IEEE CYBER SECURITY

IEEE CENTER FOR SECURE DESIGN

WearFit: Security Design Analysis of a Wearable Fitness Tracker

IEEE

IEEE computer society

CELEBRATING 70 YEARS

READ ON!

Staff

Brian Kirk, Manager, New Initiative Development
Carmen Flores-Garvey, Designer

WearFit: Security Design Analysis of a Wearable Fitness Tracker

Jacob West
Chief Architect, Security Products, NetSuite

Tadayoshi Kohno
Short-Dooley Professor, Computer Science & Engineering, University of Washington

David Lindsay
Security Researcher, Synopsis

Joe Sechman
Director, Applied Security Research, Hewlett Packard Enterprise

In 2014, the IEEE Computer Society—the leading association for computing professionals—launched a cybersecurity initiative by forming the Center for Secure Design. The mission of the Center is to expand the focus in security from merely finding bugs to identifying and avoiding common design flaws, with the hope that software architects can learn from others' mistakes.

Soon after it was founded, the Center brought together experts from industry, government, and academia at a workshop where participants discussed the types of flaws they either identified in their own internal design reviews, or that were available from external data. The group arrived at a list of what they felt were the top security design flaws, consolidated guidance on how to avoid them, and published the result as *Avoiding the Top 10 Security Flaws* (see http://goo.gl/2Gujs6).

In this document, we build on the Center's previous work by describing a fictitious wearable fitness tracking system known as *WearFit* and discussing how the system's design addresses each of the top 10 software security design flaws:

1. Earn or give, but never assume, trust.
2. Use an authentication mechanism that can't be bypassed or tampered with.
3. Authorize after you authenticate.
4. Strictly separate data and control instructions, and never process control instructions received from untrusted sources.
5. Define an approach that ensures all data are explicitly validated.
6. Use cryptography correctly.
7. Identify sensitive data and how they should be handled.
8. Always consider the users.
9. Understand how integrating external components changes your attack surface.
10. Be flexible when considering future changes to objects and actors.

This document strives to bring life to the top 10 software security design flaws by demonstrating how they apply to a specific, if fictitious, system. We selected a wearable fitness tracker because wearable devices are driving significant changes in how society uses technology, with almost half the population predicted to adopt fitness-tracking devices by 2019 (see http://goo.gl/eS0IeM). We base our analysis as much on real-world systems as possible, and aim to provide a broad analysis of threats facing users of wearable fitness-tracking devices.

The "System Overview" section describes the technical design of the *WearFit* product and outlines the fundamental categories of threats that the system takes into consideration. The "Analysis" section comprises the remainder of the document and discusses the security implications of the system's design in the context of each of the top 10 software security flaws.

System Overview

The *WearFit* system is an imaginary wearable personal health monitoring device similar, but not identical, to products from companies already on the market. Figure 1 shows the basic system architecture.

Wearable Device

The *WearFit* product comprises a device that's worn on the wrist and measures step count (from an accelerometer) and heart rate (from an optical sensor), which it encrypts and signs

BACK TO TOP OF DOCUMENT

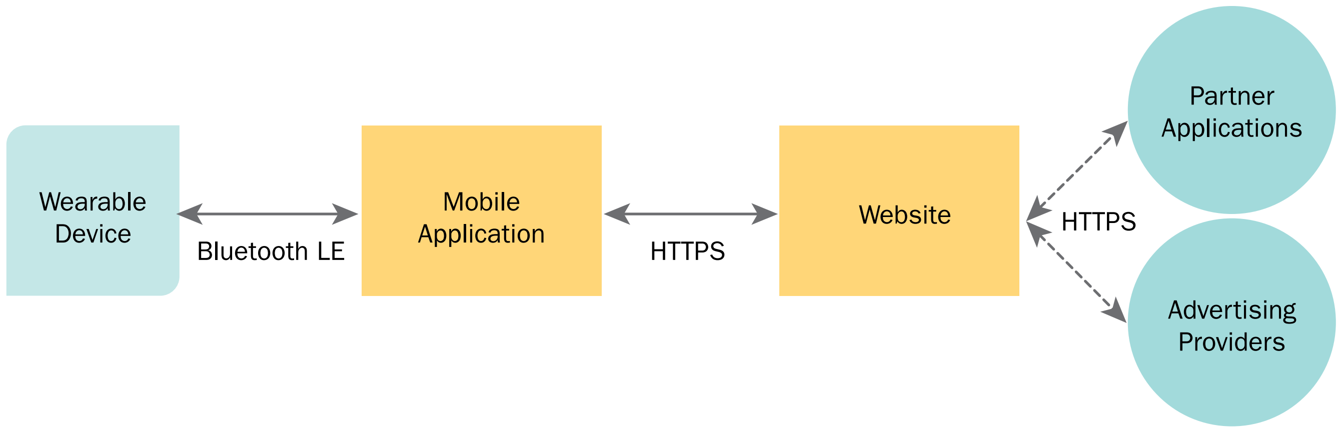

Figure 1. High-level overview of WearFit's system architecture.

before storage. The wearable responds to polling requests from native mobile applications that download the data in a JavaScript Object Notation (JSON) format and upload them to the website. Attempts to poll for and upload data are made every 15 minutes.

The device also accepts configuration instructions from authorized mobile devices, including the ability to control what's displayed by default on the wearable, setting the time and alarms, and defining specific fitness goals (such as the number of daily steps).

The wearable includes upgradeable firmware, a unique hardware identifier, and Bluetooth Low Energy (LE) communication using libraries provided by the chipset vendor. Firmware and configuration updates are signed using a private key. Signatures are validated by the wearable using a certificate provisioned on the device during manufacturing.

Mobile Application

WearFit provides native applications that run on mobile devices and allow them to interact with the wearable. The application primarily downloads activity data from paired wearables, permits users to view their activity data on the mobile device, and uploads activity data to the website.

The wearable pairs with a mobile application using Bluetooth LE and a library from the chipset vendor. A unique token is created by the wearable and is displayed on both devices. The user must verify that the tokens match and accept the pairing. Pairings can be deleted in the

BACK TO TOP OF DOCUMENT

native mobile application or the website. Pairings persist until they're deleted.

In addition to data from paired wearables, the application also polls for nearby unpaired devices and acts as a passthrough to upload activity data from those devices to the website. The application annotates activity data with geolocation information before uploading them to the website. Data uploaded via the passthrough feature can't, by design, be read or modified by the unpaired mobile device.

Website

The website lets users register and view their account, access and visualize their activity data, plus set workout targets and annotate past activity data. The website is built using open source libraries, commercial third-party libraries, and in-house developed code. It includes front-end webservers, back-end database servers, and other servers to manage third-party services on which the website relies.

Users provision accounts by registering an email address and password or by third-party credentials (such as Facebook or Google+) using the OAuth 2.0 protocol, which also lets them access the *WearFit Social System*. The user provides information such as gender, age, location, height, and weight. User accounts are automatically populated with information available in the third-party authentication services.

After a user authenticates, a unique session token is set and shared between the client and server. If a user's web session is inactive for more than 15 minutes, the server invalidates the session token, and the user must re-authenticate to generate a new valid session identifier.

Authenticated users can update their profiles, share their fitness data with other users (although the default is set to keep these data private), publish their fitness data on social media through the *WearFit Social System* (the default is off), toggle the ability to auto-approve follow requests (the default is off), toggle the ability to share data with partner applications (the default is off), and enroll in employer or insurer-sponsored *WearFit Corporate Benefits* programs to share data with the respective group (the default is no enrollment). Users can change their password by first providing their current password and retrieve their password securely through either means of account provisioning. Users can merge and unmerge accounts.

The databases and other systems behind the webserver are only accessible from a limited set of internal networks and systems. User account information is stored in a dedicated database.

Attack Categories

The following is an overview of attacks that the *WearFit* system design took into consideration. Under each top-level threat category is a list of

BACK TO TOP OF DOCUMENT

examples of representative attacks that illustrate, but do not completely enumerate, ways that attackers might target the system.

Denial of service
- Render wearable unusable with fake firmware update.
- Drain battery, CPU, or other resources.
- Lockout user from website account.

Compromising device integrity
- Malicious firmware update.
- Buffer overflow on wearable to compromise paired mobile device.

Falsifying the user's own health data
- Physically manipulate the device.
- Tamper with data on mobile device before uploading to the webserver.
- Tamper with data in transit from wearable to mobile or mobile to website.

Falsifying another user's health data
- Rewrite health data on device.
- Tamper with data on a mobile device when used as a passthrough.
- Spoof data uploads using a known device or user identifier.
- Tamper with data in transit from wearable to mobile or mobile to website.
- Direct attacks against the website (for example, SQL injection).
- Phishing, cross-site request forgery (CSRF), and other indirect attacks against end users.

Abusing health data that are intentionally shared
- Employer or insurer penalizes behavior seen through *WearFit Corporate Benefits*.
- Users of the *WearFit Social System* unintentionally view sensitive activities.
- Advertising partners target over-personalized ads.
- Share configuration that becomes out-of-sync with changes in real-world relationships.

Stealing a user's health data
- Guess or steal a user's authentication credentials.
- Direct attacks against the website (for example, SQL injection).
- Eavesdrop on communication on mobile device when used as a passthrough.
- Eavesdrop on communication from wearable to mobile or mobile to website.
- Malicious insider uses internal, or otherwise "privileged," access.
- Phishing, CSRF, and other indirect attacks against end users.

Now let's consider how the system's technical design can affect its security.

BACK TO TOP OF DOCUMENT

Analysis

This section evaluates the design of the *WearFit* system in the context of the top 10 software security design flaws.

Earn or Give, but Never Assume, Trust

Recall the following from *Avoiding the Top 10 Security Flaws*:

> *Software systems comprising more than just a single monolithic component rely on the composition and cooperation of two or more software tiers or components to successfully accomplish their purpose. These designs often depend on the correct functioning of the existing parts. They will be inherently insecure if any of those parts are run in a potentially hostile environment, such as a user's desktop computer, an unmanaged device, or a runtime or sandbox that can be tampered with by an attacker.*

The *WearFit* system as a whole includes a number of components, each with their own trust requirements and capabilities to prove their identity to the rest of the system. Adversaries might masquerade as a trusted component in an effort to steal another user's health data, but the system protects against this possibility. This way, trust becomes a property of the system that's only present once the components have actively established their identities to each other over protected communication channels.

The *WearFit* wearable uses a pairing process to establish trust with the application running on a mobile device. Both sides present a visual representation of the same token so that the user can verify a match. Once the user confirms a match, each device stores the other's identity. From that point, the devices have a trust relationship and proceed with communication. Without that trust, wearables still communicate with mobile devices acting as passthroughs, but the mobile application on passthrough devices won't have access to or trust the information.

The problem of trust also extends to partners that interface with the *WearFit* system, including partner applications and advertising providers. The trust relationship is explicitly built and verified with partners by making correct use of certificates, pre-shared keys transmitted over secure channels, and binding legal contracts.

A trust relationship also must be present to update the firmware on the wearables. Applying an unauthenticated firmware update might result in rendering the device completely inoperable or causing the device to operate in a manner not originally intended by *WearFit*. In the latter case, the device's normal operation could be forever compromised without the owner even realizing an attack has occurred. The specifics of

BACK TO TOP OF DOCUMENT

this mechanism are discussed later in the "Use Cryptography Correctly" section.

Trust is inherently time-sensitive and can be revoked. It's unsafe, generally, to assume that just because trust existed at some point in time it continues to exist over time. Trust relationships between partners are re-evaluated and re-established on a regular cadence, and expired relationships are purged. Trust between the *WearFit* device and the controlling mobile app, as well as between a mobile app and the web app, are only reset when a user changes their authentication credentials. When users change their password, they must also reconfirm the pairing with the device.

Use an Authentication Mechanism That Can't Be Bypassed or Tampered with

Avoiding the Top 10 Security Flaws suggests:

> *Authentication is the act of validating an entity's identity. One goal of a secure design is to prevent an entity (user, attacker, or in general a "principal") from gaining access to a system or service without first authenticating. Once a user has been authenticated, a securely designed system should also prevent that user from changing identity without re-authentication.*

As part of the onboarding process, *WearFit* users are required to either register a new account directly or through services that support the OAuth 2.0 authentication protocol (such as Facebook or Google). By design, the authentication mechanism implemented by the *WearFit* website is used as a service by the mobile application, which centralizes the authentication logic and simplifies the design.

For users who wish to create an account directly with *WearFit*, registration requires a strong password that meets the complexity requirements noted in the "Always Consider the Users" section of this document and stores it as a salted hash. As a result, lost passwords must be reset and can't be retrieved in plaintext. Storing the passwords in this manner minimizes the possibility of a widespread compromise of user account credentials in the event that an attacker successfully gains access to the database containing the credentials.

To combat attempts to enumerate valid user accounts by an anonymous attacker, the website displays generic messages in response to failed authentication attempts. If a user initiates a password reset request, the email address provided by the user must match their account or they won't receive further instructions to reset their credentials and, if applicable, their account will remain locked. Upon receipt of the email, the user is guided back to the *WearFit* website via a unique URL that includes a securely generated (non-guessable) randomized token that expires after 30 minutes. To defend against a similar

BACK TO TOP OF DOCUMENT

attack, five failed authentication attempts result in the user's account being locked for a fixed period of time, which requires a password reset to actively unlock.

For users who elect to register with *WearFit* using a third-party OAuth authentication service, most credential-related security operations are deferred to the third-party provider. Users are notified and must acknowledge upon registration that the selected authentication provider now has permission to access their *WearFit* activity data. *WearFit* closely monitors vulnerability disclosure reports that impact supported authentication providers that may affect users with accounts provisioned in this manner. More information regarding this practice is discussed in *Avoiding the Top 10 Security Flaws* in the section "Understand how integrating external components changes your attack surface."

To establish trust relationships with OAuth providers, the *WearFit* system maintains several important credentials, including a client ID and client secret to be able to authenticate itself, along with access and refresh tokens for all the users who choose to use the integration for sharing with social networks.

Once authenticated, the system generates a secure random session identifier with adequate size and entropy to prevent attackers from guessing it easily. The website monitors this token for any modifications that would suggest tampering and, if tampering is discovered, terminates the corresponding user's session, forcing the user to re-establish a valid session by re-authenticating. Additionally, user sessions time out and force re-authentication after 15 minutes of observed inactivity. The "Use Cryptography Correctly" section of this document discusses how transport-layer security is used throughout the system to prevent the inadvertent disclosure of session identifiers and other secrets.

Finally, the *WearFit* wearable and mobile application authenticate themselves to one another during the pairing process. Once paired, apart from occasional polling requests to upload data, the wearable only accepts commands from paired mobile applications. This minimizes the threat posed by attackers bent on draining resources by sending repeated, resource-intensive requests to the device. Unpaired devices present little value to potential attackers and are therefore unlikely targets.

Authorize after You Authenticate

Avoiding the Top 10 Security Flaws states:

> *While it is extremely important to assess a user's identity prior to allowing them to use some systems or conduct certain actions, knowing the user's identity may not be sufficient before deciding to allow or disallow the user to perform certain actions.*

BACK TO TOP OF DOCUMENT

For the *WearFit* wearable, authorization begins when it's first paired with a user's account. This process grants permission for the device to transfer fitness activity data to the *WearFit* mobile application and, ultimately, to the *WearFit* website. The pairing process typically only happens once. However, certain activities require subsequent authorization and, in some cases, re-authentication before they can be performed safely. The difference between authorization and re-authentication is distinct: Authorization is an explicitly granted or defined capability that can be revoked, whereas re-authentication is the act of repeating the authentication challenge process to validate an identity (device or user).

The process for changing a user's password for the *WearFit* website is the primary driver requiring re-authentication. The user is prompted to provide the current password in addition to the new password to complete the update process. This ensures that the user is indeed who they purport to be and not a malicious actor that has somehow accessed the system without the user's password (using, for example, account enumeration or session replay). If re-authentication weren't required, a malicious actor might update the user's password to an arbitrary value of their choice; thereby taking full and potentially ongoing control of the account. Forgotten passwords are reset using an email-based process, which is described in the "Use an Authentication Mechanism That Can't Be Bypassed or Tampered With" section of this document.

Authorization is transparent to the user, but is a key security control invoked prior to modifying or accessing data. When a user requests to do any sensitive action in the *WearFit* website, the application verifies that the user's session is active and that the user has permission to perform the given action. If a discrepancy is detected, the request is denied and the user's session is terminated with a corresponding log entry for auditing purposes.

The *WearFit* website also lets users share activity data with their friends. Users can elect to automatically approve friend requests or require explicit approval before another user may view activity details. In either case, authorization checks are explicitly made for each request before sharing activity details with other users. Should a user decide to no longer share their activity data with one or more friends, the user can elect to revoke authorization and deny access to both past and future activity updates.

As a further safeguard against attacks, requests that prompt changes on the website check the value of the Origin header. This header is validated to ensure that they originate from the same domain as the *WearFit* website, which thus prevents basic CSRF attacks. An authenticated user

BACK TO TOP OF DOCUMENT

may also elect to delete their *WearFit* account entirely, which revokes future access to the *WearFit* system and deletes all account-related data.

Strictly Separate Data and Control Instructions, and Never Process Control Instructions Received from Untrusted Sources

From *Avoiding the Top 10 Security Flaws*, recall the following:

> *Commingling data and control instructions in a single entity, especially a string, can lead to injection vulnerabilities. Lack of strict separation between data and code often leads to untrusted data controlling the execution flow of a software system. This is a general problem that manifests itself at several abstraction layers, from low-level machine instructions and hardware support to high-level virtual machine interpreters and application programming interfaces (APIs) that consume domain-specific language expressions.*
>
> *At lower layers, lack of strict segregation between data and control instructions can result in the introduction of memory-corruption vulnerabilities, which in turn may permit attacker-controlled modifications of control flow or direct execution of attacker-controlled data as machine or byte-code instructions.*
>
> *At higher levels, commingling of control and data often occurs in the context of runtime interpretation of both domain-specific and general-purpose programming languages. In many languages, control instructions and data are often segregated using in-band syntactic constructs, such as quoting and escaping. Experience has shown that use of injection-prone APIs incurs significant risk that injection vulnerabilities will indeed be introduced. Examples of such vulnerabilities include SQL query injection, cross-site JavaScript injection, and shell command injection.*

Attackers target vulnerable software systems for the majority of breaches (see http://goo.gl/Bnk6bE) and *WearFit* is unlikely to be an exception. Adversaries might want to steal or modify data for a single, targeted user, or aggregate data across many users. They might target a vulnerability in the wearable to compromise the mobile application or website, or they might use injection-style attacks to target the mobile application or website directly. Each software component in the system, as well as the data formats they exchange, is therefore responsible for maintaining a strict segregation between control and data values.

The *WearFit* system primarily deals with two kinds of data: health data recorded by a device and configuration data input by the user. Health data comprises the device identity combined with a timeline, record of steps taken, and heart rate. This information is communicated to the mobile application and website, where it's stored and processed. *WearFit* users also use

BACK TO TOP OF DOCUMENT

the mobile application and website to input various other data, including fitness profiles, workout targets, and annotations to enrich recorded activities.

No data from a device directly influences control in the mobile application or website. A device's identity is compared with a lookup table of known devices populated during pairing, and only when a match is found are the values allowed to influence program behavior. The system strictly whitelists the syntax of new device identifiers to ensure that they're formatted as expected.

WearFit enforces strict syntax whitelists at trust boundaries between and within components. The protocol for communicating health data from the device to an application on a mobile device explicitly limits messages to data, represented as structured and strongly typed values. Likewise, the device accepts a known set of control statements from the mobile application for updating the device's settings (for example, the default view, alarms, and activity targets) and strictly validates inbound commands against a list of known-good values.

At the web layer, the system assumes mobile applications communicating with it have been compromised and therefore makes no assumptions about the syntax of the data it receives. The web application strictly segregates control statements from data values, validates control statements against known-good values, and validates data against a whitelist of valid syntax.

The *WearFit* web application relies on downstream protections in the wearable and mobile application to prevent attackers from impersonating legitimate users. Regardless, it's impossible for a user to trick the web application into executing an invalid command (such as overwriting protected data values on the wearable) because these strict syntax requirements prevent data from being interpreted by adjacent systems—such as the database, *Lightweight Directory Access Protocol* (LDAP) directory, or web browser—as control statements. The system design ensures that different message formats are used to communicate between different components. Any attempt to interact with a component using the wrong format will be strictly rejected.

The web application is also the entry point for the user to input configuration settings and other data relevant to their profile and activities. As with input from the mobile application and elsewhere in the system, no user input directly influences program control. Rather, authenticated users select from a fixed set of allowed operations on strongly validated data values. When interacting with the database, the web application uses a parameterized API that utilizes prepared statements to formally segregate user input from control.

BACK TO TOP OF DOCUMENT

Although not a part of the system's core functionality, the web application also accepts input in the form of shared fitness data from partner applications and advertising supplied by a third-party system. This external content, and particularly the JavaScript used to deliver and load ads in the target application, are a known vector for malware and other types of attacks. A limited set of control statements are necessary for ads to work in the user's browser, but shared fitness data should never be executable. The application enforces a known-good list of what commands are allowed and checks URIs and IP addresses embedded in ads against a reputation database of known-bad sites before displaying them to users or acting on them in any way.

Define an Approach That Ensures All Data Are Explicitly Validated

From *Avoiding the Top 10 Security Flaws*, remember the following:

> *Software systems and components commonly make assumptions about data they operate on. It is important to explicitly ensure that such assumptions hold: Vulnerabilities frequently arise from implicit assumptions about data, which can be exploited if an attacker can subvert and invalidate these assumptions.*

Keeping in mind that the majority of breaches target vulnerable software, we note that many breaches are enabled, at least in part, by a lack of data validation that permits attacks, such as cross-site scripting (XSS), SQL injection, path traversal, and buffer overflow among others. Attackers might exploit lapses in data validation in the *WearFit* system to compromise or falsify health data by targeting a vulnerability in any one component, or a combination of vulnerabilities across components.

The *WearFit* system assumes that any given component (device, client application, or website) might have been compromised or replaced by an imposter. Therefore, each component implements a validation strategy that verifies assumptions about data types (both syntactic and semantic) and values before operating on them. Each component implements a validation strategy that verifies data as early as possible and revalidates important properties before the data are consumed.

In the web application, a centralized validation approach that enforces validation on all inbound requests is implemented using request filters and an interceptor facility provided by the underlying web framework. Additionally, a common library is used to validate known types (for example, email addresses or URIs), which ensures that all validation of different instances of the same type of data apply consistent validation semantics. Consistent use of common validation annotations (indicating, for instance, what

BACK TO TOP OF DOCUMENT

validation has been performed or what assumptions have been verified) also increases the fidelity of static analysis and makes manual code review easier.

Beyond syntactic restrictions, the validation mechanism also considers the semantics and data ranges for data values in order to refuse data that are inappropriate. For example, when the webserver first receives activity parameters from a *WearFit* device, a check is done to ensure that the values are physically possible in the real world.

In mobile applications, one area of concern is whether the validation scheme is implemented consistently across the different codebases and programming languages. In this case, the application was designed to use a common validation API with implementations in each supported language. The *WearFit* team performs frequent design and code reviews to ensure that the per-language implementations stay in line with the central design. Any platform-specific inconsistencies are well-documented and mitigated appropriately.

On the wearable, validation capabilities provided by the platform and frameworks are more limited. Custom validation logic is necessary to ensure that devices aren't compromised by malicious data or commands received from the applications. Standard communication protocols are used to narrow the types and ranges of data that may be used. However, the device explicitly validates the syntax, semantics, and expected values of all data.

Use Cryptography Correctly

The *WearFit* system transmits and stores a significant amount of information, including information related to an individual's health and activities. Protecting this and other data from unauthorized reads and modifications is therefore critical. Regarding data protection, *WearFit* considers both the data's integrity (ensuring that an unauthorized party can't modify the data) as well as the data's confidentiality (ensuring that an unauthorized party can't read that data) imperative.

To illustrate the nuances and potential impact of improper use of cryptography for *WearFit*, consider the following scenario. A user has a *WearFit Social System* feature activated, and it shares the user's exercise activities with her Facebook friends, while also showing that she's in a wonderful new romantic relationship. A jealous ex-lover could cause harm to the new relationship by modifying the user's *WearFit* activity annotations to change her running activity to something more salacious. Even something as simple as modifying the user's data to make it appear that she's taking long afternoon walks when she isn't could be sufficient to break up a new relationship. In addition to relationship

BACK TO TOP OF DOCUMENT

sabotage, the jealous ex-lover might also want to stalk his ex. To do so, he might try to obtain private information about her activities and whereabouts by illicitly extracting the (private) data recorded on her *WearFit* device.

As another motivating example, consider a situation in which a user is enrolled in his company's *WearFit Corporate Benefits* program, which means that he receives an insurance discount if he authorizes *WearFit* to share his exercise activities with his employer and insurance company and if he maintains a certain level of regular exercise. That user might want to compromise the integrity of his own data and upload falsified data to the *WearFit* servers (for example, that he has been taking 40,000 steps a day when in fact he has been taking 2,000 steps a day). To win the office pool for the healthiest division, he might at the same time falsify data for other company employees, making it seem that they exercise much less than they actually do.

There are numerous defenses that *WearFit* must employ to protect data confidentiality and integrity under such scenarios, ranging from secure authentication mechanisms (see the "Use an Authentication Mechanism That Can't Be Bypassed or Tampered With" section of this document) to implementing best practices on the server (so that it's hard to compromise the server directly).

One critical component—the component that's the focus here—is cryptography. Cryptography ultimately plays many roles in the *WearFit* system. For the aforementioned scenarios, cryptography can help protect communications between the *WearFit* device, the mobile device, and the webserver from unauthorized reads (which could compromise data privacy) and modifications (which could compromise data integrity). Quoting from *Avoiding the Top 10 Security Flaws*:

> *Cryptography is one of the most important tools for building secure systems. Through the proper use of cryptography, one can ensure the confidentiality of data, protect data from unauthorized modification, and authenticate the source of data. Cryptography can also enable many other security goals as well. Cryptography, however, is not a panacea. Getting cryptography right is extremely hard.*

Non-experts should never design cryptography protocols on their own. Moreover, even ensuring that cryptographic primitives are properly used requires some experience and domain expertise. Cryptography is notoriously hard to get right, but there are some critical cryptographic components that the *WearFit* system must have.

Focusing on the transmission of the data from the *WearFit* device to the webserver via the mobile application, it might be tempting to use cryptography to protect the data's confidentiality and integrity as it transits between the *WearFit*

BACK TO TOP OF DOCUMENT

device and the mobile device—using, for example, Secure Sockets Layer and Transport Layer Security (SSL/TLS)—and again protect the data as it transits between the mobile device and webserver (again using SSL/TLS).

However, this is insufficient because, by design, the *WearFit* device might upload the data via untrusted mobile devices. Those untrusted mobile devices could modify or leak the data that they transit. Therefore, the data from the *WearFit* device are end-to-end encrypted and authenticated as they flow from the *WearFit* device, via the mobile device, and to the webserver. That is, the data are encrypted and authenticated on the *WearFit* device, and only decrypted and verified at the final destination server.

Because the *WearFit* system requires end-to-end data integrity and confidentiality from the wearable all the way to the website, the *WearFit* device uses an immutable asymmetric private key (for which the webserver knows the public key). Additionally, the *WearFit* system is designed with algorithmic agility. If cryptographic weaknesses are discovered in the deployed algorithms, there's a path for transitioning the *WearFit* system to new cryptographic algorithms.

Cryptography manifests in other places in the *WearFit* system, as well. Data on the *WearFit* servers are cryptographically protected to mitigate the potential for malicious or curious company insiders to access or modify customer data. Encrypting data at rest also mitigates the potential harm the company would experience if a disk with customer data were discarded and recovered by a third party. See the "Identify Sensitive Data and How They Should Be Handled" section of this document for further discussion of what specific data are encrypted at rest.

Firmware updates to the *WearFit* device are cryptographically signed and authenticated using a private key that only the *WearFit* company knows. The use of signed software updates can help protect against one method by which an adversary might try to get his or her own code running on the *WearFit* device. But, as with all discussions of cryptography, *WearFit* designers must consult with cryptography experts when designing such a firmware update capability. For example, *WearFit* must ensure that the software signing mechanism is robust in the event that the private signing key is compromised, and must also consider rollback attacks in which an attacker attempts to load an old and insecure version of the firmware onto a target device.

Identify Sensitive Data and How They Should Be Handled

The "Use Cryptography Correctly" section discusses the role cryptography plays in protecting

BACK TO TOP OF DOCUMENT

sensitive data, where protecting might mean protecting against unauthorized modifications or access. However, in any complex system, it might not be possible to fully protect all data from all parties, even if it were desirable. Indeed, for the system to operate and be valuable, at least some data must be exposed to the user (for example, information about his or her activities) as well as partner organizations (for instance, if the user enrolls in the *WearFit Corporate Benefits* program) and friends (if the user enrolls in the *WearFit Social System*). This forces the following questions: What data are sensitive, how sensitive are those data, and who should have read or write access to them? Recall the following from the *Avoiding the Top 10 Security Flaws* document:

> *Data are critical to organizations and to users. One of the first tasks that systems designers must do is identify sensitive data and determine how to protect them appropriately. Many deployed systems over the years have failed to protect data appropriately. This can happen when designers fail to identify data as sensitive, or when designers do not identify all the ways in which data could be manipulated or exposed.*

To help illuminate the more challenging issues that *WearFit* addressed in their system's design, consider the following two scenarios. First, when most users enroll in the *WearFit* system, they probably expect that only those they authorize will be able to see any of their tracking data. Additionally, *WearFit* strongly promotes user privacy to ease concerns users may have about them storing health and geolocation data on their servers. Suppose now that a user is involved in some legal proceedings related to an employment dispute. In this case, law enforcement personnel may require that *WearFit* turn over all tracking and geolocation information associated with the given user.

If the *WearFit* system maintains detailed records of all of the user's raw data indefinitely, then users may view turning over this data as a violation of the privacy assurances *WearFit* made. *WearFit* proactively considered this issue and decided not to store the entire history of raw data for perpetuity, but rather to store all data for the past week, less granular data about user activity between three months and one week ago, and then even less granular data about user activity prior to that.

Consider a second scenario focused on the challenges involved in identifying sensitive data in the first place. The *WearFit Social System* permits users to share activities with their friends. When designing the *WearFit Social System*, the *WearFit* team envisioned users automatically sharing exercise activities such as running and walking rather than more sensitive activities. However, users might be surprised to learn that their sexual activities,

BACK TO TOP OF DOCUMENT

which their wearable considers to be exercise, were also displayed on their friend's newsfeed. To address this issue, *WearFit* designers made the automatic sharing of data an opt-in feature. Additionally, users who do opt-in receive periodic visual alerts reminding them about what data are shared with their friends.

Identifying sensitive data is both important and challenging, as is determining how to protect such data. The *WearFit* system errs on the side of caution and treats most data, including annotations and other fitness profile details, as sensitive. To account for evolving data values and changes to their sensitivity, the *WearFit* team consciously considers each data element, how that data element might be used or misused, and uses this to determine the data's level of sensitivity and how it should be protected.

Always Consider the Users

Avoiding the Top 10 Security Flaws states:

> *The security stance of a software system is inextricably linked to what its users do with it. It is therefore very important that all security-related mechanisms are designed in a manner that makes it easy to deploy, configure, use, and update the system securely. Remember, security is not a feature that can simply be added to a software system, but rather a property emerging from how the system was built and is operated.*

As a consumer device, the entire *WearFit* system is built around the end user. User experience is key to building customer loyalty and maintaining a profitable business. Many of the fundamental architecture decisions in the system were made with this user experience in mind. The security architecture must help support a positive user experience, and not detract from it.

The *WearFit* design acknowledges that certain user behavior, although possibly undesirable, can't be reasonably prevented. Specifically, users that wish to falsify basic health data readings by physically manipulating the wearable device (for example, bouncing the device on their desk or attaching it to another person to increase activity readings) are permitted to do so. Beyond the potential for incurring rewards from programs that incentivize exercise, such as employer- or insurer-run programs, this type of fraudulent data does little harm to other users of the overall system.

WearFit users interact most heavily with the web application, and user experience begins with authentication. New users must register a strong password with the site and a secure recovery procedure in case they forget the password. With respect to password strength, there's a tension between increased security (as an extreme example, requiring 32-character passwords) and usability (for example, allowing

BACK TO TOP OF DOCUMENT

4-digit PINs as passwords). The *WearFit* website enforces password strength requirements that are compatible with other popular, consumer-facing web applications (8 characters minimum, 3 distinct character types, and no dictionary words). Users can also authenticate using third-party services, including Google+ and Facebook.

As with weak password-strength requirements, web applications that allow too many password attempts make it easy for attackers to brute force user credentials. On the other hand, systems that are too aggressive about locking out users who have mistyped their password can worsen the user experience; this creates a new vector for attackers who wish to legitimate users. The *WearFit* application strikes a balance between these extremes by locking out users after five failed authentication attempts, but letting users unlock their account at any time by using a secure, email-based password reset process.

The *WearFit* system design also incorporates the diversity of users. Users have different cultures, geopolitical regions, ages, genders, and physical abilities. As an example of keeping in mind different physical abilities, the *WearFit* system makes it easy for both blind and sighted users to authenticate to the system. The password reset functionality includes a CAPTCHA, which allows visually impaired users to prove their humanity. Care is given to the audio CAPTCHA design to ensure that it's reasonably resistant to automated solving techniques. From a secure design standpoint, all authentication mechanisms are treated consistently, meaning that no one authentication mechanism represents a lower barrier to entry than another.

User considerations affect the *WearFit* system's architecture in more fundamental ways as well. For example, the tracking device is designed so that an end user can always keep the device on them. This means that the device needs to have a reasonably long battery life, be small, and not present a burden to carry. To accommodate these needs, the device has little memory, computational power, storage space, or long-distance communication capabilities.

All of these constraints affect the security architecture. Because storage space is small, the device needs to take advantage of every opportunity it can to upload activity data to the *WearFit* server. Not only that, the desired user experience is to have this data uploaded without any user interaction. As a result, the system is designed so that any device running the *WearFit*

“As an example of keeping in mind different physical abilities, the *WearFit* system makes it easy for both blind and sighted users to authenticate to the system.”

mobile application can relay data from a tracker to the server. This means that additional security controls must be put in place to ensure that users can't easily see or tamper with other users' activity data.

Most *WearFit* users aren't security professionals and often aren't aware of what data are sensitive or how to secure them. As discussed in *Avoiding the Top 10 Security Flaws*, there are significant advantages to making it easy for users to do the most secure thing. The *WearFit* system design incorporates this principle throughout. The website makes the default data settings private—so that no user can see another user's data unless they have explicitly shared them. The website also presents a simple dashboard for users to view what data they've shared and with whom, as well as to revoke or modify those sharing relationships at any time.

Finally, *WearFit* understands that users might share data without fully understanding the implications of doing so. To address this, the website includes a brief awareness video and FAQ to familiarize users with the sensitivity of the health data that *WearFit* collects (including unintended implications of those data—for example, determining when the user was performing sensitive activities such as going for a run when they were supposed to be working from home or having sex when they were traveling without their spouse).

Understand How Integrating External Components Changes Your Attack Surface

Recall the following from *Avoiding the Top 10 Security Flaws*:

> *The decision to use-rather-than-build means that the software as a whole inherits the security weaknesses, security limitations, maintenance responsibility, and the threat model of whatever you are including. This inheritance can amount to a deficit of security, which must be solved, mitigated, or accounted for when the system is finished.*

Like most development organizations, *WearFit* leverages multiple open source and commercial libraries, frameworks, and other APIs. In the wearable and mobile applications, Bluetooth LE communication is handled by libraries provided by the chipset vendor. The website is a traditional Java web application built on an open source model-view-controller (MVC) framework that provides various security-relevant functionalities to the application (such as user management, data access, and input validation). Moreover, the website integrates with several other open source libraries in the browser via JavaScript (to provide dynamic graphs, for example), and on the back end (to render targeted ads).

Even when no security problems are known to exist, the organization validates its assumptions about the behavior of third-party code whenever possible. Imperatives such as input validation

BACK TO TOP OF DOCUMENT

are even more critical when you don't control all of the code that's responsible for accepting or processing input and misunderstandings about the contract between in-house and third-party libraries can lead to failures.

Given the critical functionality that these components provide, *WearFit* takes great care to identify, understand, and mitigate risk introduced by third-party code. All third-party components are strictly revision-controlled and only approved versions of known components are allowed. Developers who wish to leverage a new third-party component must complete a security review before the component is approved for use. As part of regular integration builds, static analysis is used to identify specific component versions and to report any known vulnerabilities that have been identified.

> Given the critical functionality that these components provide, *WearFit* takes great care to identify, understand, and mitigate risk introduced by third-party code.

The organization maintains a detailed inventory of each component not developed in-house. For each component, a security review is conducted to determine a standard, secure configuration that disables features or functionality not in use. Depending on the component's nature, reviewers leverage static analysis to review components for unknown vulnerabilities and subscribe to notification lists of new CVEs as they're discovered. The internal security team monitors vulnerability disclosure lists and third-party library mailing lists for issues that require rapid patching or updating. The internal security team also performs penetration testing on the full application before major releases to ensure that individual components behave as expected in the overall system.

When a security problem is identified in a third-party component, the organization employs the same incident response process as they would for vulnerabilities found in internally developed code. If a patch is immediately available, a well-tuned vulnerability management process ensures that patches are tested and deployed rapidly. If no patch has been developed, the organization attempts to mitigate the risk with external controls or, as a last resort, by developing and deploying their own patch.

Be Flexible When Considering Future Changes to Objects and Actors

Avoiding the Top 10 Security Flaws states:

> *Software security must be designed for change, rather than being fragile, brittle, and static. During*

BACK TO TOP OF DOCUMENT

the design and development processes, the goal is to meet a set of functional and security requirements. However, software, the environments running software, and threats and attacks against software all change over time. Even when security is considered during design, or a framework being used was built correctly to permit runtime changes in a controlled and secure manner, designers still need to consider the security implications of future changes to objects and actors.

Securing the *WearFit* ecosystem involves planning for future security lapses, mishaps, and unknowns. It's reasonable to expect over the system's lifetime that quality and security issues will arise that require substantial system changes. Not all changes can be predicted, but advanced planning can avoid security surprises, simplify response procedures, and reduce the overall risk associated with any security breach.

The *WearFit* system is designed with a highly modular architecture where components with different functionality are developed and implemented independently. Also, third-party libraries are used for certain functionality such as for cryptographic routines. Refer to the "Understand How Integrating External Components Changes Your Attack Surface" section of this document for a discussion of how *WearFit* manages security risk in third-party libraries. This modularity increases future flexibility by allowing outdated components and libraries to be independently tested, updated, and replaced as needed.

Internally, the *WearFit* system uses several certificates to secure communication between internal services and also communication with users' mobile devices. Strong passwords are used for making connections with the database and other back-end resources. Future security breaches may require these security tokens to be updated quickly. As a result, the *WearFit* system ensures that these security tokens are never hard-coded into the application source code. Instead, the system follows security best practices by storing the security tokens in platform-provided key stores with restricted access.

The wearable includes functionality to update its firmware, to address any security vulnerabilities that are discovered after initial release. In addition, as cryptographic routines become outdated over time, firmware updates allow for substituting in improved algorithms. This process can't, however, modify the private key provisioned during manufacturing. See the "Use Cryptography Correctly" section in this document for further discussion.

Various *WearFi*t support staff have privileged access to internal webservers and databases used by the application. However, the necessary access for the support staff changes over time as their job requirements change and as system functionality changes. An audit system is

BACK TO TOP OF DOCUMENT

used to track which support staff have access to which components. The audit system is also used for granting and revoking access, as needed, over time.

Conclusion

As we mention in the introduction, this document is part of a series of practical artifacts from the Center for Secure Design. We anticipate delivering several more in 2016.

If you're interested in keeping up with the Center for Secure Design's activities, follow us on Twitter @ieeecsd or via the website (cybersecurity.ieee.org). If you would like to help with CSD activities, contact us at ieee-csd@ieee.org.

Acknowledgments

This document came to fruition through the collaborative efforts of many participants at the CSD's 2015 workshops. In particular, we thank, Jeremy Epstein, Tammy Green, Gary McGraw, Brook Schoenfield, and Greg Shannon for their significant contributions.

BACK TO TOP OF DOCUMENT