IOT SECURITY TOP 20

DESIGN PRINCIPLES

LIST OF IOT SECURITY TOP 20 DESIGN PRINCIPLES

1. Provide a manual override for any safety critical operations.
2. Ensure parameters for which a disclosure could lead to the compromise of the system (such as secret/private cryptographic keys, passwords, etc.) are unique per device.
3. Test the system to be sure that it is free of known, exploitable vulnerabilities prior to release.
4. Allow for software updates, and ensure that these updates are cryptographically authenticated prior to installation and execution. Implement 'anti-roll-back' features to prevent the installation of previous, vulnerable versions of firmware.
5. Use industry standard security protocols, with 'best practice' defaults for any remote or wireless connections and authentication of connections to management services.
6. Do not store passwords in clear text.
7. Authenticate remote access and interfaces to system management functions with session and time-out limits.
8. Ensure that methods used to generate or negotiate cryptographic keys ensures that these keys are sufficiently random.
9. Detail all customer data – including audio, video, and personal details – that can be exported to cloud systems, or third parties. Provide an opt-in for such collection.
10. Only utilize industry standard cryptographic algorithms and modes of operation for any security protocol (such as firmware authenticity checking).
11. Provide the ability for users to enable on demand features they may not want, or they may use only intermittently.
12. Implement a power-on self-test that validates core functions and the integrity of firmware prior to execution. Implement a cryptographic chain of trust from the hardware during boot where possible.
13. Ensure that any system defaults – such as passwords, certificates, or keys – are forced to be changed prior to initial operation.
14. Ensure error messages or responses to invalid messages do not expose sensitive data.
15. Ensure that cryptographic keys are only used for a single intended purpose.
16. Implement 'least privilege' in all systems.
17. Implement protections to prevent the execution of data memory.
18. Do not allow for direct execution of externally provided commands, scripts, or other execution parameters that are not within the defined functions of the device.
19. Create and compile the firmware of the device such that it contains only code and systems required for the defined functions. Always remove/disable debug and development features in devices when creating production code.
20. Implement a vulnerability management program to regularly monitor and address security flaws in the product prior to release and throughout the agreed lifetime of the device. Include as part of this a process to inform and distribute patches to customers.

IOT SECURITY TOP 20

DESIGN PRINCIPLES

In a competitive market it is important to find a differentiator for your products, and often a competitive advantage is sought through adding 'smart' features and connecting devices to the customers' home network, or the wider Internet. However, as more features and connections are added, the security of such systems is often degraded.

Additionally, the security of connected consumer devices is not just a matter of interest to the customers who use these devices, it is increasingly becoming a matter of national interest. Malware that can take control of, and subvert the operation of, connected systems has been used to launch some of the largest attacks on the Internet that have ever been seen. The connected nature of these systems also means security must be considered for any 'apps' that run on separate systems (such as the consumer's phone), as well as 'cloud' services.

Of course, fitting security into the increasingly tight time and cost budget under which consumer devices are made can be difficult. Fortunately, there are some simple steps that can be taken to increase the security of connected systems, as outlined herein. These are organized with the most important requirements first, and it is recommended that these are addressed as the initial priority in any system, considered for all aspects (device, cloud, and app). To assist with the implementation of these items, guidance is provided for these requirements.

THE FIRST FIVE.

1 PROVIDE A MANUAL OVERRIDE FOR ANY SAFETY CRITICAL OPERATIONS.

Advanced features are great; when they work. However, sometimes these features can fail – often through no fault of the system itself. The local network of the customer can be poorly configured, or their Internet connection may be intermittent. In such cases it is important that any lack of functionality that this causes dose not result in a safety problem for the consumer. Examples may be providing a physical key back-up for a 'smart' door lock, or a manual override and safety limiting feature on an IoT thermostat.

2 ENSURE PARAMETERS FOR WHICH A DISCLOSURE COULD LEAD TO THE COMPROMISE OF THE SYSTEM (SUCH AS SECRET/PRIVATE CRYPTOGRAPHIC KEYS, PASSWORDS, ETC.) ARE UNIQUE PER DEVICE.

Passwords that are used for security sensitive features should be unique per device – a password that everyone knows is not much of a password. There are simple solutions to this – for example, secure passwords can be randomly generated and printed on the serial number sticker of the device. If the device is not going to be easily accessible during normal operation, consider providing such a sticker inside the manual/quick start guide, which can be taken out and put somewhere the user will not forget. Of course, there will always be scenarios where customers forget or lose passwords, so system recovery methods should be securely implemented as well (such as through a physical 'reset' button which enters a password recovery mode when pressed).

Any secret (symmetric) or private (asymmetric) cryptographic keys should also be managed as unique for each device / application. There are a few ways in which cryptographic keys can be determined even when they are apparently being stored and managed inside a secure device – often these methods are not 'worthwhile' for extracting keys from a single device, but if the same key is used in many thousands of devices, the economics of such an attack change considerably.

3 TEST THE SYSTEM TO BE SURE THAT IT IS FREE OF KNOWN, EXPLOITABLE VULNERABILITIES PRIOR TO RELEASE.

The software in connected devices, applications, and cloud services is often comprised of various software 'components' – existing software, such as open source code and third party libraries – as well as commonly used protocols and functions (such as databases). Each of these software components may have its own vulnerabilities, and it is

important that before releasing any system that a check is made for such known vulnerabilities. This can be achieved through various software utilities, as well as scanning services for cloud based systems – look for vendors who have passed the 'ASV' requirements of the Payment Card
Industry (PCI) which has done a good job of providing a minimum validation of such scanning vendors.

4 ALLOW FOR SOFTWARE UPDATES, AND ENSURE THAT THESE UPDATES ARE CRYPTOGRAPHICALLY AUTHENTICATED PRIOR TO INSTALLATION AND EXECUTION. IMPLEMENT 'ANTI-ROLL-BACK' FEATURES TO PREVENT THE INSTALLATION OF PREVIOUS, VULNERABLE VERSIONS OF FIRMWARE.

No matter how well software is designed or tested, there will always be bugs and vulnerabilities that are missed, or that are discovered after the product has been shipped. So, it is important to allow for the update of the software to ensure that it can be patched when any such bugs are found. However, if this is not carefully implemented, it can lead to additional vulnerabilities – where a 'bad actor' can install their own software into the device to prevent its normal operation.

To prevent this, software updates should be cryptographically authenticated. This is often achieved through the use of a digital signature on the system firmware image, which can be checked by the original firmware (or bootloader of the device) prior to installation. Using a digital signature based on a public key algorithm (such as RSA, or DSA) ensures that the devices themselves do not need the part of the key (the private or secret key) that is used to generate the authentication data.

If a symmetric key system is used instead – such as a (H)MAC – this secret key is needed in each device, meaning that access to the firmware in one device provides the ability to create valid firmware signatures for any device (unless there is a unique key per device, which is not feasible for IoT systems). So, public key cryptography is strongly recommended.

It is also important to include methods to prevent a bad actor from installing a previous version of firmware; to 'reinstate' any otherwise patched vulnerabilities – this can be done by including an increasing number ('monotonic') in each firmware release that is checked before install to ensure that the firmware version attempting to be installed is not older than the current version in the device.

5 USE INDUSTRY STANDARD SECURITY PROTOCOLS, WITH 'BEST PRACTICE' DEFAULTS FOR ANY REMOTE OR WIRELESS CONNECTIONS AND AUTHENTICATION OF CONNECTIONS TO MANAGEMENT SERVICES.

It is vital to protect communications which may be subject to interception or modification using an industry standard security protocol such as TLS or WPA2. Additionally, the exact use of the security protocol is also important – for example, it is possible to use TLS, and yet still be insecure if it is not configured correctly. These protocols allow for the authentication of connections, but only when implemented properly, and validation of certificates or 'certificate pinning' should be used to ensure that the connection is both secure and private.

Use the latest version of the protocol and software library, and monitor any changes so patches can be provided when problems are fixed. Where possible the use of the security protocol should cover all communications – regardless of if they are security related or not. This will make compromise of the system more difficult, as any bad actor must first compromise the security protocol before gaining access to try to compromise the device.

This requirement also covers the use of wireless protocols, where the use of security protocols such as WPA2 is equally important. It may be necessary to allow for customers to disable security features, but consider providing them guidance on why this is not recommended.

THE FINAL FIFTEEN.

6 DO NOT STORE PASSWORDS IN CLEAR TEXT.

This item may sound a bit unnecessary – if a bad actor has sufficient access to a device to read internal storage, is there really any value in further protecting passwords (which may simply allow / protect such access)? But it is common knowledge that people tend to reuse passwords, so a password extracted from a single compromised system may be useable on other systems, accounts, and online services. Instead, passwords should be stored using a 'one-way', and computationally intensive, algorithm such as BCrypt. This item should be considered part of the first five requirements for any cloud environments.

7 AUTHENTICATE REMOTE ACCESS AND INTERFACES TO SYSTEM MANAGEMENT FUNCTIONS WITH SESSION AND TIME-OUT LIMITS.

Access into a system – whether to a device from outside the 'local' network, or into a cloud system – should be authenticated to prevent access by unauthorized parties. For back-end, or 'cloud' based systems, consider implementing two-factor security measures such as those provided by FIDO compliant tokens and software. SMS based One Time Passwords may be implemented, but these are under increasing attack and newer – more secure – methods are recommended.

For connection into local networks from an external system, consider VPN connections or routing data through a TLS connection 'tunnel' that can provide the authentication (and where certificate validation / pinning is performed as required in other requirements). Local connections may require only a password, but may also provide authentication through physical proximity (e.g., through a bluetooth / NFC connection, or through a physical button that must be pressed to access the service). If localized wireless is used, ensure security best practice is followed.

Debugging interfaces, such as JTAG, In Circuit Emulation, etc., should always be disabled on production devices. Although accessing such interfaces requires local physical access, if they are enabled it greatly simplifies the work of a bad actor in developing attacks.

These management services may have a range of functions, from turning a camera to point in a different location, through to loading new certificates, firmware, or other security related features. If a device has multiple features that can be accessed through remote administration features, consider providing different levels of authentication so that it is possible to isolate security related features from 'user' features.

Additionally, provide an absolute limit on the time during which any administration features can be accessed during any one session. This prevents people from 'forgetting' that they have left such features on.

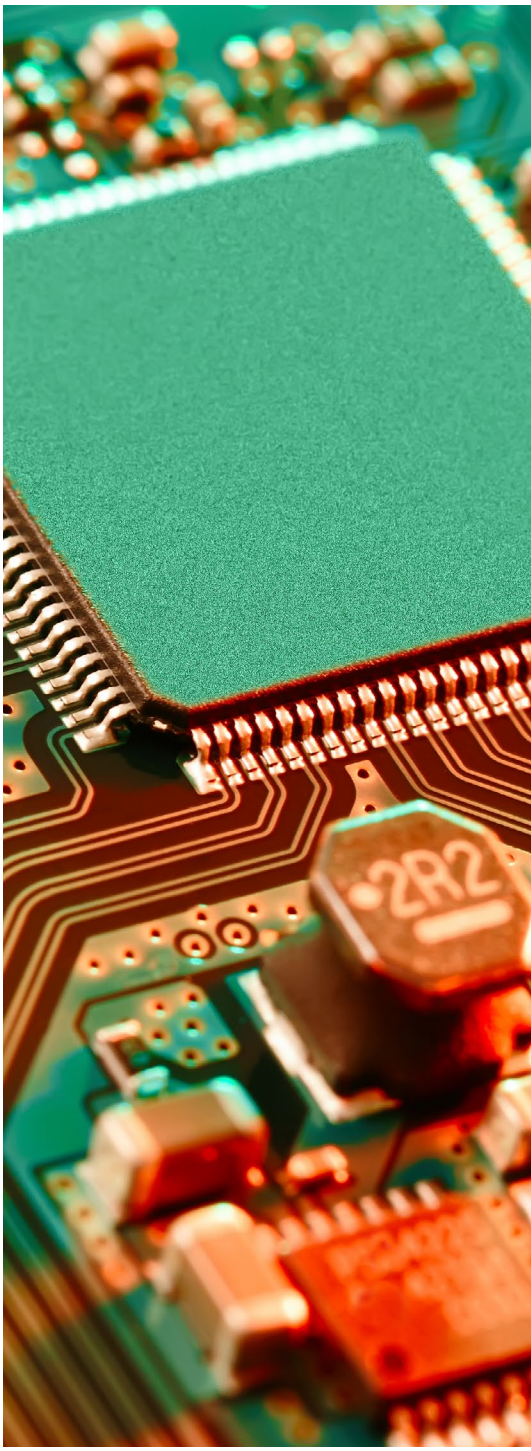
8 ENSURE THAT METHODS USED TO GENERATE OR NEGOTIATE CRYPTOGRAPHIC KEYS ENSURES THAT THESE KEYS ARE SUFFICIENTLY RANDOM.

Generating good random numbers is actually very hard, and the use of poor random numbers has been the source of many vulnerabilities. The root cause usually is based around two issues: Firstly, computing systems are deterministic – the same program, given the same inputs, will produce the same output every time. Secondly, we as human beings are not very good at spotting a lack of randomness.

How are good random numbers generated? Do not rely on input solely from an embedded device. It is often best to take input from various sources; one of these may be a standard 'random' function, but also other sources such as the least significant bits of an A/D input, network traffic timing, hard-disk seek timing, millisecond data from a real time clock source, user input timing,

etc. This can then be combined, and provided as a seed to a pseudo-random number generator, such as those outlined in NIST SP 800-90A.

Cryptographic keys should not be directly generated from passwords; it is better to use the password to enable access to the use of a key that has been generated through a strong random number process.



9 DETAIL ALL CUSTOMER DATA – INCLUDING AUDIO, VIDEO, AND PERSONAL DETAILS – THAT CAN BE EXPORTED TO CLOUD SYSTEMS, OR THIRD PARTIES. PROVIDE AN OPT-IN FOR SUCH COLLECTION.

Many systems provide for advanced features through processing in a 'cloud' environment. However, not all consumers are aware of the collection and export of this data, and privacy concerns may exceed the customer desire for the features that are provided. It is important that consumers are given control of the data they provide outside their own networks through clear disclosure and an 'opt-in' rather than 'default-on' process.

10 ONLY UTILIZE INDUSTRY STANDARD CRYPTOGRAPHIC ALGORITHMS AND MODES OF OPERATION FOR ANY SECURITY PROTOCOL (SUCH AS FIRMWARE AUTHENTICITY CHECKING).

Cryptographic algorithms are complex, and these days it is reasonable to say that there is no one person who is able to say that any particular algorithm is secure. The only way we can have any confidence in an algorithm is to subject it to study from a host of experts over many years – and even then, new research and findings may come along and reveal flaws hitherto unfound.

Therefore, it is strongly recommended that only cryptographic algorithms, key lengths, and modes of operation that have undergone such study and are generally accepted to be secure are used. A great reference for this is NIST SP 800 57, which basically defines Triple DES, AES, RSA and Elliptic Curve Cryptography (ECC) as the only algorithms for use.

The 'mode of operation' is also important – this is the way in which the cryptographic algorithm is actually used to encrypt the data or provide authentication. It can be easily overlooked that using a simple mode of operation such as Electronic Code Book (ECB) can actually expose patterns in the plaintext data, and may not achieve the security that is intended when implementing the encryption.

11 PROVIDE THE ABILITY FOR USERS TO ENABLE ON DEMAND FEATURES THEY MAY NOT WANT, OR THEY MAY USE ONLY INTERMITTENTLY.

All software has bugs, and many of these bugs expose potential security vulnerabilities. The more software a product has, the more bugs it is likely to have. Of course, these bugs can be minimized through testing, patching, etc., but many bugs will always remain unfound and unpatched until an exploit is released. In contrast to this, many products rely on an extensive feature-set to differentiate themselves in a commercially aggressive market. To balance these conflicting requirements, it is ideal to have some of the more advanced features disabled by default – so that a system can remain secure even if an exploit is found.

For example, remote access, wireless pairing, advanced functions (email, printer interfaces, etc.) may be provided but disabled by default and provided with a 'timed' access feature so that the customer can enable the feature only for a certain period.

12 IMPLEMENT A POWER-ON SELF-TEST THAT VALIDATES CORE FUNCTIONS AND THE INTEGRITY OF FIRMWARE PRIOR TO EXECUTION. IMPLEMENT A CRYPTOGRAPHIC CHAIN OF TRUST FROM THE HARDWARE DURING BOOT WHERE POSSIBLE.

Ideally firmware should be validated on each boot to make sure it has not been altered since being installed. This can be trivially achieved when there is a signature across all firmware anyway – which may be the case if the system runs a simple function executive – but is significantly more difficult when it is complex OS such as Linux. Validating all of the different files, scripts, etc. that go into ensuring that a complex OS runs correctly is complicated. Potential solutions include the validation of the device bootloader (from a hardware root of trust, which requires support in the processor being used), then use that bootloader to validate an OS 'image' which is unpacked and installed. Of course, this all takes time – but it is useful to prevent things like Ransomware which may look to install software that renders a device inoperative until a ransom amount is paid.

13 ENSURE THAT ANY SYSTEM DEFAULTS – SUCH AS PASSWORDS, CERTIFICATES, OR KEYS – ARE FORCED TO BE CHANGED PRIOR TO INITIAL OPERATION.

System defaults should be avoided where possible, but such defaults are often necessary. For example, a default may be required to allow for the 'boot-strapping' of the system for the first time. This can be acceptable, but this value should be forced to change from this default as part of the overall setup.

Ultimately, defaults should only be considered for certificates and other such items which may be required to be present for normal operation, but which must be changed by the user before installation and operation. This also covers any 'test' values that may be in the firmware – never leave these in a production system!

14 ENSURE ERROR MESSAGES OR RESPONSES TO INVALID MESSAGES DO NOT EXPOSE SENSITIVE DATA.

When a system is incorrectly accessed, it is common to return some form of error message to indicate what error has occurred. Such messages can easily reveal sensitive information about a system, and must be very carefully implemented. Consider returning only good/bad indications from production systems, at least until some form of debug state is activated (through an authenticated access, of course!). Never return details of any decrypted data if there is a failure of some sort – even to note what error has occurred in the decryption and validation – simply reject the connection in such cases.

15 ENSURE THAT CRYPTOGRAPHIC KEYS ARE ONLY USED FOR A SINGLE INTENDED PURPOSE.

Key management – the way cryptographic keys are used – is extremely important. The cryptographic algorithm is only a part of the security; the way the algorithm is used, and the way the cryptographic keys that it requires are used, are also vital.
To prevent compromise it is good practice to use cryptographic keys for only one purpose. Data encryption keys, should be used only for encrypted data. Keys that are used to secure passwords, for example, should be different. Do not mix keys (or key pairs) between uses for encryption and authentication. Each key should have a unique use.

16 IMPLEMENT 'LEAST PRIVILEGE' IN ALL SYSTEMS.

Modern processors often have different levels of 'privilege' available – that is different levels of potential for access to memory and other resources. These features can be used by the software to help secure assets within the device, by ensuring only software with the right 'privilege' can access this. Often, the interface to the hardware level privilege controls of a processor are managed by the Operating System (such as Linux), but they can also be controlled even if a device does not utilize a complex OS (e.g., if it uses a 'simple' function executive, or cut-down RToS, instead).

Whenever possible keep the use of 'root' or 'supervisor' level privileges in embedded systems to an absolute minimum, and maintain secret data – such as cryptographic keys – at a highest level of privilege (i.e., hardest to access). Similar rules hold true for 'apps' and cloud based systems; keep the use of elevated privileges to a minimum and try to isolate functions as their own 'user' or privilege set.

17 IMPLEMENT PROTECTIONS TO PREVENT THE EXECUTION OF DATA MEMORY.

Many remote attacks aim to gain execution of code supplied by the bad actor, which is provided through a specific vulnerability. Although avoiding all possible vulnerabilities in code is almost impossible, mitigating the potential for remote code execution through any coding vulnerability is possible through a number of different methods.

Many processors provide 'no-execute' functions, which can mark specific areas of memory such that they cannot be used to reference directly executable code. Alternatively, some processors may even provide entirely different code and data memory segments, which makes such direct code injection attacks impossible (although other types of attacks may still be performed).

In addition to direct hardware protections, there are also other protections that can be applied at the software level – some of these can be provided by the Operating System itself, and some may be applied through compiler settings when creating the object code to load into the device. Determine what protections are possible on the various components of any system, and implement as many as technically and operationally feasible.

18 DO NOT ALLOW FOR DIRECT EXECUTION OF EXTERNALLY PROVIDED COMMANDS, SCRIPTS, OR OTHER EXECUTION PARAMETERS THAT ARE NOT WITHIN THE DEFINED FUNCTIONS OF THE DEVICE.

In addition to direct code execution, there are also other ways that a bad actor may gain access to a system if there are other interpreters or execution environments available. For example a system may allow for non-native code to be executed, such as Javascript, which can then lead to potential vulnerabilities. Whilst this does not mean it is always necessary to disable things like Javascript completely, it is worthwhile understanding the need the system has for this type of functionality, as if it is included it will often require more complex security solutions to maintain the overall security posture of the system.

19 CREATE AND COMPILE THE FIRMWARE OF THE DEVICE SUCH THAT IT CONTAINS ONLY CODE AND SYSTEMS REQUIRED FOR THE DEFINED FUNCTIONS. ALWAYS REMOVE/DISABLE DEBUG AND DEVELOPMENT FEATURES IN DEVICES WHEN CREATING PRODUCTION CODE.

The larger the body of code, the more chance there is that there are undiscovered security flaws. Therefore, it is prudent to remove as much code as possible, to limit the chance that a flaw discovered after shipping of the product will require patching or other mitigations. This includes code that may not be executed 'normally' – even if the code is not used, having it within the device can lead to security problems down the road.

For similar reasons, it is essential to remove debug and development code from the device prior to shipping. This is often done with isolating 'ifdef' statements, which can automatically remove such features during compile time. Although it is understandable to want to have features in the code in case there are problems in production, it is often more likely that such features will become a source of vulnerability, as they provide access and information that would not normally be provided in the end-user environment.

20 IMPLEMENT A VULNERABILITY MANAGEMENT PROGRAM TO REGULARLY MONITOR AND ADDRESS SECURITY FLAWS IN THE PRODUCT PRIOR TO RELEASE AND THROUGHOUT THE AGREED LIFETIME OF THE DEVICE. INCLUDE AS PART OF THIS A PROCESS TO INFORM AND DISTRIBUTE PATCHES TO CUSTOMERS.

Security is a moving target, and no system will ever be 100% secure. As new vulnerabilities and attack methods are released, it is important to have a program to ensure that systems – both new and existing – are not vulnerable to these security flaws. This is only possible when there is a management endorsed and enforced program to ensure the on-going monitoring and update of system security. The ability to install patches into systems is of no value if the patches are not created and made available for the customer systems to install.

For further information on IoT security or to download this paper, please scan the QR code:

Or contact us at **TRANSACTIONSECURITY@UL.COM** or visit **UL-TS.COM**

ul-ts.com/IOTsecurity

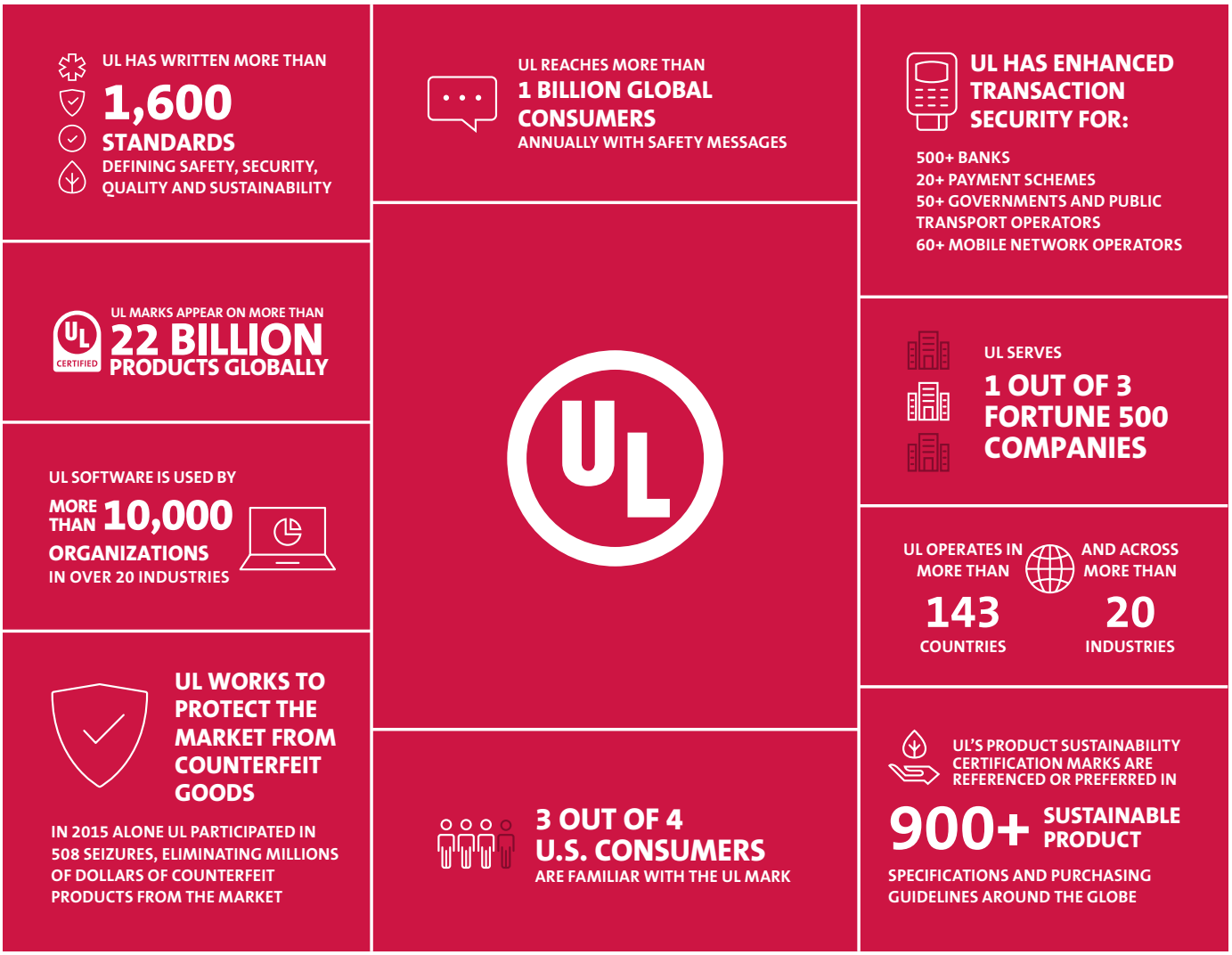
ABOUT US

UL fosters safe living and working conditions for people everywhere through the application of science to solve safety, security and sustainability challenges. The UL Mark engenders trust enabling the safe adoption of innovative new products and technologies. Everyone at UL shares a passion to make the world a safer place. We test, inspect, audit, certify, validate, verify, advise and train and we support these efforts with software solutions for safety and sustainability.

UL's Transaction Security division guides companies within the mobile, payments, and transit domains through the complex world of electronic transactions.

UL is the global leader in safeguarding security, compliance, and global interoperability. Offering advice, training, compliance and interoperability services, security services, and test tools, during the full life cycle of your product development process or the implementation of new technologies.

UL's people proactively collaborate with industry players to define robust standards and policies. Bringing global expertise to your local needs. UL has accreditations from industry bodies including Visa, MasterCard, Discover, JCB, American Express, EMVCo, UnionPay International, PCI, GCF, GlobalPlatform, NFC Forum, and many others. To learn more about us, visit UL-TS.com.



UL-TS.COM

SCAN FOR DIGITAL VERSION

ul-ts.com/IOTsecurity