SySS
THE PENTEST EXPERTS.

YOU'RE DOING IT WRONG

Real-world security issues in firmware update processes

SPEAKER

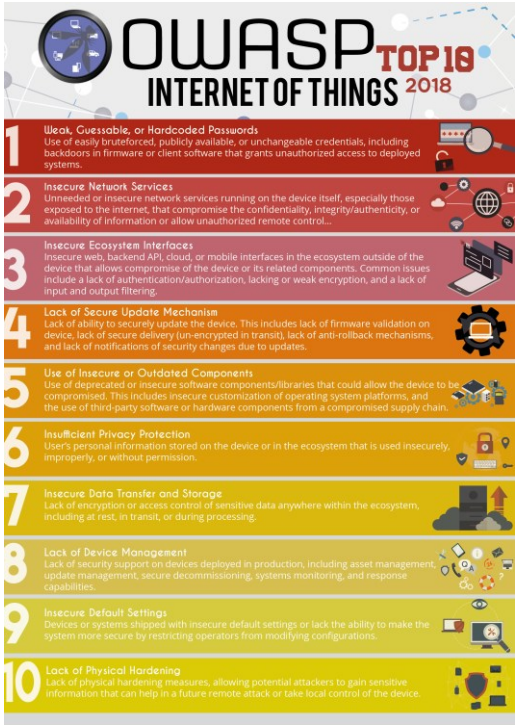
Gerhard Klostermeier

- Bachelor IT Security
- Senior Expert IT Security Consultant
- Team Manager Embedded Systems Security Department
- At SySS GmbH since 2014
- Interested in hardware hacking, IoT, automotive, wireless technologies, NFC/RFID, Android, ...
- E-mail: gerhard.klostermeier@syss.de

SYSS GMBH

- Founded in 1998 by Sebastian Schreiber
- Services: pentest, red teaming, forensics, incidence response, technical consulting, threat intelligence, live hacking, trainings
- Approximately 170 employees
- Branches: Tübingen, Frankfurt/M., Munich, Vienna

WHERE ARE WE?


OWASP TOP 10
INTERNET OF THINGS 2018
1 Weak, Guessable, or Hardcoded Passwords
Use of easily bruteforced, publicly available, or unchangeable credentials, including backdoors in firmware or client software that grants unauthorized access to deployed systems.
2 Insecure Network Services
Unneeded or insecure network services running on the device itself, especially those exposed to the internet, that compromise the confidentiality, integrity/authenticity, or availability of information or allow unauthorized remote control...
3 Insecure Ecosystem Interfaces
Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering.
4 Lack of Secure Update Mechanism
Lack of ability to securely update the device. This includes lack of firmware validation on device, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates.
5 Use of Insecure or Outdated Components
Use of deprecated or insecure software components/libraries that could allow the device to be compromised. This includes insecure customization of operating system platforms, and the use of third-party software or hardware components from a compromised supply chain.
6 Insufficient Privacy Protection
User's personal information stored on the device or in the ecosystem that is used insecurely, improperly, or without permission.
7 Insecure Data Transfer and Storage
Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing.
8 Lack of Device Management
Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities.
9 Insecure Default Settings
Devices or systems shipped with insecure default settings or lack the ability to make the system more secure by restricting operators from modifying configurations.
10 Lack of Physical Hardening
Lack of physical hardening measures, allowing potential attackers to gain sensitive information that can help in a future remote attack or take local control of the device.

input and output filtering.
4 Lack of Secure Update Mechanism
Lack of ability to securely update the device. This includes lack of firmware validation on device, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates.
Use of Insecure or Outdated Components

WHY CARE?

- You have to provide updates for medical devices
- (Personal) liability
- If an attacker can modify/exchange a firmware update it results in a compromised device
- Compromised devices can lead to:
  - Reputation damage
  - Ransom extortion
  - Unusable devices (denial of service)
  - Malware distribution
  - Harm to the patient / loss of life (especially in medical devices)

NEW ATTACK SURFACE

THE PENTEST EXPERTS.

ondrugdelivery.com/bluetooth-connected-wearable-body-injector/

A BLUETOOTH-CONNECTED WEARABLE, ON-BODY INJECTOR | ONdrugDelivery
ondrugdelivery.com

ONdrugDELIVERY

CGT INJECTION OCULAR ORAL RESPIRATORY SUSTAINABILITY MORE SUBSCRIBE

20th June 2016

A BLUETOOTH-CONNECTED WEARABLE, ON-BODY INJECTOR

CONNECTIVITY

Citation: Enable Injections, "Enable Connections: A Bluetooth-Connected Wearable, On-Body Injector". ONdrugDelivery Magazine, Issue 68 (Jun 2016), pp 12-15.

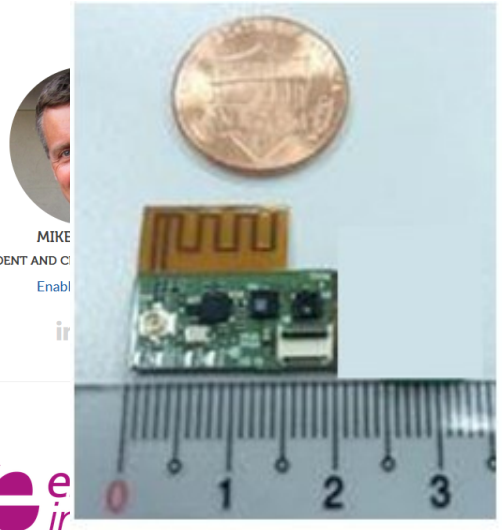
In this article, we explore the vast potential for improving patient care that comes from adding connectivity to devices that deliver biotherapeutics for the treatment of chronic conditions.

zine, for and on behalf of Enable Injections.

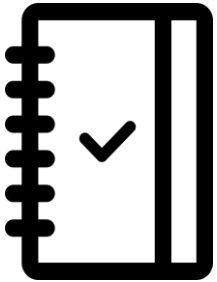
ogies within healthcare continues to improve diagnosis, in the clinical setting. It produces a large amount of data – h industry stakeholders are just coming to terms with. An ore, increasingly valuable data from digital technologies if ct the necessary resources towards building the lth technology to the cloud, for true connected health (CH).

ing etric ad. The has

"Connected Health allows the patient and other stakeholders to

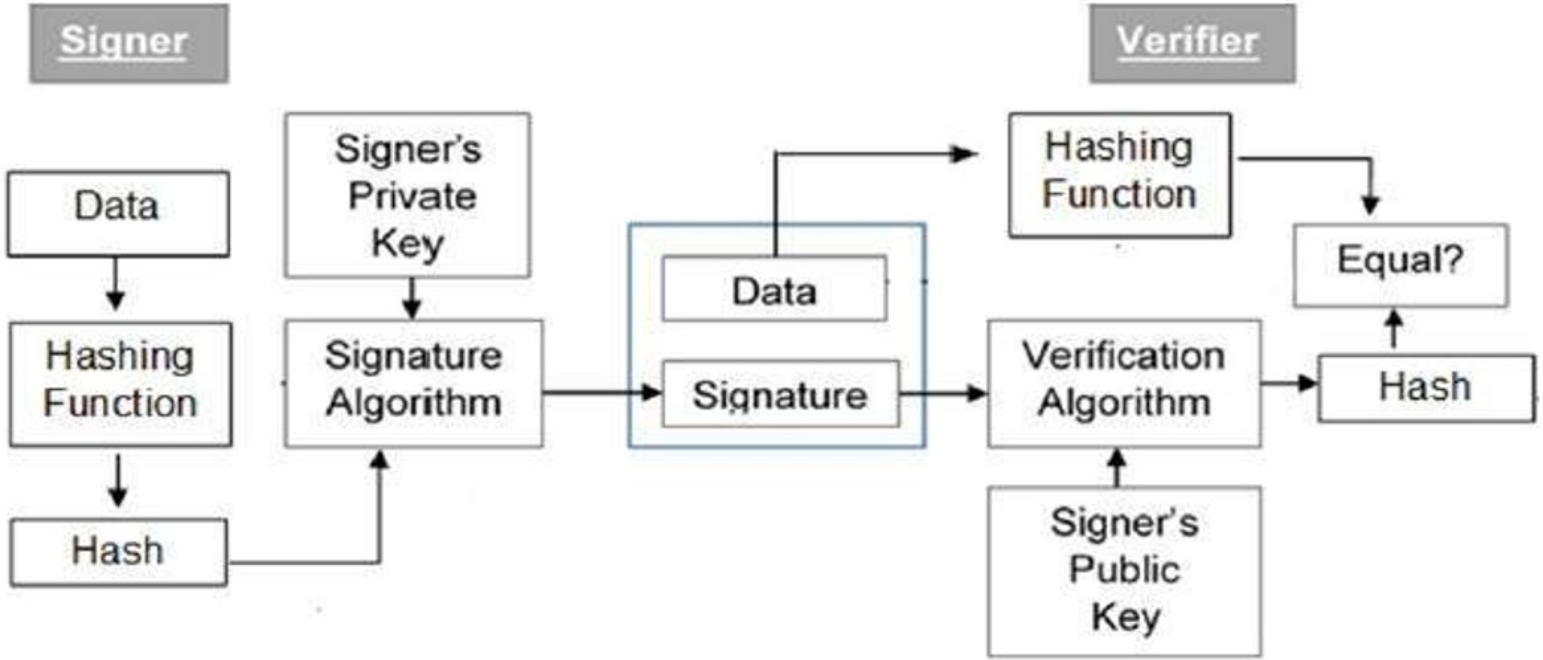


LET'S SECURE THE UPDATE PROCESS



- CIA
  - Confidentiality (C)
  - Integrity (I)
  - Availability (A)

HOW TO SIGN FIRMWARE



https://crypto.stackexchange.com/questions/71355/signature-verification-vs-decryption

THE PENTEST EXPERTS.

The end

Examples of real-world issues

EXAMPLE 1: NO SIGNATURE

- The firmware update is not signed (or encrypted)
- “The update is secure, because it is transferred via HTTPS”
  - Update could be manipulated before or after transmission
- “You can only install updates after entering a PIN”
  - User can be tricked into installing a malicious update
  - PIN verification algorithm can be extracted from firmware

EXAMPLE 2: CRC32

- The “signature” of the update is CRC32
- “We can detect manipulation because of a CRC32 mismatch”
  - CRC32 is not a cryptographically secure
  - It is easy to manipulate a firmware and still keep the same CRC32 (collision)

EXAMPLE 3: SHA512

- The “signature” of the update is a SHA512 hash of its content
- SHA512 is cryptographically secure; manipulation will be detected
- But: Who provides the hash (root of trust)?
  - Appended to the file: Attacker can manipulate file and append a new and correct SHA512 hash
  - Transmitted via HTTP: Attacker can change the hash if used in a plain text protocol
  - Transmitted via HTTPS: Attacker might be able to change the hash before or after transmission
- HMACs can be used to sign updates, but are not very practical due to symmetric cryptography (signing key = verification key)

EXAMPLE 4: ENCRYPTED UPDATE

- The update is encrypted (e.g. using AES)
- “Manipulating the update is not possible; attackers don’t know what is what and changing data will result in faulty data during the decryption”
  - Encryption is for confidentiality, not to detect/prevent manipulation
  - The key is likely the same for all devices and might be extracted from memory/flash
  - Encrypted data can be manipulated to an attackers gain: https://www.jakoblell.com/blog/2013/12/22/practical-malleability-attack-against-cbc-encrypted-luks-partitions/

EXAMPLE 4: ENCRYPTED UPDATE

```
$ echo "Pauli is owed $    1000 dollars" | openssl enc -k fnord -e -aes-
128-ctr -pbkdf2 >ciphertext
$ xxd ciphertext >data
$ cat data
00000000: 5361 6c74 6564 5f5f 0766 9237 007c 4b52  Salted__.f.7.|KR
00000010: ac81 4d00 79b2 1691 1f37 e1b8 fb26 203f  ..M.y....7...& ?
00000020: 5535 d358 be98 b049 9a93 3c02 ff51 20fb  U5.X...I..<..Q .
$ vi data
$ cat data
00000000: 5361 6c74 6564 5f5f 0766 9237 007c 4b52  Salted__.f.7.|KR
00000010: ac81 4d00 79b2 1691 1f37 e1b8 fb26 203f  ..M.y....7...& ?
00000020: 4c25 c359 be98 b049 9a93 3c02 ff51 20fb  U5.X...I..<..Q .
$ xxd -r data >ciphertext
$ openssl enc -k fnord -d -aes-128-ctr -pbkdf2
Pauli is owed $ 9000000 dollars
```

EXAMPLE 5: VERIFICATION MISSING

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - If the signature is missing, the update is always valid
  - During the installation the signature of the update is not verified (any signature will work)

EXAMPLE 6: VERIFICATION NOT ON TARGET

- The update is signed using public key cryptography
- The signature gets verified
- Good, but seen in the wild:
  - The signature verification is NOT done on the target device
  - E.g. verification is done inside the update tool running on a laptop

EXAMPLE 7: ROOT OF TRUST

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - The root of trust (public key for verification) can be easily exchanged
    → e.g. stored in external flash, not OTP area

EXAMPLE 8: STORAGE SIZE

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - The storage size is not big enough to store the update alongside the currently running firmware
  - The update process can only be done by the bootloader
  - Firmware update:
  1. Current firmware is deleted
  2. New firmware is stored
  3. New firmware gets verified before installation
  4. New firmware is invalid → device can no longer boot

EXAMPLE 9: TOC/TOU

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - Time of Check / Time of Use issues (ToC/ToU)
  1. Official firmware is read from e.g. USB flash drive for validation
  2. Official firmware gets validated (signature check)
  3. Attacker manipulates firmware on USB flash drive
  4. Manipulated firmware is read for installation
  5. Manipulated firmware gets installed
  - It was not expected that the data can change quickly in between reads

EXAMPLE 10: PROCESSING BEFORE CHECK

- The update is signed using public key cryptography
- The singed firmware is compressed into a e.g., ZIP archive
- But: Firmware needs to be extracted before validation (and installation)
  - Program code is processing untrusted data (extracting the ZIP)
  - ZIPs can contain file target paths outside the root extraction directory
    → override system files, gain system access
  - ZIP bombs can be used to threaten the availability of a system
    e.g. “42.zip” is 42 kilobytes compressed, 4.5 petabytes uncompressed
    “^.zip” is 203 kilobytes compressed, 16 yottabytes uncompressed

EXAMPLE 11: DOWNGRADE

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - As long as the signature is valid, the update gets installed
  - Old versions of the firmware can be installed
  - Old versions of the firmware may have known vulnerabilities

EXAMPLE 12: PARTIALLY SIGNED

- The update is signed using public key cryptography
- The singed firmware has a header with metadata (e.g. version number, build date, etc.)
- OK, but seen in the wild:
  - Unsigned metadata is trusted and used for different purposes
  - E.g. metadata is used to prevent downgrading firmware → bypass possible
  - E.g. metadata is written to some EEPROM before the update → faulty data bricks the device
  - E.g. metadata has length field that is trusted → buffer overflow?
  - E.g. parser reads byte 4-8 of the metadata as numerical version → what if it is text? or negative?

EXAMPLE 13: SIGNING PROCESS

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - The new firmware is stored somewhere unsafe before signing
    → Firmware might get manipulated right before signing
  - E.g. firmware is transported to air-gaped system using a USB flash drive
  - E.g. firmware is stored before signing on a shared network folder
  - E.g. an attacker can trigger the (automated) signing process for an arbitrary file of their choosing
    → No authentication and no “two-person rule”

EXAMPLE 14: KEY MANAGEMENT

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - The private key for signing is known to all employees in the development team
  - The signing key is known to the person no longer working in the company
  - The signing key can not be changed/revoked
  - The signing key is stored in some cloud for automated off-site signing
  - There is no (off-site) backup of the signing key
  - The backup of the signing key is in a storage where completely different people have access to

EXAMPLE 15: DEBUGGING INTERFACES

- The update is signed using public key cryptography
- Good, but seen in the wild:
  - Debugging interfaces like JTAG or SWD are still active on the device
  - Firmware can be installed by completely bypassing the secure update process

EXAMPLE 16: NOT ENCRYPTED

- The update is signed using public key cryptography
- Updates are not encrypted
- Ok? It depends…
  - Does the firmware include sensitive data
    → (API) keys, passwords, etc.
  - Does the firmware include sensitive code
    → Intellectual property, key generation/verification algorithms, etc.

WHAT NOW?

- Skip well-know pitfalls by knowing typical mistakes
- Use well-known frameworks (e.g. SWUpdate, RAUC, etc.); don’t reinvent the wheel
- Design good tests for security requirements
- Do different kinds of testing
  (e.g. unit tests, vulnerability scans, penetration tests, fuzzing, (AI assisted) reviews, etc.)
- Train you personal
  (e.g. a developer with no security training might struggle to implement good security)
- Think like an attacker/hacker

GOOD TESTS

- Design good tests for security requirements
- Example: “Firmware must be signed”
  - Bad: Test if firmware update has a signature appended
  - Good: Test if faulty and wrong signatures are detected and declined upon firmware installation
  - → “Firmware must be signed to prevent the installation of modified firmware updates”
  - → “Firmware must be cryptographically signed using asymmetric keys to detect and prevent …”
- Good test cases rely on good requirements engineering

THINK LIKE AN ATTACKER

- Think like an attacker
  - Assets
  - Attack surface
  - Use-cases
  - Typical security issues
  - Be creative
- Example: “Communication must be secured”
  - Developer: “I use AES to encrypt it”
  - Attacker: “AES128 or 256? What mode is used (ECB/CBC/…)? Where is the key stored? Is the key the same on all installations? How gets the key transferred? Can the data be inspected before encryption? What about manipulating encrypted data? Is authentication relevant for the protocol? …?”

COMPLIANCE VS. “REAL SECURITY”

- The intent of improving security with regulation is good, but: security is expensive, time-consuming and most customers “don’t really care”. Result:
  - Vendors work their way around to be compliant
  - Bad/wrong analysis scope is chosen to reduce impact
  - Auditors are misled
  - Risks are passed on to users/operators/integrators
  - Products are not really secure
- Think of security compliance as a chance!
  - You have to do it anyway, why not make a good product along the way?

QUESTIONS?

Contact

- E-mail: gerhard.klostermeier@syss.de
- Phone: +49 172 7239126
- Mastodon/Twitter/X: @iiiikarus

Rate this talk
(feedback)

THE PENTEST EXPERTS

WWW.SYSS.DE