Comprehensive Experimental Analyses of Automotive Attack Surfaces

Stephen Checkoway, Damon McCoy, Brian Kantor,
Danny Anderson, Hovav Shacham, and Stefan Savage
University of California, San Diego

Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno
University of Washington

Abstract

Modern automobiles are pervasively computerized, and hence potentially vulnerable to attack. However, while previous research has shown that the *internal* networks within some modern cars are insecure, the associated threat model — requiring *prior physical access* — has justifiably been viewed as unrealistic. Thus, it remains an open question if automobiles can also be susceptible to *remote* compromise. Our work seeks to put this question to rest by systematically analyzing the *external* attack surface of a modern automobile. We discover that remote exploitation is feasible via a broad range of attack vectors (including mechanics tools, CD players, Bluetooth and cellular radio), and further, that wireless communications channels allow long distance vehicle control, location tracking, in-cabin audio exfiltration and theft. Finally, we discuss the structural characteristics of the automotive ecosystem that give rise to such problems and highlight the practical challenges in mitigating them.

1 Introduction

Modern cars are controlled by complex distributed computer systems comprising millions of lines of code executing on tens of heterogeneous processors with rich connectivity provided by internal networks (e.g., CAN). While this structure has offered significant benefits to efficiency, safety and cost, it has also created the opportunity for new attacks. For example, in previous work we demonstrated that an attacker connected to a car's *internal network* can circumvent *all* computer control systems, including safety critical elements such as the brakes and engine [14].

However, the threat model underlying past work (including our own) has been met with significant, and justifiable, criticism (e.g., [1, 3, 16]). In particular, it is widely felt that presupposing an attacker's ability to *physically* connect to a car's internal computer network may be unrealistic. Moreover, it is often pointed out that attackers with physical access can easily mount non-computerized attacks as well (e.g., cutting the brake lines).

This situation suggests a significant gap in knowledge, and one with considerable practical import. To what extent are external attacks possible, to what extent are they practical, and what vectors represent the greatest risks? Is the etiology of such vulnerabilities the same as for desktop software and can we think of defense in the same manner? Our research seeks to fill this knowledge gap through a systematic and empirical analysis of the remote attack surface of late model mass-production sedan.

We make four principal contributions:

Threat model characterization. We systematically synthesize a set of *possible* external attack vectors as a function of the attacker's ability to deliver malicious input via particular modalities: indirect physical access, short-range wireless access, and long-range wireless access. Within each of these categories, we characterize the attack surface exposed in current automobiles and their surprisingly large set of I/O channels.

Vulnerability analysis. For each access vector category, we investigate one or more concrete examples in depth and assess the level of actual exposure. In each case we find the existence of *practically exploitable vulnerabilities* that permit arbitrary automotive control *without requiring direct physical access*. Among these, we demonstrate the ability to compromise a car via vulnerable diagnostics equipment widely used by mechanics, through the media player via inadvertent playing of a specially modified song in WMA format, via vulnerabilities in hands-free Bluetooth functionality and, finally, by calling the car's cellular modem and playing a carefully crafted audio signal encoding both an exploit and a bootstrap loader for additional remote-control functionality.

Threat assessment. From these uncovered vulnerabilities, we consider the question of "utility" to an attacker: what capabilities does the vulnerability enable? Unique to this work, we study how an attacker might leverage a car's external interfaces for post-compromise control. We demonstrate multiple post-compromise control channels (including TPMS wireless signals and FM radio), inter-

active remote control via the Internet and real-time data exfiltration of position, speed and surreptitious streaming of cabin audio (i.e., anything being said in the vehicle) to an outside recipient. Finally, we also explore potential attack scenarios and gauge whether these threats are purely conceptual or whether there are plausible motives that transform them into actual risks. In particular, we demonstrate complete capabilities for both theft and surveillance.

Synthesis. On reflection, we noted that the vulnerabilities we uncovered have surprising similarities. We believe that these are not mere coincidences, but that many of these security problems arise, in part, from systemic structural issues in the automotive ecosystem. Given these lessons, we make a set of concrete, pragmatic recommendations which significantly raise the bar for automotive system security. These recommendations are intended to "bridge the gap" until deeper architectural redesign can be carried out.

2 Background and Related Work

Modern automobiles are controlled by a heterogeneous combination of digital components. These components, *Electronic Control Units* (ECUs), oversee a broad range of functionality, including the drivetrain, brakes, lighting, and entertainment. Indeed, very few operations are not mediated by computer control in a modern vehicle (with the parking brake and steering being the last holdouts, though semi-automatic parallel parking capabilities are available in some vehicles and full steer-by-wire has been demonstrated in several concept cars). Charette estimates that a modern luxury vehicle includes up to 70 distinct ECUs including tens of millions of lines of code [5]. In turn, ECUs are interconnected by common wired networks, usually a variant of the Controller Area Network (CAN) [12] or FlexRay bus [8]. This interconnection permits complex safety and convenience features such as pre-tensioning of seat-belts when a crash is predicted and automatically varying radio volume as a function of speed.

At the same time, this architecture provides a broad *internal* attack surface since on a given bus each component has at least implicit access to every other component. Indeed, several research groups have described how this architecture might be exploited in the presence of compromised components [15, 24, 26, 27, 28] or demonstrated such exploits by spoofing messages to isolated components in the lab [10]. Most recently, our own group documented experiments on a complete automobile, demonstrating that *if* an adversary were able to communicate on one or more of a car's internal network buses, then this capability could be sufficient to maliciously control critical components across the *entire car* (including dangerous behavior such as forcibly engaging or disengaging individual brakes independent of driver input) [14]. However, these results raise the question of *how* an adversary might be able to access a car's internal bus (and thus compromise its ECUs) absent direct physical access, a question that we answer in this paper.

About the latter question — understanding the *external* attack surface of modern vehicles — there has been far less research work. Among the exceptions is Rouf et al.'s recent analysis of the wireless Tire Pressure Monitoring System (TPMS) in a modern vehicle [22]. While their work was primarily focused on the privacy implications of TPMS broadcasts, they also described methods for manipulating drivers by spoofing erroneous tire pressure readings and, most relevant to our work, an experience in which they accidentally caused the ECU managing TPMS data to stop functioning through wireless signals alone. Still others have focused on the computer security issues around car theft, including Francillon et al.'s recent demonstration of relay attacks against keyless entry systems [9], and the many attacks on the RFID-based protocols used by engine immobilizers to identify the presence of a valid ignition key, e.g., [2, 6, 11]. Orthogonally, there has been work that considers the *future* security issues (and expanded attack surface) associated with proposed vehicle-to-vehicle (V2V) systems (sometimes also called vehicular ad-hoc networks, or VANETs) [4, 13, 21]. To the best of our knowledge, however, we are the first to consider the full external attack surface of the contemporary automobile, characterize the threat models under which this surface is exposed, and experimentally demonstrate the practicality of remote threats, remote control, and remote data exfiltration. Our experience further gives us the vantage point to reflect on some of the ecosystem challenges that give rise to these problems and point the way forward to better secure the automotive platform in the future.

3 Automotive threat models

While past work has illuminated specific classes of threats to automotive systems — such as the technical security properties of their internal networks [14, 15, 24, 26, 27, 28] — we believe that it is critical for future work to place specific threats and defenses in the context of the entire automotive platform. In this section, we aim to bootstrap such a comprehensive treatment by characterizing the threat model for a modern automobile. Though we present it first, our threat model is informed significantly by the experimental investigations we carried out, which are described in subsequent sections.

In defining our threat model, we distinguish between *technical* capabilities and *operational* capabilities.

Technical capabilities describe our assumptions concerning what the adversary knows about its target vehicles as well as her ability to analyze these systems to develop malicious inputs for various I/O channels. For example, we assume that the adversary has access to an instance of

the automobile model being targeted and has the technical skill to reverse engineer the appropriate subsystems and protocols (or is able to purchase such information from a third-party). Moreover, we assume she is able to obtain the appropriate hardware or medium to transmit messages whose encoding is appropriate for any given channel.[1] When encountering cryptographic controls, we also assume that the adversary is computationally bounded and cannot efficiently brute force large shared secrets, such as large symmetric encryption keys. In general, we assume that the attacker only has access to information that can be directly gleaned from examining the systems of a vehicle similar to the one being targeted.[2] We believe these assumptions are quite minimal and mimic the access afforded to us when conducting this work.

By contrast, operational capabilities characterize the adversary's requirements in delivering a malicious input to a particular access vector in the field. In considering the full range of I/O capabilities present in a modern vehicle, we identify the qualitative differences in the challenges required to access each channel. These in turn can be roughly classified into three categories: indirect physical access, short-range wireless access, and long-range wireless access. In the remainder of this section we explore the threat model for each of these categories and within each we synthesize the "attack surface" presented by the full range of I/O channels present in today's automobiles. Figure 1 highlights where I/O channels exist on a modern automobile today.

3.1 Indirect physical access

Modern automobiles provide several physical interfaces that either directly or indirectly access the car's internal networks. We consider the full physical attack surface here, under the constraint that the adversary may not *directly* access these physical interfaces herself but must instead work through some intermediary.

OBD-II. The most significant automotive interface is the OBD-II port, federally mandated in the U.S., which typically provides direct access to the automobile's key CAN buses and can provide sufficient access to compromise the full range of automotive systems [14]. While our threat model forbids the adversary from direct access herself, we note that the OBD-II port is commonly accessed by service personnel during routine maintenance for both diagnostics and ECU programming.

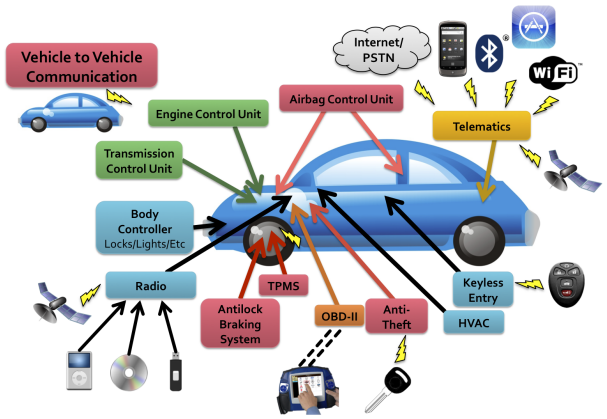

Figure 1: *Digital I/O channels appearing on a modern car.* Colors indicate rough grouping of ECUs by function.

Historically this access is achieved using dedicated handheld "scan" tools such as Ford's NGS, Nissan's Consult II and Toyota's Diagnostic Tester which are themselves programmed via Windows-based personal computers. For modern vehicles, most manufacturers have adopted an approach that is PC-centric. Under this model, a laptop computer interfaces with a "PassThru" device (typically directly via USB or WiFi) that in turn is plugged into the car's OBD-II port. Software on the laptop computer can then interrogate or program the car's ECUs via this device (typically using the standard SAE J2534 API). Examples of such tools include Toyota's TIS, Ford's VCM, Nissan's Consult 3 and Honda's HDS among others.

In both situations Windows-based computers directly or indirectly control the data to be sent to the automobile. Thus, if an adversary were able to compromise such systems at the dealership she could amplify this access to attack any cars under service. Such laptop computers are typically Internet-connected (indeed, this is a requirement for some manufacturers' systems), so traditional means of personal computer compromise could be employed.

Further afield, electric vehicles may also communicate with external chargers via the charging cable. An adversary able to compromise the external charging infrastructure may thus be able to leverage that access to subsequently attack any connected automobile.

Entertainment: Disc, USB and iPod. The other important class of physical interfaces are focused on entertainment systems. Virtually all automobiles shipped today provide a CD player able to interpret a wide variety of audio formats (raw "Red Book" audio, MP3, WMA, and so on). Similarly, vehicle manufacturers also provide some kind of external digital multimedia port (typically either a USB port or an iPod/iPhone docking port) for allowing users to control their car's media

[1] For the concrete vulnerabilities we will explore, the hardware cost for such capabilities is modest, requiring only commodity laptop computers, an audio card, a USB-to-CAN interface, and, in a few instances, an inexpensive, off-the-shelf USRP software radio platform.

[2] A question which we do not consider in this work is the extent to which the attack surface is "portable" between vehicle models from a given manufacturer. There is significant evidence that some such attacks are portable as manufacturers prefer to build a small number of underlying platforms that are specialized to deliver model-specific features, but we are not in a position to evaluate this question comprehensively.

system using their personal audio player or phone. Some manufacturers have widened this interface further; BMW and Mini recently announced their support for "iPod Out," a scheme whereby Apple media devices will be able to control the display on the car's console.

Consequently, an adversary might deliver malicious input by encoding it onto a CD or as a song file and using social engineering to convince the user to play it. Alternatively, she might compromise the user's phone or iPod out of band and install software onto it that attacks the car's media system when connected.

Taking over a CD player alone is a limited threat; but, for a variety of reasons, automotive media systems are not standalone devices. Indeed, many such systems are now CAN bus interconnected, either to directly interface with other automotive systems (e.g., to support chimes, certain hands-free features, or to display messages on the console) or simply to support a common maintenance path for updating all ECU firmware. Thus, counterintuitively, a compromised CD player can offer an effective vector for attacking other automotive components.

3.2 Short-range wireless access

Indirect physical access has a range of drawbacks including its operational complexity, challenges in precise targeting, and the inability to control the time of compromise. Here we weaken the operational requirements on the attacker and consider the attack surface for automotive wireless interfaces that operate over short ranges. These include Bluetooth, Remote Keyless Entry, RFIDs, Tire Pressure Monitoring Systems, WiFi, and Dedicated Short-Range Communications. For this portion of the attack surface we assume that the adversary is able to place a wireless transmitter in proximity to the car's receiver (between 5 and 300 meters depending on the channel).

Bluetooth. Bluetooth has become the de facto standard for supporting hands-free calling in automobiles and is standard in mainstream vehicles sold by all major automobile manufacturers. While the lowest level of the Bluetooth protocol is typically implemented in hardware, the management and services component of the Bluetooth stack is often implemented in software. In normal usage, the Class 2 devices used in automotive implementations have a range of 10 meters, but others have demonstrated that this range can be extended through amplifiers and directional antennas [20].

Remote Keyless Entry. Today, all but entry-level automobiles shipped in the U.S. use RF-based remote keyless entry (RKE) systems to remotely open doors, activate alarms, flash lights and, in some cases, start the ignition (all typically using digital signals encoded over 315 MHz in the U.S. and 433 MHz in Europe).

Tire pressure. In the U.S., all 2007 model year and newer cars are required to support a Tire Pressure Monitoring System (TPMS) to alert drivers about under or over inflated tires. The most common form of such systems, so-called "Direct TPMS," uses rotating sensors that transmit digital telemetry (frequently in similar bands as RKEs).

RFID car keys. RFID-based vehicle immobilizers are now nearly ubiquitous in modern automobiles and are mandatory in many countries throughout the world. These systems embed an RFID tag in a key or key fob and a reader in or near the car's steering column. These systems can prevent the car from operating unless the correct key (as verified by the presence of the correct RFID tag) is present.

Emerging short-range channels. A number of manufacturers have started to discuss providing 802.11 WiFi access in their automobiles, typically to provide "hotspot" Internet access via bridging to a cellular 3G data link. In particular, Ford offers this capability in the 2012 Ford Focus. (Several 2011 models also provided WiFi receivers, but we understand they were used primarily for assembly line programming.)

Finally, while not currently deployed, an emerging wireless channel is defined in the Dedicated Short-Range Communications (DSRC) standard, which is being incorporated into proposed standards for Cooperative Collision Warning/Avoidance and Cooperative Cruise Control. Representative programs in the U.S. include the Department of Transportation's Cooperative Intersection Collision Avoidance Systems (CICAS-V) and the Vehicle Safety Communications Consortium's VSC-A project. In such systems, forward vehicles communicate digitally to trailing cars to inform them of sudden changes in acceleration to support improved collision avoidance and harm reduction.

Summary. For all of these channels, if a vulnerability exists in the ECU software responsible for parsing channel messages, then an adversary may compromise the ECU (and by extension the entire vehicle) simply by transmitting a malicious input within the automobile's vicinity.

3.3 Long-range wireless

Finally, automobiles increasingly include long distance (greater than 1 km) digital access channels as well. These tend to fall into two categories: broadcast channels and addressable channels.

Broadcast channels. Broadcast channels are channels that are not specifically directed towards a given automobile but can be "tuned into" by receivers on-demand. In addition to being part of the external attack surface, long-range broadcast mediums can be appealing as control channels (i.e., for triggering attacks) because they are difficult to attribute, can command multiple receivers at once, and do not require attackers to obtain precise addressing for their victims.

The modern automobile includes a plethora of broadcast receivers for long-range signals: Global Positioning System (GPS),[3] Satellite Radio (e.g., SiriusXM receivers common to late-model vehicles from Honda/Accura, GM, Toyota, Saab, Ford, Kia, BMW and Audi), Digital Radio (including the U.S. HD Radio system, standard on 2011 Ford and Volvo models, and Europe's DAB offered in Ford, Audi, Mercedes, Volvo and Toyota among others), and the Radio Data System (RDS) and Traffic Message Channel (TMC) signals transmitted as digital subcarriers on existing FM-bands.

The range of such signals depends on transmitter power, modulation, terrain, and interference. As an example, a 5 W RDS transmitter can be expected to deliver its 1.2 kbps signal reliably over distances up to 10 km. In general, these channels are implemented in an automobile's media system (radio, CD player, satellite receiver) which, as mentioned previously, frequently provides access via internal automotive networks to other key automotive ECUs.

Addressable channels. Perhaps the most important part of the long-range wireless attack surface is that exposed by the remote telematics systems (e.g., Ford's Sync, GM's OnStar, Toyota's SafetyConnect, Lexus' Enform, BMW's BMW Assist, and Mercedes-Benz' mbrace) that provide continuous connectivity via cellular voice and data networks. These systems provide a broad range of features supporting safety (crash reporting), diagnostics (early alert of mechanical issues), anti-theft (remote track and disable), and convenience (hands-free data access such as driving directions or weather).

These cellular channels offer many advantages for attackers. They can be accessed over arbitrary distance (due to the wide coverage of cellular data infrastructure) in a largely anonymous fashion, typically have relatively high bandwidth, are two-way channels (supporting interactive control and data exfiltration), and are individually addressable.

Stepping back. There is a significant knowledge gap between these possible threats and what is known to date about automotive security. Given this knowledge gap, much of this threat model may seem far-fetched. However, in the next section of this paper we find quite the opposite. For each category of access vector we will explore one or two aspects of the attack surface deeply, identify concrete vulnerabilities, and explore and demonstrate practical attacks that are able to completely compromise our target automobile's systems without requiring direct physical access.

[3] We do not currently consider GPS to be a practical access vector for an attacker because in all automotive implementations we are aware of, GPS signals are processed predominantly in custom hardware. By contrast, we have identified significant software-based input processing in other long-range wireless receivers.

4 Vulnerability Analysis

We now turn to our experimental exploration of the attack surface. We first describe the automobile and key components under evaluation and provide some context for the tools and methods we employed. We then explore in-depth examples of vulnerabilities via indirect physical channels (CDs and service visits), short-range wireless channels (Bluetooth), and long-range wireless (cellular). Table 1 summarizes these results as well as our qualitative assessment of the cost (in effort) to discover and exploit these vulnerabilities.

4.1 Experimental context

All of our experimental work focuses on a moderately priced late model sedan with the standard options and components. Between 100,000 and 200,000 of this model were produced in the year of manufacture. The car includes less than 30 ECUs comprising both critical drivetrain components as well as less critical components such as windshield wipers, door locks and entertainment functions. These ECUs are interconnected via multiple CAN buses, bridged where necessary. The car exposes a number of external vectors including the OBD-II port, media player, Bluetooth, wireless TPMS sensors, keyless entry, satellite radio, RDS, and a telematics unit. The last provides voice and data access via cellular networks, connects to all CAN buses, and has access to Bluetooth, GPS and independent hands-free audio functionality (via an embedded microphone in the passenger cabin). We also obtained the manufacturer's standard "PassThru" device used by dealerships and service stations for ECU diagnosis and reprogramming, as well as the associated programming software. For several ECUs, notably the media and telematics units, we purchased a number of identical replacement units via on-line markets to accommodate the inevitable "bricking" caused by imperfect attempts at code injection.

Building on our previous work, we first established a set of messages and signals that could be sent on our car's CAN bus (via OBD-II) to control key components (e.g., lights, locks, brakes, and engine) as well as injecting code into key ECUs to insert persistent capabilities and to bridge across multiple CAN buses [14]. Note, such inter-bus bridging is critical to many of the attacks we explore since it exposes the attack surface of one set of components to components on a separate bus; we explain briefly here. Most vehicles implement multiple buses, each of which host a subset of the ECUs.[4] However, for func-

[4] In prior work we hypothesized that CAN buses were purposely separated for security reasons — one for safety-critical components like the radio and engine and the other for less important components such as a radio. Based on discussions with industry experts we have learned that this separation has until now often been driven by bandwidth and integration concerns and not necessarily security.

Vulnerability Class	Channel	Implemented Capability	Visible to User	Scale	Full Control	Cost	Section
Direct physical	OBD-II port	Plug attack hardware directly into car OBD-II port	Yes	Small	Yes	Low	Prior work [14]
Indirect physical	CD	CD-based firmware update	Yes	Small	Yes	Medium	Section 4.2
	CD	Special song (WMA)	Yes*	Medium	Yes	Medium-High	Section 4.2
	PassThru	WiFi or wired control connection to advertised PassThru devices	No	Small	Yes	Low	Section 4.2
	PassThru	WiFi or wired shell injection	No	Viral	Yes	Low	Section 4.2
Short-range wireless	Bluetooth	Buffer overflow with paired Android phone and Trojan app	No	Large	Yes	Low-Medium	Section 4.3
	Bluetooth	Sniff MAC address, brute force PIN, buffer overflow	No	Small	Yes	Low-Medium	Section 4.3
Long-range wireless	Cellular	Call car, authentication exploit, buffer overflow (using laptop)	No	Large	Yes	Medium-High	Section 4.4
	Cellular	Call car, authentication exploit, buffer overflow (using iPod with exploit audio file, earphones, and a telephone)	No	Large	Yes	Medium-High	Section 4.4

Table 1: *Attack surface capabilities.* The Visible to User column indicates whether the compromise process is visible to the user (the driver or the technician); we discuss social engineering attacks for navigating user detection in the body. For ($*$), users will perceive a malfunctioning CD. The Scale column captures the approximate scale of the attack, e.g., the CD firmware update attack is small-scale because it requires distributing a CD to each target car. The Full Control column indicates whether this exploit yields full control over the component's connected CAN bus (and, by transitivity, all the ECUs in the car). Finally, the Cost column captures the approximate effort to develop these attack capabilities.

tionality reasons these buses must be interconnected to support the complex coupling between pairs of ECUs and thus a small number of ECUs are physically connected to multiple buses and act as logical bridges. Consequently, by modifying the "bridge" ECUs (either via a vulnerability or simply by reflashing them over the CAN bus as they are designed to be) an attacker can amplify an attack on one bus to gain access to components on another. Consequently, the result is that compromising any ECU with access to some CAN bus on our vehicle (e.g., the media player) is sufficient to compromise the entire vehicle.

Combining these ECU control and bridging components, we constructed a general "payload" that we attempted to deliver in our subsequent experiments with the external attack surface.[5] To be clear, **for every vulnerability we demonstrate, we are able to obtain complete control over the vehicle's systems**. We did not explore weaker attacks.

For each ECU we consider, our experimental approach was to extract its firmware and then explicitly reverse engineering its I/O code and data flow using disassembly, interactive logging and debugging tools where appropriate. In most cases, extracting the firmware was possible directly via the CAN bus (this was especially convenient because in most ECUs we encountered, the flash chips are not socketed and while we were able to desolder and read such chips directly, the process was quite painful).

Having the firmware in hand, we performed three basic types of analysis: raw code analysis, in situ observations, and interactive debugging with controlled inputs on the bench. In the first case, we identified the microprocessor (e.g., different components described in this paper use System on Chip (SoC) variants of the PowerPC, ARM, Super-H and other architectures) and used the industry-standard IDA Pro disassembler to map control flow and identify potential vulnerabilities, as well as debugging and logging options that could be enabled to aid in reverse engineering.[6] In situ observation with logging enabled allowed us to understand normal operation of the ECU and let us concentrate on potential vulnerabilities near commonly used code paths. Finally, ECUs were removed from the car and placed into a test harness on the bench from which we could carefully control all inputs and monitor outputs. In this environment, interactive debuggers were used to examine memory and single step through vulnerable code under repeatable conditions. For one such device, the Super-H-based media player, we resorted to writing our own native debugger and exported a control and output interface through an unused serial UART interface we "broke out" off the circuit board.

In general, we made use of any native debugging I/O we could identify. For example, like the media player, the telematics unit exposed an unused UART that we tapped to monitor internal debugging messages as we interactively probed its I/O channels. In other cases, we

[5] In this work we experimented with two equivalent vehicles to ensure that our results were not tied to artifacts of a particular vehicle instance.

[6] IDA Pro does not support embedded architectures as well as x86 and consequently we needed to modify IDA Pro to correctly parse the full instruction set and object format of the target system. In one particular case (for the TPMS processor) IDA Pro did not provide any native support and we were forced to write a complete architecture module in order to use the tool.

selectively rewrote ECU memory (via the CAN bus or by exploiting software vulnerabilities) or rewrote portions of the flash chips using the manufacturer-standard ECU programming tools. For the telematics unit, we wrote a new character driver that exported a command shell to its Unix-like operating system directly over the OBD-II port to enable interactive debugging in a live vehicle. In the end, our experience was that although the ECU environment was somewhat more challenging than that of desktop operating systems, it was surmountable with dedicated effort.

4.2 Indirect physical channels

We consider two distinct indirect physical vectors in detail: the media player (via the CD player) and service access to the OBD-II port. We describe each in turn along with examples of when an adversary might be able to deliver malicious input.

Media player. The media player in our car is fairly typical, receiving a variety of wireless broadcast signals, including analog AM and FM as well as digital signals via FM sub-carriers (RDS, called RBDS in the U.S.) and satellite radio. The media player also accepts standard compact discs (via physical insertion) and decodes audio encoded in a number of formats including raw Red Book audio as well as MP3 and WMA files encoded on an ISO 9660 filesystem.

The media player unit itself is manufactured by a major supplier of entertainment systems, both stock units directly targeted for automobile manufacturers as well as branded systems sold via the aftermarket. Software running on the CPU handles audio parsing and playback requests, UI functions, and directly handles connections to the CAN bus.

We found two vulnerabilities. First, we identified a latent update capability in the media player that will automatically recognize an ISO 9660-formatted CD with a particularly named file, present the user with a cryptic message and, if the user does not press the appropriate button, will then reflash the unit with the data contained therein.[7] Second, knowing that the media player can parse complex files, we examined the firmware for input vulnerabilities that would allow us to construct a file that, if played, gives us the ability to execute arbitrary code.

For the latter, we reverse-engineered large parts of the media player firmware, identifying the file system code as well as the MP3 and WMA parsers. In doing so, we documented that one of the file read functions makes strong assumptions about input length *and* moreover that there is a path through the WMA parser (for handling an undocumented aspect of the file format) that allows arbitrary length reads to be specified; together these allow a buffer overflow.

This particular vulnerability is not trivial to exploit. The buffer that is overflowed is not on the stack but in a BSS segment, without clear control data variables to hijack. Moreover, immediately after the buffer are several dynamic state variables whose values are continually checked and crash the system when overwritten arbitrarily.

To overcome these and other obstacles, we developed a native in-system debugger that communicates over an unused serial port we identified on the media player. This debugger lets us dump and alter memory, set breakpoints, and catch exceptions. Using this debugger we were able to find several nearby dynamic function pointers to overwrite as well as appropriate contents for the intervening state variables.

We modified a WMA audio file such that, when burned onto a CD, plays perfectly on a PC but sends arbitrary CAN packets of our choosing when played by our car's media player. This functionality adds only a small space overhead to the WMA file. One can easily imagine many scenarios where such an audio file might find its way into a user's media collection, such as being spread through peer-to-peer networks.

OBD-II. The OBD-II port can access all CAN buses in the vehicle. This is standard functionality because the OBD-II port is the principal means by which service technicians diagnose and update individual ECUs in a vehicle. This process is intermediated by hardware tools (sold both by automobile manufacturers and third parties) that plug into the OBD-II port and can then be used to upgrade ECUs' firmware or to perform a myriad of diagnostic tasks such as checking the diagnostic trouble codes (DTCs).

Since 2004, the Environmental Protection Agency has mandated that all new cars in the U.S. support the SAE J2534 "PassThru" standard — a Windows API that provides a standard, programmatic interface to communicate with a car's internal buses. This is typically implemented as a Windows DLL that communicates over a wired or wireless network with the reprogramming/diagnostic tool (hereafter we refer to the latter simply as "the PassThru device"). The PassThru device itself plugs into the OBD-II port in the car and from that vantage point can communicate on the vehicle's internal networks under the direction of software commands sent via the J2534 API. In this way, applications developed independently of the particular PassThru device can be used for reprogramming or diagnostics.

We studied the most commonly used PassThru device for our car, manufactured by a well-known automotive electronics supplier on an OEM basis (the same device can be used for all current makes and models from

[7] This is not the standard method that the manufacturer uses to update the media player software and thus we believe this is likely a vestigial capability in the supplier's code base.

the same automobile manufacturer). The device itself is roughly the size of a paperback book and consists of a popular SoC microprocessor running a variant of Linux as well as multiple network interfaces, including USB and WiFi — and a connector for plugging into the car's OBD-II port.[8] We discovered two classes of vulnerabilities with this device. First, we find that an attacker on the same WiFi network as the PassThru device can easily connect to it and, if the PassThru device is also connected to a car, obtain control over the car's reprogramming. Second, we find it possible to compromise the PassThru device itself, implant malicious code, and thereby affect a far greater number of vehicles. To be clear, these are vulnerabilities in the PassThru device itself, not the Windows software which normally communicates with it. We experimentally evaluated both vulnerability classes and elaborate on our analyses below.

After booting up, the device periodically advertises its presence by sending a UDP multicast packet on each network to which it is connected, communicating both its IP address and a TCP port for receiving client requests. Client applications using the PassThru DLL connect to the advertised port and can then configure the PassThru device or command it to begin communicating with the vehicle. Communication between the client application and the PassThru device is unauthenticated and thus depends exclusively on external network security for any access control. Indeed, in its recommended mode of deployment, any PassThru device should be directly accessible by any dealership computer. A limitation is that only a single application can communicate with a given PassThru device at a time, and thus the attacker must wait for the device to be connected but not in use.

The PassThru device exports a proprietary, unauthenticated API for configuring its network state (e.g., for setting with which WiFi SSID it should associate). We identified input validation bugs in the implementation of this protocol that allow an attacker to run arbitrary Bourne Shell commands via shell-injection, thus compromising the unit. The underlying Linux distribution includes programs such as `telnetd`, `ftp`, and `nc` so, having gained entry to the device via shell injection, it is trivial for the attacker to open access for inbound telnet connections (exacerbated by a poor choice of root password) and then transfer additional data or code as necessary.

To evaluate the utility of this vulnerability and make it concrete, we built a program that combines all of these steps. It contacts any PassThru devices being advertised (e.g., via their WiFi connectivity or if connected directly via Ethernet), exploits them via shell injection, and installs a malicious binary (modifying startup scripts so it is always enabled). The malicious binary will send pre-programmed messages over the CAN bus whenever a technician connects the PassThru device to a car. These CAN packets install malware onto the car's telematics unit. This malware waits for an environmental trigger (e.g., specific date and time) before performing some action. Figure 2 gives a pictorial overview of this attack.

[8]The manufacturer's dealership guidelines recommend the use of the WiFi interface, thereby supporting an easier tetherless mode of use, and suggest the use of link-layer protection such as WEP (or, in the latest release of the device, WPA2) to prevent outside access.

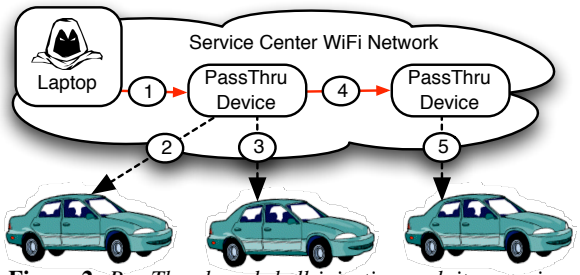

Figure 2: *PassThru-based shell-injection exploit scenario.* The adversary gains access to the service center network (e.g., by compromising an employee laptop), then (1) compromises any PassThru devices on the network, each of which compromise any cars they are used to service (2 and 3), installing Trojan horses to be activated based on some environmental trigger. The PassThru device also (4) spreads virally to other PassThru devices (e.g., if a device is loaned to other shops) which can repeat the same process (5).

To summarize, an attacker who can connect to a dealership's wireless network (e.g., via social engineering or a worm/virus à la Stuxnet [7]) is able to subvert any active PassThru devices that will in turn compromise any vehicles to which they connect. Moreover, the PassThru device is sufficiently general to mount the attack *itself*. To demonstrate this, we have modified our program, turning it into a worm that actively seeks out and spreads to other PassThru devices in range. This attack does not require interactivity with the attacker and can be fully automated.

4.3 Short-range wireless channels: Bluetooth

We now turn to short-range wireless channels and focus on one in particular: Bluetooth. Like many modern cars, ours has built-in Bluetooth capabilities which allow the occupants' cell phones to connect to the car (e.g., to enable hands-free calling). These Bluetooth capabilities are built into our car's telematics unit.

Through reverse engineering, we gained access to the telematics ECU's Unix-like operating system and identified the particular program responsible for handling Bluetooth functionality. By analyzing the program's symbols we established that it contains a copy of a popular embedded implementation of the Bluetooth protocol stack and a sample hands-free application. However, the interface to this program and the rest of the telematics system appear to be custom-built. It is in this custom interface code that we found evidence of likely

vulnerabilities. Specifically, we observed over 20 calls to `strcpy`, none of which were clearly safe. We investigated the first such instance in depth and discovered an easily exploitable unchecked `strcpy` to the stack when handling a Bluetooth configuration command.[9] Thus, any paired Bluetooth device can exploit this vulnerability to execute arbitrary code on the telematics unit.

As with our indirect physical channel investigations, we establish the utility of this vulnerability by making it concrete. We explore two practical methods for exploiting this attack and in doing so unearth two sub-classes of the short-range wireless attack vector: *indirect* short-range wireless attacks and *direct* short-range wireless attacks.

Indirect short-range wireless attacks. The vulnerability we identified requires the attacker to have a *paired* Bluetooth device. It may be challenging for an attacker to pair her own device with the car's Bluetooth system — a challenge we consider in the direct short-range wireless attacks discussion below. However, the car's Bluetooth subsystem was explicitly designed to support hands-free calling and thus may naturally be paired with one or more smartphones. We conjecture that if an attacker can independently compromise one of those smartphones, then the attacker can leverage the smartphone as a stepping-stone for compromising the car's telematics unit, and thus all the critical ECUs on the car.

To assess this attack vector we implemented a simple Trojan Horse application on the HTC Dream (G1) phone running Android 2.1. The application appears to be innocuous but under the hood monitors for new Bluetooth connections, checks to see if the other party is a telematics unit (our unit identifies itself by the car manufacturer name), and if so sends our attack payload. While we have not attempted to upload our code to the Android Market, there is evidence that other Trojan applications have been successfully uploaded [25]. Additionally, there are known exploits that can compromise Android and iPhone devices that visit malicious Web sites. Thus our assessment suggests that smartphones can be a viable path for exploiting a car's short-range wireless Bluetooth vulnerabilities.

Direct short-range wireless attacks. We next assess whether an attacker can remotely exploit the Bluetooth vulnerability without access to a paired device. Our experimental analyses found that a determined attacker can do so, albeit in exchange for a significant effort in development time and an extended period of proximity to the vehicle.

There are two steps precipitating a successful attack. First, the attacker must learn the car's Bluetooth MAC address. Second, the attacker must surreptitiously pair his or her own device with the car. Experimentally, we find that we can use the open source Bluesniff [23] package and a USRP-based software radio to sniff our car's Bluetooth MAC address when the car is started in the presence of a previously paired device (e.g., when the driver turns on the car while carrying her cell phone). We were also able to discover the car's Bluetooth MAC address by sniffing the Bluetooth traffic generated when one of the devices, which has previously been paired to a car, has its Bluetooth unit enabled, regardless of the presence of the car — all of the devices we experimented with scanned for paired devices upon Bluetooth initialization.

Given the MAC address, the other requirement for pairing is possessing a shared secret (the PIN). Under normal use, if the driver wishes to pair a new device, she puts the car into pairing mode via a well-documented user interface, and, in turn, the car provides a random PIN (regenerated each time the car starts or when the driver initiates the normal pairing mode) which is then shown on the dashboard and must then be manually entered into the phone. However, we have discovered that our car's Bluetooth unit will respond to pairing requests even without any user interaction. Using a simple laptop to issue pairing requests, we are thus able to brute force this PIN at a rate of eight to nine PINs per minute, for an average of approximately 10 hours per car; this rate is limited entirely by the response time of the vehicle's Bluetooth stack. We conducted three empirical trials against our car (resetting the car each time to ensure that a new PIN was generated) and found that we could pair with the car after approximately 13.5, 12.5, and 0.25 hours, respectively. The pairing process does not require any driver intervention and will happen completely obliviously to any person in the car.[10] While this attack is time consuming and requires the car(s) under attack to be running, it is also parallelizable, e.g., an attacker could sniff the MAC addresses of all cars started in a parking garage at the end of a day (assuming the cars are pre-paired with at least one Bluetooth device). If a thousand such cars leave the parking garage in a day, then we expect to be able to brute force the PIN for at least one car within a minute.

After completing this pairing, the attacker can inject on the paired channel an exploit like the one we developed and thus compromise the vehicle.

4.4 Long-range wireless channels: Cellular

Finally, we consider long-range wireless channels and, in particular, focus on the cellular capabilities built into our car's telematics unit. Like many modern cars, our car's cellular capabilities facilitate a variety of safety

[9] Because the size of the available buffer is small, our exploit simply creates a new shell on the telematics unit from which it downloads and executes more complex code from the Internet via the unit's built-in 3G data capabilities.

[10] As an artifact of how this "blind" pairing works, the paired device does not appear on the driver's list of paired devices and cannot be unpaired manually.

and convenience features (e.g., the car can automatically call for help if it detects a crash). However, long-range communications channels also offer an obvious target for potential attackers, which we explore here. In this section, we describe how these channels operate, how they were reverse engineered and demonstrate that a combination of software flaws conspire to allow a completely remote compromise via the cellular voice channel. We focus on adversarial actions that leverage the existing cellular infrastructure, not ones that involve the use of adversarially-controlled infrastructure; e.g., we do not consider man-in-the-middle attacks.

Telematics connectivity. For wide-area connectivity, our telematics unit is equipped with a cell phone interface (supporting voice, SMS and 3G data). While the unit uses its 3G data channel for a variety of Internet-based functions (e.g., navigation and location-based services), it relies on the voice channel for critical telematics functions (e.g., crash notification) because this medium can provide connectivity over the widest possible service area (i.e., including areas where 3G service is not yet available). To synthesize a digital channel in this environment, the manufacturer uses Airbiquity's aqLink software modem to covert between analog waveforms and digital bits. This use of the voice channel in general, and the aqLink software in particular, is common to virtually all popular North American telematics offerings today.

In our vehicle, Airbiquity's software is used to create a reliable data connection between the car's telematics unit and a remote *Telematics Call Center* (*TCC*) operated by the manufacturer. In particular, the telematics unit incorporates the aqLink code in its *Gateway* program which controls *both* voice and data cellular communication. Since a single cellular channel is used for both voice and data, a simple, in-band, tone-based signaling protocol is used to switch the call into data mode. The in-cabin audio is muted when data is transmitted, although a tell-tale light and audio announcement is used to indicate that a call is in progress. For pure data calls (e.g., telemetry and remote diagnostics), the unit employs a so-called "stealth" mode which does not provide any indication that a call is in progress.

Reverse engineering the aqLink protocol. Reverse engineering the aqLink protocol was among the most demanding parts of our effort, in particular because it demanded signal processing skills not part of the typical reverse engineering repertoire. For pedagogical reasons, we briefly highlight the process of our investigation.

We first identified an in-band tone used to initiate "data mode." Having switched to data mode, aqLink provides a proprietary modulation scheme for encoding bits. By calling our car's telematics unit (the phone number is available via caller ID), initiating data mode with a tone generator and recording the audio signal that resulted, we established that the center frequency was roughly 700 Hz and that the signal was consistent with a 400 bps frequency-shift keying (FSK) signal.

We then used `LD_PRELOAD` on the telematics unit to interpose on the raw audio samples as they left the software modem. Using this improved signal source, we hunted for known values contained in the signal (e.g., unique identifiers stamped on the unit). We did so by encoding these values as binary waveforms at hypothesized bitrates and cross-correlating them to the demodulated signal until we were able to establish the correct parameters for demodulating digital bits from the raw analog signal.

From individual bits, we then focused on packet structure. We were lucky to discover a debugging flag in the telematics software that would produce a binary log of all packet payloads transmitted or received, providing ground truth. Comparing this with the bitstream data, we discovered the details of the framing protocol (e.g., the use of half-width bits in the synchronization header) and were able to infer that data is sent in packets of up to 1024-bytes, divided into 22-byte frames which are divided into two 11-byte segments. We inferred that a CRC and ECC were both used to tolerate noise. Searching the disassembled code for known CRC constants quickly led us to determine the correct CRC to use, and the ECC code was identified in a similar fashion. For reverse-engineering the header contents, we interposed on the `aqSend` call (used to transmit messages), which allowed us to send arbitrary multi-frame packets and consequently infer the sequence number, multi-frame identifier, start of packet bit, ACK frame structure, etc. We omit the many other details due to space constraints.

Given our derived protocol specification, we then implemented an aqLink-compatible software modem in C using a laptop with an Intel ICH3-based modem exposed as an ALSA sound device under Linux. We verified the modulation and formatting of our packet stream using the debugging log described earlier.

Finally, layered on top of the aqLink modem is the telematics unit's own proprietary command protocol that allows the TCC to retrieve information about the state of the car as well as to remotely actuate car functions. Once the Gateway program decodes a frame and identifies it as a command message, the data is then passed (via an RPC-like protocol) to another telematics unit program which is responsible for supervising overall telematics activities and implementing the command protocol (henceforth, the *Command* program). We reverse-engineered enough of the Gateway and Command programs to identify a candidate vulnerability, which we describe below.

Vulnerabilities in the Gateway. As mentioned earlier, the aqLink code explicitly supports packet sizes up to 1024 bytes. However, the custom code that glues aqLink to the Command program assumes that packets will never

exceed 100 bytes or so (presumably since well-formatted command messages are always smaller). This leads to another stack-based buffer overflow vulnerability that we verified is exploitable. Interestingly, because this attack takes place at the lowest level of the protocol stack, it completely bypasses the higher-level authentication checks implemented by the Command program (since these checks themselves depend on being able to send packets).

There is one key gap preventing this exploit from working in practice. Namely, the buffer overflow we chose to focus on requires sending over 300 bytes to the Gateway program. Since the aqLink protocol has a maximum effective throughput of about 21 bytes a second, in the best case, the attack requires about 14 seconds to transmit. However, upon receiving a call, the Command program sends the caller an authentication request and, serendipitously, it requires a response within 12 seconds or the connection is effectively terminated. Thus, we simply cannot send data fast enough over an unauthenticated link to overflow the vulnerable buffer.

While we identified other candidate buffer overflows of slightly shorter length, we decided instead to focus on the authentication problem directly.

Vulnerabilities in authentication. When a call is placed to the car and data mode is initiated, the first command message sent by the vehicle is a random, three byte authentication challenge packet and the Command program authentication timer is started. In normal operation, the TCC hashes the challenge along with a 64-bit pre-shared key to generate a response to the challenge. When waiting for an authentication response, the Command program will not "accept" any other packet (this does not prevent our buffer overflow, but does prevent sending other command messages). If an incorrect authentication response is received, or a response is not received within the prescribed time limit, the Command program will send an error packet. When this packet is acknowledged, the unit hangs up (and it is not possible to send any additional data until the error packet is acknowledged).

After several failed attempts to derive the shared key, we examined code that generates authentication challenges and evaluates responses. Both contained errors that together were sufficient to construct a vulnerability.

First, we noted that the "random" challenge implementation is flawed. In most situations, this nonce is static and identical on the two cars we tested. The key flaw is that the random number generator is re-initialized whenever the telematics unit starts — such as when a call comes in after the car has been off — and it is seeded each time with the same constant. Therefore, multiple calls to a car while it is off result in the same expected response. Consequently, an attacker able to observe a response packet (e.g., via sniffing the cellular link during a TCC-initiated call) will be able to replay that response in the future.

The code parsing authentication *responses* has an even more egregious bug that permits circumvention without observing a correct response. In particular, there is a flaw such that for certain challenges (roughly one out of every 256), carefully formatted but incorrect responses will be interpreted as valid. If the random number generation is not re-initialized (e.g., if the car is on when repeatedly called) then the challenge will change each time and 1 out of 256 trials will have the desired structure. Thus, after an average of 128 calls the authentication test can be bypassed, and we are able to transmit the exploit (again, without any indication to the driver). This attack is more challenging to accomplish when the car is turned off because the telematics unit can shut down when a call ends (hence re-initializing the random number generator) before a second call can reach it.

To summarize, we identified several vulnerabilities in how our telematics unit uses the aqLink code that, together, allow a remote exploit. Specifically, there is a discrepancy between the set of packet sizes supported by the aqLink software and the buffer allocated by the telematics client code. However, to exploit this vulnerability requires first authenticating in order to set the call timeout value long enough to deliver a sufficiently long payload. This is possible due to a logic flaw in the unit's authentication system that allows an attacker to blindly satisfy the authentication challenge after approximately 128 calls.

Concrete realization. We demonstrate and evaluate our attack in two concrete forms. First, we implemented an end-to-end attack in which a laptop running our custom aqLink-compatible software modem calls our car repeatedly until it authenticates, changes the timeout from 12 seconds to 60 seconds, and then re-calls our car and exploits the buffer overflow vulnerability we uncovered. The exploit then forces the telematics unit to download and execute additional payload code from the Internet using the IP-addressable 3G data capability.

We also found that the entire attack can be implemented in a completely blind fashion — without any capacity to listen to the car's responses. Demonstrating this, we encoded an audio file with the modulated post-authentication exploit payload and loaded that file onto an iPod. By manually dialing our car on an office phone and then playing this "song" into the phone's microphone, we are able to achieve the same results and compromise the car.

5 Remote Exploit Control

Thus far we have described the external attack surface of an automobile and demonstrated the presence of vulnerabilities in a range of different external channels. An adversary could use such means to compromise a vehicle's systems and install code that takes action immediately (e.g., unlocking doors) or in response to

some environmental trigger (e.g., the time of day, speed, or location as exported via the onboard GPS).

However, the presence of wireless channels in the modern vehicle qualitatively changes the range of options available to the adversary, allowing actions to be remotely triggered on demand, synchronized across multiple vehicles, or interactively controlled. Further, two-way channels permit both remote monitoring and data exfiltration. In this section, we broadly evaluate the potential for such post-compromise control, characterize these capabilities, and evaluate the capabilities via prototype implementations for TPMS, Bluetooth, FM RDS and Cellular channels. Our prototype attack code is delivered by exploiting one of the previously described vulnerabilities (indeed, any exploit would work). Table 2 summarizes these results, again with our assessment of the effort required to discover and implement the capability.

TPMS. We constructed two versions of a TPMS-based triggering channel. One installs code on another ECU (the telematics ECU in our case, although any ECU would do) that monitors tire pressure signals as the TPMS ECU broadcasts them over the CAN bus. The presence of a particular tire pressure reading then triggers the payload; the trigger tire pressure value is not expected to be found in the wild but must instead be adversarially transmitted over the air. For our second example, the attack reflashes the TPMS ECU via CAN and installs code onto it that will detect specific wireless trigger packets and, if detected, will send pre-programmed CAN packets directly over the car's internal network. Both attacks required a custom TPMS packet generator (described below). The latter attack also required significant reverse engineering efforts (e.g., we had to write a custom IDA Pro module for disassembling the firmware, and we were highly memory constrained, so that the resulting attack firmware — hand-written object code — needed to re-use code space originally allocated for CRC verification, the removal of which did not impair the normal TPMS functionality).

To experimentally verify these triggers, we reverse-engineered the 315 MHz TPMS modulation and framing protocol (far simpler than the aqLink modem) and then implemented a USRP software radio module that generates the appropriate wireless signals to activate the triggers.

Bluetooth. We modified the Bluetooth exploit code on the telematics ECU to pair, post compromise, with a special MAC address used by the adversary and accept her commands (either triggering existing functionality or receiving new functionality). We did not explore exfiltrating data via the two-way Bluetooth channel, but we see no reason why it would not be possible.

FM RDS. Using the CD-based firmware update attack we developed earlier, we reflashed the media player ECU to send a pre-determined set of CAN packets (our payload) when a particular "Program Service Name" message arrives over the FM RDS channel. We experimentally verified this with a low-power FM transmitter driven by a Pira32 RDS encoder; an attacker could communicate over much longer ranges using higher power. Table 2 lists the cost for this attack as medium given the complexity of programming/debugging in the media player execution environment (we bricked numerous CD players before finalizing our implementation and testing on our car).

Cellular. We modified our telematics exploit payload to download and run a small (400 lines of C code) IRC client post-compromise. The IRC client uses the vehicle's high bandwidth 3G data channel to connect to an IRC server of our choosing, self-identifies, and then listens for commands. Subsequently, any commands sent to this IRC server (from any Internet connected host) are in turn transmitted to the vehicle, parsed by the IRC client, and then transmitted as CAN packets over the appropriate bus. We further provided functionality to use this channel in both a broadcast mode (where all vehicles subscribed to the channel respond to the commands) or selectively (where commands are only accepted by the particular vehicle specified in the command). For the former, we experimentally verified this by compromising two cars (located over 1,000 miles apart), having them both join the IRC channel, and then both simultaneously respond to a single command (for safety, the command we sent simply made the audio systems on both cars chime). Finally, the high-bandwidth nature (up to 1 Mbps at times) of this channel makes it easy to exfiltrate data. (No special software is needed since `ftp` is provided on the host platform.) To make this concrete we modified our attack code for two demonstrations: one that periodically "tweets" the GPS location of our vehicle and another that records cabin audio conversations and sends the recorded data to our servers over the Internet.

6 Threat Assessment

Thus far we have considered threats primarily at a technical level. Previously, we have shown that gaining access to a car's internal network provides sufficient *means* for compromising all of its systems (including lights, brakes, and engine) [14]. In this paper, we have further demonstrated that an adversary has a practical *opportunity* to effect this compromise (i.e., via a range of external communications channels) without having physical access to the vehicle. However, real threats ultimately have some *motive* as well: a more concrete goal that is achieved by exploiting the capability to attack.

This leaves unanswered the crucial question: Just how serious are the threats? Obviously, there are no clear ways to predict such things, especially in the absence of any known attacks in the wild. However, we can reason about how the capabilities we have identified can be combined in service to known goals. While one

Channel	Range	Implemented Control / Trigger	Exfiltration	Cost
TPMS (tire pressure)	Short	Predefined tire pressure sequences causes telematics unit to send CAN packets	No	Low-Medium
TPMS (tire pressure)	Short	TPMS trigger causes TPMS receiver to send CAN packets	No	Medium
Bluetooth	Short	Presence of trigger MAC addresses allows remote control	Yes*	Low
FM radio (RDS channel)	Long	FM RDS trigger causes radio to send CAN packets	No	Medium
Cellular	Global	IRC command-and-control (botnet) channel allows broadcast and single-vehicle control	Yes	Low

Table 2: *Implemented control and trigger channels.* The Cost column captures the approximate effort to develop this post-compromise control capability. The Exfiltration column indicates whether this channel can also be used to exfiltrate data. For (∗), we did not experimentally verify data exfiltration over Bluetooth.

can easily envision hypothetical "cyber war" or terrorist scenarios (e.g., infect large numbers of cars en masse via war dialing or a popular audio file and then, later, trigger them to simultaneously disengage the brakes when driving at high speed), our lack of experience with such concerns means such threats are highly speculative.

Instead, to gauge whether these threats create practical risks, we consider (briefly) how the raw capabilities we have identified might affect two scenarios closer to our experience: financially motivated theft and third-party surveillance.

Theft. Using any of our implemented exploit capabilities (CD, PassThru, Bluetooth, and cellular), it is simple to command a car to unlock its doors on demand, thus enabling theft. However, a more visionary car thief might realize that blind, remote compromise can be used to change both scale and, ultimately, business model. For example, instead of attacking a *particular* target car, the thief might instead try to compromise as many cars as possible (e.g., by war dialing). As part of this compromise, he might command each car to contact a central server and report back its GPS coordinates and Vehicle Identification Number (VIN). The IRC network described in Section 5 provides just this capability. The VIN in turn encodes the year, make and model of each car and hence its value. Putting these capabilities together, a car thief could "sift" through the set of cars, identify the valuable ones, find their location (and perhaps how long they have been parked) and, upon visiting a target of interest *then* issue commands to unlock the doors and so on. An enterprising thief might stop stealing cars himself, and instead sell his capabilities as a "service" to other thieves ("I'm looking for late model BMWs or Audis within a half mile of 4th and Broadway. Do you have anything for me?") Careful readers may notice that this progression mirrors the evolution of desktop computer compromises: from individual attacks, to mass exploitation via worms and viruses, to third-party markets selling compromised hosts as a service.

While the scenario itself is today hypothetical, we have evaluated a complete attack whereby a thief remotely disables a car's security measures, allowing a unskilled accomplice to enter the car and drive it away. Our attack directs the car's compromised telematics unit to unlock the doors, start the engine, disengage the shift lock solenoid (which normally prevents the car from shifting out of park without the key present), and spoof packets used in the car's startup protocol (thereby bypassing the existing immobilizer anti-theft measures[11]). We have implemented this attack on our car. In our experiments the accomplice only drove the "stolen" car forward and backward because we did not want to break the steering column lock, though numerous online videos demonstrate how to do so using a screwdriver. (Other vehicles have the steering column lock under computer control.)

Surveillance. We have found that an attacker who has compromised our car's telematics unit can record data from the in-cabin microphone (normally reserved for hands-free calling) and exfiltrate that data over the connected IRC channel. Moreover, as said before, it is easy to capture the location of the car at all times and hence track where the driver goes. These capabilities, which we have experimentally evaluated, could prove useful to private investigators, corporate spies, paparazzi, and others seeking to eavesdrop on the private conversations within particular vehicles. Moreover, if the target vehicle is not known, the mass compromise techniques described in the theft scenario can also be brought to bear on this problem. For example, someone wishing to eavesdrop on Google executives might filter a set of compromised cars down to those that are both expensive and located in the Google parking lot at 10 a.m. The location of those same cars at 7 p.m. is likely to be the driver's residence, allowing the attacker to identify the driver (e.g., via commercial credit records). We suspect that one could identify promising targets for eavesdropping quite quickly in this manner.

7 Discussion and Synthesis

Our research provides us with new insights into the risks with modern automotive computing systems. We begin here with a discussion of concrete directions for increasing security. We then turn to our now broadly

[11] Past work on bypassing immobilizers required prior direct or indirect access to the car's keys, e.g., Bono et al. [2] and Francillon et al. [9].

informed reflections on why vulnerabilities exist today and the challenges in mitigating them.

7.1 Implementation fixes

Our concrete, near-term recommendations fall into two familiar categories: restrict access and improve code robustness. Given the high interconnectedness of car ECUs necessary for desired functionality, the solution is not to simply remove or harden individual components (e.g., the telematics unit) or create physically isolated subnetworks.

We were surprised at the extent to which the car's externally facing interfaces were open to unsolicited communications — thereby broadening the attack surface significantly. Indeed, very simple actions, such as not allowing Bluetooth pairing attempts without the driver's first manually placing the vehicle in pairing mode, would have undermined our ability to exploit the vulnerability in the underlying Bluetooth code. Similarly, we believe the cellular interface could be significantly hardened by using inbound calls only to "wake up" the car (i.e., never for data transfer) and having the car itself periodically dial out for requests while it is active. Finally, use of application-level authentication and encryption (e.g., via OpenSSL) in the PassThru device's proprietary configuration protocol would have prevented its code from being exploited as well.

However, rather than assume the attack surface will not be breached, the underlying code platform should be hardened as well. These include standard security engineering best-practices, such as not using unsafe functions like `strcpy`, diligent input validation, and checking function "contracts" at module boundaries. As an additional measure of protection against less-motivated adversaries, we recommend removing all debugging symbols and error strings from deployed ECU code.

We also encourage the use of simple anti-exploitation mitigations such as stack cookies and ASLR that can be easily implemented even for simple processors and can significantly increase the exploit burden for potential attackers. In the same vein, critical communications channels (e.g., Bluetooth and telematics) should have some amount of behavioral monitoring. The car should not allow arbitrary numbers of connection failures to go unanswered nor should outbound Internet connections to arbitrary destinations be allowed. In cases where ECUs communicate on multiple buses, they should only be allowed to be reflashed from the bus with the smallest external attack surface. This does not stop all attacks where one compromised ECU affects an ECU on a bus with a smaller attack surface, but it does make such attacks more difficult. Finally, a number of the exploits we developed were also facilitated by the services included in several units. For example, we made extensive use of `telnetd`, `ftp`, and `vi`, which were installed on the PassThru and telematics devices. There is no reason for these extraneous binaries to exist in shipping ECUs, and they should be removed before deployment, as they make it easier to exploit additional connectivity to the platform.

Finally, secure (authenticated and reliable) software updates must also be considered as part of automotive component design.

7.2 Vulnerability drivers

While the recommendations in Section 7.1 can significantly increase the security of modern cars against external attacks and post-compromise control, none of these ideas are new or innovative. Thus, perhaps the more interesting question is why they have not been applied in the automotive environment already. Our findings and subsequent interactions with the automotive industry have given us a unique vantage point for answering this question.

One clear reason is that automobiles have not yet been subjected to significant adversarial pressures. Traditionally automobiles have not been network-connected and thus manufacturers have not had to anticipate the actions of an external adversary; anyone who could get close enough to a car to modify its systems was also close enough to do significant damage through physical means. Our automotive systems now have broad connectivity; millions of cars on the road today can be directly addressed via cellular phones and via the Internet.

This is similar to the evolution of desktop personal computer security during the early 1990s. In the same way that connecting PCs to the Internet exposed extant vulnerabilities that previously could not conveniently be exploited, so too does increasing the connectivity of automotive systems. This analogy suggests that, even though automotive attacks do not take place today, there is cause to take their potential seriously. Indeed, much of our work is motivated by a desire that the automotive manufacturers should not repeat the mistakes of the PC industry — waiting for high profile attacks before making security a top priority [18, 19]. We believe many of the lessons learned in hardening desktop systems (such as those suggested earlier) can be quickly re-purposed for the embedded context.

However, our experimental vulnerability analyses also uncover an ecosystem for which high levels of assurance may be fundamentally challenging. Reflecting upon our discovered vulnerabilities, we noticed interesting similarities in where they occur. In particular, virtually all vulnerabilities emerged at the interface boundaries between code written by distinct organizations.

Consider for example the Airbiquity software modem, which appears to have been delivered as a completed component. We found vulnerabilities not in the software modem *itself* but rather in the "glue" code calling it and binding it to other telematics functions. It was here

that the caller did not appear to fully understand the assumptions made by the component being called.

We find this pattern repeatedly. The Bluetooth vulnerability arose from a similar misunderstanding between the callers of the Bluetooth protocol stack library and its implementers (again in glue code). The PassThru vulnerability arose in script-based glue code that tried to interface a proprietary configuration protocol with standard Linux configuration scripts. Even the media player firmware update vulnerability appears to have arisen because the manufacturer was unaware of the vestigial CD-based reflashing capability implemented in the code base.

While interface boundary problems are common in all kinds of software, we believe there are structural reasons that make them particularly likely in the automotive industry. In particular, the automotive industry has adopted an outsourcing approach to software that is quite similar to that used for mechanical components: supply a specification and contract for completed parts. Thus, for many components the manufacturer does not do the software development and is only responsible for integration. We have found, for example, that different model years of ECUs with effectively the same functionality used completely different source code bases because they were provided by different suppliers. Indeed, we have come to understand that frequently manufacturers do not have access to the source code for the ECUs they contract for (and suppliers are hesitant to provide such code since this represents their key intellectual property advantage over the manufacturer). Thus, while each supplier does unit testing (according to the specification) it is difficult for the manufacturer to evaluate security vulnerabilities that emerge at the integration stage. Traditional kinds of automated analysis and code reviews cannot be applied and assumptions not embodied in the specifications are difficult to unravel. Therefore, while this outsourcing process might have been appropriate for purely mechanical systems, it is no longer appropriate for digital systems that have the potential for remote compromise.

Developing security solutions compatible with the automotive ecosystem is challenging and we believe it will require more engagement between the computer security community and automotive manufacturers (in the same way that our community engages directly with the makers of PC software today).

8 Conclusions

A modern automobile is controlled by tens of distinct computers physically interconnected with each other via internal (wired) buses and thus exposed to one another. A non-trivial number of these components are also externally accessible via a variety of I/O interfaces. Previous research showed that an adversary can seriously impact the safety of a vehicle if he or she is capable of sending packets on the car's internal wired network [14], and numerous other papers have discussed potential security risks with future (wired and wireless) automobiles in the abstract or on the bench [10, 15, 24, 26, 27, 28]. To the best of our knowledge, however, we are the first to experimentally and systematically study the externally-facing attack surface of a car.

Our experimental analyses focus on a representative, moderately priced sedan. We iteratively refined an automotive threat model framework and implemented complete, end-to-end attacks along key points of this framework. For example, we can compromise the car's radio and upload custom firmware via a doctored CD, we can compromise the technicians' PassThru devices and thereby compromise any car subsequently connected to the PassThru device, and we can call our car's cellular phone number to obtain full control over the car's telematics unit over an arbitrary distance. Being able to compromise a car's ECU is, however, only half the story: The remaining concern is what an attacker is able to do with those capabilities. In fact, we show that a car's externally-facing I/O interfaces can be used post-compromise to remotely trigger or control arbitrary vehicular functions at a distance and to exfiltrate data such as vehicle location and cabin audio. Finally, we consider concrete, financially-motivated scenarios under which an attacker might leverage the capabilities we develop in this paper.

Our experimental results give us the unique opportunity to reflect on the security and privacy risks with modern automobiles. We synthesize concrete, pragmatic recommendations for future automotive security, as well as identify fundamental challenges. We disclosed our results to relevant industry and government stakeholders. While defending against known vulnerabilities does not imply the non-existence of other vulnerabilities, many of the specific vulnerabilities identified in this paper have or will soon be addressed.

Acknowledgments

We thank our shepherd Dan Wallach and the anonymous reviewers for their helpful comments, Ingolf Krueger for his guidance on understanding automotive architectures, Conrad Meyer for his help with Bluesniff, and Cheryl Hile and Melody Kadenko for their support on all aspects of the project. Portions of this work were supported by NSF grants CNS-0722000, CNS-0831532, CNS-0846065, CNS-0963695, and CNS-0963702; by the MURI program under AFOSR Grant No. FA9550-08-1-0352; by a CCC-CRA-NSF Computing Innovation Fellowship; by an NSF Graduate Research Fellowship; by a Marilyn Fries Endowed Regental Fellowship; and by an Alfred P. Sloan Research Fellowship.

References

[1] BBC. Hack attacks mounted on car control systems. *BBC News*, May 17, 2010. Online: http://www.bbc.co.uk/news/10119492.

[2] S. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled RFID device. In P. McDaniel, editor, *USENIX Security 2005*, pages 1–16. USENIX Association, July 2005.

[3] R. Boyle. Proof-of-concept CarShark software hacks car computers, shutting down brakes, engines, and more. *Popular Science*, May 14, 2010. Online: http://www.popsci.com/cars/article/2010-05/researchers-hack-car-computers-shutting-down-brakes-engine-and-more.

[4] CAMP Vehicle Safety Communications Consortium. Vehicle safety communications project task 3 final report, Mar. 2005. Online: http://www.intellidriveusa.org/documents/vehicle-safety.pdf.

[5] R. Charette. This car runs on code. Online: http://www.spectrum.ieee.org/feb09/7649, Feb. 2009.

[6] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the KeeLoq code hopping scheme. In D. Wagner, editor, *Crypto '08*, volume 5157 of *LNCS*, pages 203–20. Springer-Verlag, Aug. 2008.

[7] N. Falliere, L. O Murchu, and E. Chien. W32.Stuxnet dossier version 1.3, Nov. 2010. Online: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[8] FlexRay Consortium. FlexRay communications system protocol specification version 2.1 revision A, Dec. 2005. Online: http://www.flexray.com/index.php?pid=47.

[9] A. Francillon, B. Danev, and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In A. Perrig, editor, *NDSS 2011*. ISOC, Feb. 2011.

[10] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive CAN networks – practical examples and selected short-term countermeasures. In M. D. Harrison and M.-A. Sujan, editors, *SAFECOMP 2008*, volume 5219 of *LNCS*, pages 235–248. Springer-Verlag, Sept. 2008.

[11] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A practical attack on KeeLoq. In N. Smart, editor, *Eurocrypt '08*, volume 4965 of *LNCS*, pages 1–18. Springer-Verlag, Apr. 2008.

[12] ISO. *ISO 11898-1:2003 - Road vehicles – Controller area network*. International Organization for Standardization, 2003.

[13] F. Kargl, P. Papadimitratos, L. Buttyan, M. Müter, E. Schoch, B. Wiedersheim, T.-V. Thong, G. Calandriello, A. Held, A. Kung, and J.-P. Hubaux. Secure vehicular communication systems: implementation, performance, and research challenges. *IEEE Communications Magazine*, 46(11):110–118, 2008.

[14] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In D. Evans and G. Vigna, editors, *IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2010.

[15] U. E. Larson and D. K. Nilsson. Securing vehicles against cyber attacks. In A. Mili and A. Krings, editors, *CSIIRW '08*, pages 30:1–30:3. ACM Press, May 2008.

[16] J. Leyden. Boffins warn on car computer security risk. *The Register*, May 14, 2010. Online: http://www.theregister.co.uk/2010/05/14/car_security_risks/.

[17] P. Magney. iPod connections expected in more than half of U.S. car models in 2009. Online: http://www.isuppli.com/Automotive-Infotainment-and-Telematics/MarketWatch/Pages/iPod-Connections-Expected-in-More-than-Half-of-U-S-Car-Models-in-2009.aspx, Oct. 2008.

[18] J. Markoff. Stung by security flaws, Microsoft makes software safety a top goal. *The New York Times*, Jan. 2002.

[19] C. Mundie. Trustworthy computing. Online: http://download.microsoft.com/download/a/f/2/af22fd56-7f19-47aa-8167-4b1d73cd3c57/twc_mundie.doc, Oct. 2002.

[20] NPR. 'Rifle' sniffs out vulnerability in bluetooth devices. All Things Considered, Apr 13, 2005.

[21] M. Raya and J.-P. Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15(1):39–68, 2007.

[22] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In I. Goldberg, editor, *USENIX Security 2010*, pages 323–338. USENIX Association, Aug. 2010.

[23] D. Spill and A. Bittau. Bluesniff: Eve meets alice and bluetooth. In D. Boneh, T. Garfinkel, and D. Song, editors, *WOOT 2007*, pages 1–10. USENIX Association, 2007.

[24] P. R. Thorn and C. A. MacCarley. A spy under the hood: Controlling risk and automotive EDR. *Risk Management*, Feb. 2008.

[25] J. Vijayan. Update: Android gaming app hides Trojan, security vendors warn. *Computerworld*, Aug. 17, 2010. Online: http://www.computerworld.com/s/article/9180844/Update_Android_gaming_app_hides_Trojan_security_vendors_warn.

[26] M. Wolf, A. Weimerskirch, and C. Paar. Security in automotive bus systems. In C. Paar, editor, *ESCAR 2004*, Nov. 2004.

[27] M. Wolf, A. Weimerskirch, and T. Wollinger. State of the art: Embedding security in vehicles. *EURASIP Journal on Embedded Systems*, 2007.

[28] Y. Zhao. Telematics: safe and fun driving. *Intelligent Systems, IEEE*, 17(1):10–14, Jan./Feb. 2002.