Security analyse omtrent de Source-code van SOFA ATM's.

Inhoudsopgave

Inleiding

Dit document is het rapport van de security analyse van een door SOFA ontwikkeld besturingssysteem voor ATM's. De software is beoordeeld op basis van vier categorieën: structuur, naamgeving, netheid en veiligheid. De bevindingen van dit onderzoek worden in de volgende hoofdstukken uitvoerig beschreven worden. Ook zullen voor ieder gevonden probleem één of meerder alternatieven worden geopperd, waarna de conclusie van de analyse zal worden gepresenteerd. Teven zijn er enkele aanbevelingen opgesteld waarmee SOFA in de ogen van de analisten in de toekomst haar kwaliteit mee verbeteren kan.

Bevindingen

De software is getest op basis van vier categorieën: structuur, netheid, naamgeving en veiligheid. Bij structuur word beoordeeld hoe de code gestructureerd is. Dat wil zeggen: is de code logisch opgebouwd? Zijn alle klassen en variabelen logisch ondergebracht en gemakkelijk vindbaar? Bij netheid wordt beoordeeld of de code op een nette manier geschreven is. Dat wil zeggen: staan er witregels waar die moeten staan? Is alles op correcte wijze geïndend? Is de code goed leesbaar? Bij naamgeving word beoordeeld of de namen van variabelen, klassen en methodes wel of niet logisch zijn gekozen en begrijpelijk zijn. Bij veiligheid wordt gekeken naar beveiligingsrisico's zoals onbeveiligde communicatie processen, onveilige code die ruimte bied aan exploits, enz.

Structuur

Op de structuur van de software valt weinig aan te merken. Het is duidelijk zichtbaar hoe de flow van het programma verloopt, de backend is grotendeels gescheiden van de GUI, en alle methodes en variabelen zijn op logische wijze in hun klassen onderverdeeld. Het is dan ook direct duidelijk welke rol iedere klasse vervuld in het programma.

Netheid

Ook hier geen problemen. De wijze waarop de code vormgegeven is is netjes volgens het boekje, en de code is derhalve ook goed leesbaar.

Naamgeving

De naamgeving van de software voelt grotendeels adequaat aan. Toch zijn er enkele namen die beter gekozen hadden worden, hoewel de leesbaarheid van de code hier weinig door beïnvloed wordt. Eén incident springt echter wel direct in het oog, namelijk in de klasse Saldo, op regel 16 en 17. Hier worden twee variabelen geïnitieerd en zij krijgen de namen i en j. Het is op het eerste gezicht zeer onduidelijk waar deze variabelen voor gebruikt worden, en deze variabelen werken dan ook zeer verwarrend. Uit verder inspectie van de code wordt duidelijk dat in deze variabelen enkele tussenresultaten worden opgeslagen van een complexere berekening. Dat roept de vraag op of deze variabelen dan ook wel noodzakelijk zijn. Het advies is hier dan ook om deze variabelen achterwege te laten en de bewerking in één statement uit te voeren. Het resultaat zou er als volgt uitzien.

In plaats van:

```
string str = MainBackend.strDbQuery(attribute, Program.Rfid);
double i = Convert.ToDouble(str);
double j = i / 100;
int saldo = 0;
doubleBedrag = Convert.ToString(j);
```

het onderstaande:

```
string str = MainBackend.strDbQuery(attribute, Program.Rfid);
doubleBedrag = Convert.ToString(Convert.ToDouble(str) / 100);
int saldo = 0;
```

Veiligheid

Veiligheid is helaas een gebied waar deze software aardig wat te wensen overlaat. Er zijn twee grote beveiligingsrisico's ontdekt.

Het eerste en tevens grootste beveiligingsprobleem is het feit dat er directe communicatie plaatsvind tussen de cliënt en de server, zonder tussenkomst van een autorisatie console of ander controlemiddel. Dit houdt in dat wanneer er in geslaagd wordt om de broncode te reverse-engineeren er ongehinderd verbinding kan worden gemaakt met de server en er allerlei wijzigingen in de database kunnen worden aangebracht. Het is absoluut essentieel dat dit probleem wordt aangepakt. Dit kan worden gedaan door een interface op de server te maken die als liaison tussen de cliënt en de server dient. Communicatie met de server vindt dan indirect plaats via deze interface. De interface kan controleren of de verbinding wel of niet is toegestaan, en kan maar een gelimiteerd aantal handelingen uitvoer (alleen wat noodzakelijk is voor het functioneren van de pinautomaat). Op deze manier wordt dus voorkomen dat er vanaf de pinautomaat zelf met de database gerotzooid wordt.

Het tweede probleem bevind zich in de klasse Pincode. In deze klasse wordt gecontroleerd of een ingevoerde pincode correct is. Dit wordt gedaan met behulp van een boolean: approval genaamd. Deze boolean heeft initieel de waarde true. Wanneer een pincode incorrect is wordt de waarde false gemaakt. Afhankelijk van de waarde van deze variabele kunnen er vervolgens twee dingen gebeuren: de waarde is true, de pincode wordt goedgekeurd en het programma wordt voorgetzet, of de waarde is false waardoor de pincode wordt afgewezen en deze situatie verder afgehandeld wordt. Deze structuur is echter kwetsbaar voor een techniek die fault-injection heet. Bij deze techniek wordt aan de hand van het voltage wat door een systeem heen loopt gemeten wanneer een bepaalde variabele true of false gemaakt wordt. Vervolgens kan er met een exact getimed stroomstootje door het systeem voor gezorgd worden dat de regel waarin de variabele, in dit geval approval, gewijzigd wordt word overgeslagen. Dat wil zeggen dat in de hierboven beschreven software constructie de regel waarin approval false wordt gemaakt overgeslagen kan worden, met als gevolg dat approval true blijft ongeacht de ingevoerde pincode, en het programma verder gaat. De oplossing voor deze vulnerability is echter simpel: maak de waarde van approval initieel false, en maak deze pas true wanneer de pincode goedgekeurd is. Door deze simpele wijziging kan fault-injection niet meer worden toegepast hier.

Conclusies

Al met al kan men hier spreken van een degelijk stuk software. Echter zijn er enkele aandachtspunten, voornamelijk wat beveiliging betreft, die nog aangepakt moeten worden. Een ander aandachtspunt is efficiënt programmeren. Enkele bewerkingen die in de software uitgevoerd worden, zouden ook in minder regels code uitgevoerd kunnen worden terwijl zij dezelfde functionaliteit behouden. Dit zou rekenkracht besparen en tot efficiëntere software leiden. Qua structuur en netheid is het programma echter prima in orde.

Aanbevelingen

Op basis van de bevindingen en conclusies zijn de volgende aanbevelingen opgesteld:

- Sla tussenresultaten van berekeningen niet op in aparte variabelen, maar tracht de berekening in slechts één statement uit te voeren. Dit maakt de code overzichtelijker en beter leesbaar.
- Initieer de waarde van booleaanse variabelen, tenzij het echt niet anders kan, altijd op false. Hiermee wordt voorkomen dat de software vatbaar wordt voor fault-injection.
- Gebruik bij het communiceren met remote servers altijd een interface zodat de communicatie indirect verloopt. Hiermee wordt de server beschermt in het geval dat de cliënt-side software gekraakt wordt.
- Lees code altijd na nadat zij is geschreven, vaak kunnen namen beter en structuren simpeler en efficiënter. Code wordt hierdoor beter leesbaar en overzichtelijker, en het scheelt de processor berekeningen.