December 2023

It’s Time to Elevate.

Moving the Discussion Forward on Secure by Default.

AUTHORS
James Dolph
David B. Cross

Contributions from the IT-ISAC CSaaS-SIG

CSaaS SIG
An IT ISAC Community

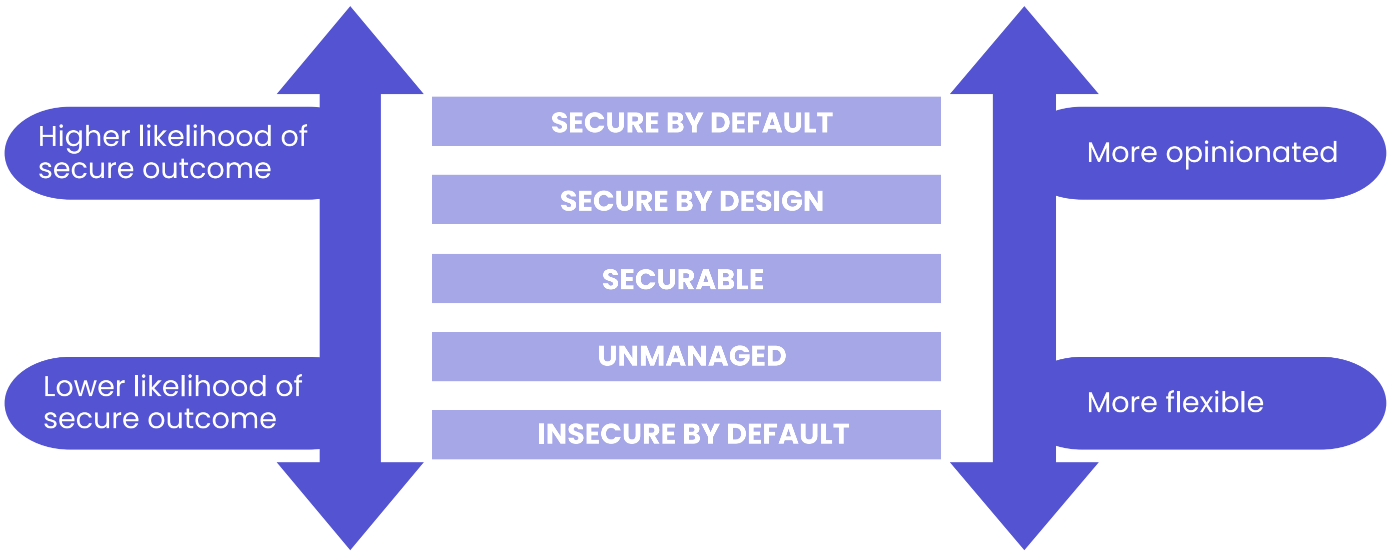
INTRODUCTION

The complexity of systems and software is on the rise, along with the acceleration of attacker sophistication. Making it harder and harder for technology implementors and security teams to understand what they need to do to result in more secure outcomes. This applies to both the consumer realm and the business world. Providers of software and services can play an important role in cybersecurity by actively designing systems that ensure security is the default configuration, and making the software less secure takes extra effort by the end user. Rather than the end user needing to take extra steps to enable security, making something less secure should take extra effort. SaaS providers, in particular, have an opportunity to leverage the scale benefits that fundamentally drive their success to improve the security outcomes of their customers, critical infrastructure, and the internet as a whole.

There is no shortage of articles, blogs, and writings about secure by default, secure by design, and similar concepts recently. However, many of these seem to miss the real point and end up sounding like a hardening guide, a how to build a secure development lifecycle, or a generic laundry list of things that would make software better. Like a lot of good ideas, (think zero trust) "Secure by Default" runs the risk of being used as a marketing fodder rather than a clear set of principles that can be applied to the user experience of security and the overall design of software and systems. What we need is both a clear point of view on what secure by default is and what engineers, user experience designers, security engineers, and providers of technology should do to increase the likelihood of secure outcomes for their consumers and technology implementers. This article is an attempt to provide a point of view that can move the discussion forward.

WHAT IS SECURE BY DEFAULT?

Let us start by defining what secure by default is and is not. Secure by default is defined as a system or software that is configured to the most secure settings possibly right "out of the box". It's helpful to describe secure by default in the context of a hierarchy between insecure by default and secure by default. The closer to secure by default, the higher the likelihood of secure outcomes and the more opinionated the functionality becomes. Part of being secure by default is understanding how functionality will be used in a secure way and very clearly defining the most secure implementation or use.



Secure by Default

The full user experience of secure use and secure implementation has been considered from the start and an opinionated set of default configurations are in place. It's not assumed that the user or implementor knows how to deploy security appropriately or will do the right thing. This includes consideration for a broader set of security capabilities that may be needed beyond the immediate functionality such as logging, alerting on insecure configurations, updatability, preventing breaking changes, isolation, defense in depth, etc.

The user or implementor must take explicit and deliberate action to use, operate, and configure the software/functionality in an insecure manner if insecure use is even allowed. It should be considered if an insecure state or usage pattern is actually needed or supported. This might result in taking insecure choices away or providing additional mitigating functionality for less secure use cases. It should be obvious to the user or implementor both that a security choice is being made and what that specific impact entails.

It is preferred to make the configuration descriptive so that users do not inadvertently make an insecure choice where a choice is given (e.g. UI toggle says something is insecure or dangerous, code methods use words like dangerously or insecurely, and system configuration properties use words like insecure or dangerous). When there are options or configurations to change, reduce, or eliminate protection, that means security will be reduced. It also creates an opportunity to accidentally or maliciously reduce security.

Finally, insecure choices should continue to be visible and "noisy" - ensuring the user or implementor does not forget their change of security. Often an insecure choice is made with the intention of fixing it later and that choice can be easily forgotten or buried in other technical debt. A final consideration should be building feedback loops like field tests with users or implementors with measures to determine if the functionality leads to the expected secure outcomes.

Secure By Design

Secure by design is a prerequisite for secure by default, but it is not as opinionated, explicit, or comprehensive. There are defined ways to secure functionality or use products and it has been considered during the design/spec process. What can go wrong has been thought through and solutions are pre-defined rather than bolted on. Insecure choices may be available or the default behavior may not be secure. It is assumed that the user or implementor will want a secure outcome, know what to do, and will do the right thing.

A common and recommended approach that organizations follow for driving secure design is using Adam Shostack's Four Question Framework for threat modeling:

- What are we working on?
- What can go wrong?
- What are we going to do about it?
- Did we do a good job?

Securable

Securable means security has at least been considered and there are ways to secure functionality. It is often bolted on or requires additional software, libraries, or capabilities. Sometimes it is complex (e.g. 10 lines of code to check access control) and it has not necessarily been thought about during the design, but you can accomplish security.

Unmanaged

Unmanaged means you are on your own to figure it out either way. There may be communal knowledge on ways to get to a secure outcome, but the functionality maintainers don't actively manage security or security capabilities.

Insecure by Default

Insecure by default means that security capabilities are intentionally not considered. This may be because it is disruptive, creates performance issues, or there are point-in-time assumptions - for example, it will never be on the internet.

It is no surprise that there is a considerable amount of effort and thought required to be secure by default. In the same way that the user experience of software generally has become a higher priority and systematized, ensuring the user experience of security will become more common as well. It's also reasonable to assume that there may be tradeoffs in ease of use or integration in highly secure by default products or services. This can be mitigated to some extent by actively considering the user experience and supported use cases. At the end of the day, secure by default involves removing dangerous choices and making secure choices the default or only choice.

EXAMPLES OF GOOD WORK

Secure by default is not a new idea in the world of software and services, but until recently it has been more of a theoretical concept that had minimal examples in the technology world. Many open source projects, commercial software providers, and SaaS providers have internalized the responsibility and opportunity that they have to improve security outcomes by considering the security user experience and implementing secure by default functionality. There are some examples that exemplify the spirit and execution of secure by default well in the section below. It is important to note that there is a lot of good work out there and this is not intended to be an exhaustive list, just a few examples to illustrate the concepts.

Amazon S3 - Block Public Access by Default

This year Amazon took an opportunity to change a long-standing recommendation related to public access of S3 buckets and make it secure by default. Starting this spring newly created S3 buckets have Block Public Access enabled which makes granting public access an intentional decision and potentially prevents inadvertent misconfigurations. This is a great example of taking lessons from both the use and the intended outcome and introducing an opinionated secure by default setting. It's not easy to retrofit existing settings to be secure by default particularly for popular services because of the complicated change management, enablement, and communication required. However, it appears that Amazon made a bet that this would be impactful to its users and would make a long-standing recommendation more likely to stick, thus leading to better security outcomes. It also shows that there is a potential path for other products and services to incrementally introduce secure by default not only for new functionality but also for existing products and services.

React - Dangerously Setting the Inner HTML

Cross Site Scripting (XSS) is both a versatile attack vector and a challenging problem to comprehensively solve in web applications. It has been either referenced or explicitly listed on the OWASP top 10 since the beginning of the top 10 project. There have been many attempts to solve the issue in browsers, by guiding developers to take some action (i.e. making frameworks and apps Securable), and by building complex testing tools and web application firewalls.

With the increased popularity of client-side frameworks, the React team took an opportunity to creatively introduce a secure by default posture for the framework that prevents the developer from inadvertently including arbitrary unsanitized HTML into a page that might lead to XSS. In the default case when a developer includes HTML into a page, it is encoded or otherwise sanitized to prevent inadvertent XSS vulnerabilities. In the case (e.g a preview for a microsite) where a developer would like to knowingly introduce arbitrary HTML into a page, and they know that protections, mitigations, or isolation have been applied there is a way to introduce the HTML using dangerouslySetInnerHTML. The brilliant part of this pattern is that the name of the function itself may cause the developer to stop and think about what they are about to do. This in combination with clear documentation that is intended to educate the engineer increases the chance that the outcome will be secure or that the implementor clearly understands that there is a specific security choice being made. Additionally, it is easy for a reviewer or auditor to see that this is a potentially dangerous choice that warrants more scrutiny.

iOS - Photo Access

Photos are hands down one of the most personal or sensitive data that exists on our mobile devices. There are also many ways that photos can be used in applications for improved productivity, communication, and fun. On the other hand, there is a potential for intentional or unintentional abuse, mishandling, or malicious use of photos by 3rd party applications. Even the potential for this abuse may create a chilling effect or decrease the trust users have in 3rd party applications on their mobile devices.

Beginning in iOS 14, Apple introduced new capabilities to both increase transparency in the access to photos that 3rd party applications have, and also start from a secure by default position where the user has granular control over what is shared. The functionality allows a user to permit even a single photo that the application can access. This means a few things when it comes to security outcomes. First, when a user doesn't explicitly grant access to a photo, the app cannot access it in the background or without their knowledge. Second, granularity allows users to have confidence that they understand what data may be available outside of their devices which may increase their trust and engagement with these apps. A simple example use case that this opens up is for an application to use a single photo for an avatar or profile picture without granting access to all photos. Lastly, it creates an opportunity for the end user to go back and review what has been granted in the past and decide if that access is still appropriate.

OPPORTUNITIES FOR CLOUD APPLICATIONS

There are numerous popular cloud and SaaS applications that have been implemented and deployed across the internet that provide great examples of secure by default functionality. The best example applications are implemented and deployed with the following attributes by default:

- No access is permitted without explicit addition.
- All user accounts and activations require MFA.
- All user accounts are disabled or adjusted automatically when role changes occur (terminations, org changes, role changes etc.).
- All elevated privileges are time-restricted based on approval.
- All secrets are rotated automatically after each usage.
- All roles are segregated and may not be combined.
- All stored data is encrypted.
- All sessions and data in transit is encrypted.
- All interfaces and APIs require authentication and authorization.
- Customer Identity providers are either required or available by default.
- Enable security logging and monitoring.

CSaaS SIG
An IT ISAC Community

CALL TO ACTION

While implementing secure by default may sound like a completely logical approach to implementing security and driving secure outcomes for users, it is not as easy as defining it as a requirement. It takes collaboration between security teams, user experience (UX) teams, product managers, engineers, and others to come to common ground on the way we think about it. There is as much work to be done in understanding the customer as there is in understanding the idiosyncrasies of software and how it can be abused.

We must focus on the outcome of the end user (the customer) to guide the approach that security, UX, product managers, and engineering teams take to effectively translate complex security needs into actionable requirements that are enabled by default. Users and consumers have a very important role in demanding security by default and to ensure that all their purchase decisions and deployments are designed with security by default.

WHAT ARE YOU DOING TO ELEVATE YOUR SECURE BY DEFAULT STATUS?

Thank you to the SaaS companies and providers that are taking the lead across the industry to overcome this ongoing challenge and helping to improve the security of all businesses and consumers.

INFORMATION TECHNOLOGY - INFORMATION SHARING AND ANALYSIS CENTER (IT-ISAC)

Founded in 2000, the Information Technology - Information Sharing and Analysis Center (IT-ISAC) is a non-profit organization that augments member companies' internal capabilities by providing them access to curated cyber threat analysis, an intelligence management platform, and a trusted forum to engage with senior analysts from peer companies.

CRITICAL SaaS SPECIAL INTEREST GROUP (CSaaS SIG)

The Critical SaaS Special Interest Group (CSaaS SIG) is part of the IT-ISAC and serves as a forum for CSaaS companies to collaborate on a collective defense strategy to improve the security and operational resiliency of their services and share intelligence information with the industry. It enables companies who are essential to the internet to share cyber threat intelligence and effective security practices. The SIG holds a weekly analysts meeting and is designed for security managers, analysts, and IT executives from Critical SaaS companies.

Questions? Interest in joining IT-ISAC and the CSaaS SIG?
Email us at csaas@it-isac.org.

IT-ISAC.ORG