Integrating an 802.11 Wireless Application in a Linux Kernel Module with a Bare PC Application

Rasha A. Almajed, William Agosto-Padilla, Ramesh K. Karne, Alexander L. Wijesinha
Department of Computer & Information Sciences, Towson University
Towson MD, 21252, USA
{ralmajed, rkarne, awijesinha}@towson.edu, agopad@gmail.com

Abstract

We describe the novel integration of an 802.11 wireless Linux kernel module application with a bare PC application that runs with no OS support. Integration is achieved by modifying the Linux 802.11 wireless driver. Integration enables the bare PC to control the Linux system at the kernel level, and to serve as a backend for offloading kernel operations. Furthermore, the bare PC can filter and/or process packets received by the Linux system. Integration with a bare PC has security advantages since it is not vulnerable to OS-related attacks, and performance advantages since it has no OS overhead. We first present the system architecture and its implementation. We then demonstrate how: 1) Linux kernel operations can be controlled using a bare PC; 2) Linux kernel functions can be offloaded to a bare PC; and 3) packets can be exchanged directly between the Linux kernel and the bare PC. The implemented prototype can serve as a basis for studying performance aspects of integrating Linux and bare systems in the future.

1 Introduction

OS-based systems are continually patched to eliminate newly discovered vulnerabilities [1]. Unfortunately, they make available a large attack surface due to offering a proliferation of services for the convenience of users. While hardening OS-based systems is a solution to some security issues, the presence of an OS or kernel, however small, provides capabilities that enable exploitation. A bare machine is an alternative approach to run applications without the support of an OS, kernel, or intermediary software. Bare systems have many security advantages including the following: 1) Bare applications are not vulnerable to attacks that require OS support; 2) It is not possible for an attacker to run scripts or escalate privileges on a bare machine; 3) Bare code is statically compiled (there are no dynamically linked libraries); 4) Bare systems do not use any local mass storage, thus reducing the ability of an attacker to compromise the system; 5) Bare machine code is also easier to analyze and secure due to its simplicity and smaller size, enabling vulnerabilities in a bare application to be more easily detected and fixed. In addition, running a bare PC application has performance benefits since it has no OS overhead. Many bare PC applications including mail servers [2], Web servers [3], peer-to-peer VoIP clients [4], SIP servers and user agents [5], and IPv6/v4 translators [6] have been previously built. Bare PC applications have also been used to measure protocol performance in the absence of OS overhead [7].

We implement an 802.11 wireless application as a Linux kernel module and modify the 802.11 Linux driver to integrate the application with a bare PC application. The integrated system is a prototype that could be enhanced to run 802.11 applications in the kernel to provide better security than that afforded by user space applications (which could be more easily compromised by an attacker). In essence, the prototype demonstrates how a bare PC application could directly control kernel operations on Linux wireless devices and also serve as a Linux backend for offloading security operations. It enables the bare PC to send control commands such as start/stop and kill to the Linux system; and to execute input/output commands such as readln or printk for the Linux system. The Linux system uses kernel sockets to directly exchange packets with the bare PC. A similar integrated system could be used in the future to offload functions from Linux (or Android) wireless devices to a bare PC.

The rest of this paper is organized as follows. In Sections 2 and 3, we provide an overview of bare machine computing and related work respectively. In Section 4, we describe the system architecture. In Section 5, we give design and implementation details. In Section 6, we demonstrate operations using the prototype. In Section 7, we present the conclusion.

2 Bare Machine Computing

Figure 1 illustrates the difference between conventional computing and bare machine computing. In conventional computing, devices run an OS or some form of a kernel, or are embedded systems. In addition, most use local mass

978-1-943436-10-1 / copyright ISCA, CATA 2018
March 19-21, 2018, Las Vegas, Nevada, USA

storage or a disk. The basic input/output system (BIOS) is commonly used to start up the machine and load an OS. The OS also uses a variety of vendor-specific device drivers. Conventional computer applications are platform dependent as they use system calls or an API to reach the hardware. Java applications are platform independent, but they require a platform dependent JVM. Conventional computing applications thus depend on their execution environment which is provided by an OS.

Bare applications are implemented as application objects that are self-contained, self-managed, and self-executable [8]. Applications include the necessary boot and load code, network protocols, and device drivers and are stored on removable mass storage (such as a USB) under individual user control. They are written in a single programming language such as C/C++ with very little assembly code, and independent of computing environments [9]. A single bare application or a bare application suite consisting of several bare applications is run on ordinary PCs with IA-32 (Intel) hardware. The bare applications can communicate with other bare or conventional applications using conventional access control and authentication mechanisms if needed. A bare machine application includes a main task that runs continually when no other tasks are running. It uses a receive task to receive packets from an Ethernet connection and application-specific tasks such as HTTP tasks in case of a bare Web server.

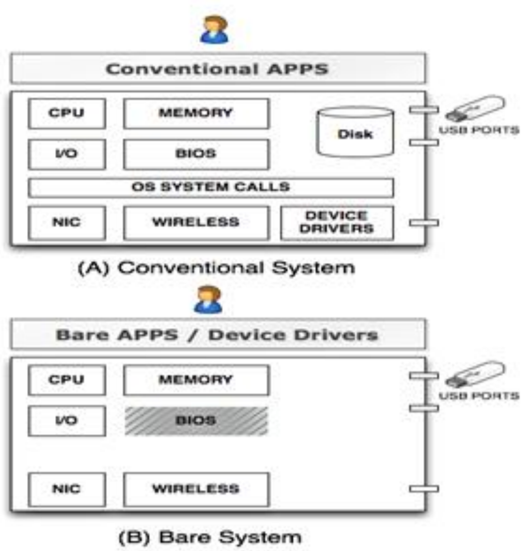

Figure 1. Differences between conventional and bare systems.

Each bare application contains its own (OS-independent) lean versions of protocols in the TCP/IP suite for communication with OS-based machines and other bare machines. Security protocols such as IPsec or TLS are included as needed. Bare machines can also be used to evaluate the impact of OS overhead on protocol performance [6].

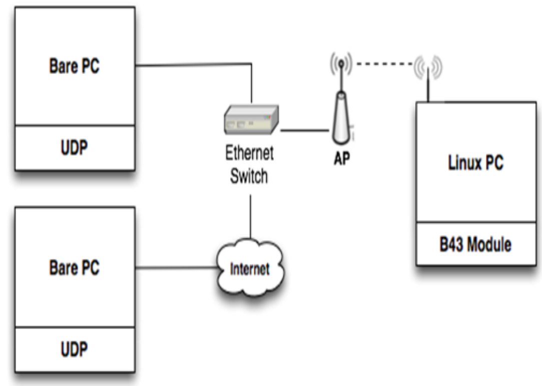

Figure 2. Integration architecture.

3 Related Work

Bare machine computing is similar to other approaches for reducing OS overhead and improving application performance such as Exokernel [10], OS-Kit [11], Sand-boxing [12] and Palacio/Kitten [13]. The main difference is that bare machine applications run without the support of any OS or kernel. A comprehensive treatment of Linux kernel networking is provided in [14]. The use of Netlink sockets to communicate between user space applications and the Linux kernel is discussed in [15]. The details of an in-kernel FTP client are given in [16]. A system for communicating between Linux kernels in cluster computers using Ethernet broadcast is implemented in [17]. To the best of our knowledge, there are no systems that offload Linux kernel functions to a system with no OS or kernel (for security purposes or otherwise).

4 System Architecture

The architecture shown in the upper part of Figure 2 is used for the prototype, which integrates the Linux system (Kernel 3.9.3/Ubuntu 12.10) with the bare PC. Both machines are Dell GX 960 PCs. The Linux system associates with an 802.11n access point connected to a gigabit Ethernet switch. The bare PC connects to the switch.

Other integration architectures are possible: the bare PC can be on the Internet (lower part of Figure 2); or the wireless (802.11) connection for the Linux system can be replaced by an Ethernet connection. While this prototype only uses the 802.11 Linux driver for integration, other modules in the Linux kernel could also be integrated with a bare system.

As shown in Figure 3, we added a new Linux kernel module called Udpthreads for communicating with a bare PC. It opens two UDP kernel sockets (send and receive) for use by the Linux kernel space application. The Udpthreads module is inserted between the mac80211 (802.11 wireless MAC) and b43 (Broadcom wireless driver) modules in the

Linux kernel. We modified the code in [18] enabling Udpthreads to send/receive commands to/from a bare PC, and replaced a callback routine (that did not work) with work queues [19].

While UDP is used in the prototype to communicate between the systems for simplicity and efficiency, TCP connections could also be used. IP-level communication between the Linux system and the bare PC can be secured with IPsec. We do not discuss securing Linux-bare communication in this version of the prototype.

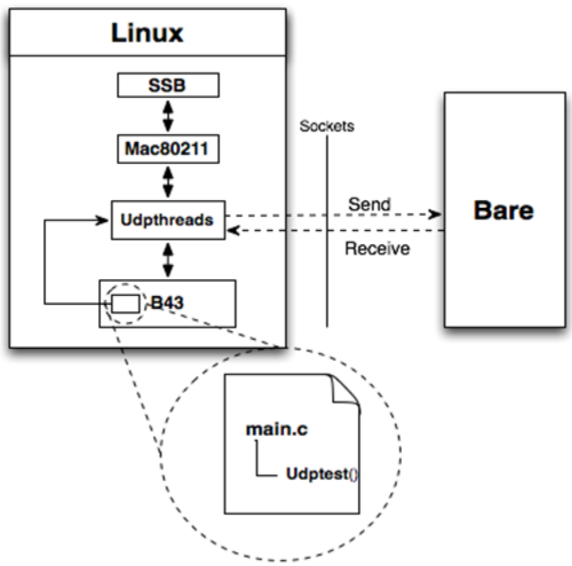

Figure 3. Bare-Linux communication.

5 Design and Implementation

The integrated system is implemented using the Udpthreads and b43 wireless driver modules in the Linux kernel (written in C), and the UDP application in the bare PC (written in C++). The design flow for the Udpthreads module is shown in Figure 4. Since the Udpthreads module is inserted before the modified b43 module, the Send socket is created first and used for sending dummy packets to the bare PC until the modified b43 module starts running. In the b43 module, main.c has calls to the Udptest() function in Udpthreads. This function initiates actions in the Udpthreads module based on commands (Bare OPs) to trigger the exchange of packets with the bare PC. The packets, which are exchanged via Send/Receive, once the b43 module is running, enable the receiving system to perform the corresponding operation.

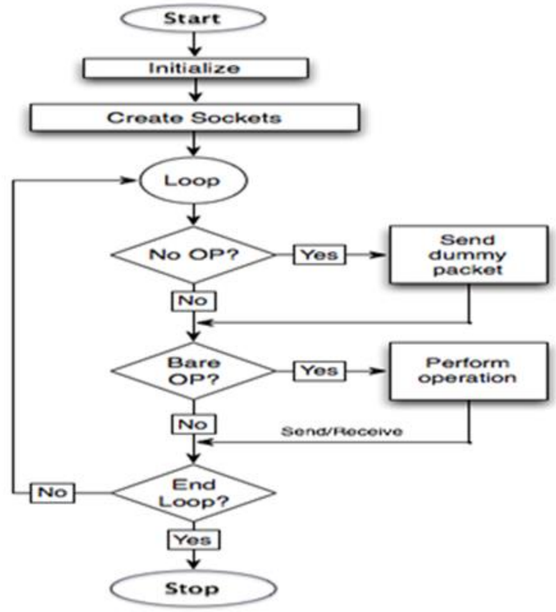

Figure 4. UDPthreads module flow.

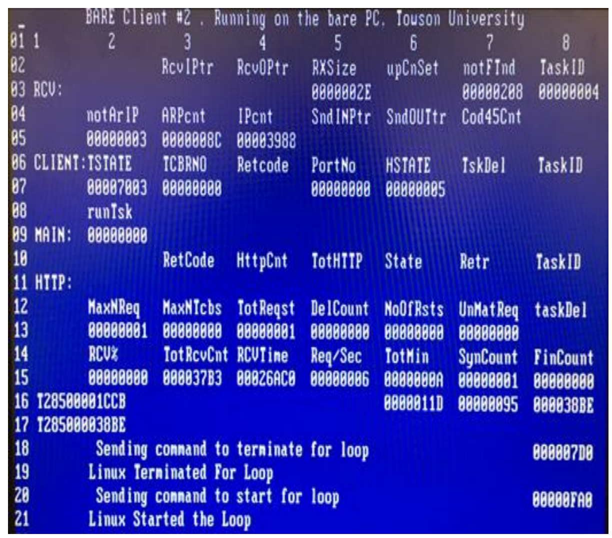

Figure 5. Start-Stop command in the bare PC.

Due to the delay between inserting the Udpthreads and b43 modules, the bind on the receive socket has be done only after the system is ready to receive packets. Also, we found that it is necessary to do a bind each time to receive a packet to ensure that receive does not block. For reasons of space, we omit details of the methods in Udpthreads that are needed to enable our Linux kernel space socket application to work. The bare PC UDP application is designed so that every time a packet is received from the Linux system, it is processed and a response is sent.

```
Sep 25 16:30:56 rkkw-OptiPlex-960 kernel: [415.656288]
  RX-130: status flag: 0 <6>[  415.656292] RKKW, dma.c,
    update_max_used_slots, 0145
Sep 25 16:30:56 rkkw-OptiPlex-960 kernel: [415.656294]
  RKKAW IN SEND ANSWER
Sep 25 16:30:56 rkkw-OptiPlex-960 kernel: [415.656294]
  RKKAW IN SEND ANSWER WHILE LOOP
Sep 25 16:30:56 rkkw-OptiPlex-960 kernel: [415.656296]
  RKKAW: DATARCVD-SKBUFF: 8 message: EndLoop
Sep 25 16:30:56 rkkw-OptiPlex-960 kernel: [415.656297]
  RKKAW: BARECMDRCVD: message: EndLoop
Sep 25 16:30:56 rkkw-OptiPlex-960 kernel: [415.656297]
  RKKAW: DATARCVD-UDPBAREBUFF: message: EndLoop
Sep 25 16:30:57 rkkw-OptiPlex-960 kernel: [416.340017]
  RECEIVED COMMAND TO TERMINATE FOR LOOP: LCOUNTER: 4196
Sep 25 16:30:57 rkkw-OptiPlex-960 kernel: [416.340019]
  INFIRSTLOOP: 0
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.500641]
  RKKAW IN SEND ANSWER
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.500642]
  RKKAW IN SEND ANSWER WHILE LOOP
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.500644]
  RKKAW: DATARCVD-SKBUFF: 10 message: StartLoop
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.500645]
  RKKAW: BARECMDRCVD: message: StartLoop
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.500647]
  RKKAW: DATARCVD-UDPBAREBUFF: message: StartLoop
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.964745]
  YY100: DATATOSEND: T2850000085C size: 12
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.972010]
  RECEIVED COMMAND TO START FOR LOOP: LCOUNTER: 8148
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.972012]
  LOOP Start AA100
Sep 25 16:31:13 rkkw-OptiPlex-960 kernel: [432.972014]
  RKKWT7-0 In ksocket_send() udpthreads.: 4278
```

Figure 6. Start-Stop command in Linux.

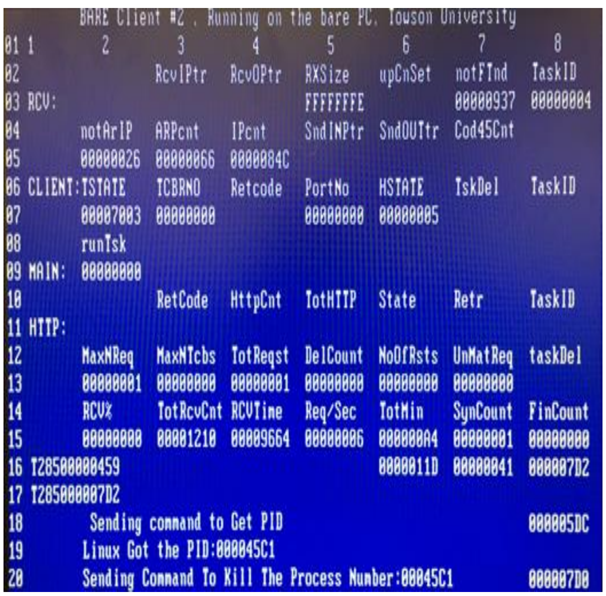

Figure 7. Kill command in the bare PC.

6 Prototype Operations

The following operations are presented only to illustrate the feasibility of enhancing 802.11 security on Linux by leveraging bare PC security advantages. In case of an 802.11 Linux client communicating with an Internet TLS Web server for example, TLS functions could be offloaded to the bare PC for additional security (larger keys, memory-based protection of TLS parameters and state, lesser chance of compromising the wireless link etc.).

```
Oct 20 12:27:18 rkkw-OptiPlex-960 kernel: [3633.148314]
  RKKAW: DATARCVD-SKBUFF: 7 message: GetPid
Oct 20 12:27:18 rkkw-OptiPlex-960 kernel: [3633.148314]
  RKKAW: BARECMDRCVD: message: GetPid
Oct 20 12:27:18 rkkw-OptiPlex-960 kernel: [3633.148315]
  RKKAW: BARECMDRCVD to Get The Pid: message: GetPid
Oct 20 12:27:18 rkkw-OptiPlex-960 kernel: [3633.148316]
  RKKAW: DATARCVD-UDPBAREBUFF: message: GetPid
Oct 20 12:27:18 rkkw-OptiPlex-960 kernel: [3633.149130]
  the current pid: 17857
Oct 20 12:27:18 rkkw-OptiPlex-960 kernel: [3633.149131]
  this is the current pid that we have in decimal: 17857,
    pidnum is the pid in hex: 000045C1
Oct 20 12:27:22 rkkw-OptiPlex-960 kernel: [3636.880626]
  RKKAW: DATARCVD-SKBUFF: 9 message: KillProc
Oct 20 12:27:22 rkkw-OptiPlex-960 kernel: [3636.880627]
  RKKAW: BARECMDRCVD: message: KillProc
Oct 20 12:27:22 rkkw-OptiPlex-960 kernel: [3636.880628]
  RKKAW: BARECMDRCVD To Kill The Process: message: KillProc
Oct 20 12:27:22 rkkw-OptiPlex-960 kernel: [3636.880629]
  RKKAW: DATARCVD-UDPBAREBUFF: message: KillProc
Oct 20 12:27:22 rkkw-OptiPlex-960 kernel: [3636.880732]
  kill the pid:17857
```

Figure 8. Kill command in Linux.

Figure 9. Clear command in the bare PC.

```
Nov 12 23:22:56 rkkw-OptiPlex-960 kernel: [357.880147]
  Linux send command to Bare to Clear the Screen
Nov 12 23:22:56 rkkw-OptiPlex-960 kernel: [357.880162]
  XX100
Nov 12 23:22:56 rkkw-OptiPlex-960 kernel: [357.880163]
  BB100
Nov 12 23:22:56 rkkw-OptiPlex-960 kernel: [357.880179]
  RKKAW: DATARCVD-SKBUFF: 80 message: Bare Received A
    Command From Linux To Clear The Screen
```

Figure 10. Clear command in Linux.

```
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.576750]
  The command flag will change from 2 to 3: 2
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.576771]
  The command flag will change from 2 to 3: 2
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.578004]
  the current pid: 16515
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.578056]
  the current Pid: 16515
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.578058]
  this is the pid in decimal: 16515
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.578079]
  this is the pid in decimal: 16515
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.578081]
  this is the pid in hex: 00004083
Oct 28 16:07:08 rkkw-OptiPlex-960 kernel: [487.578101]
  this is the pid in hex: 00004083
Oct 28 16:07:12 rkkw-OptiPlex-960 kernel: [491.524779]
  The command will change from 3 to 4: 3
Oct 28 16:07:12 rkkw-OptiPlex-960 kernel: [491.524795]
  The command will change from 3 to 4: 3
```

Figure 11. Printk command in Linux.

6.1 Start – Stop Commands

UDPthreads has a never-ending for loop, which sends and receives packets as long as the thread is running. The stop command sent from the bare PC is used to stall the for-loop and force it wait until the start command arrives (for example, till keys are generated on the bare PC). Figure 5 shows the bare PC screen, and Figure 6 shows the corresponding Linux trace.

6.2 Kill Command

This command shows how the bare PC application can kill a thread in the Linux kernel. First, the bare PC sends a command to the Linux system and requests a PID for the target process. When the bare PC receives the PID, it sends a Kill command with the PID to terminate the process. Figure 7 shows the bare PC screen and Figure 8 shows the corresponding Linux log file for the Kill command. This could be used after a TLS Close Notify from a remote system to Linux for example.

6.3 Clear Command

The Clear command sent by Linux clears the bare screen. Figure 9 shows the bare PC screen after the clear command, and Figure 10 shows the Linux log file trace. This command was found to be convenient during system testing.

6.4 Printk and Readln Commands

The Printk command allows Linux to offload the printing of its log files to the bare PC system. Figure 11 shows a trace of the Linux printk data, which is to be sent to the bare PC for printing. Its counterpart Figure 12 shows this data printed on the bare PC screen. It is also possible to encrypt and save this log file in a bare PC file system (on a removable mass storage device). The Readln command sent from Linux to bare allows a line to be input at the bare PC and sent to Linux for processing. Figure 13 shows the data entered in the bare PC, and Figure 14 shows the Linux log file trace.

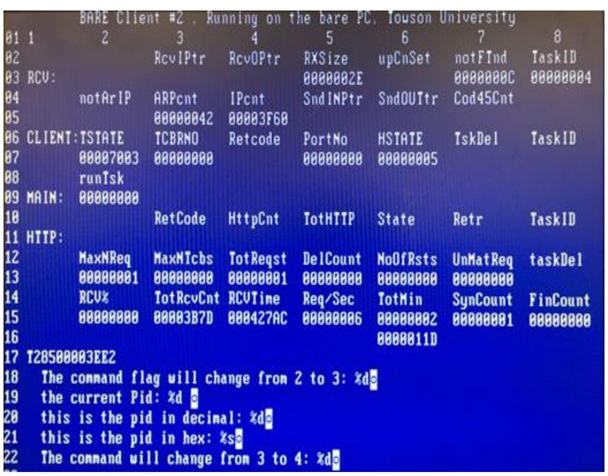

Figure 12. Printk command in the bare PC.

6.5 Packet Transfer

A bare PC can receive packets (via Ethernet) from the Linux kernel (via 802.11) and return them after processing. Figure 15 shows a 105-byte UDP packet (34- byte header and 71 bytes data) in the bare PC's memory prior to sending it directly to the Linux kernel. For this packet, Figure 16 shows the Linux log file trace and Figure 17 shows the packet details in Wireshark. In an actual system, packets from the Linux kernel on a wireless device can be filtered and/or processed by the bare PC.

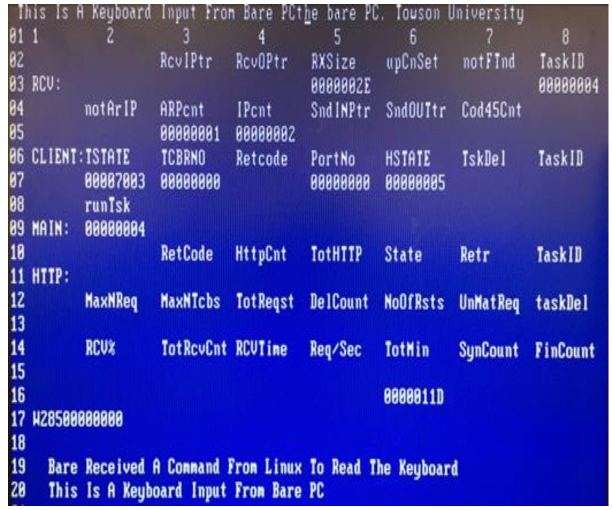

Figure 13. Readln command in the bare PC.

```
Nov 5 12:10:05 rkkw-OptiPlex-960 kernel: [737.513386]
  Linux sent command to Bare to Read Line From Bare
    Kayboard
Nov 5 12:10:05 rkkw-OptiPlex-960 kernel: [737.661153]
  RKKAW: DATARCVD-SKBUFF: 80 message: This Is A Keyboard
    Input From Bare PC
Nov 5 12:10:05 rkkw-OptiPlex-960 kernel: [737.661157]
  RKKAW: DATARCVD-UDPBAREBUFF: message: This Is A Keyboard
    Input From Bare PC
Nov 5 12:10:05 rkkw-OptiPlex-960 kernel: [737.661642]
  ksocket: RKKAW-UDPTEST-RECEIVE CMD RCVD  updcount: 12891
    DATAFORB43: This Is A Keyboard Input From Bare PC
```

Figure 14. Readln command in Linux.

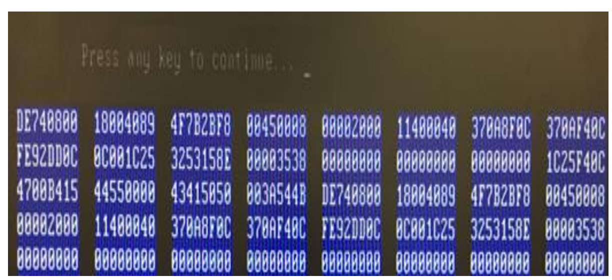

Figure 15. Ethernet packet in the bare PC.

```
Nov 17 13:31:36 rkkw-OptiPlex-960 kernel: [397.372519]
  RKKAW: DATARCVD-SKBUFF: 63 message: UDPPACKT:
Nov 17 13:31:36 rkkw-OptiPlex-960 kernel: [397.372522]
  RKKAW: Ethernet Packet From Bare
Nov 17 13:31:36 rkkw-OptiPlex-960 kernel: [397.372522]
  5544505041434b543a00000874de89400018f82b7b4f08004500
    0020000040004011 0c8f0a370cf40a370cdd92fe251c000c8e
      15533238350000000000000006a
```

Figure 16. Ethernet packet in Linux.

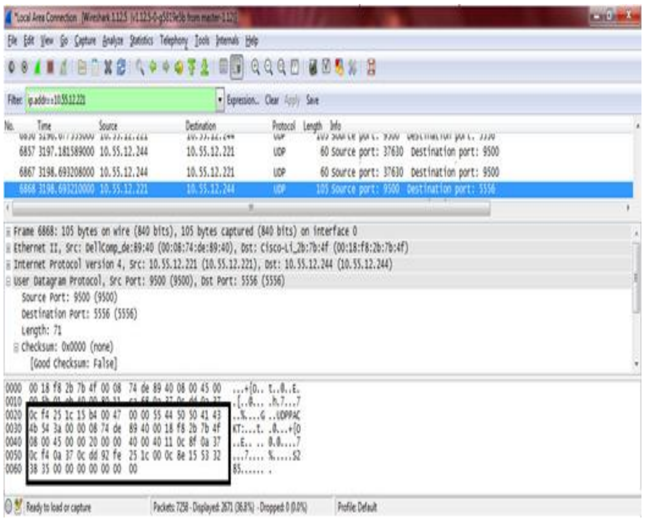

Figure 17. Ethernet packet in Wireshark.

7 Conclusion

We presented a novel system for wireless security that integrates a Linux kernel-space application on an 802.11 wireless device with a bare machine application. When kernel security-related operations are offloaded from the wireless device to the bare machine, security of the integrated system is improved since the bare machine does not have an OS or kernel. The implemented prototype can be enhanced to enable more Linux kernel operations to be executed on the bare PC. It can also be used for performance studies of integrated Linux-bare systems.

References

[1] Packet Storm, https://packetstormsecurity.com, accessed October 2017.

[2] G. H. Ford, R. K. Karne, A. L. Wijesinha, and P. Appiah-Kubi, "The design and implementation of a bare PC email server", 33rd IEEE International Computer Software and Applications Conference (COMPSAC), 2009, pp. 480-485.

[3] B. Rawal, R. K. Karne, and A. L. Wijesinha., "Splitting HTTP requests on two servers", 3rd Conference on Communication Systems and Networks (COMSNETS), 2011, pp. 94-100.

[4] G. Khaksari, A. L. Wijesinha, R. K. Karne, L. He, and S. Girumala, "A peer-to-peer bare PC VoIP application", IEEE Consumer Communications and Networking Conference (CCNC) 2007.

[5] R. Yasinovskyy, A. Alexander, A. L. Wijesinha, and R. K. Karne, "Bare PC SIP user agent implementation and performance for secure VoIP", International Journal on Advances in Telecommunications, vol 5 no 3 & 4, 2012, pp. 111-119.

[6] A. Tsetse, A. Wijesinha, R. Karne, A. Loukili and P. Appiah-Kubi, "An experimental evaluation of IP4-IPv6 IVI translation", ACM SIGAPP Applied Computing Review, March 2013, Vol. 13, No. 1, pages 19-27.

[7] A. Loukili, A. L. Wijesinha, R. K. Karne, and A. K. Tsetse, TCP's retransmission timer and the minimum RTO", 21st International Conference on Computer Communications and Networks (ICCCN), 2013.

[8] R. K. Karne, K. V. Jaganathan, N. Rosa, and T. Ahmed. DOSC: Dispersed operating system computing. 20th ACM Object-oriented Programming, Systems, Languages, and Applications Conference (OOPSLA), 2005, pp. 55-62.

[9] R.K. Karne, K. V. Jaganathan, and T. Ahmed, "How to run C++ applications on a bare PC," 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Net-working, and Parallel / Distributed Computing (SNPD), 2005, pp. 50-55.

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

[11] "The OS Kit Project," School of Computing, University of Utah, Salt Lake, UT, June 2002, http://www.cs.utah.edu/flux/oskit, accessed Oct. 2017.

[12] B. Ford, and R. Cox, Vx32: "Lightweight user-level sandboxing on the x86", USENIX Annual Technical Conference, June 2008.

[13] 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, 2010.

[14] R. Rosen, Linux Kernel Networking: Implementation and Theory. Apress, 2013, vol. 1.

[15] P. Neira-Ayuso, R. M. Gasca, and L. Lefevre, "Communicating between the kernel and user-space in Linux using Netlink sockets", Software-Practice & Experience, vol. 40, issue 9, pp. 797-810, Aug. 2010.

[16] P. Padala and R. Parimi, "Network programming in the kernel", Linux Journal, issue 138, pp. 22-32, Oct. 2005.

[17] P. Werstein, M. Pethick, and Z. Huang, "Locabus: A kernel to kernel communication channel for cluster computing", 5th Conference on Parallel and Distributed Computing: Applications and Technologies (PDCAT), pp. 497-504, 2004.

[18] Linux Kernel Newbies, "Simple UDP server", http://Kernelnewbies.org, accessed: Oct. 2017.

[19] GitHub, "iptables_dev_examples", http://github.com/joninvski, accessed: Oct. 2017.