//consult.red

6 key considerations for Device Cybersecurity compliance

The security of connected devices and systems should be a primary consideration for any product manufacturer or designer, ensuring their products and end-users are safe from hackers and other online threats.

Connected. Conformant. Compliant

// In this guide you will learn:

- 6 key considerations for device cybersecurity
- What can go wrong if they are not addressed
- How cybersecurity threats can be mitigated as part of good device design and development

// An introduction to device cybersecurity

ATMs. Baby monitors. Thermostats, vending machines, and more. The number of internet-connected devices has skyrocketed in recent years, a trend that shows no signs of slowing down. By 2030, there could be anywhere between 31.8 billion [Transforma Insights] and 125 billion [IHS Markit] connected devices, according to some.

There's good reason to bring those objects online, too. Connecting them to the network opens up opportunities that range from greater personal convenience to data-driven industrial optimisation. Naturally, though, there's an inherent risk that comes with connectivity.

In December 2022 alone, more than 10.54 million cyberattacks were launched against mobile and "Internet of Things" (IoT) devices [Statista]. From botnets and ransomware to data theft and AI-enhanced attacks, the objects around us are now under near-constant assault.

Unsurprisingly, governments and other relevant authorities have begun to take note. The UK's Product Security and Telecommunications Infrastructure (PSTI) Act, in addition to the EU's Radio Equipment Directive (RED), make it a legal requirement for connected devices to comply with cybersecurity standards.

Not only is this a new requirement, but it's also one that carries significant weight as well. Both PSTI and RED threaten severe penalties for any manufacturer that fails to protect their equipment.

The EU Cyber Resilience Act will broaden the scope even further—bringing ethernet-only devices into play—and will carry fines of up to €15m or 2.5% of worldwide annual turnover.

Against this backdrop, manufacturers find themselves needing to guarantee the security of their devices. Across storage, communications, authentication, and more, they need to show they've done everything in their power to eliminate vulnerabilities and defend against threats.

For some, the capabilities to do that will already be in place. For many more, though, this will represent unfamiliar — if not entirely new — territory.

This resource is a guide to 6 key aspects of device cybersecurity as identified by our expert team. We'll talk you through each aspect individually, the associated risks, and how to mitigate these risks through best practices in system design and development.

"Consult Red is an expert cybersecurity partner, able to guide you through your entire cybersecurity journey – ensuring product compliance and your brand reputation."

// Secure Storage

1

What is secure storage for IoT or embedded devices?

Any device will have at least some information which must be kept securely, that is, in a way which ensures both confidentiality and integrity:

- **Confidentiality** – preventing unauthorised access to the data
- **Integrity** – preventing unauthorised modification of the data

Typically, there are two types of secure storage in an embedded device:

- **Low volume, very high security**: specialist hardware storage (such as within a Trusted Platform Module, Hardware Security Module, Secure Enclave, Trusted Execution Environment or other specialist secure storage within a SoC). This storage is very resistant to unauthorised access (for example, using hardware measures to protect against physical tampering) but cannot store large volumes of data. Typically, this storage would be used to store cryptographic keys, which could, in turn, be used to protect encrypted drives for high-volume storage, secure communication, secure software update/secure boot, etc.

- **High volume, general-purpose security**: commodity storage such as flash memory, disk drives, SD cards, etc., which uses an encryption mechanism to secure the data stored on the drive. Such storage can store large volumes of data (limited only by the size of the storage device) and can be considered fully secure, provided that a suitable encryption scheme has been used and that the keys used for encryption have been secured and managed appropriately.

The EN 18031 standards require devices to use secure storage mechanisms for storing sensitive data, although the specific nature of the secure storage is not mandated by the standard.

What can go wrong if you don't get secure storage right?

Secure storage is the foundation of many other security features of a device. Therefore, if the secure storage mechanism is absent or not properly implemented, the device is open to a range of security attacks. This could include exfiltrating user data; gaining access to keys and certificates used for secure communication, which could allow an attacker to spoof the device or intercept communication; or modifying the data stored on the device, which could include software and configuration files, allowing an attacker to install malicious code on the device.

Some examples of real-world attacks that revolve around a lack of proper secure storage include:

- **Tesla Model S key fob attack** (2018) - Researchers found that the cryptographic keys used to secure the Tesla Model S's key fob used a weak encryption algorithm and were stored insecurely, allowing attackers to clone the key fob and unlock the vehicle. [wired.com]

- **Foscam security camera credential vulnerability** (2017) - Vulnerabilities in Foscam security cameras were discovered whereby sensitive information, such as user credentials and Wi-Fi passwords, were stored in plain text in the camera's filesystem. In addition, hard-coded credentials were present, which could not be disabled. The lack of secure storage and proper key management gave attackers complete control of all cameras from the same product range, including the ability to install modified software on the device. [thehackernews.com]

- **ZigBee Light Link vulnerability (Philips Hue)** (2016) - Philips Hue smart bulbs, which use the ZigBee Light Link protocol, were found to store encryption keys insecurely. Researchers were able to use an attack to recover the Philips Hue OTA update verification and encryption keys. With those keys, they could create a malicious software update and load it to a Philips Hue light, which, in turn, could be used to attack other devices on the network. [iacr.org]

How do I design a good secure storage implementation for my device?

Designing an effective, secure storage implementation for an embedded device must start at the very beginning of the design phase since good security relies on having appropriate security support built into the hardware (such as a trusted platform module, secure element or hardware security module to store cryptographic keys and other highly sensitive data).

Without some amount of hardware security capability, there is the possibility that a device is vulnerable to physical attack to obtain cryptographic keys and certificates, which could be used to view or modify data on the device or spoof the device when communicating with cloud or backend services.

The hardware security functionality must then be used appropriately by software in order to implement secure storage on the device's main filesystems. Things to consider include:

- Selecting **strong, modern encryption algorithms** and appropriate modes of operation to encrypt data at rest.
- Using **cryptographic hashing** or **authenticated encryption** to provide integrity protection as well as confidentiality, preventing unauthorised modification of data.
- Assessing whether, for specific data items, it is appropriate to rely on the security provided by the **encrypted filesystem** or whether **per-file encryption** is required for more sensitive data.
- Implementing mechanisms to control and manage per-device keys, whether **initial provisioning** during manufacturing or **key revocation and replacement** after device deployment.
- Using appropriate **authentication and access control mechanisms** to limit which users or processes can access data from the secure storage.
- Implementing a **secure software update process** to ensure that secure storage mechanisms cannot be overridden by an attacker installing modified firmware on the device.

// How Consult Red can help with secure storage

Some examples of applications of secure storage within our experience include:

- **Secure management of video content** using Digital Rights Management – and secure storage of DRM keys
- **Secure storage of keys and certificates** used to authenticate a device with headend and cloud-based services
- **Secure storage and management of wireless network credentials** in connected devices and access points
- **Secure storage of logs and telemetry data**, including those containing business-sensitive and personally identifiable data
- **Hardware audit of security mechanisms** provided by the chip vendor to ascertain effectiveness and identify potential weaknesses.

// Secure Software Update

2

What is secure software update for IoT or embedded devices?

Any internet-connected embedded device needs a mechanism for the device manufacturer (or operator) to issue software updates (sometimes called firmware updates) to the device after it has been deployed into the field. There are two crucial aspects to this:

- A **software update mechanism** is necessary to secure the device against newly discovered vulnerabilities.
- The software update mechanism **must itself be secure**. Otherwise, malicious actors could exploit the software update mechanism for nefarious purposes.

A secure software update process will have the following features:

- The device manufacturer/operator can **receive telemetry** showing which software version is running on which device in the field.
- The manufacturer/operator can **remotely initiate software updates** targeted at one device or a class of devices.
- The device can **verify that the software update** is genuine and has not been tampered with. It is not possible for a non-authorised entity to get a software update installed on the device.
- The device has a **fallback mechanism**. Should a software update fail for some reason, the device will still be able to function.
- The device should have an **anti-rollback mechanism** so that once a new software image has been booted successfully, it cannot be forced to run older software.

The EN 18031 standards require devices to have at least one mechanism for securely updating each part of the device's software. In the UK, the Product Security and Telecommunications Infrastructure Regulations 2023 require that manufacturers publish information about the duration of the support period during which software updates will be available at the time of marketing the product. In some contexts, a secure software update may be referred to as an Over-The-Air (OTA) update, which originates from devices such as set-top boxes which receive updates via an "over the air" broadcast mechanism such as terrestrial or satellite TV signals. It is now much more common for updates to be delivered over the internet, although the terminology of OTA update may still be used.

What can go wrong if you don't get secure software update right?

If a device cannot receive software updates promptly, it can be left vulnerable to exploits discovered after it has been deployed to the field. Malicious actors can use such exploits for various nefarious purposes, including:

- **Direct denial-of-service (DoS) attacks**: the attacker can stop the device from functioning correctly or at all.
- **Distributed denial-of-service (DDoS) attacks**: a compromised device can be used as part of a botnet to orchestrate a DDoS attack on other targets on the internet.
- **Ransomware attacks**: an attacker could attempt to hold the device or the data it contains hostage in an attempt to gain money from the victim.
- **Network entry point**: once an attacker has control of a compromised device within the user's local network, they can use it to attempt to mount attacks on other devices on the user's network.
- **Data exfiltration**: an attacker could use the compromised device to steal sensitive information from the user or from other devices on their network.
- **Spying**: an attacker could use a compromised device to obtain audio or video footage from a device which contains a microphone or camera.

An attacker who is able to install a compromised software image on a device could also use such an image to achieve any of the above forms of attack. An example of a series of real-world attacks that involve a lack of a proper secure software update mechanism is the Mirai Botnet. Starting in 2016, the Mirai malware has been used to create a botnet, which has been used in a series of large distributed denial-of-service attacks. It has been very successful because it targets low-cost IoT devices, which often use weak security credentials and do not have an effective software update mechanism. These devices remain vulnerable to the Mirai malware several years after its first discovery.

How do I design a good secure software update implementation for my device?

- Designing an effective, secure software update mechanism requires full knowledge of the device's hardware and software stack, as you will need to consider mechanisms to update the bootloader, firmware, runtime images and applications.
- Simpler devices may use a monolithic runtime image that updates all application software in one go. In contrast, more complex devices often use package- or container-based solutions to update applications independently of the core runtime image. Each approach has advantages and disadvantages, and you should consider the use cases for your device before deciding on the approach that is right for you.
- Where possible, you should leverage any secure update mechanism or infrastructure provided by your device's platform (e.g. Android) in preference to rolling your own solution, but not all devices will use a platform that has such a mechanism built in.
- Parts of the software update mechanism are likely to interact with the device's bootloader, which can use keys stored in the device's secure storage to verify the signature of software images at boot time, prior to executing the code. The bootloader will also likely manage any fallback mechanism, such as an "A/B" partition system.
- The device will also require middleware functionality to communicate with the remote update service, identify when images are available, and schedule these for installation at a convenient time. Often, updates may have a priority status set, depending on the severity of the security issues the update might fix, so that the most urgent patches can be installed as soon as possible, while less urgent fixes may be applied at a more convenient time for the user.
- Where the update mechanism permits non-sequential updates, careful testing is required to ensure the device functions correctly, even if updates are installed in a different order.
- The software update mechanism's design requires careful thinking to ensure that the system works as intended when deployed at scale. Many issues can be encountered when deploying software updates to large numbers of devices in the field, which may not have been spotted in even the best test lab. One typical example of what can go wrong is as follows: devices normally reboot as part of the software update process, and devices typically connect to several cloud and backend services just after they reboot. Therefore, if an operator inadvertently schedules too many devices to install an update at the same moment, they could end up overloading their backend systems by having too many devices rebooting and initiating connections to the backend at the same time.

// How Consult Red can help with secure software update

We have worked with various aspects of secure update for deployments with up to 20 million devices deployed in the field, including:

- **Software update mechanisms** for Android, Linux, RTOS and bare-metal-based devices.
- **Bootloader configuration**, including A/B partition systems and cryptographic signature verification of downloaded images
- **Device middleware** to manage software updates
- **Backend and cloud services** to schedule and orchestrate software updates on a fleet of devices
- **Telemetry reporting and remote diagnostics** for software update and boot failures

// Secure Communication

3

What is secure communication when applied to an IoT or embedded device?

Any IoT device, by definition, needs to communicate across the internet, generally to some form of cloud backend services which control the device and provide it with useful functionality. Information that is communicated in this way could include:

- **Commands from the backend to the device**, instructing it to do something (for example, telling an IoT door lock to open a door)
- **State information sent from the device to the backend**, which is used by the backend to trigger other actions (for example, trigger a heating system when a thermostat tells the backend that a room is cold)
- **Data from the backend to the device**, for example, a software update image or a configuration file
- **Data sent from the device to the backend**, as part of the device's functionality (for example, video streamed from a security camera to a cloud storage system)

Other forms of embedded device also require secure communication channels, for example, wireless remote car keys. Most devices communicate through channels such as the internet, which are not inherently secure, so the device architecture must allow for a secure channel to be layered on top of the communication mechanism. Considering an IoT or embedded device, the following properties are the most significant in making the communication system secure:

- **Confidentiality** – it should not be possible for an unauthorised party to access the data being communicated
- **Integrity** - it should not be possible for the data to be modified in transit
- **Authenticity** – it should be possible to know with certainty that a message originated from a trusted source – i.e. the device must know that it is communicating with the genuine backend, and vice versa
- **No replay** – it must not be possible for an attacker who records a genuine message and replays it later to trick the device into executing an action.

What can go wrong if you don't get secure communication right?

Attacks on devices that fail to secure their communication channels adequately can lead to:

- Attackers **gaining control of a device and misusing its functionality** (such as unlocking a house secured by an IoT door lock)
- Attackers **gaining access to data** transmitted by a device (for example, CCTV footage from a camera)
- Attackers **gaining control of a device to use it for some other nefarious purpose** (for example, as part of a botnet)
- Attackers **tricking the backend into performing some action** (for example, turning on someone's lights when they didn't want it)

Some examples of real-world attacks that revolve around a lack of proper secure communication mechanisms include:

- **TP-Link smart bulb hack** (2023) - flaws in secure communication meant that authenticity and confidentiality were not guaranteed on the communication channel between the device and the control app. This allowed an attacker to gain control of all devices on the user's account and obtain the user's Wi-Fi password. [arxiv.org]
- **Ring doorbell hack** (2019) - Ring doorbells communicated with an app using an insecure communication channel. Attackers could intercept video data from the doorbell camera and potentially modify footage. [bitdefender.com]

How do I design a good secure communication system for my device?

- EN 18031 requires that devices **use appropriate secure communication mechanisms** and that these mechanisms **must protect the confidentiality, integrity and authenticity of the communications**, as well as **protect against replay attacks**. However, the standard does not detail the specific measures that should be used to achieve this since the appropriate mechanisms depend on the nature of the device and the constraints placed on the environment in which it operates.
- Designing a secure communication mechanism between a device and a backend service sounds simple at first – with technologies such as HTTPS being commonplace.
- However, while “just use HTTPS” can solve many problems around confidentiality, integrity and replay attacks, the issue of authenticity – that the device “knows” it is communicating to the genuine backend – and vice versa – is more challenging to achieve. HTTPS, as it is used on millions of websites, relies on TLS using PKI certificates and chains of trust to allow a web browser to confirm to a user that when they visit https://www.amazon.com they are communicating with the genuine Amazon.com Inc, and not an imposter or a “man in the middle” attack. But HTTPS does not help Amazon to know who their customer is – they must layer a separate user authentication and login process into their website to provide that.
- A solution is possible using TLS mutual authentication, but this requires the device to be issued with a unique TLS certificate that must be stored securely on the device. There are logistical challenges to overcome to ensure that devices are securely provisioned with device certificates – either during manufacture in the factory or over some provisioning mechanism in the field. Of course, steps must then be taken to ensure that the provisioning mechanism is itself secure...
- Other communications mechanisms, either over the internet using protocols other than TLS/HTTPS or over channels such as Bluetooth or Zigbee, face similar problems. While standard mechanisms are typically available to allow encrypted communication and provide authentication, their security relies on using keys, which again depends on a secure provisioning mechanism.
- Another consideration is to ensure that cryptographic methods used to secure the communication channel used by a device are effective and do not rely on outdated and potentially vulnerable cryptographic schemes. EN 18031 has a general requirement that devices should use best practices for cryptography; while the standard is not prescriptive, this would exclude the use of outdated cryptographic algorithms such as 3-DES, MD5, SHA-1, AES with short key lengths (AES-128), and ineffective block cypher modes of operation such as ECB mode.

// How Consult Red can help with secure communication

We have experience with various aspects of secure device communication, including:

- Design and implementation **of secure, standards-based**
- **device-to-backend and device-to-device communication protocols**
- Secure communication implementations on **ultra-low-cost, resource-constrained devices** (which might not have the resources necessary to support traditional public key-based algorithms)
- **Device provisioning** and key/certificate management
- **Device security review**

// Logging

4

Why is logging important for security in an IoT or embedded device?

In embedded devices, logging is useful for various purposes: verifying that the device is operating as intended, finding and fixing bugs, monitoring how users use the device, and measuring device performance. But is logging really a security measure?

Think about a bricks-and-mortar store. There are several security measures that the store is likely to have in place to actively prevent shoplifting: enforcing a one-way system that only allows exit at the cash registers, locking away the most valuable items behind the counter, and so on. But the store is likely to have a CCTV system installed as well. On the face of it, the CCTV system does nothing to physically prevent shoplifting – but it does provide a very good mechanism to provide evidence of what happened after an incident has occurred. By reviewing CCTV footage after a theft, the store management may be able to identify a flaw in their other security measures, which they can address to reduce the chances of further thefts in future.

Logging in an embedded device is similar. The logs don't themselves prevent a security incident from occurring, but they can help developers conduct a post-mortem of what went wrong after an incident, which means they can take steps to prevent it from happening again.

EN 18031-2 considers logging important to protect personal data that may be stored in a device, and EN 18031-3 has similar requirements aimed at protecting financial data. By logging activities such as the addition or removal of users and permitted and denied access attempts, an investigator can trace a pattern of user access that may have occurred during an attack.

What can go wrong if you don't get logging right?

Because logging does not directly protect against security incidents, failure to have an effective logging system is not as devastating as a lack of other security measures. However, a lack of proper logging could prevent engineers from identifying security vulnerabilities in their device that may have been exploited by attackers, which could potentially put their users at increased risk by extending the amount of time an exploitable vulnerability is present in a device in the field.

The same is true for issues outside of the security realm – a lack of effective logging could result in a buggier and less reliable product, a degraded user experience, and, ultimately, damage to a manufacturer's brand.

It is also crucial that any logging system which is implemented is done securely. Carelessly designed logs can lead to information leaking, which could include sensitive personal data or security-related information such as passwords. See example - [cve.org]

How do I design a suitable logging system for my device?

To design a good logging system, you must understand **why** and **what** before thinking about **how**.

Why

- Why do you need logs? From a security perspective, think about how having logs will help make your device secure; for example, "having logs of user accesses will help us to identify any irregular login attempts".

What

- What information is needed to design a system which fulfils the purposes you have set out? For example, if you need a log of user accesses, then you probably should log the date, time, user ID, and whether the login was successful, associated with each login attempt.
- What information don't you need? For example, there is no need to log the password used – and doing so could lead to information leakage.
- How long does this information need to be kept?

How

- How will you store the logs on your device? Are there constraints, for example, the amount of storage space that can be used? A resource-constrained device may need to use a very compact format, while other devices may focus on a format that allows logs to be read and searched efficiently.
- How will the logs be used? This might inform the storage format or other considerations, such as whether and how frequently logs are uploaded to a backend or cloud service. If you are using a SaaS logging service, you may wish to design your logging system around your vendor's libraries and APIs – or you may want to keep your device more vendor-agnostic.
- How will the logs be gathered? How will logging be hooked into your existing code and any library components you use?
- How will the logs be kept secure?
- Will you require different levels of logging for different deployments (e.g. debug level logging for use during development or in beta-test scenarios?)
- How much will it cost? Logging has the potential to generate enormous volumes of data. While cloud storage and SaaS logging tools can appear cheap initially, the costs can soon mount up, especially if you have many devices in the field. If you rely on a vendor's proprietary APIs to capture log data, you can also face significant costs should you wish to switch to a different vendor in the future.

// How Consult Red can help with Logging

Some examples of applications of secure storage within our experience include:

- Designing **compact log storage formats** for resource-constrained devices
- Use of **SaaS logging tools** such as New Relic
- **High throughput, low latency logging** for monitoring system performance
- **Log analysis tools** and techniques

// Access Control and Authentication

5

What is access control and authentication in an IoT or embedded device?

Access control and authentication are two related areas of functionality which are essential to secure any device.

- **Authentication** is about making sure that an entity really is who they claim to be
- **Access control** is about making sure that entities can only perform the actions they are permitted to perform

The two mechanisms go together to secure a system: effective user authentication is a prerequisite to an effective access control system.

Some examples of requirements for effective access control systems could include:

- An IoT thermostat must only be able to be controlled by the residents of the home in which it is installed.
- The video stream from a smart CCTV camera must only be visible by the owner of the CCTV system.
- A Wi-Fi router must allow one set of users to access the main Wi-Fi network, a different set of users to only access the Guest Wi-Fi network, and only the administrator to change the Wi-Fi settings on the device.

EN 18031 is not prescriptive on the access control model that should be used, but it requires that some form of access control mechanism is in place when it would be required to protect the security of the device, and that the manufacturer verifies that the access control mechanism is effective for its intended purpose.

Authentication mechanisms can vary. Passwords are a very common authentication mechanism but are imperfect in many ways. In the past, it was very common for IoT devices to use factory default passwords (such as ‘0000’) which led to many devices being vulnerable to attacks, such as those described below. EN 18031 requires that if passwords are used, they must be either uniquely set per device or required to be configured by the user prior to the device being connected to the internet. Very similar laws are in place in the UK (Product Security and Telecommunications Infrastructure Act & Regulations) and California (Senate Bill 327).

EN 18031 also requires that devices are resilient against brute-force attacks on their authentication mechanisms, such as using a time delay between consecutive login attempts, a lockout period after failed login attempts, or multi-factor authentication systems.

What can go wrong if you don’t get access control and authentication right?

Many security vulnerabilities have been caused by improper access control and user authentication mechanisms. Some examples include:

- **Epson printers –** in many Epson printers, in the default configuration, no administrator password was set, allowing an attacker to set the password and take control of the device. [nist.gov]

- **Security cameras** – a security camera used a hardcoded root password, allowing an attacker with one such device to obtain the root password for any similar device. [nist.gov]

- **Mirai Botnet** – a widespread attack which has been ongoing since 2016 has seen the Mirai malware attack a large number of different low-cost IoT devices; a common factor being they all use default passwords. [cloudflare.com]

How do I design a good access control and authentication system for my device?

To ensure that your IoT or embedded device has an effective access control and authentication system, you should:

Identify all interfaces that provide access to the device. This could include:

- Local user interfaces, such as via a local display or TV screen
- Remote user interfaces, such as a web-based administration user interface
- Application-specific interfaces and APIs, which could be over a network, Bluetooth, etc.
- Debugging interfaces, such as SSH or a serial interface
- Interfaces which are intended to be internal to the device, but which could be physically accessible via the hardware, such as using physical test points and vias on the PCB

For each and every interface, you should document:

- Who or what should be allowed to access the device via that interface (the authorised entities)
- What data or control is available via that interface
- .How the authorised entities authenticate themselves over that interface, if applicable. (Note that in some cases, physical presence in the same location as the device may be considered an appropriate level of authentication, but this would depend on the nature of the device and its intended operating environment.)
- Whether simple authentication is sufficient to provide the appropriate level of access control, or whether a more nuanced permissions model is required
- If a permissions model is required, how does that work?

You should then revisit each interface outlined in the previous step and think "what could go wrong?"

- What weaknesses exist in the authentication method? For example, are you relying on a password? If so, could an attacker try a brute-force attack to guess the password? Are credentials encrypted both in transit and at rest on the device?
- If an unauthorised entity does manage to gain access via a particular interface, what would they be able to do, and what would the impact of this be - is this an acceptable risk? Is there a way to reduce the impact and therefore make the risk more acceptable?
- If there is a permissions model, does it restrict access sufficiently to appropriate entities? For example, if some functions require an “administrator” or “root” level of access, is there a robust method to stop a user from gaining unauthorised access to the root privileges (privilege escalation)?

Depending on your findings from the previous step, you may find some points which need addressing to secure your device. Steps could include:

- Eliminate the interface entirely (e.g. changing the design of a PCB to reduce the number of test points or moving to a SoC rather than separate components connected by an accessible interface).
- Reducing the circumstances in which the interface is active (e.g. disable a debugging interface in production software builds).
- Restrict the data or functionality that is available via the interface.
- Add additional or more robust authentication methods on the interface (for example, using two-factor authentication, or adding protection against brute-force attacks).
- Adding additional or more fine-grained access control methods (e.g. differentiating between “administrator” and “root” level functionality).

// How Consult Red can help with access control and authentication

We have experience with all aspects of secure access control and authentication, including:

- **Secure hardware designs and selection of components** to support secure use cases
- **Designing software architectures** to support security requirements
- **Secure device provisioning** with security keys, certificates and other authentication credentials
- **Security assessment** and vulnerability analysis

// Denial of Service

6

What are Denial of Service attacks and why are they of concern in an embedded or IoT device?

A Denial of Service (DoS) attack is a cyber-attack in which the perpetrator aims to make a network service unavailable to its users, typically by flooding the service with more requests or network traffic than it is able to handle. IoT and embedded devices could be the target of a DoS attack, although this is less common in practice.

The more significant risk for an IoT device is for them to be used as part of a Distributed Denial of Service attack (DDoS). In a DDoS attack, the embedded device is turned into the 'attacker', rather than the 'attacked'. The device is taken over and becomes part of a 'botnet' - thousands of compromised devices are used together to deliver a DoS attack on a target, typically by sending so much traffic to an online service that it is unable to cope with demand, and therefore denying genuine users access to that service.

Why should device manufacturers care about Denial of Service attacks?

Denial of Service attacks can be extremely damaging to the operation of the internet. As such, EN 18031-1 requires manufacturers to help protect the internet by building measures into relevant types of network devices that will mitigate and help prevent DoS and DDoS attacks. These include:

- A **resilience mechanism**, so that a device which is targeted in a DoS attack can detect that it is under attack, and take steps to mitigate the impact of the attack and ensure it recovers to a known-good state once the attack passes. This can help make DoS attacks less effective since they only knock the device offline for the duration of the attack, and not afterwards.
- A **network monitoring mechanism**, so that a device such as a router or gateway which connects other devices to the internet can monitor the traffic it is passing and identify traffic that is potentially part of an attack.
- A **traffic control mechanism**, so that a device such as a router or gateway can block traffic which is suspected to be part of a DDoS attack originating from inside the network from accessing the internet.

How do I include Denial of Service attack prevention in my device?

The first and most straightforward step to reduce the chances of your device being affected by denial of service attacks is to minimise the attack surface by shutting down all non-essential services and closing unneeded ports using firewall rules. If, for example, your device's web interface is only accessible via HTTPS on port 443, then standard HTTP on port 80 is not needed, and by simply blocking port 80, you can reduce your susceptibility to attacks on that port. Similarly, disabling debugging interfaces and other services that are not needed in a production environment can reduce the number of routes for an attack on your product.

The next step is to ensure that if your device does come under attack, it can recover and, ideally, still operate (perhaps in a degraded state) while the attack is in progress. For example, a smart home gateway device that has an external web-based API, which comes under attack from a large number of HTTP requests on this interface in a short space of time, could detect this spike in requests and disable the external interface. Although this would also block any legitimate external web API requests while the attack is ongoing, the device can still operate using its internal-facing interfaces, which is better than nothing, whereas if the external web interface was left up, the weight of traffic could have caused the device's CPU to be overloaded. After a period (for example, 1 to 3 hours) has passed, the device could tentatively reactivate the external web interface and either close it again if the attack is still ongoing or reopen it and resume normal operation if the attack has passed.

For routers and other devices which bridge traffic between networks, an additional, more sophisticated network monitoring and traffic control mechanism is needed to monitor and rate limit suspicious traffic originating from other devices inside the network, which could form part of an outgoing DDoS attack on a device on the internet.

// How Consult Red can help prevent Denial of Service attacks

When it comes to denial of service attacks, we can help with:

- **Analysing** where your device may be vulnerable
- **Advising** how to reduce your exposure risk
- **Designing and implementing** mitigation and resilience mechanisms
- **Network monitoring and traffic control** mechanisms for gateways and routers

// Summary

Device cybersecurity is a critical requirement for IoT and embedded device manufacturers. With increasing regulatory oversight through standards like EN 18031, compliance is essential not just to avoid legal and financial penalties but to ensure user trust and brand reputation.

This guide has explored six fundamental aspects of device cybersecurity: secure storage, secure software updates, secure communication, logging, access control and authentication, and denial of service prevention. Each of these components plays a vital role in safeguarding devices from ever-evolving threats.

Neglecting cybersecurity measures carries significant risks, ranging from data theft and operational disruptions to enabling large-scale cyberattacks such as botnets. Real-world examples, such as the Mirai botnet and security vulnerabilities in devices like smart bulbs and routers, demonstrate the importance of proactive, robust security design.
By implementing secure architectures and adhering to best practices, manufacturers can not only protect their devices but also contribute to the broader security of our connected world.

At Consult Red, we bring over two decades of expertise in embedded devices and cybersecurity, offering a range of advisory and compliance services tailored to your needs.

Whether you are looking to design secure systems from the ground up or ensure your existing systems meet regulatory standards, our team is here to help.

Together, we can ensure your devices are not just compliant but resilient, enabling innovation and security to go hand in hand.

Be confident in your device cybersecurity, with Consult Red.

Get in touch to learn more about our Device Cybersecurity services and how we can ensure the security of your products and their users.

Connected. Conformant. Compliant

//consult.red

Contact us to find out more

v1.0-Nov-24