Isolating System Faults on Vehicular Network Gateways Using Virtualization

Sung-Moon Chung    Hyun-Wook Jin
Department of Computer Science and Engineering
Konkuk University
Seoul, Korea
{antistar,jinh}@konkuk.ac.kr

Abstract — The traditional vehicular network gateway takes charge of communication between different internal networks and helping the electric control units in vehicle to collaborate each other. Due to the increasing requirements on innovative applications such as infotainment systems and cyber-physical systems, there are significant efforts to have an external wireless network connection on the vehicles. Accordingly, the secure architecture of the network gateway that can avoid or isolate the malicious behavior of external nodes is very critical for the next-generation vehicles. In this paper, we design a safe vehicular network gateway by exploiting full virtualization technology. Since the virtualization adds additional overheads, we try to minimize this side effect while considering the security by carefully choosing the communication mechanisms in the virtualized gateway. In our preliminary implementation, we use VirtualBox to run Linux and QNX as guest operating systems, which handles external (Wi-Fi) and internal (CAN) networks, respectively. The performance measurement results show that the virtualization-based gateway adds only 10% overhead compared with non-virtualized gateway while improving the security. We also show that the multi-core processor can leverage performance improvement.

Keywords - network gateway; virtualization; fault isolation; security; vehicle

I. INTRODUCTION

The ground and aerial vehicles comprise many Electric Control Units (ECUs) which are connected through vehicular internal networks. In general, several sub-networks having different charactersitics or requirements construct the internal network and collaborate each other through a network gateway. For instance, in an automobile, CAN [1], LIN [2], FlexRay [3], and MOST [4, 5] networks can be utilized for powertrain, body, safety, and infotainment systems, respectively. The network gateway takes charge of the protocol translation between different networks. Since the vehicular internal networks do not provide a common network layer such as IP, the routing is a tricky issue. This can be resolved by using a static routing table [6], which is acceptable in the vehicular systems because the network is not dynamically changed at the run-time unlike Internet and the number of nodes is already known at the startup time.

Due to the recent requirements on innovative applications such as infotainment systems and cyber-physical systems, there are significant efforts to have an external wireless network connection (e.g., Wi-Fi or WiMAX) on the next-generation vehicles. Figure 1 shows an example of unmanned aerial vehicles that have an external network connection. In this example, the internal network A is the real-time network that connects ECUs and delivers the sensing and avionic control data between ECUs. The ground control system sends control messages to and receives the status information from the ECUs. The camera node streams the video data to the ground control system through the internal network B of high-bandwidth and wireless network adjusting its angle based on the sensing data (e.g., GPS and heading information) forwarded from the network A. Here the network gateway performs the key role of enabling the message exchange between internal networks but also between internal and external networks.

Figure 1. Network gateway on an unmanned aerial vehicle

The connection to the external network, however, can expose the possibility of facing either malicious or unexpected behavior of outside world that causes the system fault. Since the security of vehicles related directly with critical domains such as human life and national defense, we should deal with the system faults as one of most critical issues in vehicles. Especially the system faults on the network gateway are propagated to several system areas and fatal to overall operation of the vehicle. It is because many of vehicle applications are based on the collaboration between nodes on different networks as discussed in Figure 1. Thus the secure architecture of the network gateway that can avoid or isolate the system faults is very critical.

* This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency) (NIPA-2010-C1090-1031-0003).
In this paper, we design a secure vehicular network gateway by exploiting virtualization technology. We utilize the features of virtualization that can isolate system faults and prevent them propagating to other domains. The security can be enhanced by various schemes such as authentication, encryption, intrusion detection, filtering, etc. but in this paper we aim to suggest a software stack to prevent the system faults propagating to whole system, on which other schemes mentioned above can be applied. Since the virtualization adds additional overheads, we also try to minimize this side effect and analyze its impact on performance. Though there have been studies on the vehicular network gateways [6, 7, 8, 9, 10, 17], the actual implementation considering security has not been presented in detail. Hergenhan and Heiser [7] and Navet et. al. [17] suggest utilizing the virtualization technology for safety but they present only very high-level pictures. Sangorrín et. al. [18] also shows the benefits of virtualization with respect to the safety; however, they assume that the real-time domain can be always trusted. Our preliminary implementation shows the potential of virtualization-based vehicular network gateway with reasonable additional overhead and improved security.

The rest of the paper is organized as follows: Section II discusses the design alternatives of non-virtualized vehicular network gateways and their limitations with respect to the security. Section III details the suggested design of virtualization-based vehicular network gateway. The performance measurement results of the suggested design are presented in Section IV. Finally we conclude this paper in Section V.

II. VEHICULAR NETWORK GATEWAY WITHOUT VIRTUALIZATION

In this section, we describe possible designs for vehicular network gateway without virtualization. We classify the designs into two based on where the protocol translation is performed: i) kernel-level gateway and ii) user-level gateway. Since the vehicular internal networks, such as CAN, LIN, FlexRay, and MOST, do not target Internet, they do not support a common network layer such as IP. Thus the routing schemes based on the IP layer are not applicable in our target systems. Instead, an additional protocol translation layer, which exists on top of the existing protocol stacks, is required. The protocol translation layer decides the destination using a static routing table. The packet should contain additional information to find a matched entry in the routing table. The format and meaning of the information embedded in the packet is pre-defined by applications.

In the kernel-level gateway, the protocol translation between external and internal networks is performed in the Operating System's (OS's) kernel as shown in Figure 2. This design can minimize the number of data copy during the routing operations and prevent significant performance degradation. The protocol translation layer, however, has to be implemented as a kernel thread to establish external network connections. Otherwise it is not clear who takes care of the TCP connections because the internal networks do not understand TCP/IP. Thus in this design if the kernel thread misbehaves the gateway can face fatal system faults, which results in halt of internetworking between internal networks as well. In addition this design requires the kernel to implement all protocol stacks for internal networks but many of protocol stacks for vehicular internal network have been implemented at the user-level [19]. Therefore, it would take significant time to move them into the kernel and stabilize them with respect to the security.

The design of user-level gateway is shown in Figure 3. As we can see, the protocol translation layer exists in the user space. In this design, the protocol stacks for internal networks can be either user or kernel-level. Therefore, we can reuse existing implementations of protocol stacks without any modifications. The performance, however, can be worse than the kernel-level gateway because the network data should be copied between user and kernel spaces. Compared with the kernel-level gateway, this design is also lax in terms of security.

We can consider both Real-Time Operating System (RTOS) and general-purpose OS as OS of the network gateway. An RTOS that is already used for a vehicular network gateway but without support for external network might not be verified its safety and security sufficiently for having external network. On the other hand, if we use a general-purpose OS that has been widely used for network servers, this can guarantee the security on the external network together with various existing security tools but can require new implementations for some of protocol stacks of internal networks. This new implementations can induce system faults until they are verified through time-consuming processes.
whole gateway system. Thus it is highly required to isolate the system fault when it occurs. For example, though the external network becomes disconnected because of a network attack, the data exchange between internal networks still need to work so that the control systems can be operated by human or software driver/pilot without experiencing any problems.

III. DESIGN AND IMPLEMENTATION OF VIRTUALIZATION-BASED SECURE NETWORK GATEWAY

As we have described in Section II, the network gateway without virtualization is not sufficient to provide secure software architecture due to the possibility of fault propagation over the whole gateway system. To tackle this issue, we suggest a virtualization-based vehicular network gateway, which is able to isolate the system fault on the external connection from the rest of the networks. The proposed design can provide a secure software platform that can improve the safety and security together with various existing security tools and schemes.

A. Virtualization-based Design

Our basic idea is to have two OSes on the network gateway: one for external network and the other for internal networks. We use the virtualization technology to run two different guest OS domains on the network gateway. The legacy network gateways that support only internal networks run RTOSes that are verified thoroughly in terms of correctness and safety. Thus it is desirable to use them for internal networks without any modifications. In our design, we exploit the full virtualization such as VirtualBox [11], and VMware Workstation [12] that can provide OS virtualization without any modification of guest OSes though it can have worse performance than paravirtualization such as Xen [13]. In order to deal with external network and arbitrary behavior of outside world, we consider using a general-purpose OS that has been applied to Internet servers. In this way we run general-purpose and real-time guest OSes for external and internal networks, respectively. Then the Virtual Machine Monitor (VMM) isolates a system fault by a guest OS from the other.

We, however, still need to exchange data between guest OSes to perform data forwarding to internal and external networks. The level of security can vary based on where the network data goes through in the gateway node. We classify the possible data paths into two: i) kernel-level and ii) user-level data forwarding. As shown in Figure 4, the data can be forwarded by the guest OS that takes charge of external network at the kernel level while the protocol translation is done on the guest RTOS. Thus the general-purpose guest OS works as an Internet router. This design, however, exposes the user process for protocol translation to the outside world because the process has to establish the TCP connection to an external node, which results in high chance to attack internal networks.

In the user-level forwarding, an additional layer called External Networking performs data forwarding to the guest RTOS and external network as shown in Figure 5. Since the external networking layer running on the external network domain and handles all TCP connections to outside world, the domain for internal networks is hidden safely. Though this design induces more network data copies, we choose this because of better security than the kernel-level forwarding. The proposed design can guarantee the security on the internal network domain by hiding it from the external nodes and preventing the faults occurred on the external network domain propagating to the whole system of network gateway. Therefore, even if such faults disconnect the external network, the internal networks can still work together.

![Figure 4. Kernel-level data forwarding in the virtualization-based network gateway](image)

![Figure 5. User-level data forwarding in the virtualization-based network gateway](image)

B. Protocol Translation

In this section, we detail the internals of the protocol translation layer. This layer has two routing tables called Route-In Table and Route-Out Table as shown in Figure 6. We target the CAN network as internal network for example. A CAN message includes the identifier, which specifies its attribute, in the message header. The meaning of each identifier is defined at the software design phase and highly dependent on application. Instead of using source and destination addresses, the CAN network uses this identifier information to decide the receivers. Once the nodes specify interesting identifiers, they get the messages having the same identifiers in a multicast manner. MsgID in the routing tables represents the CAN message identifier. In addition, there are network-independent operation codes called OpType. This is also defined at the software design phase and used by both external and internal nodes of vehicles. By mapping MsgID and OpType through the routing tables, we can control or monitor ECUs in the internal network without knowing the details of the internal network from outside world. Both tables are static and initialized at the vehicle startup time. As we have mentioned earlier, this static
Routing table is acceptable because vehicular internal network is not dynamic. It is thanks to the nature of distributed embedded systems in vehicles. Since the protocol translation introduced is independent on the legacy routing tables and schemes implemented already for traditional vehicular network gateway, adding the protocol translation layer does not affect on existing implementations.

The external networking layer also has a routing table called Route-Out Table, which is shown in Figure 6. This table maps an OpType with a TCP socket descriptor that specifies a TCP connection to external node. Using this table, the external networking layer decides the destination node of the message forwarded from the internal network (i.e., CAN). Since the value of socket descriptor is varied whenever the connection is established, this table is initialized at the run time. On the other hand, in this layer, we do not need a routing table for messages from the external network because the message embeds the OpType information.

![Figure 6. Routing tables and protocol translation](image)

C. Communication between Domains

In order to minimize the overhead of virtualization we carefully choose the communication mechanisms between different OS domains. At the same time, we also consider the security of each communication mechanism. In our preliminary implementation of vehicular network gateway, we use VirtualBox for VMM. We use Linux and QNX for guest OSes for external network (Wi-Fi) and internal network (CAN), respectively, which is shown in Figure 7. Though the protocol translation layer accesses the internal network device transparently in our original design depicted in Figure 5, the guest QNX on the current implementation of VirtualBox is not able to recognize the CAN device. Thus, we have added the internal network protocol to the host Linux as a user process and let it bridge between protocol translation layer and the internal network device. Consequently, there are four network connections as shown in Figure 7: i) between external network and external networking layer, ii) between external networking and protocol translation layers, iii) between protocol translation and internal network protocol layers, and iv) between internal network protocol layer and internal network. Since the last case is straightforward we do not detail it in this paper.

The communication between external network and external networking layer can be done by either Network Address Translation (NAT) or bridged networking. The NAT allows the guest domain to only connect to external network while external node cannot connect to the guest domain. Therefore, the guest domain cannot run server software. On the other hand, the guest domain can accept the connection from outside with the bridged networking. This uses the software bridge and constructs a virtual network assigning IP addresses to the guest domains, which are visible from outside. The vehicles, however, barely runs server software and are better to block connections from outside. By this reason, we use NAT for external network connection.

![Figure 7. Communication mechanisms chose for better security and performance](image)

For the communication between two guest domains, we can use bridged networking, internal networking, or host-only networking. Since the bridged networking provides communication to external network, the network messages always go through the protocol stacks of host OS, which induces additional overhead. On the other hand, internal networking can bypass the protocol stacks of the host OS but only supports inter-domain communication. The IP addresses assigned for internal networking are used only for private LAN; therefore, the addresses are not visible from external network. The host-only networking is similar with internal networking but it allows communication between guest and host domains as well by means of virtual loopback device. Thus it has higher overhead than internal networking. Since the internal networking has least communication overhead but also can hide the guest RTOS from outside, we use the internal networking for communication between two guest domains. If
VirtualBox provides higher performance and more secure inter-domain communication mechanisms [14, 15, 16], we can also benefit from this.

We have two alternatives (i.e., bridged networking, and host-only networking) for communication between protocol translation and internal network protocol layers. As we have mentioned earlier, bridged networking is not suitable for inter-domain communication because it exposes the domain to the external network. Thus we decide to use host-only networking for this case. We do not consider internal networking here because it cannot provide communication with the host OS.

IV. PERFORMANCE MEASUREMENT

We have implemented the design suggested in Section III and measured its performance. We setup an experimental system that consists of a network gateway, a CAN node, and an Internet node. The network topology is similar with that shown in Figure 1 in which the wireless network and internal networks are Wi-Fi and CAN, respectively. Table I shows the software and hardware configurations of each node.

<table>
<thead>
<tr>
<th>NODES</th>
<th>SOFTWARE</th>
<th>HARDWARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Gateway</td>
<td>Fedora 8 (kernel version 2.6.23.1)</td>
<td>Intel Core 2 (2.13GHz)</td>
</tr>
<tr>
<td></td>
<td>VirtualBox 3.0.12</td>
<td>PEAK-CAN PCI</td>
</tr>
<tr>
<td>CAN Node</td>
<td>SUSE 10.3 (kernel version 2.6.22.19)</td>
<td>Intel Pentium M (1.4GHz)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PEAK-CAN PCI</td>
</tr>
<tr>
<td>Internet Node</td>
<td>SUSE 10.3 (kernel version 2.6.22.19)</td>
<td>Gigabit Ethernet (BCM5721)</td>
</tr>
</tbody>
</table>

**TABLE I**

**EXPERIMENTAL SYSTEMS**

Figure 8. Round-trip time varying the number of processor cores

Figure 8 shows the inter-domain communication throughput. As we have mentioned in Section III.C, the internal networking shows the best performance for the inter-guest domain communication while the host-only networking does for communication between host and guest domains.

Figure 9. Round-trip time between CAN and Internet nodes via the network gateway

Figure 9 shows the measurement results with 8-byte message. In this figure we compare the communication latency of network gateways with and without virtualization. As we can observe in the figure, the gateway with virtualization shows higher round-trip time but the additional overhead is only about 10%. It is to be noted that the experimental system has been setup in the same room, where the overhead on the external wired network is hardly added. If we measure the performance in a general scenario with a long distance wired network area, the overhead of virtualization will occupy even less portion of the overall communication overhead.

Figure 10. Round-trip time varying the number of processor cores

We have measured the communication latency using a test program running on CAN and Internet nodes, which sends and receives the same size control messages in a ping-pong manner repeatedly for a given number of iterations. We report the average time taken for each loop, which is round-trip communication latency. As we can observe in the figure, the gateway with virtualization shows higher round-trip time but the additional overhead is only about 10%. It is to be noted that the experimental system has been setup in the same room, where the overhead on the external wired network is hardly added. If we measure the performance in a general scenario with a long distance wired network area, the overhead of virtualization will occupy even less portion of the overall communication overhead.

We also measure the round-trip time varying the number of processor cores on the network gateway, where we use Intel Core i7 (2.80GHz). The measurement results are shown in Figure 10. As we can see, as the number of cores increases the round-trip time decreases. As a result, the round-trip time has been reduced up to 10%. This is because if every domain can occupy own processor core, the parallelization between the domains maximizes and thus the communication progress can happen in timely manner.
As a scenario-based case study, we generate a fatal system fault artificially on the guest Linux domain to observe the system behavior when the system faces a fault as shown in Figure 11. This situation can happen when a malicious attack from the external network stops the guest domain. To emulate such situation, we insert a kernel module that induces the kernel panic into the guest Linux. We again try to measure the round-trip time between the external node and internal node. Obviously the measurement fails but we note that the protocol translation layer simply consumes the network messages for out-side world when the connection to the external networking layer is disconnected. Thus, the system fault on the external networking domain does not cause misbehavior on the other domain. Consequently we can observe that the communication between internal nodes still sustain without performance degradation. If both external and internal networking is managed by a single domain without virtualization, the collaboration between nodes on different internal networks cannot take place at all by such system fault.

V. Conclusions and Future Work

In this paper, we have presented a virtualization-based vehicular network gateway in order to enhance its safety. More specifically, we have exploited full virtualization technology to avoid or isolate the malicious behavior of external nodes supporting existing OSes without modifications. In our preliminary implementation, we use VirtualBox to run Linux and QNX as guest operating systems, which handles external (Wi-Fi) and internal (CAN) networks, respectively. Since the virtualization adds additional overheads, we try to minimize this side effect while considering the security by carefully choosing the communication mechanisms in the virtualized gateway. The performance measurement results have shown that the virtualization-based gateway adds only 10% overhead compared with the non-virtualized gateway while improving the security. We have also showed that the multi-core processor can help to improve the performance and carried out a scenario-based case study to prove the fault isolation.

As future work, we plan to enhance the performance of the network gateway by allowing the guest domain (i.e., QNX domain) to access the internal network device transparently. In addition, we intend to study inter-domain scheduling to guarantee the real-time requirements on the RTOS domain.

REFERENCES