PROTECTION PROVIDED BY DIGITAL SIGNATURES AND CERTIFICATES FOR EMBEDDED SYSTEMS

When designing for security, the operating environment needs to determine the degree of robustness required. A security architecture must include not only the target device, but all endpoints and users within the overall system. While there are serial numbers, MAC addresses, white lists, and black lists - these designs are not foolproof. Most embedded hacks are accomplished by monitoring network traffic to reverse engineer commands, then replaying the same or modified version from somewhere else.

In the IoT world, everyone is a stranger until proven otherwise. Even private networks are subject to data breaches and are just as vulnerable as the Internet. Given no network is secure, all remote endpoints must be authenticated before commands and data can be trusted. Transitioning from people to computers, the equivalent of a secret word is the key. Designs using secret keys may be feasible in smaller environments, but as the number of devices grow, so does the complexity and cost to protect those keys. If a secret key is exposed due to compromise of a single device, then every other system in the environment is vulnerable and cannot distinguish a stranger from a trusted system.

User names and passwords help dynamically generate unique secret keys, which are entered upon demand and don't need to be stored in nonvolatile memory, reducing the risk of compromise. While this works in some environments, many products don't have the concept of users and must operate securely immediately when powered on. These environments require shared secret keys between both endpoints for authentication and encryption. But how do the keys get in the system? In military environments, special handheld computers called key fill devices are used to keys so they are never exposed. Key fill devices and key management systems protect keys during distribution until stored safely within a device's security boundary.

Public key encryption simplifies the complexity of shared keys by providing each endpoint with a single unique asymmetric key pair to use for encryption and authentication. Best practice security never uses the same key for both. If an endpoint is compromised and private key exposed, the vulnerability is limited to just that system.

Asymmetric cryptography allows endpoints to communicate securely without needing to pre-share keys. However, how does a device know if it's securely communicating to the right endpoint? Certificates are used to prevent man-in-the-middle attacks during the exchange of public keys and prove the identity of a remote endpoint. In most basic form, a certificate consists of metadata including name, serial number, expiration data, etc; bound to the public key by a digital signature from a Certificate Authority (CA). Using mutual authentication, devices will never attempt to process incoming commands unless coming from a valid source. Company IoT services are also protected because they too only respond to properly credentialed devices.

When two systems exchange certificates, public keys can be trusted if they are both digitally signed by the same CA. Since both systems trust the CA, that trust can be extended to the remote system assuming private keys are not compromised. Once authenticated and trusted, software can read the certificate to determine what action to take, or system features to make available. This allows devices to be triggered to enter debug mode for a limited time based on the training level of the technician.

Certificate management

Several certificate generation and management designs exist to help embedded system developers incorporate digital identities into their products. The IEEE 802.1ar specification is increasingly used in embedded products, like networking and industrial control systems. 802.1ar addresses the management and use of a single iDevID and multiple LDevIDs certificates. The iDevID certificate is also known as the "birth certificate" - used to identify the manufacturer of the system, board, or component. This is typically the first certificate on the system and generated during manufacturing. The purpose of LDevIDs varies depending on the environment. For example, in one product an LDevID certificate may be created to protect customer profiles and data. In another example, a board may be sold to multiple

PROTECTION PROVIDED BY DIGITAL SIGNATURES AND CERTIFICATES FOR EMBEDDED SYSTEMS

An LDevID certificate may be generated for each integrator so they can communicate securely within their own local enclaves and protect their data without risk of compromise.

When incorporating certificates into an embedded design, developers and manufacturers must take the following into consideration:

Programming the Root CA certificate into immutable memory – keeps an attacker from replacing to something else

Asymmetric key pair generation – Private keys should never be exposed outside of the device, assuming proper random number generation is feasible

Private key protection – Spoofing attacks are possible if private keys are compromised

Sending of certificate signing requests to CA system – If using a hosted CA system, what is the impact to production due to a network outage?

Receipt of certificate and protected storage – How are certificates and keys protected on the device?

Loading initial revocation lists – Relates to the entire inventory management process and tracking of credentialed systems and RMAs.

An increasing number of product developers are building their own public key infrastructures (PKI) and generating certificates right on the assembly line, where private keys can remain inside each device and production is unaffected by Internet outages. The Device Lifecycle Management (DLM) System, developed by INTEGRITY Security Services, protects a company's CA keys from IT networks susceptible to data breach at remote and third-party manufacturing sites for high availability certificate generation. DLM supports developers wanting to incorporate certificates, tailored to their design and supply chain, without having to build and support their own PKI.

The most important consideration in the discussion of authentication is how are we able to trust others if we can't trust ourselves? Unfortunately, this isn't a philosophical question. If system software is compromised, then the chain of trust is broken from the beginning and nothing can be guaranteed. Hacked software can skip verification, accept any certificate, and modify message contents. Keys can be compromised if not properly protected and used to attack other devices by manipulating commands and data. Backdoors can be opened to collect and send data, making any

Before an embedded system can trust itself to authenticate remote endpoints, it must check that software has not been modified. This is accomplished through a process called secure boot, where system software is verified before executed. At each power on, secure boot checks the authenticity of each software layer before allowing it to execute. This ensures software is not corrupted and comes from a valid source. A component is never executed unless proven trustworthy.

Hashes and checksums only verify the integrity of software, not authenticity. As long as an attacker modifies the hash along with the code, malicious software can still execute undetected. Authentication comes from digitally signing the hash using an asymmetric key. A company's security infrastructure protects the private key and digitally signs the release software. The corresponding public key is programmed into the device during manufacturing and used during verification. Now, if an attacker changes both code and hash, they still can't update the digital signature without the corresponding private key regenerating the digital signature. INTEGRITY Security Services works with companies in all industries to deploy secure boot solutions including digital signing infrastructures, with zero exposure protection of private keys and integration into everyday software build processes.

The security architecture of today's networked IoT products must account for more unknowns than ever before. With every new device added to the network, comes an additional unknown threat and risk of conflict causing support and development costs. These unknowns are mitigated by controlling communication to only trusted endpoints through authentication. Starting with hardware to verify software before extending remotely, the chain of trust is preserved.

"A security architecture must include not only the target device, but all endpoints and users within the overall system."

TE EXTC

Omkar Bhushankar

Soham Naik

Sumit Mondal

Aman Takshak