Integritetsverifikation inom Inbyggda System

Måldokument

Axel Blomén Douglas Fjällrud

2026-03-13

Innehållsförteckning

Bakgrund

Vid utveckling av inbyggda system förekommer ofta löpande underhåll och vidareutveckling med därtill följande uppgraderingar av system i fält. För att genomföra dessa på ett säkert sätt använder man modern kryptografi för att utvinna vissa garantier om ett slutsystems integritet och authenticitet, och säkerställer därmed att ingen obehörig aktör har modifierat systemet. Dessa säkerhetsmekanismer refereras ofta till som paraplybegreppet 'secure boot' i vardagligt språk.

Ökad driftsäkerhet står som huvudsaklig sammhällsnytta vid applikation av denna typen av säkerhet. Problemets natur medför en vis tolkningsöppenhet som resulterat i en mångfald olika metoder, tekniker och implementationer. Tydliga riktlinjer för tillvägagångssät, underhåll och teknikval är något som för många ses som en öppen fråga, i synnerlighet rörande exotisk eller ny hårdvara.

Ämnet säkerhet präglas av utsmyckade kvalitetskrav, hotbildsanalyser och auditeringar. Dessa säkerhetskrav varierar beroende på applikation. För en bild av varför säkerhet inom inbyggda system är viktigt vänder vi oss till vanliga applikationer:

- Medicinteknik
- Larm och telekommunikation
- Autopiloter för flyg, tåg och bilar
- Reglertekniska lösningar inom industri

I många fall finns det tydlig koppling till samhällskritisk verksamhet. I dessa fall innebär ett kompromminterat system **nästan alltid en fara för liv och hälsa**. Flera av dessa går även att kategorisera under **hot mot rikets säkerhet**.

Vårt arbete, kommer att beröra just dessa frågor. Det är en fördel om läsaren har förståelse inom applicerad kryptografi samt datorarkitektur.

Teknisk Beskrivning

De säkerhetsmekanismer vi avser behandla befinner sig på olika lager av målsystemet beroende på vilka garantier som önskas. Något som kännetecknar dessa system är att nästkommande lagers säkerhet ofta beror på föregående lager.

De övre lagrena är ofta flexibla, och erbjuder därmed stor frihet i sin tillämpning. Eftersom denna flexibiliteten också bidrar till ökad felpotential måste systemutvecklaren utöva extra aktsamhet i designstadiet då systemets natur är mycket svårförändrat efter provisionering.

I kontrast är de nedre lagrenas funktionalitet utom utvecklarens kontroll. Ofta implementeras de via diskreta kretsar eller icke-muterbart minne från fabrik och exponerar i regel enbart ett fåtal minnesregioner för modifikation av systemutvecklaren. Dessa lagrenas funktionalitet och omfattning är nästan alltid specifik till processortillverkaren.

Dessa minnesregioner, hädanefter refererade till som 'register', innefattar ofta en eller flera nyckelregister. Här placeras nyckelrelaterad metadata från produktutvecklaren, vars syfte är att utgöra en sorts 'root-of-trust'. Vid plantering av dessa nycklar sker en sorts envägstransak-

tion, och nycklarna blir en permanent tillsats till målenheten. Tanken med systemet är att enbart part med tillgång till korresponderande signeringsnycklar skall kunna signera mjukvara till enheten efter att denna transaktion tagit plats.

Förlust eller läckage av dessa signeringsnycklar innebär att ingen, respektive alla, kommer att kunna signera mjukvara för målenheten. Skadeverkanspotentialen varierar helt beroende på enhetens verkningsområde, men är ofta omfattande.

Ofta är implementationsdetaljer kring de ovan nämnda 'nedre lagrena' sekretessbelagda. Systemutvecklaren tilldelas information på begäran, ibland under tystnadsplikt. Detta bedöms inte vara ett hinder i denna utredningen, men är nämnvärt i kontext.

Något som komplicerar problemet är att det ofta är praxis att använda unika nycklar för olika enheter och lager. Termer som 'fleet-wide' och 'device specific' används för att beskriva hur pass generella nycklar är.

Fortsättningsvis har nycklar ofta en livstid som begränsar dess giltighet. De flesta nycklar är utbytbara efter provisionering. Det är som ovan nämnt praxis att använda sig av olika nycklar för olika typer av mekanismer.

Problemformulering

Flera stående frågor uppstår inom vårt applikationsområde. Nedan presenteras ett urval.

- Vilka nyckelkategorier bör befinna sig samt vanligtvis befinner sig på ett system i denna kategorin?
- Vilka nycklar skall gälla, och för vilken enhet?
- Hur länge skall en nyckell gälla innan den roteras?
- Vem skall ha tillgång till signering av maskinvara?
- Hur provisioneras nycklar i tillverkningsprocessen?
- Hur planterar systemutvecklaren produktionsnycklar på målenhetenen efter provisionen?
- Vilka typer av attacker bör tas särskild hänsyn till i designprocessen kring ett system i denna kategori?

Flera olika standardiseringsorganisationer lägger i varierande grad mandat på flera av de ovanstående punkterna. Den centrala frågan blir då; vad har de gemensamt? Till följd; går de att sammanfatta på ett tillgängligt sätt som aktivt bidrar till säkrare system i vårt samhälle?

Målformulering

Vårt uppdrag är att redogöra för existerande lösningar i en jämnförande kvalitetsanalys avseende kryptografi, verktyg, brister och relation till rådande standarder. Med denna kontext presenterar vi en hållbar metodik inom nyckelhantering samt provisionering inom vårt problemområde.

Vi planerar att initialt beröra följande aspekter:

- Tillgänglighet och öppenhet. Finns verktyg, kod och dokumentation tillgängligt under öppna licenser?

- Kryptografi. Använder sig plattformen av säkra och erkända algoritmer på ett idiomatiskt sätt?
- Simplicitet och användbarhet. Hur pass tillgänglig är plattformens säkerhetsmekanismer?
- Hållbarhet. Vilken typ av säkerhetsmässig livstid kan förväntas?
- Verktyg. Använder sig tillverkaren av erkända, väl auditerade verktyg?

Vi understryker likheter och skillnader mellan olika implementationer och verktyg och bedömer vilka av dessa som går att generalisera med målsättning att kategorisera olika mekanismer till en specifik familj.

Vi utvärderar systematiskt samtida litteratur samt ämnesrelevanta arbetsflöden från industrin rörande ämnet. Vi framför en sammanfattning av olika processer, metoder och modeller för en hållbar och säker strategi. Vårt mål är att detta arbete skall utforma en mall för liknande processer.

Arbetsbeskrivning

För att förankra våra fynd och spekulationer i verkligheten krävs en praktisk implementation. En implementation validerar arbetet och är avgörande för att säkerställa att detta arbete inte förblir en akademisk övning.

Under arbetets gång kommer en praktisk implementation att genomföras på en NXP i.MX 93 -processor. Vi kommer att behandla NXP's egna AHAB (Advanced High Assurance Boot) för implementering av Secure Boot. Vi kommer att dokumentera och utvärdera processen att integrera Secure Boot fristående samt via Yocto [1], en av industrins vanligaste system för att bygga skräddarsydda Linux-system [2]. Särskild uppmärksamhet kommer ges till CI-aspekten ("Continous Integration") och säkerhetsrutinerna kring dessa.

Vidare kommer vi även att behandla RAUC [3]. RAUC är ett slags ramverk som erbjuder funktionalitet för kontinuerliga uppdateringar i fält, där målenheten ofta befinner sig utom fysiskt räckhåll. Flera snarlika system finns öppet att tillgå. Diverse säkerhetsaspekter rörande dessa typer av system kommer att beröras.

I mån av tid kommer vi även att beröra linuxkärnans egna säkerhetsmekanismer så som:

- dm-verity
- dm-integrity
- dm-crypt

Vi begränsar oss till praktisk bekantskap om NXP's AHAB. Med denna kunskap har vi möjligthet att kontrastera övriga implementationer med högre träffsäkerhet.

Ett antal andra processorartillverkare och processorarkitekturer på marknaden har löst secure boot problemet på annat men liknande sätt. Dessa inkluderar, men är inte begränsade till:

- UEFI Secure Boot (vanligt ffa för x86-baserade system)
- Texas Instruments ARM processorer (tex Sitara AM6x)
- Rockchip ARM baserade processorer

- Qualcomms Snapdragon processorer

Vi planerar att studera och dokumentera de sakrelevanta aspekterna av dessa i den mån information finns tillgänglig.

Referenser

[1] Yocto Project Contributors, “The yocto project.” https://www.yoctoproject.org, 2024.
[2] Linux Kernel Developers, “The linux kernel.” https://www.kernel.org, 2024.
[3] RAUC Contributors, “RAUC: Robust auto-update controller.” https://rauc.io, 2024.