Editors: Mohamed Kaâniche, mohamed.kaaniche@laas.fr | Aad van Moorsel, aad.vanmoorsel@ncl.ac.uk

Integrity in Embedded Control Networks

Philip Koopman and Christopher Szilagyi | Carnegie Mellon University

Many embedded systems, such as in cars, use a network to coordinate control actions in real time. Usually, the system doesn't employ an Ethernet network but rather a specialized real-time control protocol running on severely resource-constrained computing nodes (mostly 8-bit and 16-bit CPUs). This presents unique challenges. Ensuring integrity in such systems involves a combination of safety and security needs, and the usual big-system security approaches aren't necessarily practical. Nonetheless, as threats of attacks increase, integrity approaches in networked embedded systems will have to evolve to provide both the security and safety aspects of integrity in a unified way. And they'll have to do it on a shoestring, using only a few bits per message.

For example, networks using the Controller Area Network (CAN)[1] protocol are ubiquitous in cars (for instance, the diagnostic port in recent cars runs CAN). Because high automotive production volume drives down cost, CAN also appears in all manner of embedded systems, from telecommunication backup power systems to robots.

Rather than being optimized for data transfer, embedded networks are optimized to carry short, periodic, real-time control messages. (It takes only a dozen or so bits to report an engine speed, but that speed changes quickly and must be updated frequently.) CAN provides prioritized message transmission to permit real-time scheduling of network messages, with a maximum payload of only 8 bytes plus a message header field. The maximum network speed is only 1 Mbit per second, with many systems running at half that speed or less to keep hardware costs low. A 15-bit cyclic redundancy check (CRC) is the primary built-in integrity mechanism and is designed to only detect hardware bit corruptions.

Integrity is a key concern in almost any embedded control system because there's some level of concern over safety, or at least an expectation that the network will detect corrupted messages. Let's look at how integrity measures must improve as fault models (analogous to threat models for security concerns) become more demanding.

The Threats

Consider a situation in which multiple computers in an embedded system share a communication network. The CAN CRC is designed to detect some fraction of message corruptions caused by random bit errors and other sources of non-malicious hardware network interference. But in some cases that might not be enough, so the payload might include an additional CRC. For example, the ARINC-825 aircraft data network standard includes an optional 16-bit secondary CRC for high-integrity messages.[2] However, using it requires sacrificing 16 of the 64 payload bits. Even this still deals just with hardware bit corruption.

If you're concerned with faults beyond hardware bit corruption, CAN and most other embedded-network protocols offer no protection from spoofed messages. (Spoofing attacks inject or modify a message while falsifying the source.) This applies to both malicious attackers and simple software defects that accidentally send an unsafe command.

Although it would be nice if you could trust every piece of software that can transmit a message on a network, that's not reality. Rather, it's common for some network nodes to be trusted while others aren't. From a security viewpoint, this might have to do with

- whether the software has been analyzed for security vulnerabilities,
- what protections have been applied (for example, tamper resistance or access control), and
- whether the software is subject to attack from an external network interface or other source.

Copublished by the IEEE Computer and Reliability Societies

There's an analogous issue for safety. In most systems, only some software is trusted from a safety viewpoint (it's often called high Safety Integrity Level, or high-SIL, software). Other software is less critical and must be considered untrusted because the resources necessary to create nearly perfect software aren't spent on noncritical functions. In other words, low-SIL software might have bugs that could send an unsafe network message. So, from either a security or safety viewpoint, it's important to ensure that untrusted nodes can't send unsafe messages. (Multiple isolated networks at different integrity levels are usually infeasible owing to cost, size, weight, and other constraints.)

In current embedded systems, a successful spoofing attack typically lets attackers make a system unsafe in essentially limitless ways. For example, Karl Koscher and his colleagues demonstrated that an attacker who can connect to an automotive control network (for example, via a wireless connection through an attached laptop or via physical access through the CAN diagnostic port) can inject messages to control safety-critical actuators.[3] An attacker might engage a car's emergency brake while it's on a highway, unlock doors and start the engine, or shut off headlights while the car travels at night. More severe attacks are conceivable, such as reprogramming controllers via the network to perform arbitrarily unsafe functions.

Spoofing might originate from a malicious attacker or just a software defect. For example, a low-SIL network node might have a bug that accidentally sends a "full throttle" message by putting the wrong message type in a CAN header field. Or, the node might receive a safety-critical message and resend it with altered contents. Whether the spoofing is accidental or malicious doesn't matter—integrity still must be preserved. To prevent such induced system failures, we need some mechanism to ensure both data integrity (the message content hasn't been changed) and data origin authenticity (the message's source is as claimed).

Ensuring Integrity and Authenticity

Encryption is a common first thought when it comes to embedded system security. If attackers can't tell what a message's contents are, how can they modify the contents? However, encryption aims to provide secrecy, not necessarily integrity.

Depending on the encryption method, a message receiver might happen to detect some accidental modifications to the ciphertext but likely can't defend against all cases of malicious tampering. An attacker can flip any bits of a ciphertext he or she wishes or just generate random noise data as ciphertext. The resulting tampered ciphertext will often decrypt to something that passes error detection checks, even though the attacker might not be able to precisely control the result.

But precise control might not be required. Random commands to control systems might be enough to violate some safety constraint. Moreover, with only a few tries, an attacker can set small actuator command fields of a few bits to any desired malicious value, just by sending random data that gets past integrity checks.

For a cryptologically unsophisticated attacker model (for example, a nonmalicious bug using the wrong message identifier), error detection codes can provide sufficient authenticity. One way to do this is to use a different starting seed value for a message's CRC for critical messages versus noncritical messages.[4] In this scheme, an untrusted node can't spoof trusted nodes so long as it doesn't know the proper seed to use for critical messages (that's the secret "key" in this scheme). In effect, this becomes a non-cryptographically-secure authenticator.

To handle spoofing attacks, we need a function that provides message integrity and authenticity, such as a secure message authentication code (MAC) or a digital signature. These ensure an attacker (or a low-SIL module software defect) can't manipulate message traffic to violate a system's safety constraints.

Suppose we use an MAC to authenticate a message to a single receiver. We run into a problem for embedded networks because they're optimized for short messages. A 256-bit secure authenticator won't fit into a 64-bit payload. It will swamp the network with additional traffic even if it's fragmented across multiple messages. Techniques to authenticate batches of messages usually won't work because of real-time latency constraints. At best, we might be willing to pay 16 bits of a 64-bit payload to detect malicious faults because we were willing to pay that much to detect nonmalicious faults.

Fortunately, whereas some characteristics of embedded networks act as constraints, other characteristics can be exploited to create an efficient authentication mechanism. Two such characteristics are the periodic sampling of messages and the system's inertia.

In many embedded control network applications, nodes periodically broadcast current values of state variables and sensor inputs to the rest of the network. Receivers then update outputs and actuators on the basis of the current system state. This information is typically sampled much faster than the time constraints of control stability requirements[5] (perhaps 10 times the control system step response time). Choosing such a sample rate reduces the delay between a command and the system response, smooths system

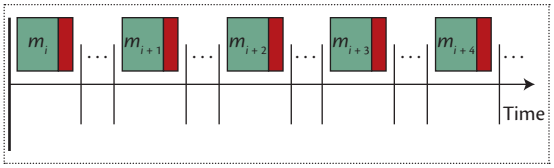

Figure 1. A rolling authentication window verifies messages across multiple truncated authenticators. Each packet contains a truncated authenticator (in red).

responses, and tolerates messages lost via corruption.

Periodic oversampling designed to provide robust system control typically also grants resilience to occasional transient faults—malicious or otherwise. A spoofed message might cause a slight "bump" in control response. However, in most cases it won't cause a catastrophic failure unless a sufficient number of successfully spoofed messages arrive in a short time period. So, it's not necessary to guarantee that a spoof can never be successful. Rather, it's sufficient to ensure that it's unlikely that a large number of spoofing attempts can succeed within a relatively short time window. (How unlikely? Unlikely enough that hardware faults and other problems will dominate in causing catastrophic system failures.)

Given this insight, we can exploit the temporal redundancy of messages and system inertia to scale down an MAC tag's size. Instead of making the MAC 256 bits to make spoofing any single message essentially impossible, we need only a few bits (sometimes only one) per message to make spoofing many messages in a short time period unlikely (see Figure 1).

This idea extends to multicast authentication, which is important because many embedded control networks use broadcast transmissions. A significant advantage of this authentication approach is that the system designer can perform a tradeoff among

- authentication bits per packet,
- application level latency for state changes and physical actuations, and
- the acceptable probability of induced system failure for each message type.

For example, assume that we need a 10^{-9}-per-hour probability of undetected spoofing attempts and that we must verify within no more than four message samples. We would need 16 MAC bits per sample under a reasonable set of network operating assumptions. On the other hand, if we have room for just two MAC bits per packet, we can instead verify over 26 samples. (A detailed description of this approach appears elsewhere.[6])

Implementing such an approach in a real system might also entail the following actions. The system might synchronize nodes to establish a time base for message freshness (that is, to prevent replay attacks). It might enter a degraded mode when it detects too many invalid messages in a short time period. It will need to distribute keys during commissioning. Finally, it must update the keys after nodes are replaced during maintenance.

Although safety and security might seem very different design goals, in embedded systems their respective integrity concerns can end up in the same place. Historically, embedded safety has considered faults to be just short of malicious. But in the Internet-connected future, safety designers will have to assume malicious faults and adopt appropriate countermeasures. To do this, they'll have to adapt existing security approaches to meet embedded control systems' unique constraints and opportunities. Important techniques to enable this will likely include ultralightweight secure MAC algorithms that can run on small, slow microcontrollers; mitigation strategies for attacks that manipulate message arrival times to destabilize control loops; and novel approaches to dealing with the control-centric nature of many embedded systems. ■

References

1. *CAN Specification*, ver. 2, R. Bosch GmbH, Sept. 1991.
2. *ARINC Specification 825-2, General Standardization of CAN (Controller Area Network) Bus Protocol for Airborne Use*, ARINC, July 2011.
3. K. Koscher et al., "Experimental Security Analysis of a Modern Automobile," *Proc. 2010 IEEE Symp. Security and Privacy*, IEEE, 2010, pp. 447–462.
4. J. Morris and P. Koopman, "Critical Message Integrity over a Shared Network," *Proc. 5th IFAC Int'l Conf. Fieldbus Systems and Their Applications* (FeT 03), Elsevier IFAC Publications, 2003, pp. 139–145; www.ece.cmu.edu/~koopman/roses/fet03/fet03_critical_message_integrity.pdf.
5. H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.
6. C. Szilagyi, "Low Cost Multicast Network Authentication for Embedded Control Systems," PhD dissertation, Dept. of Electrical and Computer Eng., Carnegie Mellon Univ., 2012; www.ece.cmu.edu/~koopman/thesis/szilagyi.pdf.

Philip Koopman is an associate professor in Carnegie Mellon University's Department of Electrical and Computer Engineering. Contact him at koopman@cmu.edu.

Christopher Szilagyi is a recent graduate of Carnegie Mellon University's Department of Electrical and Computer Engineering. Contact him at christopher.szilagyi@gmail.com.