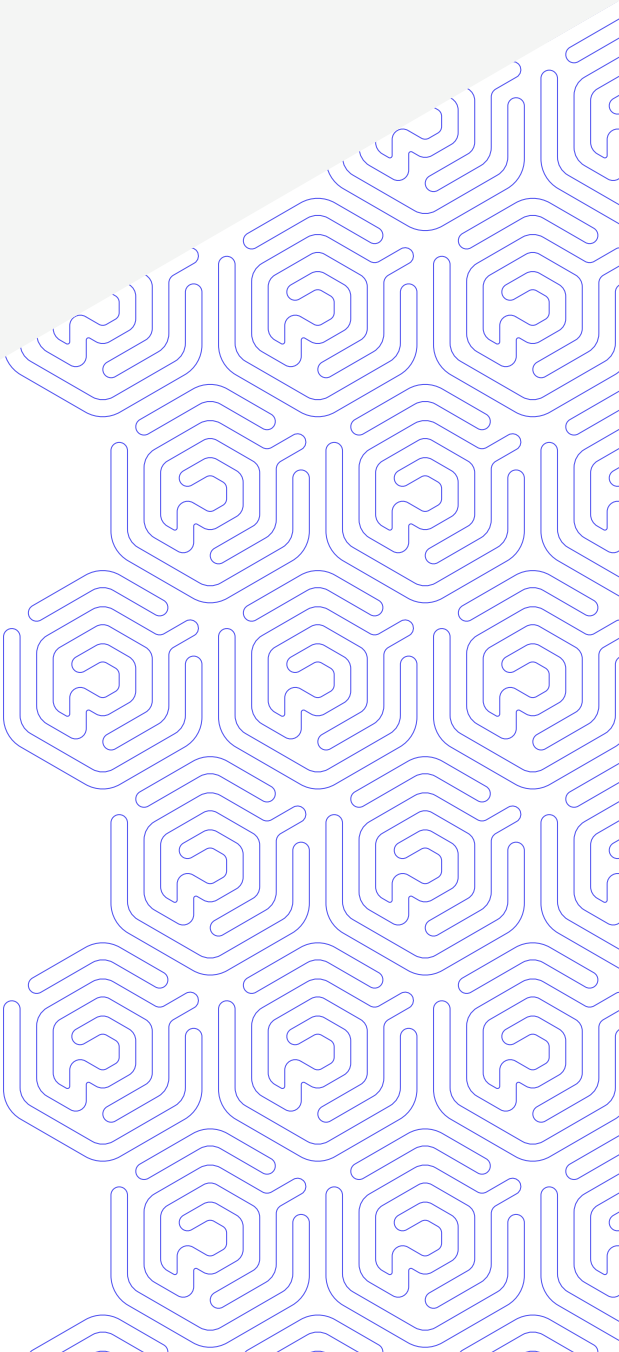
WHITE PAPER

The 9 Most Prevalent Internet of Things Security Issues and How to Avoid Them



praetorian

praetorian

Observations from over 20,000 hours of IT security assessments

INSIDE THIS REPORT

- This report describes the top Internet of Things (IoT) security issues and how to avoid them.
  - Authorization Issues
  - Platform / Vendor weaknesses
  - User Information Leakage
  - Hardcoded Secrets
  - Outdated Third-Party Components
  - Insufficient Account Protections
  - Resource Enumeration
  - Debug Services Enabled
  - TLS Weaknesses

In the past two years Praetorian has performed more than 20,000 hours of IoT security assessments. While every assessment is unique, there are several types of vulnerabilities that we see occur over and over, across widely different products and use cases. We first noticed these patterns in 2016, and in early 2017 we published a white paper documenting the most common issues that we observed at that time. In the year and a half since, our perspective on the most common issues has evolved as the IoT marketplace has evolved, but we still see many of the same issues. What's striking about this is just how common these issues are. The 20,000 hours of assessments included every portion of an IoT ecosystem – apps, APIs, devices, portals, platforms – in different combinations. They also included a wide swathe of industries and use cases: consumer appliances, industrial automation, connected vehicles, connected locks, power and SCADA, medical devices, IoT platforms, ATMs, building automation, and kiosks. Despite all of these differences, the same nine issues appeared again and again.

These commonalities reaffirmed the observation we'd made in 2017: Development teams, regardless of their industry and product, struggle with similar problems.

Top IoT Security Issues

DAs a caveat, this analysis is not statistically based. Very few of our assessments were similar in terms of scope, so we cannot make meaningful apples-to-apples comparisons across them. We consequently cannot rank order the issues, but a raw count of issues across assessments identifies the issues discussed here as the most common. It is also worth noting that the risk of these and other issues will always be context dependent.

A low-powered device that only collects telemetry and sends it back for processing is different from a device that records video or audio in and around people's homes. In some situations, the issues described here may present minimal or no risk. It is always important to evaluate each unique device and use case for the ramifications of potential abuse. What in some situations are benign use cases can be avenues for attacks in others, such as the case reported by The New York Times highlighting abusers using smart home devices to harass victims.

> *Virtually any Android, Linux, or Windows device that hasn't been recently patched and has Bluetooth turned on can be compromised by an attacking device within 32 feet. It doesn't require device users to click on any links, connect to a rogue Bluetooth device, or take any other action, short of leaving Bluetooth on.*

Dan Goodin

SECURITY EDITOR, ARS TECHNICA

1. Authorization Issues

Over the past year, we have discovered an increasing number of authorization issues, or locations where a user's permission to perform a specific action is not checked by the application. This allows an attacker to perform actions for which they do not have permission. These can include things like administrative actions or impersonating other users.

Architectures made up of many micro services are particularly prone to this problem. We've seen multiple cases in which the architecture performs authentication checks on all users, but not authorization. For example, if an authenticated user makes a request directly from a service, the service will perform the requested action.

Another common root cause has been mistakenly exposed endpoints. In these cases, an endpoint was intended to be internal only and servicing requests only from other trusted services. However, when that assumption of trust is broken due to a misconfiguration, these sorts of endpoints will perform an action from any requester.

Solution

In some instances, a centralized authorization service is the best solution. This functionality is integrated with the authentication service. Regardless of where authorization occurs architecturally, for each request the system needs to verify not only whether the user is authenticated, but whether the user has permission to perform the requested action. In many cases, this is as simple as verifying that the user has requested an action affecting their own account or resources.

2. Platform / Vendor Weaknesses

There's a growing market of service providers that provide a backend and SDKs for IoT device developers. These offerings can decrease development costs and time to market, as well as reducing the administrative overhead of architecting and maintaining a cloud backend.

These vendors face market pressures to make their platforms as accessible and friendly as possible. That can place business decisions at odds with security due to security vs. usability tension.

There are also additional complications from integrating components made by different development teams at different times. It's a security maxim that problems often occur at boundaries. In this case there are added complications to ensuring your developers are making the same assumptions as the platform's developers, which leads to underlaps and security flaws.

Solution

The best fixes for these issues are in process changes, not technical. Customers should be proactive in asking vendors for a third-party assessment or perform their own. Customers should also ask vendors for information on their secure development life cycle as part of their evaluation process. Don't be afraid to ask hard questions and challenge your vendor on the adequacy of their responses.

Buyers should also ensure their security or risk management organization(s) is involved in the procurement process. Those stakeholders will be best able to evaluate the vendor's level of risk.

Platform providers should provide customers with clear documentation on their security model and features. Ensure customers are clear on which security functions you will manage and which they are responsible for. Finally, try to make design decisions that default toward more security. That ensures customers taking a less secure route do so deliberately, and not through omission.

> Customers should also ask vendors for information on their secure development life cycle as part of their evaluation process.
>
> Don't be afraid to ask hard questions and challenge your vendor on the adequacy of their responses.

3. User Information Leakage

We've increasingly seen issues in which APIs leak customer information. Most commonly, the leaked information yielded is email addresses, but server responses often include related information. We've seen profile pictures, full PII (name, address, email, phone, social), and account details leaked this way. In several instances the affected API endpoint was never intended to be externally facing. For example, we've seen endpoints intended to be restricted to customer service representatives that were accidentally exposed.

There's immediate brand and reputational risk from these sorts of vulnerabilities. These details are easily understood by both customers

and media. This leaked information can also be a stepping stone to further attacks, such as enabling social engineering attacks against your customers or seeking to compromise leaked customer accounts.

Solution

The solution to this problem is partly process and partly technical.

As a process control, identify all of your endpoints that are intended to provide customer information. For each endpoint in this inventory, verify that it has appropriate access controls. In most cases in which we've discovered this issue, the root cause was a simple misconfiguration that exposed the offending API.

For those endpoints that do need to be exposed, perform a scrub as part of a QA process to ensure they only provide intended info. Another common root cause is for server responses to include more information than is displayed in the UI, leading to the excess information to go unnoticed.

Ensure any endpoints that do provide user information include authentication and check for authorization, for example, returning a given user's info only to that user or an admin.

4. Hardcoded Secrets (Keys, Credentials)

"Hardcoded secrets" refers to things like default credentials, keys, or passwords that are set in source code. Since they are set in the code itself, these secrets are shared between products. Once a product is released, attackers can often recover these secrets from devices in their possession.

In the case of default credentials, they aren't treated as secrets, and instead are included in documentation. Since the secret is shared, a compromise in one location means a compromise of that secret for all devices.

These default values are often not changed by users.

One final consideration is that when secrets are shared in source code, they are typically included in the source code repository for perpetuity. This means that any malicious insider with access to the repository will have access to that key.

> *Starting on January 1st 2020, any manufacturer of a device that connects "directly or indirectly" to the internet must equip it with "reasonable" security features, designed to prevent unauthorized access, modification, or information disclosure."*

Adi Robertson
SENIOR REPORTER, THE VERGE

The exact impact varies based on the circumstance, but when these secrets are compromised, the impact is often significant. Compromised credentials can create opportunities for trivial remote compromise. We've recently seen attacks automated and used by worms to spread across IoT devices.

If the device included something like hardcoded API keys, an attacker who gains the keys can then authenticate with that API and launch further attacks against backend infrastructure.

Solution

Use unique secrets for each device. These should be a value that is accessible to the user but not broadcast in any fashion (i.e., Mac addresses and device names aren't appropriate).

Store secrets used by web services in environmental variables. Don't include them in source code.account or resources.

5. Outdated Third-Party Components

Preventing the use of outdated services or libraries tends to be an especially vexing problem for device manufacturers. Managing an inventory of which third-party components are in each version of each device is not trivial administrative overhead.

It's also a challenge to maintain when time pressures push for prioritizing activities that will contribute directly to bringing a product to market.

The result of this is that many products contain vulnerabilities due to the third-party services that were included. The impact will vary based on what sort of issue is present, but it can include severe issues like remote code execution, authentication bypasses, or information leakages.

Solution

Maintain an inventory of third-party components – libraries, services, binaries, etc. Monitor for new vulnerabilities and patches for those components. Include those in your own product.

Consider having a process to rapidly test and release an emergency patch, outside of a typical patch cycle.

Build your update functionality so that it's automatic and requires little or no user interaction.

Consider compiling custom binaries that include only the functionality you need. You'll be immune to vulnerabilities that don't affect the functionality in your build.

> Build your update functionality so that it's automatic and requires little or no user interaction.

6. Insufficient Account Protections

This issue appears across our assessments. It's not unique to IoT ecosystems. The problem often results from several lower risk issues appearing together – weak password policies, lack of lockout, and user enumeration (discussed separately previously).

When two or more of these three issues are present, attackers can launch automated brute-force attacks. These can be targeted (thousands of password attempts on a single account) or wide (guessing 1 - 3 common passwords on thousands of accounts). Due to unfortunate statistics about people's password practices, this almost always succeeds in production environments. Another common tool for attackers is searching breached password dumps for users, and then testing for reused passwords.

When successful, the result is compromised user accounts.

Solution

Implement account AND IP-based lockouts or throttling.

Inform users by email when their account is accessed from a new device. If they are not the one who logged in, this gives them notification to reset their password and potentially limit their exposure.

Consider using a blacklist to prevent users from setting their passwords to a vulnerable value. The blacklist can be based on lists of the most.

7. Resource Enumeration

Often a system needs some sort of schema to refer to ecosystem components other than users – things like devices, stored files, or organizations. For these instances there's usually a concept of a resource identifier, which is often something like UIDs, UDIDs, etc.

When these values are sequential or otherwise predictable, it can facilitate attacks on the ecosystem. This issue is not an explicit problem on its own, but knowledge of resource identifiers is often a prerequisite for other attacks. For example, if an authorization flaw is also present, an attacker may be able to harvest PII by sequentially requesting other users' resource IDs. Denying an attacker the knowledge they would need to perform an attack can prevent it altogether or limit its blast radius.

The most easily identifiable examples are sequential resource identifiers. We've also seen things like hashed timestamps or usernames used as unique identifiers. While these look random, a determined attacker can often reverse engineer the values, and thereby predict or enumerate resource identifiers.

Solution

Use random, non-deterministic values for resource identifiers. Many frameworks provide built-in functionality to support this.

8. Debug Services Enabled

We often find production devices that have debug services enabled through things like JTAG, UART, serial, or USB. The purpose of these services is to enable developers or technicians access either to control the device directly or receive debugging information.

For an attacker, this often presents a quick path to compromise of the device if we can compromise the debug service. This then provides an

avenue to pull firmware, identify secrets, and generally identify other vulnerabilities in the device. Consequences can include compromise of intellectual property, loss of control of deployed devices, or providing attackers with a foothold to attack your infrastructure through the compromised device. It can be especially damaging in cases where there are also shared secrets, as an attacker typically can compromise those secrets the debugging access. By sharing other secrets, they enable others to compromise your devices.

We have also seen cases where physical connectors for debug services aren't included on the devices' circuit boards, but we are able to connect by soldering on our own interfaces.

Solution

Inventory all the mechanisms and services you've enabled to facilitate interaction with a device during development – UART, JTAG, serial, USB, SD card readers, adb, telnet, ssh, qcon, etc. Plan for steps near the end of the development process to disable all such services that are not intended for use in the deployed system, and then test those changes. Consider compiling and using custom binaries that include only the required functionality.

For any services that are needed in production devices, ensure that they run at the lowest required privilege level.

Removing physical interfaces is insufficient. Ensure debugging services are disabled at the hardware or software level as appropriate.

9. TLS Weaknesses

Transportation Security Layer weaknesses are endemic across our assessments. We frequently see TLS servers configured to use old versions of protocols, weak algorithms, or weak keys. These poor

configurations can allow a suitably positioned attacker to encrypt sensitive communications. The sophistication required of the attacker can range from nation-state to moderately skilled, depending on which misconfigurations are made. Happily, this problem typically is easy to change with a configuration tweak.

Another common issue is simply failing to encrypt sensitive information. This again leaves communications open to eavesdropping by a suitably positioned attacker.

Solution

Unless you have a power or processing limitation, default to encryption with TLS. We discourage you from trying to implement your own encryption. Instead, rely on one of the many mature open source libraries or services, or core OS functionality, to implement transport layer security.

Ensure you are using appropriately future-proof protocols, algorithms, and key sizes. Understanding all of the nuances is hard. Mozilla offers a great service to provide secure TLS configurations based on your use case.

We Are the Security Experts

Praetorian provides a suite of security solutions that enable today's leading organizations to solve cybersecurity problems across enterprise IT assets, software development teams, and IoT product portfolios. Praetorian's exceptional reputation is supported by its talent density, "customer first" mentality, success-oriented culture, and drive for innovation.

Gain From Our Team's Deep Security Expertise Across:

- Internet of Things
- SaaS Applications
- Mobile Applications
- Cloud Infrastructure
- Corporate Infrastructure
- Critical Infrastructure

Read to Get Started?

We provide deep security expertise to teams in today's leading organizations.

Are you ready to discuss your next security initiative?

Contact us at (866) 477-1028

www.praetorian.com

sales@praetorian.com