Koppla upp, säkra upp,

Så skyddar du säkerhetskritisk IoT mot hackare

Av Dan Smith, chefsingenjör och Andrew Girson, vd, Barr Group

Dan Smith har drygt 20 års erfarenhet av produktutveckling och projektledning inom inbyggda system för konsumentelektronik, industriell styrning, telekom, medicinteknik och fordonselektronik. Han är ofta talare på branschkonferenser och har en kandidatexamen i elektroteknik från Princetonuniversitetet.

Under **Andrew Girsons** ledning har mer än ett företag fått se sina intäkter och lönsamhet växa tvåsiffrigt flera år i rad. Han är medgrundare av Barr group och inledde en gång sin karriär som programvaruingenjör inom inbyggda system. Andrew Girson tog sin magisterexamen i elektroteknik på Virginiauniversitetet.

Uppkoppling av produkter skapar säkerhetsproblem av helt nya slag. Första steget mot en lösning att är förstå relationen mellan data- och personsäkerhet. Därefter finns massor av god praxis att plocka fram i form av kodningsstandarder, statisk analys, kodrevisioner och hotmodellering.

Välkonstruerade program för inbyggda system har klassiskt alltid beaktat både personsäkerhet och datasäkerhet (”safety” och ”security”). Men när inbyggnadssystemen kopplas upp uppstår sårbarheter som är oacceptabla i säkerhetskritiska sammanhang inom exempelvis medicin, autonoma fordon och IoT.

IoT utsätter system för risker ”på distans”. Ett aktuellt exempel är Sonykameror som visade sig ha odokumenterade konton. Dessa fungerade som bakdörrar för hackare som kunde infektera systemen med botnätsprogram som bas för ytterligare attacker.

I detta specifika fall kunde Sony täppa till hålet via en firmwareuppdatering, men kod- och konstruktionsfel är ofta irreparabla. Och ibland katastrofala.

För att bevisa det senare visade två säkerhetsforskare att det enkelt gick att hacka en bil i rörelse. De tog över styrning, transmission och bromsar. 1,4 miljoner fordon fick återkallas.

Farliga inbyggda system fanns långt innan elektroniken blev uppkopplad. Strålterapimaskinen Therac-25 från 1983 används som skolexempel på dålig systemkonstruktion. Patienter utsattes för dödliga stråldoser på grund av programfel, brist på hårdvaruspärrar och allmänt dåliga konstruktionsbeslut.

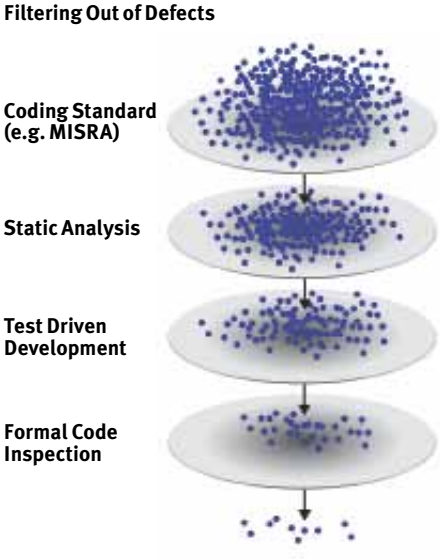
Här är några av Therac-25:s problem:

- Programkoden gick inte att testa
- Analysen av robusthet och feltålighet var bristfällig
- Programvaran granskades inte av en oberoende part
- Tidigare programvara återanvändes på ett felaktigt sätt

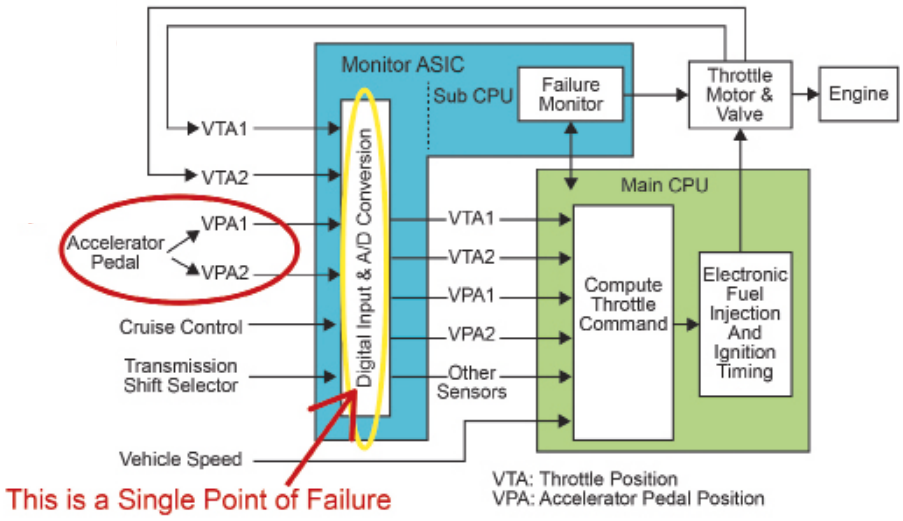
ETT AV FLERA ÖDESDIGRA FEL involverade en åttabitarsräknare i en testrutin. Den överflödade ofta och om en operatör samtidigt gjorde en manuell inmatning sattes en mjukvaruspärr ur spel.

I juni 1996 sprängde en Ariane 5-raket sig själv när den upptäckte att den avvek från avsedd kurs. Ett register hade flödat över, men detta upptäcktes inte eftersom testet hade rationaliserats bort.

Än idag förbises ofta kritiska sårbarheter. På Barr Group gjorde vi år 2016 en enkät



Figur 1. Många lager av kvalitetssäkring och skydd måste användas under hela konstruktionsprocessen om mjukvara och hårdvara ska bli av god kvalitet.



Figur 2. Säkerhetskritiska system får inte ha akilleshälar.

koppla av

bland ingenjörer verksamma i projekt med Internetuppkopplade säkerhetskritiska system – det skulle sätta liv på spel om de hackades. Vi fann följande:

- 50 procent av ingenjörerna tillämpade ingen kodningsstandard
- 17 procent gjorde aldrig kodrevisioner
- 24 procent kunde ”eventuellt” göra kodrevisioner
- Mer än en tredjedel gjorde inte statisk analys

DATASÄKERHET (security) och personsäkerhet (safety) blandas ofta samman. Vissa lider under missuppfattningen att bara deras programkod är bra, så kommer den att automatiskt också vara säker i båda dessa betydelser. Det stämmer bevisligen inte.

Ett personsäkert system är ett system som inte orsakar skada på sina användare eller andra. Ett säkerhetskritiskt system kan orsaka skada eller dödsfall när det inte fungerar, vilket det är konstruktörens uppgift att så långt möjligt förhindra .

Datasäkerhet handlar om ett systems förmåga att se till att behöriga användare kommer åt alla resurser medan obehöriga hålls utanför. Resurserna kan bestå av dynamiska data, programkod, intellektuell egendom, processorer, kontrollcentraler, kommunikationsportar, minnen och databanker.

Därmed är det uppenbart att ett system kan vara datasäkert utan att samtidigt vara personsäkert – en produkt med hög personskaderisk kan ha samma datasäkerhet som en ofarlig produkt.

Något man däremot alltid kan säga är att ett system som saknar datasäkerhet alltid utgör en säkerhetsrisk eftersom det kan tas över av obehöriga.

Konstruktion för säkerhet har många aspekter. Här ska vi fokusera på firmware.

ETT BRA EXEMPEL på en verksamhetskritisk tillämpning är en bil, som kan innehålla uppåt 100 miljoner kodrader. Den är i händerna på en ofta dåligt utbildad och ofokuserad användare – en förare. För att kompensera för användaren adderas nya funktioner i form av kameror, sensorer, V2I och V2V. Mängden programkod fortsätter att öka. Exponentiellt.

Den stora kodmängden gör kodning och felsökning svårare, men mycket av debugtiden kan elimineras om några grundregler följs:

- Partionera hårdvara och mjukvara med avseende på realtidsprestanda, kostnad, uppgraderbarhet, personsäkerhet, tillförlitlighet och datasäkerhet.

Development Process	Integrity Level				
	0	1	2	3	4
Specification and design	ISO 9001	Structured method.	Structured method supported by CASE tool.	Formal specification for those functions at this level.	Formal specification of complete system. Automated code generation (when available).
Languages and compilers		Standardized structural language.	A restricted subset of a standardized structural language. Validated or tested compilers (if available).	As for 2.	Independently certified compilers with proven formal syntax and semantics (when available).
Configuration management products		All software products. Source code.	Relationships between all software products. All tools.	As for 2.	As for 2.
Configuration management processes		Unique identification. Product matches documentation. Access control. Authorized changes.	Control and audit changes. Confirmation process.	Automated change and build control. Automated confirmation process.	As for 2.

Figur 3. MISRA motar buggar och gör kod mer lättläst, konsistent och portabel.

> **”Ett system som saknar datasäkerhet utgör alltid också en säkerhetsrisk eftersom det kan tas över av obehöriga”**

- Skapa avgränsade regioner mellan vilka fel inte kan spridas.
- Undvik akilleshälar, se figur 2.
- Hantera alla sorters undantag (exceptions), oavsett om de orsakas av programlogik, kodbuggar, minneshantering eller sporadiska interrupt.
- Testa för overflow – glöm aldrig Therac-25 och Ariane.
- Tvätta data med okänt ursprung – använd intervallkontroll och checksummor.
- Testa systemet på alla nivåer: enhetstest, integrationstest, systemtest, fuzzing, verifiering och validering, med mera.

KONSTRUKTÖREN MÅSTE BEHÄRSKA de komplexiteter som kommer med autentisering, Public Key Infrastructure (PKI), och datakryptering. Säkerhet måste också innebära att system inte gör oväntade eller farliga saker när något oväntat inträffar eller en attack sker.

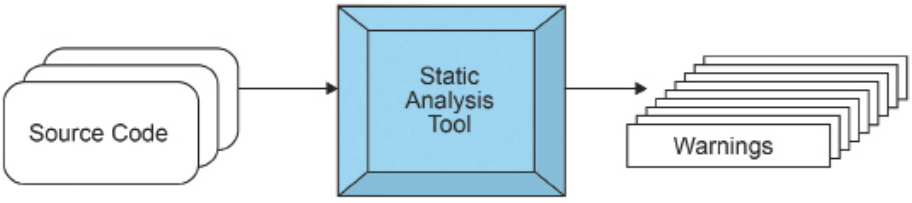
Firmwareuppdatering är en svag länk i säkerhetskedjan, när RFU (Remote Firmware Up-date) är aktiverad. Här är det bra att ha en policy, som att användaren kan inaktivera RFU eller att den kräver auktorisering.

Kryptering är sällan den svagaste länken, även om det kanske låter ointuitivt. Angriparen söker snarare attackvägar i svaga implementationer, protokoll, programgränssnitt, användningsmönster och sidokanaler.

Här är några generella åtgärder som minskar en produkts sårbarhet:

- Använd en styrkrets utan externa minnen
- Inaktivera JTAG-gränssnittet.
- Använd secure boot.
- Generera enhetsspecifika nycklar ur en huvudnyckel.
- Förvräng (obfuscate) objektkoden.
- Använd POST (power-on-self-test) och BIST (built-in-self-test).

APROPOS FÖRVRÄNGNING finns en föreställning om security-through-obscurity. Men den är livsfarlig eftersom hemligheten i sig utgör en akilleshäl. Förr eller senare läcker alla hemligheter ut om det så sker genom social ingenjörskonst, missnöjda anställda, dumpning eller reverse engineering. Hemlighållande har sin roll, förstås, exempelvis ▶



Figur 4. Statisk analys simulerar källkoden och analyserar syntax och programlogik. Utdata är varningar, inte objektkod.

när det gäller kryptonycklar.

Några grundläggande åtgärder kan säkerställa att ett system säkerhetsoptimeras så mycket som är rimligt.

FÖR DET FÖRSTA kan man använda industri- och tillämpningsspecifika standarder och standarder för kodning och funktionssäkerhet. Här finns bland annat MISRA och MISRA-C, ISO 26262, Automotive Open System Architecture (Autosar), IEC 60335 och IEC 60730.

En kodningsstandard som MISRA motverkar buggar och gör dessutom koden mer lättläst, konsistent och portabel (figur 3).

För det andra: använd så kallad statisk analys (figur 4) som analyserar programkoden utan att exekvera den. Koden simuleras, eller exekveras symboliskt till skillnad från dynamisk analys som identifierar defekter under exekvering i målsystemet.

Statisk analys är ingen universallösning men adderar ytterligare ett lager av säkerhet, eftersom den är mycket bra på att upptäcka potentiella fel som oinitierade variabler, risk för over- och underflow och typfel som att blanda heltal med och utan tecken i samma uttryck. De statiska analysverktygen fortsätter hela tiden att förbättras.

VANLIGTVIS INNEBÄR statisk analys användning av dedikerade verktyg som PC-Lint eller Co-verity, men utvecklare bör också överväga att omanalysera sin egen kod.

För det tredje: utför kodrevisioner. Det ökar kodens korrekthet och stöder underhåll och utbyggbarhet. Kodrevisioner är också till nytta vid återkallelser eller garantireparationer och utkrävande av produktansvar.

För det fjärde, skapa attackträdsmodeller. Detta kräver att utvecklaren tänker som en angripare:

- Identifiera attackmål (träd).
- Bestäm vilka attacker som är möjliga för varje mål.
- Identifiera steg och alternativ för varje attack.

SÄKERHETSOPTIMERING TAR TID och man måste sätta budgetanpassade realistiska mål.

Konkret kan det betyda att man ökar utvecklingstiden med mellan 15 och 50 procent som vigs åt kodgranskning. Vissa system behöver revidera all källkod, andra klarar sig utan det.

Statiska analysverktyg kan ta tiotals till hundratals timmar att installera, men när de väl är en del av utvecklingsprocessen adderar de ingen tid till produktutveckling, och de betalar för sig själva genom att de leder till bättre system.

■