Commonalities in Vehicle Vulnerabilities

Corey Thuen
Senior Security Consultant

IOActive

Abstract

With the Connected Car becoming commonplace in the market, vehicle cybersecurity grows more important by the year. At the forefront of this growing area of security research, IOActive has amassed real-world vulnerability data illustrating the general issues and potential solutions to the cybersecurity issues facing today's vehicles.

This paper explains the differences in testing methodologies, with recommendations on the most appropriate methods for testing connected vehicle systems. Detailed findings follow, including the impact, likelihood, overall risk, and remediation of vulnerabilities IOActive consultants have discovered over the course of thousands of testing hours. The paper concludes with a recommendation for an "ounce of prevention" that may ensure more secure vehicle systems in the future.

IOActive®

Contents

Introduction

This paper provides a metadata analysis of the multitude of private vehicle security assessments IOActive has conducted, also incorporating other publicly available research to provide an informative picture of the vulnerabilities that researchers are uncovering, the specific systems and attack vectors most prevalently affected, and the real-world significance of these vulnerabilities. This data is extremely useful for organizations considering cybersecurity strategy and planning.

Vehicle cybersecurity has become a focused and growing area of security research for IOActive. In 2013, the company conducted about 2,000 hours of combined research and services in the vehicle cybersecurity space, doubling to 4,000 in 2014 and spiking to 10,000 hours in 2015, and we anticipate that number continuing to grow going forward.

This experience puts IOActive's Vehicle Cybersecurity Division in a unique position to provide valuable insight into common struggles, failures, and solutions facing the automotive and related transportation and components industries. This research uses hard data taken from vulnerability assessments of real-world vehicle systems. We have conducted enough of these assessments to properly anonymize the sources of this information and extract the valuable "big-picture" aspects.

First, in order to make use of the data we must establish a foundation of cybersecurity terminology and comprehension. We will discuss threat modeling, attack vectors, and attacker methodologies in order to explain how we discovered these vulnerabilities. Second, we will cover categorization and walk through a vulnerability evaluation. Finally, we will look at the data itself and answer some key questions:

- What kind of vulnerabilities most commonly affect the Connected Car?
- What attack vectors are most commonly used to compromise a vehicle?
- What percentage of vulnerabilities would defense product XYZ mitigate?
- How do automotive and component manufacturers best manage their limited cybersecurity resources to maximize effectiveness?

Threat Modeling the Connected Car

Understanding the attack surfaces of the connected car is an important first step. This means noting the possible ways to attack a target, which might be the entire vehicle or just a component therein. A threat model does not focus on attack *methods*, but rather looks at possible attack *vectors*.

Holistic Model

For a holistic threat model of the Connected Car, we are interested in looking at how data can enter the vehicle, and what the potential impact to the vehicle is if a given vulnerability is exploited. These boundaries are where an attacker will focus their efforts, and therefore where defense efforts should be prioritized.

Data can enter vehicles in a variety of ways, including:

- Cellular Radio
- Bluetooth
- Wi-Fi
- Companion Apps
- V2V Radio (802.11p)
- OBDII
- Infotainment Media
- Zigbee Radio (e.g. TPMS)

Component Model

Threat modeling is also quite effective at the system or component level. For example, a threat model of an infotainment unit will include inputs from any media inserted, such as Bluetooth pairing with smartphones, physical media access via USB, and possibly cellular data communications. The threat model aids in organizing the overall development effort. For example, time spent searching an entire code base for buffer overflow vulnerabilities may not be as effective as focusing on code that interacts with higher-risk attack vectors, such as Wi-Fi.

Test Methodologies

Cybersecurity researchers will vary their testing methodologies depending on the component being analyzed, but most testing falls under two categories: black box and white box.

Black box testing is so named because the researcher is given no foreknowledge or insight into how a system operates. The researcher takes the role of an attacker who must evaluate the system, discover how it works, and attempt to find and exploit vulnerabilities. This usually involves dynamic testing utilizing protocol fuzzing, hardware analysis, chip desoldering, observation of serial bus lines, capturing firmware updates, and other methods.

White box (which is often actually grey box - somewhere in between true black box and white box) testing is so named because the researcher works with the product developer to evaluate the system. The developer will often provide code or a debug testing harness. The methods and “attacker mindset” approach used in white/grey box testing by IOActive are similar to black box testing, but may also include code analysis, protocol specifications, and design review.

In general, white box testing has provided the best ROI to clients and most of the useful data for this paper. This is because greater insight into a system makes for more meaningful bug reports and more accurate assessments of impact that can be better aligned to the specific business risks of the client.

Categorizing Vulnerabilities

To provide a meaningful, quantitative analysis of its findings, IOActive uses an impact-versus-likelihood approach to scoring. For each individual finding, the assessment team assigns two ratings, one for impact and another for likelihood; that is, the likelihood the given vulnerability will be exploited. Each vulnerability is then assigned a rating of Critical, High, Medium, Low, or Informational, with corresponding numeric

scores ranging from 5 (Critical) to 1 (Informational). This table explains each rating in terms of score, impact, and likelihood.

Rating (Score)	*Impact*	*Likelihood*
Critical (5)	Extreme impact to vehicle if exploited. Would receive media attention.	Vulnerability is almost certain to be exploited. Knowledge of the vulnerability and its exploitation are in the public domain.
High (4)	Major impact to vehicle if exploited. Could be a regulatory violation.	Vulnerability is relatively easy to detect and exploit by an attacker with little skill.
Medium (3)	Noticeable impact to vehicle if exploited.	An expert attacker could exploit the vulnerability without much difficulty.
Low (2)	Minor impact if exploited, or could be used in conjunction with other vulnerabilities to perform a more serious attack.	Exploiting the vulnerability would require considerable expertise and resources.
Informational (1)	Poor programming practice or poor design decision that may not represent an immediate risk on its own, but may have security implications if multiplied and/or combined with other vulnerabilities.	Vulnerability is not likely to be exploited on its own, but may be used to gain information for launching another attack.

Table 1 – Vulnerability rating system

IOActive assigns aggregate risk scores to identified vulnerabilities; specifically, the impact score multiplied by the likelihood score. For example, a vulnerability with high likelihood and low impact would have an aggregate risk score of eight (8); that is, four (4) for high likelihood multiplied by two (2) for low impact. The Aggregate Risk Score determines the finding's Overall Risk Level.

Overall Risk Level	*Aggregate Risk Score (Impact multiplied by Likelihood)*
Critical	**20–25**
High	**12–19**
Medium	**6–11**
Low	**2–5**
Informational	**1**

Table 2 - Overall Risk Levels and corresponding Aggregate Risk Scores

The Common Vulnerability Scoring System (CVSS) has proven effective for other organizations, but experience has taught us that a simpler and more relatable scoring

metric facilitates better understanding of overall vulnerability. CVSS does have scoring parameters that are useful for the type of meta-analysis performed for this paper, but the research goes beyond CVSS parameters to evaluate other aspects of a vulnerability.

In order to further categorize vulnerabilities in a meaningful way, IOActive collected additional data for each vulnerability finding. For instance, testers gathered data relating a vulnerability to a particular attack vector, a given methodology, or type (e.g., buffer overflow, authentication issue, etc.).

Example Categorization

A recent high-profile example[1] effectively illustrates how a vulnerability is evaluated. A vehicle manufacturer released a companion smartphone app to interface with the vehicle, enabling users to control climate settings and view fuel status, among a myriad of other capabilities. The application itself requested association with a user account, requiring the user to log in before using the app.

This example vulnerability shows that HTTPS requests made to the backend servers from the app did not contain any user authentication information. The application made a request to get battery status (example, fictitious URL):

GET https://host/BatteryStatus.php?VIN=JNFAAZE0U60XXXXX

This request did not contain any session parameters or use Authentication Headers. The server returns a JSON response. By changing the VIN, an attacker can gain access to information about any vehicle that sends data to the backend servers, without authentication.

Further, using the app to turn climate control on and off issues requests with similar parameters to the URLs:

```
/orchestration_example/ACRemoteRequest.php?XXX
/orchestration_example/ACRemoteOffRequest.php?XXX
```

We are looking at a vulnerability that is remote, unauthenticated, low-skill, capable of controlling or changing a process or system, and automatable.

[1] https://www.troyhunt.com/controlling-vehicle-features-of-nissan/

CATEGORY	VALUE
IMPACT	Critical (4.5). Media attention. API servers can affect all vehicles.
LIKELIHOOD	Critical (5). Almost certain to be exploited. Very low skill required. Easily discoverable.
ACCESS VECTOR	Network
ACCESS COMPLEXITY	Low
AUTHENTICATION	None
IMPACT: CONFIDENTIALITY	Complete
IMPACT: INTEGRITY	Partial
IMPACT: AVAILABILITY	None
LABELS	Telematics, Web API, App, Insecure By Design
EXPLOIT DELIVERY MEDIA	Web
TOOLS USED	burp
VULNERABILITY	Unauthenticated API
IMPACT: VEHICLE	Secondary system loss of control, multiple vehicles compromised
TESTING METHOD	Black Box
TEMPORARY REMEDIATION	Easy (0). Disable API immediately.
REMEDIATION: DISSEMINATION	Medium (3). Prod server update to remediate the issue but a new app required to use new server API.
REMEDIATION: COMPLEXITY	Medium (3). Proper security of web API requires authentication/session information and CSRF tokens (if a web interface exists).
REMEDIATION: TIMELINE	Medium (3)
EXPLOIT AUTOMATABILITY	Easy (5)

Table 3 – Vulnerability report table

Common Vulnerabilities and Analysis

With this understanding of how we find and classify vulnerabilities we can look at the raw data. We evaluate the data organized by:

- Type
- Severity
- Attack Vector
- Methodology
- Remediation Difficulty
- Vehicle Impact

Impact

Looking at the impact level across all vulnerabilities, we see that half are Critical or High impact. This means half of the vulnerabilities result in a compromise of components, communications, or data that causes complete or partial loss of control over the system. Business impact, such as regulatory violation fees or negative media attention, is also factored into the generic Impact score.

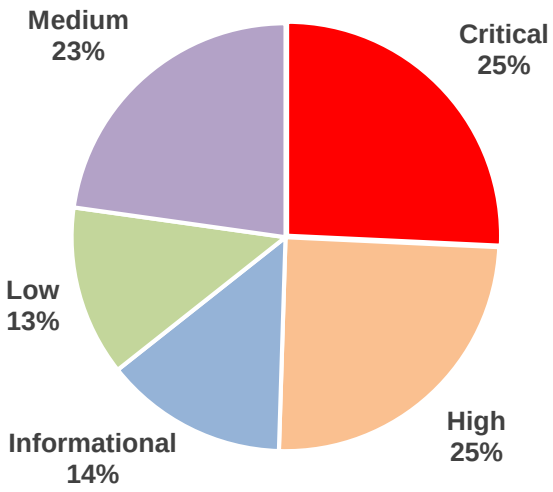

Figure 1 - Vulnerabilities by Impact

Likelihood

The second generic vulnerability category is Likelihood. Here we see the largest classification is Medium, meaning “an expert attacker could exploit the vulnerability without much difficulty.” The nature of embedded systems security causes a barrier to entry for attackers. The customized nature of each system makes developing attack tools difficult. However, in time, more vulnerabilities will shift toward the Critical value as more vehicle tools are released to the public and embedded security becomes more accessible to a would-be attacker.

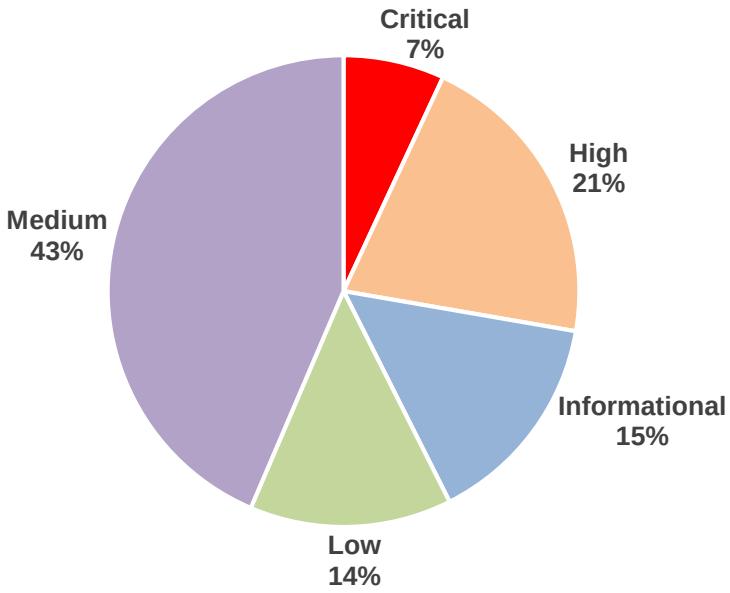

Figure 2 - Vulnerabilities by Likelihood

Overall Risk Level

Combining the previous two data segments gives us an Overall Risk Level, a general quick look at vulnerability severity. 22% of all vulnerabilities sit in the Critical range. These are the high-priority "hair on fire" vulnerabilities that are easily discovered and exploited and can cause major impacts to the system or component. These vulnerabilities should drive cybersecurity resource allocation and spending.

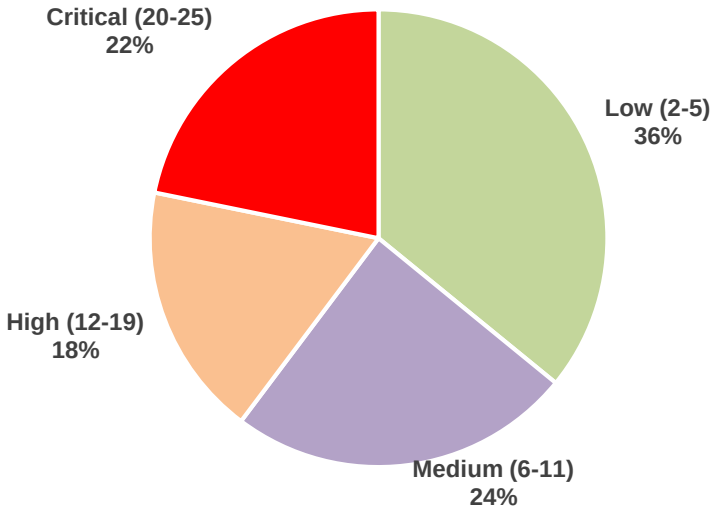

Figure 3 - Vulnerabilities by Overall Risk Level

Attack Vector

The Attack Vector is a useful data set to compare against the threat model for a given component or system. The attack surface and trust boundaries should be considered during the design phase of a system. Every decision that increases the attack surface

provides new opportunities for an attacker, and should be evaluated accordingly for risk.

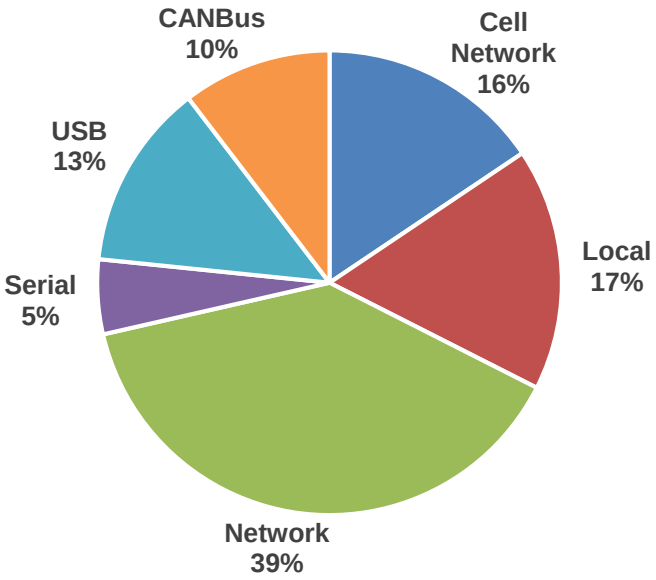

Figure 4 - Vulnerability Attack Vectors

Within our vulnerability set, 39% of vulnerabilities are related to the network. This is a general category that includes all network traffic, such as Ethernet or web. However, cellular network and CANBus vulnerabilities appear in separate categories, representing 16% and 10% (respectively) of the attack vectors for vulnerabilities found.

The local attack vector requires an attacker to be on the system and obtain privilege escalation. At first this may seem like an inconsequential or unlikely attack vector, but after considering the availability of app stores and third-party software modules, this attack vector is just as significant as network-based attacks.

USB and serial attack vectors require an attacker to be physically present at the system or otherwise chain an attack. For example, malware on a desktop system may deploy a payload to a USB storage device, which the unsuspecting user then delivers to the vehicle. These attack vectors generally coincide with a lower likelihood of vulnerability.

Top Eight Vulnerabilities

Vulnerabilities related to system design occupy four of the top eight vulnerability types. Vendor-introduced backdoors, incorrect utilization of the principle of least privilege, authentication systems requiring hardcoded credentials, and accidental information disclosure are all products of the system design process.

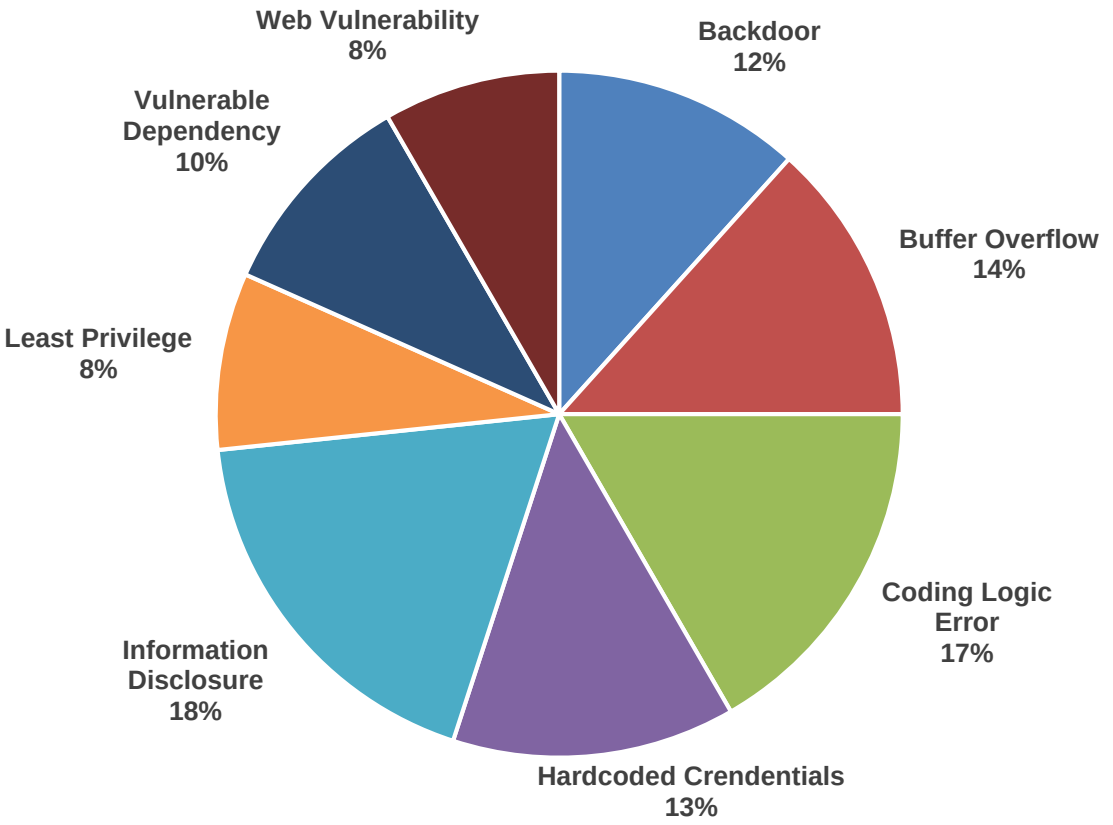

Figure 5 – Top Eight Vulnerabilities

Engineering problems are the root cause of three of the top eight vulnerability types. Coding logic errors, such as format strings, buffer overflows, and integer overflows, account for the most vulnerabilities, with buffer overflows the most common sub-type. Additionally, web vulnerability implementation problems fall under this category, which are generally High to Critical risk due to widely available attacker tools and knowledge base.

Problems in deployment mechanisms and testing cause vulnerabilities in the Vulnerable Dependency and Backdoor categories. The Vulnerable Dependency category stems from technical and cultural issues. It is extremely common for embedded systems to use outdated libraries simply because it is hard enough to get a system to work in the first place, let alone introduce changes that could cause further issues by updating a dependency. Further, updating embedded systems is non-trivial, so a discovered vulnerability in one of these dependencies is likely to be a fruitful vector for attackers for much longer than in the general realm of IT systems.

Backdoors, information disclosure, and hardcoded credentials also stem from issues in the deployment process. Production systems should not include developer debug interfaces or other information that enables attackers. Security-aware deployment procedures and testing are necessary to verify removal of these interfaces from production systems.

Critical Impact Remediation

To create this chart, we took all vulnerabilities with a Critical impact score and made a "remediation complexity" estimate to evaluate the difficulty of fixing a specific vulnerability. For example, patching code to remove a buffer overflow is relatively easy. Interestingly, an overwhelming majority of Critical impact vulnerabilities can be remediated with relatively simple fixes.

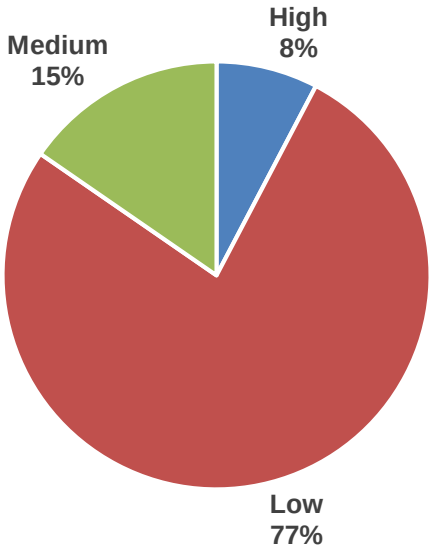

Figure 6 - Critical Vulnerability Remediation Difficulty

However, vulnerabilities stemming from design-level issues occupy the Medium and High remediation complexity categories. These can be extremely difficult, if not impossible, to remediate after the design phase, which results in a system that is "insecure by design" and can never really be remediated. Some design issues do fall in the Low category as a result of not following industry best practices that are widely published with how-to guides (e.g., the use of CSRF tokens to prevent web application CSRF attacks).

Testing Method

The Testing Method chart below is useful when considering how to proceed with a vulnerability assessment. It includes normalized data on Overall Risk Level for vulnerabilities, broken down by the testing method used when the vulnerabilities were discovered. IOActive generally conducts assessments in a black or white box format wherein the client will either share (white box) or not share (black box) data, code, or other information.

The usual motivation for a black box test is to evaluate what a "real-world attacker" would see or do, but in reality the assessment rarely plays out as intended. Real attackers are not limited in scope or timeline when attacking a system. During such an assessment, more time is required by researchers to discover and evaluate how to use a system and develop the harnesses or methodologies required for testing. This limits the time remaining for the testing itself.

Conversely, in a white box test, researchers have more information about the system, giving them three advantages:

1. Researchers can spend less time figuring out how a system works and more time discovering vulnerabilities.
2. Researchers are better able to evaluate the impact and likelihood levels for any vulnerabilities that are discovered.
3. Researchers can provide insight and assistance in areas of the system that may not be directly attackable but might be accessible in the future or by chaining another vulnerability. Generally, these insights fall into the Informational Overall Risk Level.

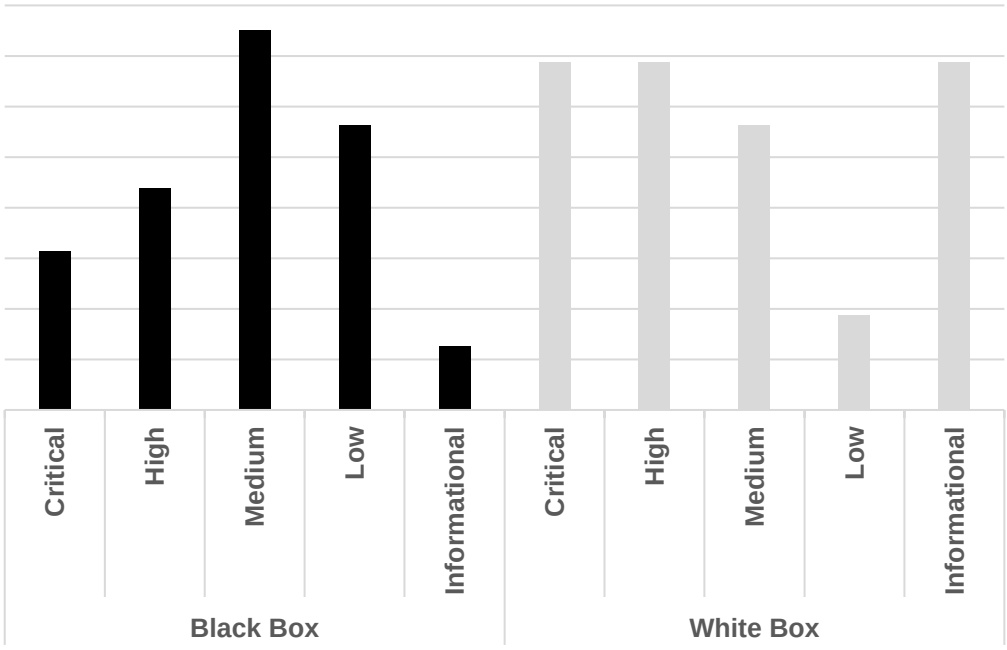

Figure 7 - Testing Method

In black box testing, most vulnerabilities sit at the Medium Overall Risk Level, and fewer vulnerabilities are discovered overall. Comparatively, white box testing uncovers fewer Medium and more Critical and Informational vulnerabilities. The additional information available in white box testing enables researchers to better assess the actual risk of a vulnerability. Further, researchers spend less time working to discover information about the system and how it operates and more time focusing on finding and evaluating vulnerabilities.

Grey box testing can be similar in methodology to black box, but benefits from the addition of information that enables testers to focus on critical attack paths or components, often resulting in the discovery of more Critical or High Overall Risk Level vulnerabilities. However, this also results in fewer Informational level vulnerabilities than a white box test, often limiting the additional insights, such as incorporating industry best practices, that white box testing can offer.

In summary: white box testing is the most effective at identifying high priority bugs and improving the development process.

Impact on Vehicle

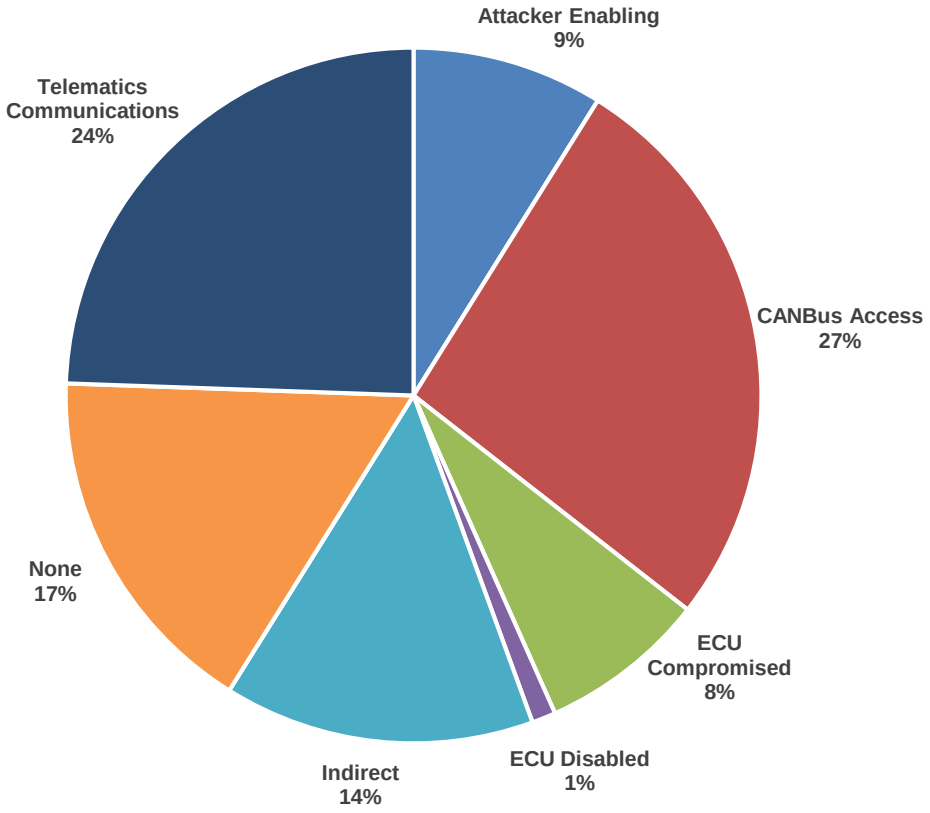

Figure 8 - Impact on Vehicle

An important point to consider when reviewing this vulnerability data is the actual impact on the vehicle or fleet. Starting from the benign, 17% of vulnerabilities evaluated had zero impact on the vehicle itself. These might be vulnerabilities in unrelated backend systems or other components that are tangentially connected to the vehicle system being evaluated.

Attacker-enabling impact (9%) encompasses those vulnerabilities that provide attackers with additional information or attack chains they can use to gain access or target another vulnerability. Similarly, indirect impact is a secondary consequence of an attack against a vehicle. For example, compromise of a telematics backend might allow an attacker to then communicate with the vehicle and gain an additional direct attack vector against the vehicle.

Vehicle telematics communications are effected in 24% of vulnerabilities. This might occur directly on the component, in transit over a cellular or other network, or on the backend systems that gather and utilize the data.

More significantly, 27% of vulnerabilities can be used to gain CANBus Access. This is important when considering the increasing body of public research showing that an attacker on the CANBus can control the vehicle[234].

Actual ECU control is gained in 8% of the vulnerabilities IOActive evaluated, while 1% of vulnerabilities disable the ECU without gaining actual control. Compromised ECUs may allow the attacker to control all of their normal functionality, add further functionality, or otherwise result in complete compromise of the vehicle component in a manner that may be extremely difficult to detect.

Defenses and Effectiveness

As part of the assessment process, IOActive provides recommendations to the customer for vulnerability remediation. This might include re-working of code, replacement or removal of a feature or component, the purchase and implementation of a defensive tool or product, or other techniques depending on the nature and impact of the vulnerability.

In the assessments analyzed for this research, no defensive products were tested. This is likely a result of the relative infancy of automotive cybersecurity products, as well as a general tendency of defensive cybersecurity products to not undergo third-party testing. Organizations can improve this deficiency by including testing requirements in any procurement language for such products. This has become commonplace in other industries, such as the Industrial Control System space, where manufacturers must deliver security reports alongside products that are to be integrated into production systems.

Ounce of Prevention

In conducting this research, we evaluated an "Ounce of Prevention" measure to best determine what techniques, policies, or methodologies might have prevented a vulnerability from existing in the first place.

Disclaimer: while this aspect of the report was a collaborative effort involving opinions of multiple researchers, and the results are subjective.

[2] http://www.ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf

[3] http://opengarages.org/handbook/2014_car_hackers_handbook_compressed.pdf

[4] https://www.sans.org/reading-room/whitepapers/internet/developments-car-hacking-36607

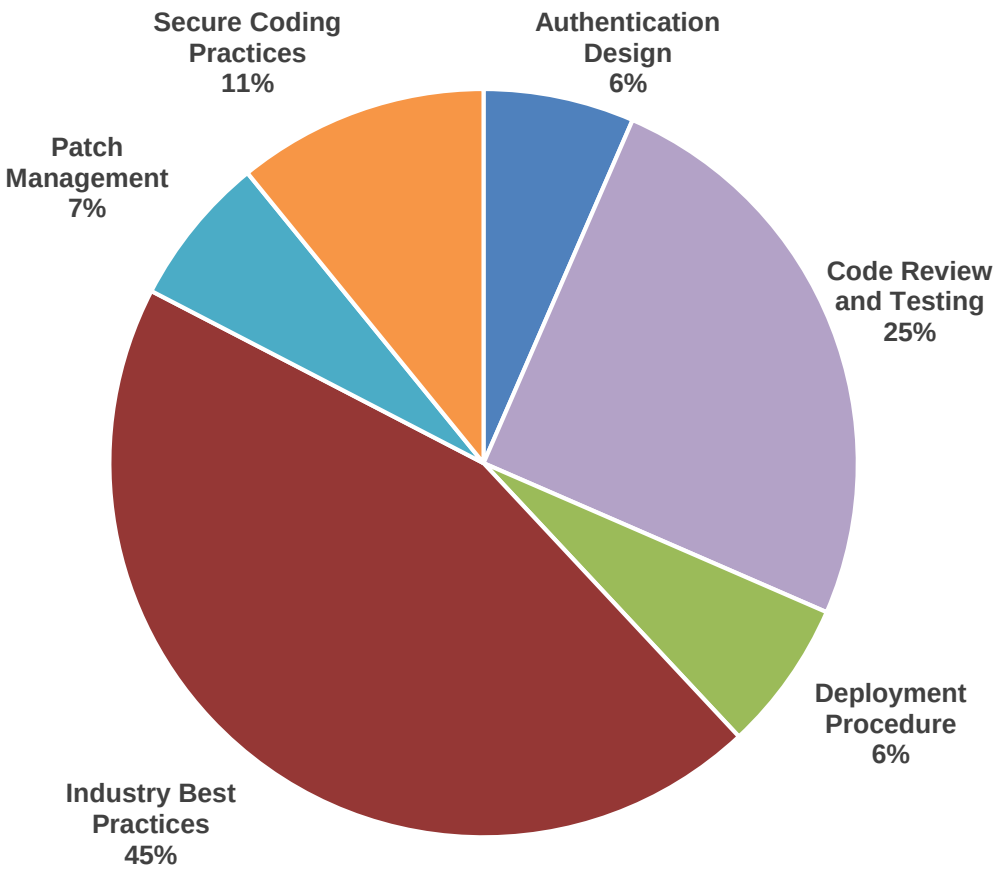

Figure 9 - An "Ounce of Prevention" Analysis

The largest category in our analysis is Industry Best Practices. An increasing number of security best practices publications are made available every year. The Auto-ISAC[56], Microsoft, OWASP, ARM, and others publish best-practice documents to help software and hardware developers to create secure systems. Utilizing this information could result in up to a 45% decrease in vulnerabilities across the board.

Authentication design may fall under industry best practices, but becomes its own category due to its prevalence and the more unique nature of some of the components evaluated. Authentication mechanisms may be difficult to change after a system is deployed and thus should be thoroughly evaluated during system design. Similarly, secure coding practices could help to prevent 11% of vulnerabilities. Simple steps, such as including Microsoft's "banned.h" to avoid using insecure coding patterns like strcpy(), help to improve overall code security (and consequently overall quality) of any system.

The most difficult category to judge with certainty is Code Review and Testing. Catching coding logic errors can be extremely difficult, but following modern software

[5] http://www.prnewswire.com/news-releases/automotive-industry-collaborates-in-developing-vehicle-cybersecurity-best-practices-to-address-cybersecurity-challenges-300301805.html

[6] http://www.autoalliance.org/index.cfm?objectid=1E518FB0-BEC3-11E5-9500000C296BA163

development principles such as test-driven development can do wonders for improving a code base and hardening against unexpected behavior and bugs.

A smaller number of issues stem from poor deployment procedures or patch management. Shipping a production system with enabled backdoors (a.k.a. "debug features") is preventable with a proper deployment procedure and verification. Similarly, shipping products with vulnerable dependencies or a difficult update mechanism further increases the risk for attack.

In evaluating the past few years of vulnerabilities, the old adage rings truer than ever: an ounce of prevention is worth a pound of cure. Incorporating industry best practices and a secure development lifecycle is crucial to avoiding pitfalls that, frankly, have been a non-issue in other industries for years. Using modern design principles improves the ability to patch and maintain a codebase, and improves overall code quality.

Conclusion

The majority of vehicle cybersecurity vulnerabilities are not solvable using “bolt-on” solutions, instead relying on sound engineering, software development practices, and cybersecurity best practices. The most effective cybersecurity work occurs during the planning, design and early implementation phases of products, with the difficulty and cost of remediation increasing in correlation with product age and complexity.

Future Work

Emerging vehicle technologies, such as V2V and V2I communication components, are underrepresented in the data thus far. These emerging technologies typically require self-funded research, and thus are rarely published.

Similarly, the data set does not contain any defensive tools or products. It would be interesting to perform a similar analysis on the defensive tools present in the current vehicle cybersecurity market. Adding additional components to the system, even if they are designed to improve security, always adds complexity, introducing new attack vectors and possibly new vulnerabilities. In the future we should evaluate these systems on their own cybersecurity merit, and add their vulnerabilities to this research.

For more information on IOActive Transportation Security research and services, please visit http://www.ioactive.com/services/air-auto-rail-satcom-ship-transportation-security.html or contact IOActive at info@ioactive.com.

About IOActive

IOActive is the industry’s only research-driven, high-end information security services firm with a proven history of better securing our customers through real-world scenarios created by our security experts. Our world-renowned consulting and research teams deliver a portfolio of specialist security services ranging from penetration testing and application code assessment to chip reverse engineering across multiple industries. IOActive is the only security services firm that has a dedicated practice focusing on Smart Cities and the transportation and technology that connects them. Global 500 companies across every industry continue to trust IOActive with their most critical and sensitive security issues. Founded in 1998, IOActive is headquartered in Seattle, US, with global operations through the Americas, EMEA, and Asia Pac regions. Visit www.ioactive.com for more information. Read the IOActive Labs Research Blog: http://blog.ioactive.com/. Follow IOActive on Twitter: http://twitter.com/ioactive.

