewed as an event based system, where the occurrence of an event triggers execution of the relevant parts of the code. Fig. 1 illustrates the internals of the TBB architecture.

When the bare PC is powered up after inserting the USB with the TBB application, the boot sector of the USB is loaded into a boot address (0x7c00 in our prototype) using a BIOS (basic input output system) interrupt. This boot code, which consists of 512 bytes, has a mini-loader using interrupt 13h. The mini-loader loads an initial program called the prcycle (protected/real-mode cycle), which enables the system to switch between real and protected modes. The TBB application can be customized based on an individual user's requirements. It is possible to authenticate the user, the USB, and the TBB application code.

If a packet arrives, the main task (MT) gives control to the Receive Task (RT). The RT processes the packet stored in the Ethernet buffer after copying it to the TCP buffer and sends a response to the server if needed. The state of each TCP connection is stored in the TCP Table (TCB). The Web client's port number and server's IP are used as a hash index to the TCB entry.

The Interface Task (IT) provides a keyboard interface to the TBB. The IT is started by the MT. It runs when a user event occurs. The TCP Task (TT) implements the TCP protocol interactions for the TBB as shown in Fig. 2. The Web Task (WT) provides functions associated with browsing the Web, parsing packet content, and displaying the text. These five tasks (MT, RT, IT, TT, WT) provide the complete functionality needed for the TBB application.

The MT and RT are single tasks whereas TT, IT and WT are allocated for each client connection. The latter three tasks are created as task pools and stored in their respective stacks. When a task is needed, it is placed in the Task Circular List (TCL) that is serviced in a first-come-first-serve (FCFS)

978-1-5386-9223-3/19/$31.00 ©2019 IEEE

manner as shown in Fig. 1. The task system as described provides the multi-processing capabilities in the TBB.

The TBB interfaces with the user via the simple menu shown in Fig. 3. The menu enables a user to enter an IP address for a Web site and access its content. The browser provides multiple windows to show Web content. A desired window can be accessed by using the UP and DOWN arrow keys. Menu option 1 allows user to enter the link address, option 2 provides browsing capabilities, and option 3 allows the user to search a given web site. Other information shown in Fig. 3 is used for debugging and monitoring purposes.

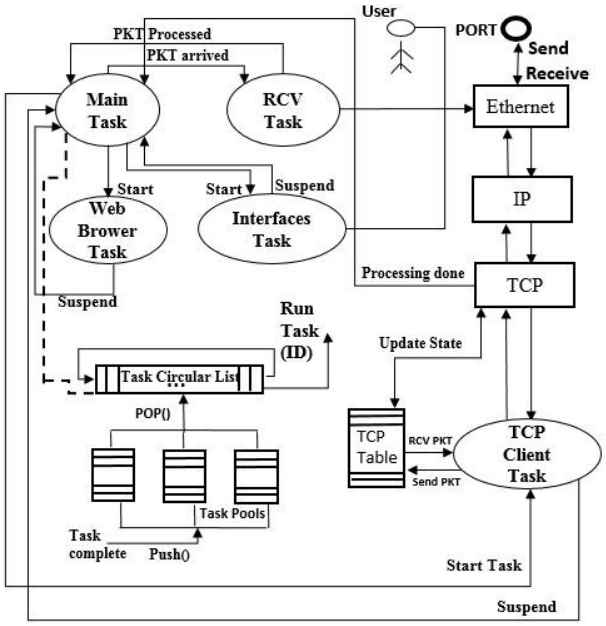

Figure 1. TBB architecture

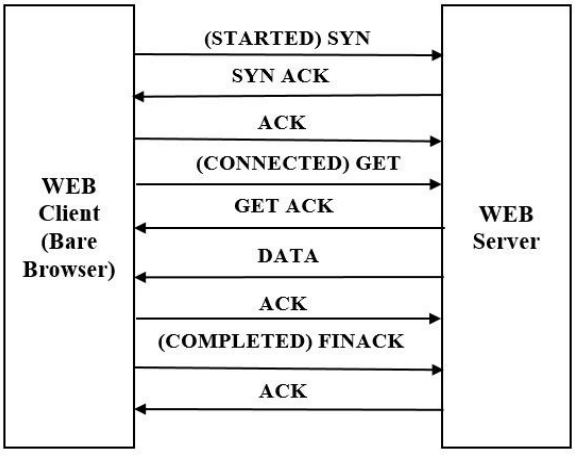

Figure 2. TCP client/server message exchange

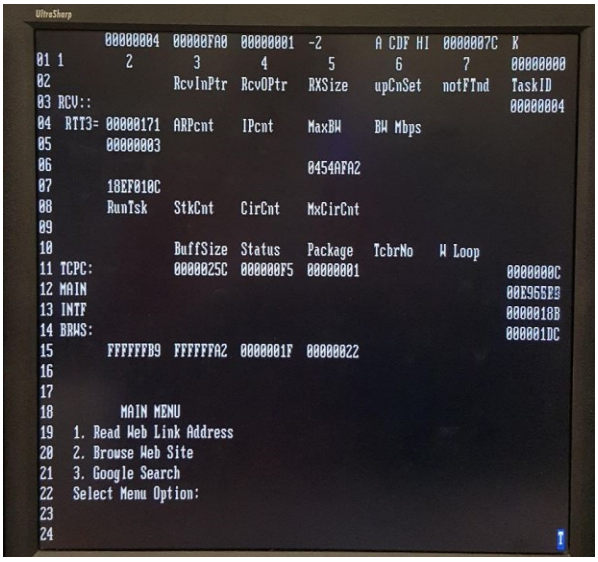

Figure 3. Web browser interface menu

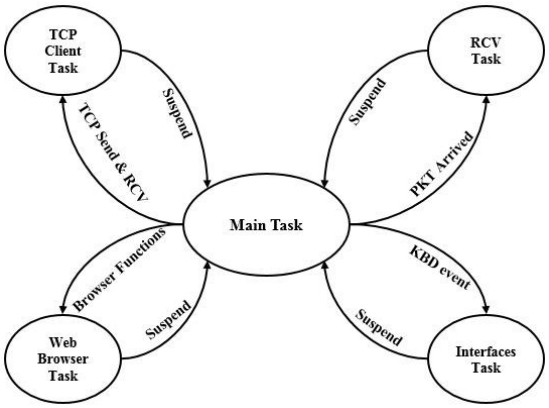

Figure 4. Tasks

III. DESIGN

The TBB functions as a single monolithic system. Unlike in a conventional system, the TBB application includes the necessary system components including interfaces for directly accessing the hardware. While some conventional applications include system functionality in user space, a BMC application does not have any OS or kernel modules. We now discuss more details of the TBB design.

A. Task Interactions and TBB Control Flow

The interactions between the MT and the other tasks in the TBB are shown in Fig. 4. The BMC paradigm requires that when another task has no work to do, it should suspend itself and return control back to the MT. As noted earlier, when a TT, WT or IT task is ready to run, it is placed in a circular list; and when it is done, it is pushed back onto its respective stack. When these tasks are placed in the FCFS circular list, each task/client request is considered to have the same priority. The TBB application controls all aspects of task creation, execution and termination.

The interaction between the IT, WT and TT tasks are shown in Fig. 5. There are no complex OS-type concurrency control mechanisms required for a BMC application running in a single core system. This is because each task runs a small program thread and then returns to the MT. The IT communicates with the WT and TT tasks using a shared object called the Interface Object. The IT stores interface commands and command information in this object so that the TT and WT tasks can access the information needed to start. There is also a cache object shared between the TT and WT tasks. This object stores the data received from the server in cache memory and makes it available to the WT task.

The unique TCB entry that corresponds to a client request and maintains the state of the request is also used to provide concurrency among the requests. The control flow of each request is shown in Fig. 6. The control flow in the TBB transitions through the normal TCP states (connection, data transfer, and termination). The state of each HTTP request is also maintained in the TCB and access to a TCB entry is provided using a hash index as noted earlier. The TBB interacts with the server to access the Web content. Once the data for a given request has been collected, the TBB parses the data and displays the results on the screen.

B. Parsing and Display

The TBB parses only the input data that is relevant. We used state transition diagrams to simplify the parser and discard the unnecessary information. Fig. 7 shows the design of the parser. The design can be easily extended to add other features such as graphics and images. The HTML tags parsed are shown in the diagram. Notice that we only parse web page title, printable text, and related links.

In text only mode, we use video memory to display the text. No graphics or images are displayed in the system. The video memory area is 4000 bytes (at 0xb8000) which is in real memory. Each character requires 2 bytes, one for the ASCII code and another for the color codes. The text in a screen has 25 lines, where each line is 80 characters wide. This requires 4000= 25*80*2 bytes in memory.

In order to have multiple screens, we used UP and DOWN arrows (keys) as the means to move from one screen to the next. The browser is currently limited to 100 screens (0-99), but the design enables extension to more screens if needed. The UP and DOWN arrow enable respectively higher or lower screen numbers. The video memory shown in Fig. 8 is copied to its screen location depending on the screen number. Each screen also maintains its own cursor position and is saved in its own screen memory. The printing functions automatically print the text starting at the current cursor position. When a screen is full, the current screen is automatically saved and the next screen is displayed.

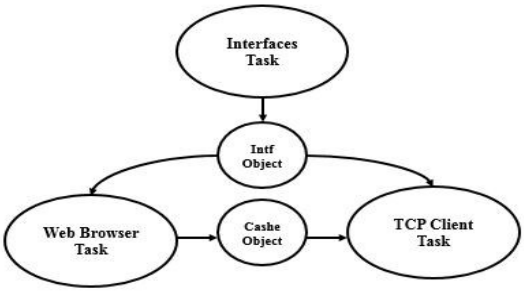

Figure 5. IT, WT and TT task interactions

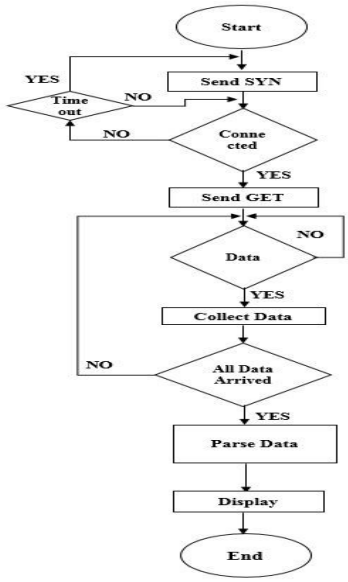

Figure 6. TBB control flow

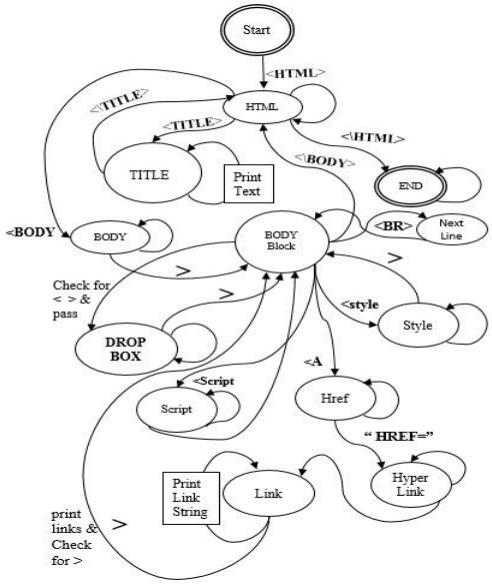

Figure 7. Parser state transition diagram

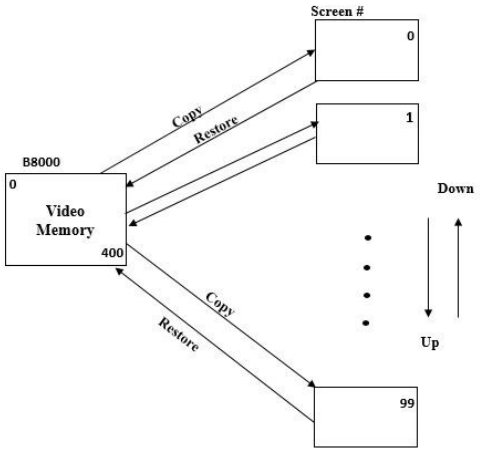

Figure 8. BMC video memory

IV. Implementation and Usage

The TBB is implemented in C/C++ including the API to the hardware. It uses an on-board Intel Gigabit Ethernet network interface controller (NIC) and a BMC NIC driver. The executable size for the entire TBB is 241,664 bytes. This includes the application program and the required execution environment code that enables the TBB application to be self-contained, self-managed and self-controlled. A user controlled USB drive contains the TBB executable including its boot code, which is used to boot up a bare PC and run this application. The TBB runs on any Intel x86 architecture based PC or laptop. It can be extended to work with other CPU architectures by simply developing a hardware API for other instruction set architectures (ISAs).

To launch the TBB, the USB drive containing its application is inserted into a PC or a laptop and the power is turned on. Once the application boots, a mini loader in the boot code will load a BMC menu. This menu can be used to load the application using a BMC loader, run the TBB, or debug the application if needed. When the TBB is running, it is the only running application in the bare PC (nothing else can be run unless the PC is shut down). The user menu shown earlier in Fig. 3 is used to access the Web. The IP address of a site is entered using menu option 1, and menu option 2 is used to browse the site.

We accessed the Web sites listed in Table 2 using the TBB. Fig. 9 shows the content for a sample Web site using a conventional browser (page 0). Fig. 10 shows the same content using the TBB in a bare PC environment (screen 0 as shown in the top left corner). Web links for this page are shown on each line and the title is shown on the top of the page. Figs. 11 and 12 respectively compare display of multiple pages in a conventional browser and the TBB (page 14/page e). The TBB shows each bullet and links on separate lines in the bare PC screen.

TABLE I. Sample Web Sites

Website Name	Website Address	Website IP
Dr. Ramesh K. Karne	http://baremachinecomputing.com/	http://73.133.93.20
LADBS: Build Safe, Well,	http://www.ladbs.org/	http://161.149.40.13
Web Scanner Test Site	http://www.webscantest.com/	http://69.164.223.208
ABC NEWS Website	http://abcnews.go.com/	http://68.71.209.234
LexiConn Internet Services	https://www.lexiconn.com/	http://69.65.13.216

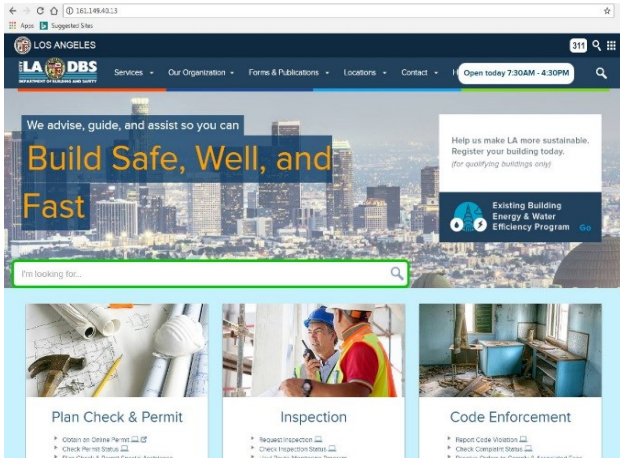

Figure 9. LADBS Web site in conventional browser (page 0)

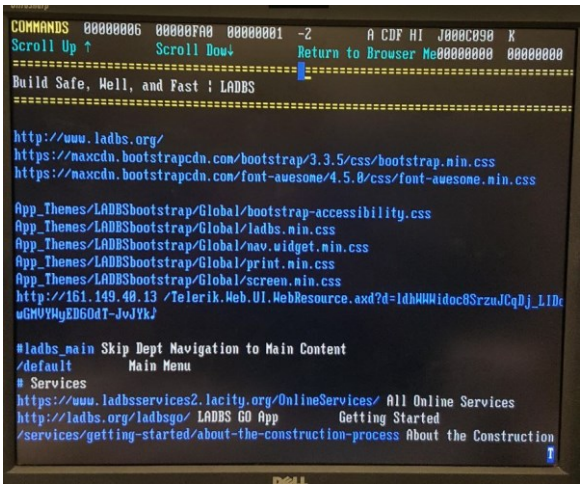

Figure 10. LADBS Web site in TBB (page 0)

V. Related Work

Similar approaches to the BMC paradigm include [8] and [9] in which the OS footprint is reduced to improve application performance. Systems with a hardened OS such as SELinux [10] have some similarities to BMC systems. The main difference between applications running on a minimal OS and BMC applications is that no OS or kernel runs in the background when a BMC application executes. BMC

applications including the TBB are based on the BMC paradigm, which originated from the dispersed operating system concept [11]. Existing BC applications include a Webserver [12], a TLS Webmail server [13], and an HTTP split server [14]. A general approach to implementing BMC applications is detailed in [15], and the development of gigabit Ethernet BMC NIC drivers is described in [16].

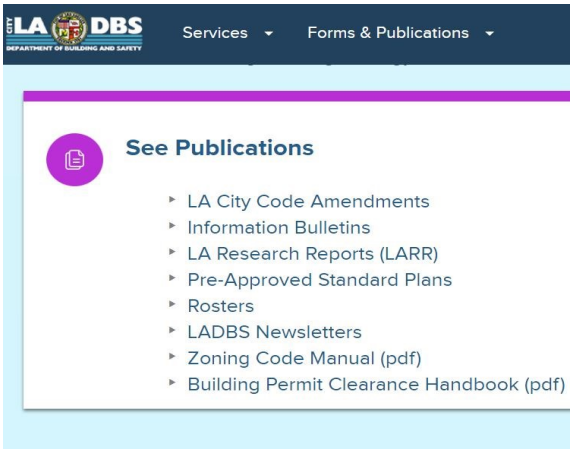

Figure 11. LADBS Web site in conventional browser (page 14)

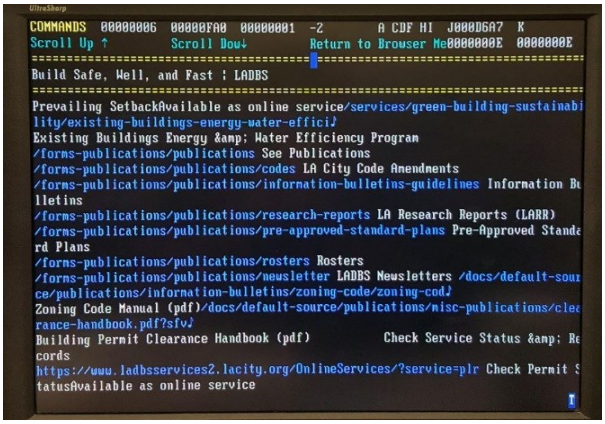

Figure 12. LADBS Web site in TBB (page e)

VI. Conclusion

Conventional browsers are complex and provide a variety of features making them difficult to secure and often degrading their performance. While OS-based TBBs can provide better performance and enhanced security by eliminating unnecessary features, the underlying OS or kernel can be targeted by attackers. We presented a novel TBB that runs on a bare PC with no OS. We described the architecture, design and implementation of the TBB including its task design and parser. It is possible to extend the TBB enabling it to handle links, display images and graphics, and resolve Web link addresses. Further studies are needed to investigate the security and performance aspects of the TBB.

References

[1] Lynx Information [online] Available at: http://lynx.browser.org/ [Accessed 27 July 2018].

[2] Twibright Labs: [online] Available at: http://www.jikos.cz/~mikulas/links// [Accessed 27 July 2018].

[3] W3m Homepage [online] Available at: http://www.w3m.org/ [Accessed 27 July 2018].

[4] Browsh. [online] Available at: https://www.brow.sh/ [Accessed 27 July 2018].

[5] Lynx Lynx: List of security vulnerabilities [online] Available at:https://www.cvedetails.com/vulnerability-list/vendor_id-5836/product_id-9869/Lynx-Lynx.html [Accessed 27 July 2018].

[6] M. Šilić, J. Krolo, and G. Delač, "Security vulnerabilities in modern web browser architecture", 33rd IEEE International Convention (MIPRO), 2010, pp. 1240-1245.

[7] H. J. Wang, X. Fan, J. Howell, and C. Jackson, "Protection and communication abstractions for web browsers in MashupOS", ACM SIGOPS Operating Systems Review, 41(6), 2007, 1-16.

[8] D. R. Engler and M.F. Kaashoek, "Exterminate all operating system abstractions", Fifth Workshop on Hot Topics in Operating Systems, USENIX, 1995, p. 78.

[9] J. Lange. et. al, "Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing," 24th IEEE International Parallel and Distributed Processing Symposium , Apr. 2010.

[10] SELinux wiki [online] Available at: https://selinuxproject.org/page/Main_Page [Accessed July 27, 2018].

[11] R. K. Karne, K. Venkatasamy, N. Rosa, and T. Ahmed, "Dispersed Operating System Computing (DOSC)," Object-Oriented Programming, Systems, Languages and Applications, Onward Track (OOPSLA) 2005.

[12] L. He, R. Karne, and A. Wijesinha, "The Design and Performance of a Bare PC Web Server", International Journal of Computers and their Applications, vol. 15, June 2008, pp. 100 - 112.

[13] P. Appiah-Kubi, R. K. Karne, and A. L. Wijesinha, "A bare PC TLS Webmail Server", International Conference on Computing, Networking and Communications, (ICNC), 2012.

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

[15] G. H. Khaksari, R. K. Karne and A. L. Wijesinha. A Bare Machine Application Development Methodology, International Journal of Computers and Their Applications (IJCA), Vol. 19, No.1, March 2012, pp. 10-25.

[16] F. Almansour, R. K. Karne, A.L. Wijesinha, H. Alabsi and R. Almajed, Middleware for NICs in Bare PC Applications, 26th International Conference on Computer Communications and Networks (ICCCN), 2017.