A Security Model for Dependable Vehicle Middleware and Mobile Applications Connection

Shengzhi Zhang[1], Omar Makke[2], Oleg Gusikhin[2], Ayush Shah[2] and Athanasios Vasilakos[3]
[1]*Florida Institute of Technology, Melbourne, Florida, U.S.A.*
[2]*Ford Motor, Detroit, Michigan, U.S.A.*
[3]*LTU, Luleå, Sweden*

Keywords: Cyber Security, Infotainment, Mobile Application, IoT.

Abstract: Nowadays automotive industry has been working on the connectivity between automobile and smartphones, e.g., Ford's SmartDeviceLink, MirrorLink, etc. However, as the interoperability between the smartphone and automotive system increase, the security concern of the increased attack surface bothers the automotive industry as well as the security community. In this paper, we thoroughly study the attack vectors against the novel connection framework between automobile and smartphones, and propose a generic security model to implement a dependable connection to eliminate the summarized attack vectors. Finally, we present how our proposed model can be integrated into existing automotive framework, and discuss the security benefits of our model.

1 INTRODUCTION

As nowadays vehicles move towards increased connectivity, automotive manufacturers have started integrating more software modules inside the vehicle. Software modules and connectivity open up the door towards giving the passenger/driver a seamless experience and increasing safety while driving the vehicle. For instance, mobile applications can leverage vehicle data, for example, tire pressure or fuel level and assist the driver in finding the next gas station. Such type of connectivity from cloud/mobile devices to vehicles is implemented by introducing middleware technologies which reside within the vehicle's head unit, such as SmartDeviceLink, Entune, MirrorLink, Android Auto, CarPlay and so on. Each of these technologies has a middleware software module to extend mobile applications features to the vehicle. Such connectivity offers drivers to use intelligent assistant like Siri to read text messages, make calls, start navigation, launch and interact with applications. News agencies like National Public Radio (NPR) can read news directly in the Ford vehicle via AppLink (Ford's implementation of SmartDeviceLink) (npr.org, 2012). GM's Middleware like NGI gives access to more than 350 data points in the vehicles like suspension, vehicle speed, fuel level, tire pressure, and so on (developer.gm.com, 2017). Further, middleware like Android auto, CarPlay and MirrorLink offer users the great convenience of using the projection, a technique that allows mobile applications to project the navigation or even create a customized user interface on vehicle head unit display. Such an improvement in connectivity increases safety by allowing drivers to keep hands on the wheel and eyes on the road. However, such connectivity to the vehicle also gives a path to the malicious application developers and hackers to penetrate into these connected vehicles and try to compromise the driver's privacy or various vehicle modules.

Automotive security issues were brought up by the research that demonstrated the possibility of hacking vehicles by injecting commands into the Controller Area Network (CAN) bus. By either physical access (typically via the OBD port connection as in (Koscher et al., 2010)) or remote access (mostly through Multimedia Head Unit or Telematics as in (Checkoway et al., 2011)) to the vehicle, researchers successfully controlled a wide range of automotive functionalities such as disabling the brake, stopping the engine, etc. Recently, the remote hacking on an unaltered passenger vehicle further convinced automobile manufacturer as well as the general public the reality of remote exploitation

Zhang, S., Makke, O., Gusikhin, O., Shah, A. and Vasilakos, A.
A Security Model for Dependable Vehicle Middleware and Mobile Applications Connection.
DOI: 10.5220/0006704903790386
In *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2018)*, pages 379-386
ISBN: 978-989-758-293-6

without physical access to the vehicle (Miller and Valasek, 2013). With the integration of vehicle middleware into head unit system to enable mobile application connection, the attack surface for the hackers to manipulate the vehicle will be enlarged, e.g., compromising mobile phone to control vehicles, hijacking communication between mobile phone and head unit systems, etc.

In order to guard the new functionality (vehicle middleware) provided by automotive manufacturers, we first performed a thorough study of potential attack threats against such the deployed middleware. Then we proposed a novel security model that integrates cryptography, network security, system security approaches to eliminate the above summarised threats. Such security model is well compatible with existing implementation of such middleware by vehicle manufacturer like Ford's SmartDeviceLink, etc., thus easy to be widely deployed. Furthermore, it incurs little burden on mobile application developers, and negligible communication overhead for vehicle head units.

The rest of the paper is organised as follows. Section 2 summarizes background of middleware technologies available by various manufacturers. Section 3 explores the potential attack vectors against the vehicle with the mobile application connection framework. In section 4, we propose our security model to counter the attack vectors, and in section 5 we discuss the easy deployment of the proposed security model to SDL. Finally, we conclude in section 6.

2 BACKGROUND

Rapid adoption of mobile devices has led automotive manufacturers to integrate a middleware layer into vehicles to connect the mobile device with the vehicle ecosystem. There are many middleware technologies available currently, such as AppLink by Ford, HondaLink by Honda, MirrorLink by Car Connectivity Consortium, SmartDeviceLink by Smart Device Link Consortium, Apple's CarPlay, Google's Android Auto. The purpose of developing such middleware is to keep drivers' hands on the wheel and eyes on the road.

The middleware provides an interface for mobile applications to communicate with the vehicle modules. The middleware can display the compatible mobile applications on the vehicle's HMI and can allow the driver to interact with it through voice and natural language, or even through touch screen. Automotive manufacturers, with support from the advancements of the operating systems capabilities and the hardware of the mobile devices, have implemented this middleware with use of Remote Procedural Calls (RPC) and Projection technology. Apple CarPlay, Android Auto, and MirrorLink use projection to display a User Interface (UI) on the vehicle HMI. In such implementation all the business code and business logic resides in the mobile application. For these applications using projection technology, a prior approval is required by the owners of the technology. For example, Apple has to approve an application for projection use before it can be used by CarPlay and downloaded through Apple App Store. On the other hand, middleware such as SmartDeviceLink and BlueLink (hyundaiusa.com, 2017) uses Remote Procedural Calls to initiate a request. RPC Communication can be done over USB, Bluetooth, or TCP/IP. Once the request is received by the vehicle, the vehicle will perform corresponding actions and send back a response or notification.

Furthermore, it is possible for the middleware to communicate with brought-in and built-in sensors, IoT devices and provide all the information to mobile phone via Bluetooth (Yeung et al., 2017). The mobile device can be connected to brought-in sensors and IoT devices via Bluetooth Low Energy (BLE) e.g., changing climate control can be based on measurements from wearable devices or dust sensors (Yeung et al., 2017). Moreover, automotive companies are introducing cloud services, such as OnStar (Onstar.com, 2017), which connect each vehicle to the cloud. Users can get vehicle details, remote access vehicle, road assist and more. All information is sent to mobile device from the vehicle to mobile devices through cloud.

3 ATTACK VECTORS AGAINST VEHICLE MIDDLEWARE

In this section, we summarize the potential threats of deploying the vehicle middleware on the head units.

3.1 Mobile App Masquerading

The legitimate mobile applications, approved by the vehicle manufacturer, can invoke the corresponding middleware APIs to interact with vehicle, e.g., obtaining vehicle mechanical status, controlling entertainment, etc. However, reverse engineering of either Android apps or iPhone apps is possible

according to (Enck et al., 2011), (Gitbub.com, 2017), etc. With the knowledge of the implementation of the apps and the middleware APIs, it is straightforward for attackers to either clone or repackage those legitimate applications, and release their own masqueraded apps on Google Play or Apple App Store, which has been proved weak in detecting cloned applications or repackaged applications (Viennot et al., 2014), (Han et al., 2013), etc. Such masqueraded apps with malicious logic by attackers could leak vehicle status, manipulate vehicle functionalities, and even impact vehicle safety.

3.2 Privilege Escalation

To ease the development of mobile applications, the middleware APIs and the corresponding description need to be available to developers. Depending on how open different vehicle manufacturers would like their middleware to be, some of the APIs can be quite security-sensitive, thus deserving auditing before invocation, while the others may not. If not properly protected, such critical APIs could be invoked by a malicious application in a hidden fashion even it is not allowed to do so, e.g., dynamic loading of functions during runtime has been used to compromise Apple private Object C function calls (Han et al., 2013).

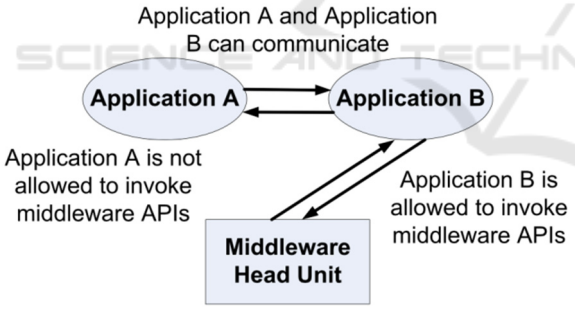

Figure 1: Confused Deputy Attack for Mobile App and Vehicle Middleware Connection.

Even worse, if there are other mobile applications installed on the phone to interact with vehicle middleware, a malicious application can launch confused deputy attack. As shown in Figure 1, a legitimate application B from trusted collaborator has been approved by vehicle manufacturer to use security-sensitive APIs. On the same mobile phone with the above application B installed, a malicious application A leverages a flawed design of B, e.g., an open service, and successfully invokes the critical APIs with the help of B, even if A itself does not have such privilege.

3.3 Replay Attack

Replay attack can be launched by attackers to intercept the communication in one session and retransmit the messages in another session. For instance, the permissions assigned by vehicle manufacturer to the mobile application to use a subset of vehicle middleware APIs can be intercepted by attackers, and later reused in their own malicious application. Another example is that the attacker can intercept the middleware API requests from the legitimate mobile application with permissions, and later resend the requests to the middleware to receive services.

3.4 Compromised Mobile Systems

As the prevalence of smartphones these years, the attacks against such small but complicated devices advance significantly. Researchers have identified a large amount of vulnerabilities on both Android (Wang et al., 2016), (Zhou et al., 2014), etc. and iOS (Han et al., 2013), (Wang et al., 2013), etc. systems that allow attackers to exploit and manipulate. Since the legitimate mobile applications are running on the mobile systems, either Android or iOS, the attackers can exploit vulnerability in the mobile systems, and then leverage the compromised systems to manipulate the mobile application. Note that the mobile system is running in a more privileged mode than the mobile applications, so the attackers can leverage a compromised system to manipulate the application in various ways, e.g., intercepting the middleware API calls from the application and replaying, revising the communication between the application and middleware, etc. Actually, protecting the legacy application from a compromised system is a challenging issue in the research community (Chen et al., 2008), (Guan et al., 2017), etc.

3.5 Compromised Head Unit System

Similarly, the vehicle middleware is running on the head unit system. Recently, researchers have demonstrated the feasibility of either direct (Koscher et al., 2010) or remote hacking into head unit system (Checkoway et al., 2011), and then manipulating the vehicle CAN. Since the head unit system is running in a more privileged mode and also vulnerable to remote exploitation, all the applications/services including the middleware could be tampered considering the system is compromised. The problem of a compromised head unit system is even harder to address, compared to that of the compromised mobile

system. One reason is that it is relatively easier to patch the mobile systems, considering most of the smartphones are Internet-capable and can maintain active as well reliable connection. However, not all head unit systems are equipped with such capability. Even if a vulnerability is identified and vehicle manufacturer releases patch immediately, it might exist in some vehicles for a long time till the vehicle owner drives to the dealership for maintenance, then the patch can be applied. Such a late patch leaves the door open for the attackers to exploit the vulnerability on vehicles head unit systems, and leverage them to manipulate the middleware to easily control the vehicle maliciously.

3.6 DoS or DDoS Attack

Denial of Service (DoS) or Distributed Denial of Service (DDoS) attack can be launched by attackers via various channels, if their goal is only to disturb the routine service. Below we try to provide taxonomy of DoS/DDoS attack:

- **DoS Against Mobile Application:** the attackers can leverage the compromised mobile system to discard the middleware API calls from the legitimate mobile application, or the compromised head unit system to reject response for the API calls from the legitimate mobile application.
- **DoS Against Vehicle Control:** the malicious mobile application intensively invokes the middleware API calls to query vehicle status, thus flooding messages to vehicle CAN bus to interference the vehicle control functionality, or broadcasts messages to head unit screen to disturb the regular display, e.g., navigation, etc.
- **DDoS Against Vehicle Manufacturer:** vehicle manufacturer need to maintain a facility (e.g., cloud servers) to involve some of the following activities, e.g., releasing patches for head unit systems, managing all end users' credential, assigning permissions to invoke various middleware APIs, or negotiating secret keys with different parties to encrypt/decrypt secure communication, etc. Attackers can control a large amount of zombie machines, even as simple as IoT (Internet of Things) devices, to launch DDoS attack against vehicle manufacturer facility, thus failing the middleware functionality.

3.7 Man in the Middle Attack

As there are multiple stakeholders involved, like vehicle manufacturer facility, mobile application, vehicle head unit, attackers can "jump in the middle" of the interaction between any two parties, and pretend to be one party to the other involved party. Such MITM (Man In The Middle) attack typically can break communication semantics protected by encryption/decryption. For instance, via MITM attack, one can intercept the encrypted credential of users when users initiate the connection with vehicle manufacturer facility, or obtain the permissions to invoke security-sensitive middleware APIs from the requests of mobile applications and later reuse such permissions in their own illegitimate application.

4 SECURITY MODEL

In this section, we present our proposed security model for vehicle middleware, and discuss its effectiveness of mitigating various threats mentioned in the previous section.

4.1 Capability based API Management

Developers can be from vehicle manufacturers, their collaborators, or other third parties to connect their mobile applications to vehicle middleware to offer various functionalities. Depending on the affiliation of developers, some mobile applications can be fully trusted, while the other may not. Furthermore, as discussed above, some vehicle middleware APIs can be quite security-sensitive, e.g., query vehicle mechanic status, like velocity and location, etc., but the others may not, e.g., tune the volume of speaker, etc. Hence, it is critical to regulate the middleware API usage by different applications based on how much trust we can place on their developers. We propose to use capability to protect the usage of middleware APIs. Specifically, vehicle manufacturer can define capabilities that are needed for the mobile applications to call the middleware APIs on head unit. We leave the definition of capabilities and the corresponding middleware APIs to individual vehicle manufacturer, who has better understanding of their product and security requirements. It is suggested to have a complete documentation of available middleware APIs and their corresponding capabilities. The vehicle manufacturer may deploy cloud servers to manage the distribution of the capabilities upon requests from developers for their applications. The middleware core on the head units

needs to check the appropriate capability before approving any API call from mobile applications.

4.1.1 Approving Capability Request

When receiving capability requests from mobile application developer, manufacturer cloud server can make decision based on the trustworthiness of the developer, e.g., a legitimate collaborator or a third party unknown developer, the reputation/rating of the developer, etc. Different capabilities may require different trust level or reputation of developers, which can be defined flexibly by various manufacturers. In order to prevent developers from requesting more capabilities than needed by their application, manufacturer can also demand them to submit the application for review together with the capability request. Once approved, the cloud server will respond with the requested capabilities to the mobile application, which can use them to invoke corresponding middleware APIs during runtime. The manufacturer cloud server also needs to choose a unique AppID for the manufacturer with the capabilities request. The manufacturer cloud needs to record the AppID together with the approved capabilities for later reference and management (like expiration or update etc.). This AppID will be sent back to application developer.

4.1.2 The Ownership of the Approved Capabilities

The approved capabilities are the "tickets" to present to the vehicle middleware when mobile applications need to invoke the middleware APIs. However, the attackers can steal such "tickets" from other mobile applications if one developer does not have any or the one they need. The stealth is feasible because typically the approved capabilities need to be included in the application package, which can be easily obtained by downloading the application and disassembling the package. Note that the AppID is the unique identifier to distinguish different applications from the manufacturer cloud servers, but it cannot used to identify a particular application's ownership of capabilities. Such AppID is assigned to the application by manufacturer cloud, thus also easy to be obtained by attackers from application package to launch masquerading attack.

We notice that the application name is unique on the application store/market, which typically cannot be changed after submitting application store for review. Hence, it is required that the application name needs to be sent to manufacturer cloud when requesting capabilities. The approved capabilities together with the unique AppID and application name will be sent back to the dedicated application for middleware API calls. The middleware SDK on smartphone forwards the API calls together with the above information to the middleware core on head unit. Also it needs to retrieve the application's name from the smartphone system, and send it to the middleware core as well. Then the middleware core compares the application name retrieved by the middleware SDK (indicating the identify of the running application) and the one from the manufacturer cloud (indicating the owner of the approved capabilities). Hence, any mismatch implies application masquerading attack.

4.1.3 Approved Capability Revocation

When application developers apply for capabilities approval from manufacturer cloud, they also need to indicate the period of validity they would like the approved capabilities to be. Based on the trust level of the application developers and the requested capabilities, the manufacturer cloud approves the capabilities request with the expiration time. It is the developers' responsibility to renew the capabilities approval before expiration. Once the renewal request is approved, the new tuple {application name, AppID, approved capabilities, expiration time} with the updated expiration time can be available to mobile applications via update. If the application developers need extra capabilities for new functionalities in their mobile applications, they can follow similar procedure as renewal. When the mobile application initializes connection to the middleware core, the middleware SDK should first pull the expiration status from the manufacturer cloud to validate the validity of the approved capabilities. If the approval expires, the middleware SDK refuses connection request to the middleware core and responds with a capability expiration error.

4.1.4 Integrity of the Approved Capabilities

If the tuple {application name, AppID, approved capabilities, expiration time} from the manufacturer cloud is sent to the application developers in clear text, attackers can steal it from other applications and revise any item to over-privilege their own applications. For instance, by changing the application name in the tuple, the attackers can pretend to be another application with the approved capabilities. They can also manipulate the approved capabilities or expiration time to over-privilege their application in functionalities or period of validity.

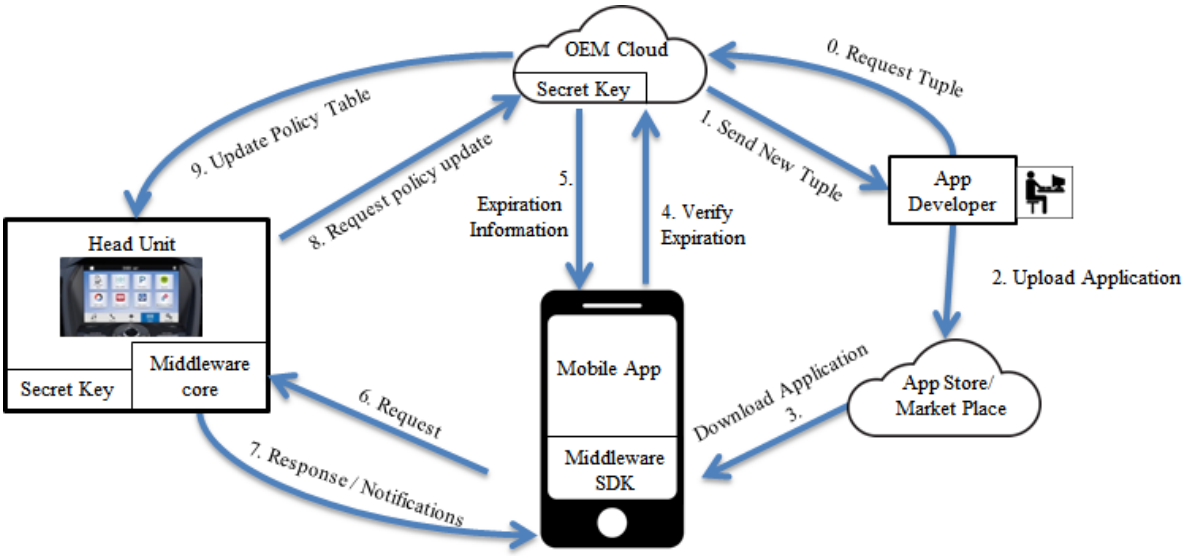

Figure 2: Security Model.

Hence, the tuple from the manufacturer cloud has to be authenticated to preserve its integrity. Simply the manufacturer cloud server can sign the tuple with its private key, and the middleware core can verify the signature with its public key. Hence, any revision to the tuple from one who does not hold the private key of cloud server will cause mismatched signature.

4.2 Workflow of the Model

Figure 2 demonstrates the workflow of the proposed security model. Below we present the details of each step.

0. The mobile application developers request capabilities from manufacturer cloud, indicating the set of middleware APIs they need to invoke. Together with the request, they also send their application's name and expected validation time.
1. If the request is approved based on Section 4.1.1, the manufacturer cloud server selects a unique AppID for the application, and then build the tuple {application name, AppID, approved capabilities, expiration time}. It uses its private key to sign the tuple, and send the tuple together with the signature to the application developer. The tuple is recorded by manufacturer cloud in its policy for update management.
2. The application developers need to integrate the tuple and the signature in their application package. Then they publish their application in the application store/market (via regular review process if any).
3. The application package is downloaded into a smartphone and installed.
4. Whenever the mobile application needs to invoke middleware API call, it also sends the tuple and the signature. For any new session, the middleware SDK retrieves the application name from smartphone system, and sends it together with the tuple and signature to the manufacturer cloud to verify expiration status.
5. Based on AppID, the manufacturer cloud queries its policy database for the corresponding expiration status and then sends back the expiration information to the middleware SDK.
6. If the approval is still valid, the middleware SDK sends API request, tuple and signature to middleware core on head unit.
7. The middleware core verifies the signature and the application name, and then responds the request if verification succeeds.
8. Whenever possible, e.g., the head unit can be connected to Internet or the vehicle is under maintenance in dealership, the middleware core can request any update of the middleware APIs and the corresponding capabilities. The initial setup of such configuration can be done before the delivery of the vehicle.
9. The manufacturer cloud responds such update request to the middleware core if any.

4.3 Trusted Execution Environment for the Middleware Core and the Mobile Application

In Section 3.4 and 3.5, we discussed the compromised systems running on smartphone or the head unit can manipulate the execution of the mobile application or the middleware core. Due to the common belief that smartphone system and the head unit system are vulnerable to remote exploitation, building a trusted execution environment for security-critical programs, e.g., the application, the middleware SDK, as well the middleware core, is highly desired. Fortunately, most smartphones and head units are running on ARM processors, which recently released a hardware based security extension, ARM TrustZone, partitioning hardware and software resources for separate uses. We have built a trusted runtime environment, TrustShadow, based on TrustZone for legacy applications on edge devices (Guan et al., 2017). The TrustShadow prototype can be easily ported into the proposed model, to protect the middleware core from the untrusted head unit system, and the mobile application as well as the middleware SDK from the untrusted smartphone system. For example, we run the middleware core and the corresponding device driver interacting with CAN bus inside the built trusted runtime environment, while the head unit system runs outside. Intuitively, the communication with the middleware SDK, CAN bus, and the processing of the corresponding information should be in the trusted runtime, protected by the hardware partition from the system running outside. The middleware SDK and the mobile applications can be protected in a similar fashion.

5 DISCUSSIONS

In this section, we first introduce SmartDeviceLink, an implementation of connection framework between vehicle middleware and mobile phones, review trade-off between privacy and convenience for fully automated control applications, and finally present a case study showing the effectiveness of the proposed model against app masquerading attack.

5.1 SmartDeviceLink Example

SmartDeviceLink (SDL) is an open source middleware framework led by Ford Motor. SDL allows mobile application developer to implement an in-vehicle experience for their users. To develop a SDL compatible mobile application, application developers would have to request the permissions that their application will need, e.g., speed, GPS and so on. Once the cloud servers approve the request, developers can receive a unique AppID. The cloud server maintains a global policy table mapping an individual AppID with the approved permissions, and share it with all the head unit systems. Then the mobile application can use this unique AppID to register with the vehicle middleware (SDL_CORE), through the SDL API available in the software development Kit (SDK) on the smart phones. Once registered successfully, SDL_SDK will use this APPID to generate any future RPC request to SDL_CORE on behalf of the mobile application. SDL_CORE would request for updated policy table every time it connects to the Internet. Whenever there is a new mobile application registering to SDL_CORE, it would request policy table update according to the AppID of the new application to make decisions for the incoming API calls from the application (Github.com, 2017).

5.2 Automated Control Applications

The mobile applications can be designed to support fully automated control, to offer better user experiences. For instance, such application can also read user's calendar from his/her mobile phone, and automatically start corresponding settings on the incoming vehicle when the uber drive is 5 minutes far away. Moreover, for the users with a smart wearable device with temperature sensor and heart rate monitor, the application on mobile phone can read the information and intelligently adjust the air conditioning in the vehicle. Such application can also request automobile head unit play music based on mobile user's personal preference, which can be obtained on user's mobile phone as well.

Utilizing the appropriate vehicle middleware APIs, such kind of fully automated control applications can be implemented. Enjoying the convenience offered by such applications, mobile users should be warned to pay attention to the information collected by such applications, since most of the information can be private and even sensitive. It is the mobile users' decision to either agree or disagree with the permission requirements made by such applications when installing them on their mobile phones. The security model proposed in this paper does not handle the privacy leakage on users' smartphone since that is the issue for smartphone security.

5.3 Attack Case Study

Below we present one case study to demonstrate how the proposed security model can preserve the dependable connection between vehicles and mobile phones under *Application Masquerading Attack*. The attackers can simply search for the applications that can connect to the vehicle head unit, and download it from Google Play. Through reverse engineering, they can obtain the middleware API calls from the application, as well as any confidential that the application received from manufacturer cloud. With such information, the attackers implement their own application and pretend to be legitimate, but with malicious logic that will leverage the middleware APIs to manipulate the navigation functionality. With our proposed security model, such attack is not feasible. On one hand, the confidential received by the mobile application contains its name, and is signed by the manufacturer servers, which will be detected by head unit middleware if modified by attackers. On the other hand, the middleware SDK extracts the application name from mobile phone system directly, to check the ownership of the confidential. In the scenario of this attack, mismatch will be detected and the corresponding API calls from the malicious application will be denied.

6 CONCLUSIONS

In this paper, we present a novel security model that can initiate dependable connection between vehicle systems and smartphones. In particular, such model incurs minimal burden on mobile application developers, and negligible communication overhead. Our analysis and comparison with the existing methods demonstrate that the proposed model is effective in defeating most of the security threats that are introduced by the new communication channel between vehicle systems and smartphones.

REFERENCES

Developer.gm.com, 2017. Next Generation Infotainment. [online] Available at: https://developer.gm.com/ngi.

Npr.org, 2012. Ford enables voice control of NPR app delivering in-car, On-demand access to news, programs and stations. [online] Available at: http://www.npr.org/about/press/2012/010912.NPRAnnouncesFordSYNC.html.

Miller, S. and Valasek, C. 2013. Remote Exploitation of an Unaltered Passenger Vehicle, [online] Available at: http://illmatics.com/Remote%20Car%20Hacking.pdf.

Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H. and Savage S. 2010. Experimental security analysis of a modern automobile. *IEEE Symposium on Security and Privacy, pp. 447–462.*

Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, A. and Kohno, T. 2011. Comprehensive experimental analyses of automotive attack surfaces. *USENIX conference on Security.*

Hyundaiusa.com, 2017. Hyundai Blue Link | 3 Years Free on Eligible 2018 Models. [online] Available at: https://www.hyundaiusa.com/bluelink/index.aspx.

Yeung, J., Makke, O., MacNeille, P. and Gusikhin, O. 2017. SmartDeviceLink as an Open Innovation Platform for Connected Car Features and Mobility Applications. *SAE Int. J. Passeng. Cars – Electron. Electr. Syst. 10(1):231-239*.

Onstar.com, 2017. Home | OnStar. [online] Available at: https://www.onstar.com/us/en/home.html.

Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S. 2011. A Study of Android Application Security. *USENIX conference on Security*.

Gitbub.com, 2017. iOS reverse engineering. [online] Available at: https://github.com/iosre/iOSAppReverse Engineering.

Viennot, N., Garcia, E. and Nieh, J. 2014. A Measurement Study of Google Play, *ACM SIGMETRICS*.

Han, J., Mon Kywe, S., Yan, Q., Bao, F., Deng, R., Gao, D., Li, Y., and Zhou, J. 2013. Launching Generic Attacks on iOS with Approved Third-Party Applications, *ACNS*.

Wang, K., Zhang, Y. and Liu, P. 2016. Call Me Back! Attacks on System Server and System Apps in Android through Synchronous Callback. *ACM CCS*.

Zhou, X., Lee, Y., Zhang, N., Naveed, M. and Wang, X. 2014. The peril of fragmentation: Security hazards in android device driver customizations. *IEEE S&P*.

Wang, T., Lu, K., Lu, L., Chung, S. and Lee, W. 2013. Jekyll on iOS: When Benign Apps Become Evil. *USENIX Security Symposium*.

Chen, X., Garfinkel T., E. Lewis, C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., Dwoskin, J. and Ports DRK. 2008. Overshadow: virtualization-based approach to retrofitting protection in commodity operating systems. *ASPLOS*.

Guan, L., Liu, P., Xing, X., Ge, X., Zhang, S., Yu, M. and Jaeger, T. 2017. TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone. *ACM Mobisys*.

Github.com, 2017. smartdevicelink/sdl_core. [online] Available at: https://github.com/smartdevicelink/sdl_core.