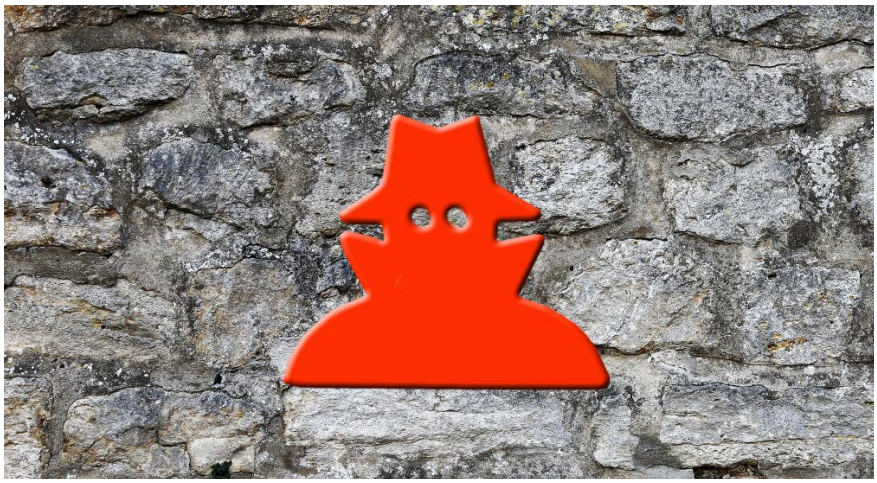


IIoT-Tutorial, Teil 2: Firmware- und Netzwerksicherheit verbessern

Stein auf Stein

Alexander Poth

Kommunikationsschnittstellen, Bootloader, Speicher, Dienste – auch die Software in Embedded Devices bietet viele Angriffspunkte. Hier hilft nur ein systematisches Vorgehen beim Absichern und das automatisierte Scannen von Sicherheitslücken.

Der erste Teil des Tutorials zur sicheren Gestaltung von IoT-Geräten erläuterte, wie man Hardware weitgehend vor physischen Angriffen schützen kann. Dieser zweite und abschließende Teil stellt die sichere Implementierung von IoT-Software vor.

Embedded-Firmware enthält oft sensible Daten. Das gilt auch für Images, die man zum Aktualisieren der Firmware herunterlädt. Wer deren Daten extrahiert, kann damit im Zweifelsfall exponierte Webservices, Datenbanken, Datei- oder Terminaldienste ausnutzen, um weitere Informationen zu ergaunern oder zu manipulieren.

Die Methoden zur Absicherung von Speicherbausteinen, mit denen sich der erste Teil des Tutorials beschäftigte, stellen zwar eine Hürde, aber keine vollständige Absicherung gegen Zugriffe auf den Speicherinhalt dar. Nahezu vollständig kann man das nur mit dem Verschlüsseln sämtlicher Inhalte erreichen. Allerdings ist hier auf die richtige Implementierung zu achten. Erfahrungsgemäß ist oft wichtiges Schlüsselmaterial wie Private Keys ungesichert, also im Klartext abgelegt. Erbeutet ein Angreifer den Schlüssel, kann er die Firmware ohne große Schwierigkeiten entschlüsseln.

Damit ein Angreifer den Speicherinhalt respektive die Firmware nicht manipulieren kann, ist die eingesetzte Firmware zu signieren. Nur so lässt sich die Quelle als vertrauenswürdig verifizieren. Besonderes Augenmerk sollte dabei dem Bootloader gelten. Oft gelingt es einem Angreifer, den Bootprozess so zu manipulieren, dass er nachgelagerte Signaturen des Kernels oder des Dateisystems nicht berücksichtigt oder ein alternatives Betriebssystem startet. Deshalb ist bereits die Authentizität des Bootloaders zu prüfen, um die nachfolgenden Prozesse zu sichern.

Zur Implementierung eines solchen Secure-Boot-Mechanismus bei Embedded-Systemen existieren mittlerweile viele Anleitungen. Zur sicheren Entwicklung solcher Mechanismen ist aber der Einsatz eines Hardwaresicherheitsmoduls erforderlich. Es stellt neben den gängigen kryptografischen Verfahren das sichere Speichern von PINs und Schlüsselmaterial bereit. Hardwareangriffe wie das Auslesen integrierter Schlüssel sind damit nahezu unmöglich. Ein bekanntes Beispiel sind die bei Notebooks oft verwendeten TPM-Chips.

Update-Mechanismen absichern

Ist das Gerät bereits in Verwendung und soll durch ein Update Bugfixes oder neue Features erhalten, stehen diverse Update-Mechanismen zur Auswahl. Einspielen lässt sich die Firmware etwa manuell über eine USB-Schnittstelle oder – die modernere Variante – drahtlos over the air. In beiden Fällen ist das Image über einen verschlüsselten Kanal zu übertragen. Allerdings reicht eine Verschlüsselung des Kommunikationskanals allein oft nicht aus. Die übertragene Datei selbst sollte zunächst verschlüsselt, über einen per TLS/SSL gesicherten Kanal übertragen und erst auf dem Gerät für den Update-Prozess entschlüsselt werden.

Das Firmware-Image sollte zudem signiert sein, damit die Vertrauenswürdigkeit der Quelle geprüft werden kann. Der zugehörige private Schlüssel zum Signieren sollte sich nicht, wie Erfahrungen bereits gezeigt haben, auf derselben Backend-Instanz befinden. Erlangen Angreifer die Kontrolle über diese Instanz und damit über die

iX-TRACT

- Kann man mit dem Härten der IoT-Hardware Angriffe zumindest erschweren, liegt die hohe Kunst im Härten der Software.
- Unter Zuhilfenahme fertiger Skripte, Checklisten und Sicherheitsscanner unterschiedlicher Art lassen sich Fehlkonfigurationen und Sicherheitslücken in der eigenen Software aufspüren.
- Eines aber ist auf allen Kommunikationskanälen Pflicht: verschlüsseln, verschlüsseln, verschlüsseln.

Schüssel, können sie Updates manipulieren, signieren, auf die Geräte pushen und installieren. Unabhängig davon können starke Verschlüsselungen und Signaturen Man-in-the-Middle-Angriffe verhindern.

Eines der häufigsten Sicherheitsrisiken ist das Verwenden veralteter Software mit bereits bekannten Schwachstellen. Der Grund für das Angriffspotenzial liegt in der zum Teil detaillierten öffentlichen Dokumentation der Schwachstellen. Zudem existieren bereits Tausende fertige Exploits, mit denen ein Angreifer quasi per Knopfdruck ein Programm oder einen Service kompromittieren kann, um einen Zugang zum System zu erhalten. Aus diesem Grund sollte sämtliche verwendete Software auf einem sicheren, aktuellen Stand betrieben und gehalten werden. Dies betrifft neben exponierten Netz- und Webservices auch den Kernel und den Bootloader.

Um zu prüfen, ob Software bereits von bekannten Schwachstellen betroffen ist, lassen sich CVE-Datenbanken (Common Vulnerabilities and Exposures) wie cvedetails.com heranziehen. Auch wenn bei der Entwicklung meist die neueste und sicherste Software eingesetzt wird, wird diese über die Lebenszeit des Systems oft nicht aktualisiert. Ein kontinuierliches Patch-Management, das neue Veröffentlichungen frühzeitig auf die jeweiligen Geräte ausrollt, ist das Mindeste, was in Betracht gezogen werden sollte.

Webapplikationen nicht unterschätzen

Eine weitere oft vernachlässigte Komponente von IoT-Geräten sind Webapplikationen. Oft befinden sich auf den Geräten selbst entwickelte Webanwendungen, die einem Benutzer verschiedene Darstellungs- und Konfigurationsmöglichkeiten bieten. Meist sind diese Applikationen durch eine Anmeldung gesichert, sodass nur authentifizierte Nutzer auf das Gerät zugreifen können.

Gerne wird angenommen, dass sich ein Angreifer höchstens Zugang zur Webapplikation verschaffen kann. Durch Brute-Force-Angriffe, schlechte Passwortrichtlinien oder das unterlassene Deaktivieren von Standardpasswörtern sind solche Angriffe zwar realistisch, aber bergen nicht das größtmögliche Risiko.

Vor allem bei IoT-Geräten implementiert die Webapplikation oftmals Funktionen, die Einstellungen auf Systemebene vornehmen. Manipuliert ein Angreifer solche Konfigurationsfunktionen, kann er etwa per Command Injection Systembefehle ausführen. Je nach Berechtigung des Webservers oder durch weitere Fehlkonfigurationen kann er das Gerät so vollständig übernehmen. Ist das Gerät beispielsweise in eine Cloud eingebunden, kann er das Authentifizierungsmaterial ab- und auf die Cloud-Instanz zugreifen, die unter Umständen selbst von unsicheren Fehlkonfigurationen betroffen ist. Das hier beschriebene Ausbreiten innerhalb einer IT-Infrastruktur und das Übernehmen weiterer Systeme nennt man Lateral Movement.

Die sichere Entwicklung von Webanwendungen umfasst sehr viele Teilaspekte. Aus diesem Grund sollte man beim Entwickeln und Testen von Webapplikationen die OWASP Top 10 konsultieren (siehe auch ix.de/z37t). Hier hat das Open Web Application Security Project die zehn kritischsten Schwachstellen von Webanwendungen detailliert beschrieben. Die Aufstellung beinhaltet außerdem Anleitungen für Entwickler zur Vermeidung dieser zehn Risiken. In der IT-Sicherheitsbranche hat sich dieser eher technisch orientierte Standard zu einem globalen Referenzwerk entwickelt.

Berechtigungen zurückhaltend vergeben

Gelingt es einem Angreifer, einen exponierten Service zu hacken, hat er Zugriff auf das System mit den Rechten des angegriffenen Dienstes. Damit er dadurch nicht umgehend Admin- respektive Root-Zugriff auf das Gerät erhält, sollte man Dienste mit eingeschränkten Rechten unter einem eigenen User-Account ausführen. Zudem dürfen schützenswerte Daten nicht für alle Benutzer oder Gruppen auf dem System lesbar oder gar schreibbar sein. Oft führen World-Writeable-Files zur Ausweitung der Zugriffsrechte.

Zur Erläuterung dieser Privilege Escalation soll ein Beispiel aus einem vergangenen Penetrationstest dienen: Während des Tests eines IoT-Gerätes ließ sich unter Zuhilfenahme einer bekannten Schwachstelle im exponierten Webserver eine System-Shell aufrufen. Diese Linux-Konsole erlaubte das Ausführen von Befehlen als Benutzer www-data. Da dieser Benutzer über wenige Rechte verfügt, suchte der Analyst nach schreibbaren Dateien, die eine Rechteerhöhung ermöglichen, und wurde fündig.

Die Webapplikation des Gerätes war aus bis dato unerklärlichen Gründen nach 30 Stunden Laufzeit nicht mehr erreichbar. Der Entwickler des Gerätes behalf sich mit einem Workaround und erstellte einen

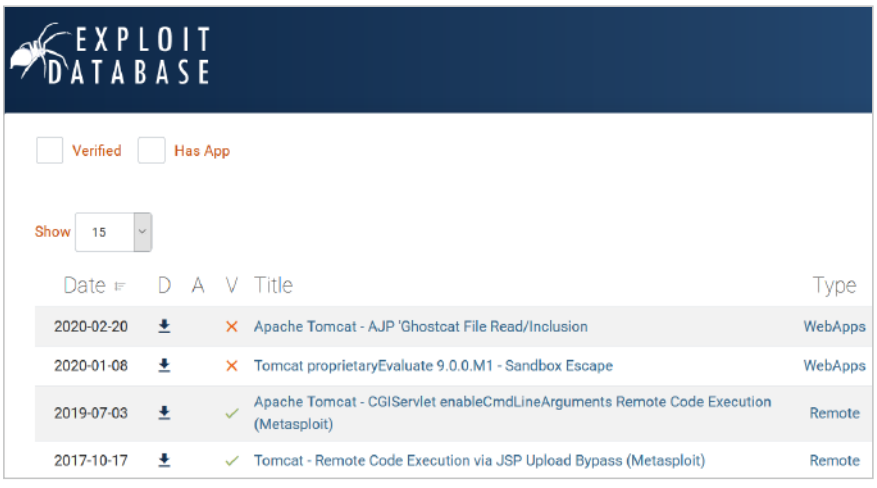
Tutorialinhalt

Teil 1: Hardwaresicherheit von IoT-Geräten

Teil 2: Firmware- und Netzwerksicherheit von IoT-Geräten

Quelle: Microchip

Ein Hardwaresicherheitsmodul besteht aus einem kryptografischen Coprozessor und Speicher für die Schlüssel (Abb. 1).



Quelle: NSIDE

In der Exploit Database sind fertige Exploits zu Tausenden Services öffentlich verfügbar (Abb. 2).

Skripte und Checklisten zum Erkennen von Privilege-Escalation-Lücken

Betriebssystem	Projekt	Beschreibung	URL
Linux	LinEnum	Linux Privilege Escalation Script von rebootuser	github.com/rebootuser/LinEnum
	linPEASS	Linux local Privilege Escalation Awesome Script (.sh)	github.com/carlospolop/privilege-escalation-awesome-scripts-suite
	HackTricks	Checklist for privilege escalation in Linux	book.hacktricks.xyz/linux-unix/linux-privilege-escalation-checklist
Windows	PowerUp	PowerUp des PowerSploit Frameworks (kein Support mehr)	github.com/PowerShellMafia/PowerSploit
	JAWS	Just Another Windows Enum Script	github.com/411Hall/JAWS
	winPEASS	Windows local Privilege Escalation Awesome Script (C#.exe and .bat)	github.com/carlospolop/privilege-escalation-awesome-scripts-suite
	HackTricks	Checklist for privilege escalation in Windows	book.hacktricks.xyz/windows/checklist-windows-privilege-escalation

Cron-Job, der alle 24 Stunden ein Shellskript mit Root-Rechten ausführte, um den Webserver neu zu starten. Das Shellskript war jedoch für alle vorhandenen Benutzer schreibbar. Der Analyst konnte es so bearbeiten, dass bei Ausführung eine Reverse-Shell für den Analysten geöffnet wurde. Da Cron das Skript alle 24 Stunden mit Root-Rechten ausführte, musste der Analyst lediglich auf die eingehende Verbindung der Root-Shell warten.

Eine weitere, in der Praxis oft gesehene Schwachstelle bilden Sticky-Bits und SUID-Rechte. In Linux lassen sich Dateien so konfigurieren, dass sie immer mit den Rechten des Erstellers ausgeführt werden – in den meisten Fällen Root. Findet ein Angreifer innerhalb der ausführbaren Datei beziehungsweise des Programms eine Möglichkeit, Dateien zu beschreiben oder anzulegen, kann er eine vollständige Eskalation zum Admin herbeiführen, indem er etwa die Shadow-Datei überschreibt, die sämtliche Benutzerpasswörter beinhaltet und zur Authentifizierung dient.

Es gibt einige öffentliche Skripte zum Erkennen solcher Fehlkonfigurationen. Diese haben sich bei Penetrationstests bewährt und lassen sich ebenso in der späten Entwicklungsphase nutzen. Die Tabelle „Skripte und Checklisten zum Erkennen von Privilege-Escalation-Lücken" listet die wichtigsten auf (siehe auch ix.de/z37t).

Die Angriffsfläche verkleinern

Um exponierte Services auszuhebeln oder Rechte zu erhöhen, nehmen Angreifer häufig die Speicherverwaltung des Dienstes ins Visier. Buffer Overflows beispielsweise lassen sich zum Ausführen von Systembefehlen nutzen. Aus diesem Grund empfiehlt es sich, auf Sicherheitsmechanismen zurückzugreifen, die das Betriebssystem bereitstellt. Dazu zählen:

- **ASLR:** Die Address Space Layout Randomization weist Programmen einen zufälligen Adressbereich zu. Hierdurch ist er praktisch nicht mehr vorhersagbar. Das soll Angriffe wie Buffer Overflows erschweren. Seit dem Linux-Kernel 3.14 gibt es eine vollständige ASLR-Implementierung.
- **PIE:** Ein als Position Independent Executable kompiliertes Programm und alle für seine Ausführung notwendigen Komponenten lädt das Betriebssystem bei jeder Ausführung an zufällige Stellen im virtuellen Speicher. Dies erschwert ROP-Angriffe (Return-oriented Programming) erheblich.
- **Stack Canaries:** Ein separater Schutz der auf dem Speicherstack gespeicherten Rücksprungadresse bildet eine weitere Möglichkeit, ein System vor Buffer Overflows zu schützen. Hierzu wird zwischen dem Buffer und der Rücksprungadresse ein zusätzlicher Canary-Parameter abgelegt. Die meisten gängigen Compiler wie der GCC implementieren Stack Canaries.
- **RELRO:** Mit der generischen Exploit-Mitigation-Technik Relocation Read-only härtet man Datenabschnitte eines ELF-Binary oder -Prozesses. RELRO kennt zwei Modi: Beim Partial RELRO ordnet der Compiler die ELF-Abschnitte neu an, indem er den Datenabschnitten des Programms .data und .bss die internen ELF-Datenabschnitte, etwa .got und .dtors, voranstellt. Dabei ist die Nicht-PLT-abhängige (Procedure Linkage Table) GOT (Global Offset Table) schreibgeschützt, während die PLT-abhängige GOT beschreibbar bleibt. Im Unterschied dazu nimmt der Compiler beim Full RELRO ein Remapping der gesamten GOT als schreibgeschützten Datenabschnitt vor.

Ob Programme solche Techniken nutzen, lässt sich manuell oder automatisiert prüfen. Für das automatisierte Suchen von Schwachstellen innerhalb von Linux-Programmen empfiehlt sich das Werkzeug Checksec (siehe auch ix.de/z37t).

Bei einer vollständig selbst entwickelten Firmware können die erwähnten Tools bereits einige versehentlich implementierte Fehlkonfigurationen identifizieren. Ein universelles Werkzeug, das einen Großteil der hier erwähnten Schwachstellen aufdecken kann, ist FACT. Das vom Fraunhofer-Institut für Kommunikation, Informationsverarbeitung und Ergonomie FKIE entwickelte Firmware Analysis and Comparison Tool erkennt eine Vielzahl der vorgestellten Schwachstellen innerhalb eines Firmware-Image. Das Tool bietet eine – wenn auch noch nicht ausführlich dokumentierte – API, wird regelmäßig Updates unterzogen und ist kostenlos auf GitHub verfügbar (siehe auch ix.de/z37t).

Den Datenverkehr verschlüsseln

Nicht nur bei Firmware-Updates ist es zwingend erforderlich, sämtlichen Datenverkehr zu verschlüsseln. Bereits in der Planungsphase ist zu überlegen, welche Protokolle oder Dienste von Haus aus TLS/SSL oder weitere Sicherheitsmechanismen nutzen. Unverschlüsselte Daten sind für einen erfahrenen Angreifer sehr einfach abgreifbar. Oft genügt es, wenn er sich im selben lokalen Netz befindet. Möglich ist dies etwa per ARP-Spoofing, also dem Versenden gefälschter ARP-Pakete, sodass der lokale Datenverkehr über den Angreifer läuft.

Erfahrungsgemäß wird TLS/SSL aber oft falsch konfiguriert. Beispielsweise verwenden viele weiterhin schwache Chiffrierverfahren wie RC4. Auch wenn ein Angreifer in diesem Fall mehrere GByte verschlüsselter Daten sammeln muss, reicht dies unter Umständen aus, um wenige Pakete mit vertraulichem Inhalt zu entschlüsseln. Zum Überprüfen der TLS/SSL-Konfiguration kann das Open-Source-Werkzeug TestSSL herhalten, das automatisiert gängige Schwachstellen respektive Fehlkonfigurationen aufdeckt (siehe auch ix.de/z37t).

Selbstredend sollte nicht jeder Client im lokalen Netz auf Services zugreifen können, die nur für ausgewählte Geräte bestimmt sind. Im IoT haben sich X.509-Client-Zertifikate zur Zugriffskontrolle durchgesetzt, weshalb gängige IoT-Techniken wie der MQTT-Service Mosquitto diese von Haus aus einsetzen. Diese Zertifikate sollten allerdings nicht unverschlüsselt auf den Speicherbausteinen liegen.

Ein großes Einfallstor bilden offene Ports, die für die Funktion des Geräts nicht nötig sind. Erfahrungen haben gezeigt, dass

Dienste wie SSH, Telnet oder VNC für Entwicklungszwecke konfiguriert, aber für den Produktivbetrieb nicht abgeschaltet werden. Einen Überblick über offene Ports und verfügbare Services auf einzelnen Rechnern oder im Netz verschafft man sich am einfachsten mit dem Netzwerkscanner nmap.

Dienste zum Schweigen bringen

Darüber hinaus kann nmap aufzeigen, welche für einen Angreifer nützlichen Informationen Systeme exponieren. Dienste geben oft Informationen wie Serviceart und -version zurück. Meist lässt sich diese Art von Banner in den jeweiligen Servicekonfigurationen deaktivieren. Ein Gerät sollte also nur für den Verwendungszweck benötigte Services bereitstellen, die wiederum keine unnötigen Informationen preisgeben sollten.

Auch die im IoT häufig anzutreffenden Funktechniken wie WLAN oder Bluetooth spielen eine wichtige Rolle bei der Datensicherheit. Der Datentransfer darüber ist selbstredend zu verschlüsseln und über sichere Authentifizierungsverfahren zu initiieren. Im WLAN sollte das Verwenden veralteter Verschlüsselungsprotokolle wie WEP oder WPA1 ausgeschlossen sein.

Über diese Protokolle kann ein Angreifer den Vier-Wege-Handshake zur WLAN-Authentifizierung mitschneiden, den darin übermittelten PSK in gehashter Form extrahieren und diesen mit Werkzeugen wie aircrack-ng entschlüsseln. Ein weiteres Einfallstor bietet die standardgemäß aktivierte WPS-PIN-Authentifizierung. Ist sie unsicher konfiguriert, kann ein Angreifer per Brute-Force-Attacke den WPS-PIN erraten und damit Zugriff auf das lokale Netz erlangen.

Bei Bluetooth und BLE (Bluetooth Low Energy) ist von veralteten Pairing-Mechanismen abzusehen. Hier muss aber der Schutzbedarf der Anwendung und der Daten gegen die Rückwärtskompatibilität abgewogen werden. Wer einen hohen Schutzbedarf hat, sollte Geräte mit älteren Versionen vom Verbindungsaufbau ausschließen. Wer das nicht kann, sollte über die Umsetzung der Sicherheitsziele auf der Anwendungsebene nachdenken.

Zu beachten ist auch der Secure Connection Only Mode in BTLE. Erst kürzlich wurde eine Schwachstelle in der Spezifikation festgestellt, durch die sich sensible Daten unverschlüsselt abfangen lassen.

Was bleibt

Obwohl externe Schnittstellen wie Cloud-Services oder Mobile-Apps ebenfalls zur IoT-Infrastruktur gehören, seien sie hier nur kurz erwähnt. Eine detaillierte Betrachtung würde den Rahmen des Tutorials sprengen. Auch für sie gilt: Die Verschlüsselung des Datenverkehrs ist ein Must-have. Wer tiefer in die Materie einsteigen will, sollte die ausführlichen Anleitungen von OWASP zu Angriffsmethoden auf APIs oder Mobile-Apps heranziehen (siehe auch ix.de/z37t). (sun@ix.de)

Quellen

Alle Dokumentationen und Werkzeuge unter ix.de/z37t

Alexander Poth

ist IT-Security-Analyst bei NSIDE ATTACK LOGIC. Zu seinen Schwerpunkten zählt die Sicherheit von IoT-Geräten.