Reguliere expressies configureren en valideren in Cisco ESA en CES

Inhoud

Inleiding

In dit document wordt beschreven hoe ESA en CES reguliere expressies gebruiken in filters, belangrijke gedragsverschillen en de noodzaak om te testen voordat ze worden gehandhaafd.

Achtergrondinformatie

In dit document wordt beschreven hoe Cisco Email Security Appliance (ESA) en Cisco Cloud Email Security (CES) omgaan met reguliere expressies wanneer deze worden gebruikt in berichtenfilters en inhoudsfilters. Het is specifiek gericht op het begrijpen van hoe reguliere expressies zich gedragen in deze componenten en hoe ze omgaan met e-mailheaders, body-inhoud en bijlagen.

Het is belangrijk om vanaf het begin duidelijk te maken dat de reguliere expressiemotor die in de DLP-module wordt gebruikt, zich anders gedraagt. Daarom is alles wat in dit document wordt beschreven uitsluitend van toepassing op berichtenfilters en inhoudsfilters en niet op DLP-beleid.

Bij het werken met reguliere expressies in ESA moeten beheerders begrijpen dat e-mailinhoud niet wordt geëvalueerd op dezelfde manier als het visueel wordt weergegeven in een e-mailclient. E-mailberichten bevatten envelopinformatie, gestructureerde headers, MIME-onderdelen en mogelijk gecodeerde inhoud. Als gevolg hiervan kunnen vergelijkingen met filters onverwachte resultaten opleveren als de berichtstructuur en het regex-gedrag niet volledig worden begrepen.

Om deze reden kan elk nieuw filter dat reguliere expressies gebruikt altijd worden ingeschakeld in de monitormodus voordat de afdwinging wordt uitgevoerd. Dit maakt validatie tegen echt verkeer mogelijk en voorkomt onbedoelde blokkering of impact op de prestaties.

Woordenboeken en zoektermen

Bij het maken van een berichtenfilter of een inhoudsfilter wordt de term die in veel omstandigheden is ingevoerd, geïnterpreteerd als een reguliere expressie. Dit is een cruciaal concept: zelfs wanneer de beheerder van plan is letterlijke tekst te matchen, kan ESA de invoer verwerken met behulp van regex-logica.

Dit geldt niet uniform voor alle soorten voorwaarden. Wanneer u bijvoorbeeld in bepaalde gestructureerde omstandigheden naar een specifiek IP-adres zoekt, wordt de waarde niet geïnterpreteerd als een reguliere expressie. Wanneer u echter zoekt in de onderwerpkop, de berichttekst, een specifiek kopveld of een bestandsnaam voor bijlagen, wordt de waarde meestal behandeld als een regex-patroon.

Een gemeenschappelijk voorbeeld illustreert dit duidelijk. Stel dat het doel is om e-mails te blokkeren met het onderwerp:

```
Receipt number (123456)
```

Aangezien haakjes speciale tekens zijn in reguliere expressies (gebruikt voor groeperen), moeten ze worden ontlopen.

De juiste uitdrukking zou zijn:

```
Receipt number \(123456\)
```

Als de haakjes niet zijn ontsnapt, interpreteert de regex-engine ze als groeperende operatoren in plaats van letterlijke tekens. Afhankelijk van het patroon kan dit leiden tot onbedoelde matches of ander gedrag dan verwacht.

Daarom is het essentieel om te begrijpen welke tekens een speciale betekenis hebben in regex en ervoor te zorgen dat ze op de juiste manier ontsnappen wanneer letterlijke matching vereist is.

Voorbeelden van speciale personages en hun ontsnapte syntaxis

De eerste kolom toont een voorbeeldtekst met speciale tekens en de tweede kolom laat zien hoe de juiste syntaxis voor reguliere expressie moet worden geschreven om overeen te komen met die letterlijke tekst in Cisco ESA (Python-stijl regex).

Letterlijke tekst om aan te passen	Syntaxis voor reguliere expressies corrigeren
Ontvangstnummer (123456)	Ontvangstnummer \(123456\)
user@example.com	user@example\.com
www.test.ab	www\.test\.abc
file_name.txt	file_name\.txt
Prijs is 10,50	De prijs is 10,50
C:\Users\Admin	C:\\Users\\Admin
[VERTROUWELIJK]	\[VERTROUWELIJK\]
{factuur}	\{factuur\}
+34 600 123 456	\+34 600 123 456
vraag?	Vraag\?
100% gegarandeerd	100% gegarandeerd (% hoeft niet te ontsnappen)
* asterisk-symbool	Symbool voor sterretje *
A|B	A|B
caret ^start	caret \^start
Dollar $ 100	Dollar \$ 100

Beperking van het gebruik van reguliere expressies

Regelmatige expressies moeten zorgvuldig worden gebruikt en alleen wanneer dat nodig is.

Hoewel ze krachtige matching-mogelijkheden bieden, kunnen overmatige of slecht ontworpen expressies de verwerkingstijd van berichten verhogen en onbedoelde matches produceren.

Een bepaalde constructie die voorzichtigheid vereist is .* , die "elk teken, nul of meer keren" vertegenwoordigt. Wanneer het aan het begin of einde van een expressie wordt geplaatst, kan dit leiden tot overmatige backtracking en onnodige verwerkingsoverhead.

Cisco-documentatie geeft aan dat vermeldingen met .* aan het begin of einde ertoe kunnen leiden dat het systeem onder bepaalde omstandigheden vergrendeld raakt wanneer specifieke MIME-onderdelen worden gebruikt. Om deze reden raadt Cisco aan om het gebruik van leading of trailing te vermijden.* waar mogelijk.

In veel scenario's gebruiken beheerders patronen zoals . * factuur. * wanneer ze eenvoudig een factuur konden schrijven en hetzelfde praktische resultaat in ESA konden produceren. Aangezien de scanengine al zoekt in de relevante inhoudsgebieden, is het omringen van een woord met .* vaak overbodig en computertechnisch inefficiënt.

⚠ Let op: De algemene aanbeveling is om reguliere expressies zo eenvoudig en nauwkeurig mogelijk te houden.

Berichtenfilters, inhoudsfilters en woordenboeken

Cisco ESA biedt meerdere mechanismen om berichten te evalueren en acties toe te passen. Berichtenfilters werken aan het begin van de pijplijn en gebruiken een syntaxis in scriptstijl. Ze zijn uiterst flexibel en maken geavanceerde logica mogelijk met betrekking tot envelope-gegevens, headers en bijlagen. Omdat ze echter vroeg in de verwerkingsketen worden uitgevoerd, kunnen inefficiënte berichtenfilters de prestaties negatief beïnvloeden.

Inhoudsfilters worden geconfigureerd via de grafische interface en werken nadat het bericht is geaccepteerd. Voor de meeste use cases voor inhoudsinspectie zijn inhoudfilters gemakkelijker te beheren en veiliger vanuit een prestatieoogpunt.

Binnen zowel berichtenfilters als inhoudsfilters kunnen reguliere expressies direct in een bepaalde conditie of indirect worden geïntroduceerd door het gebruik van woordenboeken.

Met woordenboeken kunnen beheerders herbruikbare zoektermen centraliseren. Elk item is geschreven op een afzonderlijke regel en kan platte tekst of een reguliere expressie zijn. Woordenboeken ondersteunen ook niet-ASCII-tekens, waardoor ze geschikt zijn voor meertalige omgevingen.

In sommige situaties kunnen bepaalde complexe reguliere expressieconstructen zich niet identiek gedragen binnen woordenboeken. Wanneer dit gebeurt, moet de reguliere expressie direct in de filterconditie worden geplaatst in plaats van in het woordenboek.

Cisco ESA maakt het mogelijk om tot 150 contentwoordenboeken te maken. Standaard kunnen

100 woordenboeken worden geconfigureerd, tenzij de limiet via de CLI wordt gewijzigd met behulp van de opdracht dictionary config.

Woordenboeken kunnen ook term weging implementeren. Elke term kan een numeriek gewicht krijgen, en wanneer ESA een bericht scant, vermenigvuldigt het het aantal keren dat die term voorkomt met zijn gewicht. De resulterende score wordt vergeleken met een drempelwaarde die in het filter is gedefinieerd. Dit scoremodel maakt een flexibelere en meer gegradueerde handhaving van het beleid mogelijk.

Bovendien kunnen woordenboeken Smart Identifiers bevatten, wat algoritmische detectoren zijn voor gestructureerde numerieke patronen zoals socialezekerheidsnummers of bancaire identifiers.

engine voor reguliere expressies

Cisco ESA gebruikt reguliere expressies op basis van de stijl van de Python-module. Hoewel dit compatibiliteit biedt met algemene Python-regex-syntaxis, wordt niet elke geavanceerde functie die wordt ondersteund in volledige Python-omgevingen noodzakelijkerwijs ondersteund in ESA.

Voor exacte string matching moeten expressies verankerd worden met ^ aan het begin en $ aan het einde. Zonder deze ankers kan de regex-engine subtekenreeksen matchen in plaats van volledige waarden.

Bijvoorbeeld de uitdrukking:

```
sun.com
```

Overeenkomende tekenreeksen zoals:

```
thegodsunocommando
```

De uitdrukking:

```
^sun\.com$
```

Komt alleen overeen met de exacte tekenreeks sun.com.

Bij het matchen van een lege string is het belangrijk om geen "" te gebruiken, omdat dit effectief overeenkomt met alle strings. In plaats daarvan is de juiste uitdrukking:

```
^$
```

Omdat Cisco ESA reguliere expressies in Python-stijl gebruikt, zijn er een paar manieren om een hoofdletterongevoelige vergelijking uit te voeren.

Standaard zijn reguliere expressies hoofdlettergevoelig. Dat betekent zoeken naar:

```
foo
```

Alleen match foo, maar niet foo, foo of foo.

Als u een hoofdletterongevoelige overeenkomst wilt uitvoeren, kunt u de inline-markering (?i) gebruiken aan het begin van de reguliere expressie. Dit vertelt de regex-engine om de case voor de rest van het patroon te negeren.

Voorbeeld:

```
(?i)foo
```

Deze expressiematch:

- miskleun
- FOO
- Foo
- OfO

Als je de hele string exact wilt matchen, zonder hoofdletter, kun je de hoofdletter-ongevoelige vlag combineren met ankers:

```
(?i)^foo$
```

Dit zorgt ervoor dat de volledige waarde precies "foo" is, ongeacht de kapitalisatie.

Een ander (minder praktisch) alternatief zou zijn om alle mogelijke combinaties expliciet te definiëren met behulp van tekenklassen, bijvoorbeeld:

```
[Ff][Oo][Oo]
```

Deze aanpak wordt echter moeilijk te handhaven en wordt niet aanbevolen wanneer de (?i) vlag in plaats daarvan kan worden gebruikt.

In de meeste ESA-scenario's is de meest geprefereerde en schoonste methode voor case-insensitive matching het gebruik van:

```
(?i)
```

aan het begin van de reguliere expressie.

Niet-ASCII-tekens en woordgrenzen

In talen die double-byte tekensets gebruiken, kunnen de concepten van woordgrenzen of hoofdletters zich niet gedragen zoals verwacht. Complexe expressies die afhankelijk zijn van constructies zoals \w kunnen inconsistente resultaten opleveren wanneer codering of locale onbekend is.

In dergelijke gevallen kan het raadzaam zijn om woordenlijsthandhaving in woordenboekconfiguratie uit te schakelen of de uitdrukking te vereenvoudigen om afhankelijkheid van dubbelzinnige tekenklassen te voorkomen.

Wanneer u werkt met niet-ASCII-woordenboeken, kan CLI tekens niet correct weergeven, afhankelijk van de terminalcodering. In die gevallen wordt aanbevolen het woordenboek naar een tekstbestand te exporteren, het extern te bewerken en het opnieuw te importeren.

Efficiënte filters schrijven

Efficiëntie is van cruciaal belang bij het schrijven van filters, vooral in omgevingen met een hoog volume. Een veel voorkomende fout is het schrijven van lange ketens van OR-voorwaarden voor soortgelijke wedstrijden.

Het controleren van tientallen bevestigingsextensies dwingt de regex-motor bijvoorbeeld om herhaaldelijk te initialiseren. Dit verhoogt het CPU-gebruik en vermindert de onderhoudbaarheid.

In plaats van veel afzonderlijke vergelijkingen te schrijven, vermindert het groeperen ervan met behulp van afwisseling binnen een enkele reguliere expressie de verwerkingsoverhead aanzienlijk. Dit vermindert het aantal keren dat de regex-motor wordt aangeroepen en maakt het filter gemakkelijker te onderhouden.

Efficiënt filterontwerp gaat niet alleen over leesbaarheid - het heeft directe invloed op de systeemprestaties.

PDF's en reguliere expressies

Als u inhoud in PDF-bestanden vergelijkt, kunt u onverwachte resultaten krijgen, afhankelijk van de manier waarop de PDF is gegenereerd. Sommige PDF's bevatten geen logische spaties of regeleinden in hun interne representatie. De scanengine probeert logische spatiëring te reconstrueren op basis van woordpositionering.

Als een woord is opgebouwd met behulp van meerdere lettertypen of lettergroottes, kan de interne representatie de tekst fragmenteren. Het woord "callout" kan bijvoorbeeld intern worden geïnterpreteerd als "call out" of "call lout".

In dergelijke gevallen kan een poging om de uitdrukking "callout" te matchen mislukken omdat de interne representatie niet die exacte aaneengesloten string bevat. Beheerders moeten zich bewust zijn van deze beperking bij het ontwerpen van op inhoud gebaseerd beleid dat is gericht op PDF-bijlagen.

Reguliere expressies testen

Het testen van reguliere expressies voordat deze in productie worden genomen, is een essentiële operationele vereiste. Een reguliere expressie die syntactisch correct lijkt, kan zich heel anders gedragen wanneer deze wordt geëvalueerd aan de hand van echt e-mailverkeer. Zonder de juiste tests kan een filter valse positieven genereren, beoogde patronen niet detecteren, prestatieoverhead introduceren of de legitieme e-mailstroom onbedoeld verstoren.

Het testen moet worden benaderd als een gestructureerd proces in twee fasen om het risico te minimaliseren voordat een filter in productie wordt genomen.

Fase 1 – Regelmatig expressieontwerp en validatie
De eerste fase richt zich op het ontwerpen en valideren van de reguliere expressie zelf voordat deze wordt geïntegreerd in Cisco ESA.

1. Gebruik van regex101 of soortgelijke instrumenten

Online platforms zoals http://regex101.com (of gelijkwaardige tools) zijn zeer nuttig tijdens de ontwerpfase. Bij het gebruik van deze tools moet de Python-smaak worden geselecteerd om de regex-engine van ESA te benaderen.

Met deze platforms kunnen beheerders:

- Juistheid van syntaxis valideren
- Bevestig dat speciale tekens correct zijn ontsnapt
- Test zowel matchende als niet-matchende cases
- Groepering en kwantificeergedrag visualiseren
- Identificeer mogelijk hebzuchtige constructies zoals .*

Deze tools simuleren echter standaard Python-regex-gedrag en kunnen functies ondersteunen die

niet volledig zijn geïmplementeerd in Cisco ESA. Daarom moeten zij worden beschouwd als voorlopige validatie-instrumenten in plaats van als definitieve compatibiliteitstests.

2. Gebruik van AI-modellen (ChatGPT, Copilot, ...)

AI-gebaseerde assistenten kunnen de creatie van regex versnellen, vooral voor complexe matching-scenario's. Door het gewenste gedrag in natuurlijke taal te beschrijven, kunnen beheerders een eerste regex-voorstel krijgen dat vervolgens kan worden verfijnd.

AI-tools zijn vooral nuttig voor:

- Complexe gegroepeerde expressies genereren
- Bedrijfsvereisten omzetten in regex-syntaxis
- Vereenvoudiging van lange OR-gebaseerde voorwaarden in gegroepeerde alternatieven

AI-gegenereerde expressies moeten echter altijd kritisch worden beoordeeld. Ze kunnen inefficiënties, niet-ondersteunde constructies of overdreven complexe logica introduceren. AI-hulp moet worden behandeld als redactionele hulp, niet als definitieve validatie. Elke AI-gegenereerde expressie moet nog steeds worden getest met behulp van gestructureerde validatiemethoden.

Fase 2 – Filtergedragsvalidatie in Cisco ESA

Nadat de expressie zelf is gevalideerd, richt de tweede fase zich op het bevestigen van hoe deze zich gedraagt binnen Cisco ESA wanneer deze wordt toegepast op de verwerking van echte berichten.

1. De functie Trace gebruiken in de CES-console

Met de functie Trace in de Cisco Email Security (CES)-console kunnen beheerders simuleren en analyseren hoe een specifiek bericht wordt verwerkt. Dit is een van de meest betrouwbare methoden voor het valideren van filtergedrag vóór handhaving.

Trace biedt inzicht in:

- Hoe wordt de boodschap ontleed
- Welke filters worden beoordeeld
- Of de voorwaarde wordt geactiveerd
- De volgorde van regel uitvoering

Omdat ESA MIME-parsing, header-normalisatie en inhoudsdecodering uitvoert, kan het gedrag in het toestel verschillen van externe regex-testtools. Voor gedetailleerde instructies moeten beheerders de officiële Cisco-documentatie raadplegen:

https://www.cisco.com/c/en/us/td/docs/security/ces/ces_16-0-3/user_guide/b_ESA_Admin_Guide_ces_16-0-

3/b_ESA_Admin_Guide_12_1_chapter_0101001.html

Het gebruik van Trace zorgt ervoor dat het filter zich gedraagt zoals verwacht in de echte verwerkingsmotor.

2. Het filter maken met een logboekactie

Een andere veilige en aanbevolen aanpak is om het filter in te zetten met een niet-verstorende actie, zoals logboekregistratie, in plaats van een agressieve actie uit te voeren, zoals het laten vallen, stuiteren of in quarantaine plaatsen van berichten.

Door het filter zo te configureren dat een item wordt vastgelegd wanneer deze overeenkomt, kunnen beheerders:

- Bekijk de wedstrijdfrequentie
- Detecteer onverwachte triggers
- Impact op prestaties valideren
- Reëel verkeersgedrag analyseren

Deze aanpak plaatst het filter effectief in een gecontroleerde bewakingsfase binnen het productieverkeer. Zodra voldoende validatie is voltooid en is bevestigd dat het gedrag correct is, kan de actie veilig worden gewijzigd in handhavingsmodus.

De expressie in een inhoudsfilter en in een woordenboek introduceren

Zodra de reguliere expressie goed is ontworpen en gevalideerd, is de volgende stap begrijpen hoe deze moet worden ingevoerd binnen Cisco ESA. De syntaxis kan enigszins verschillen, afhankelijk van het feit of de expressie direct is geconfigureerd in een inhoudsfilter of in een woordenboek. Dit verschil zorgt vaak voor verwarring.

De expressie in een inhoudsfilter invoeren

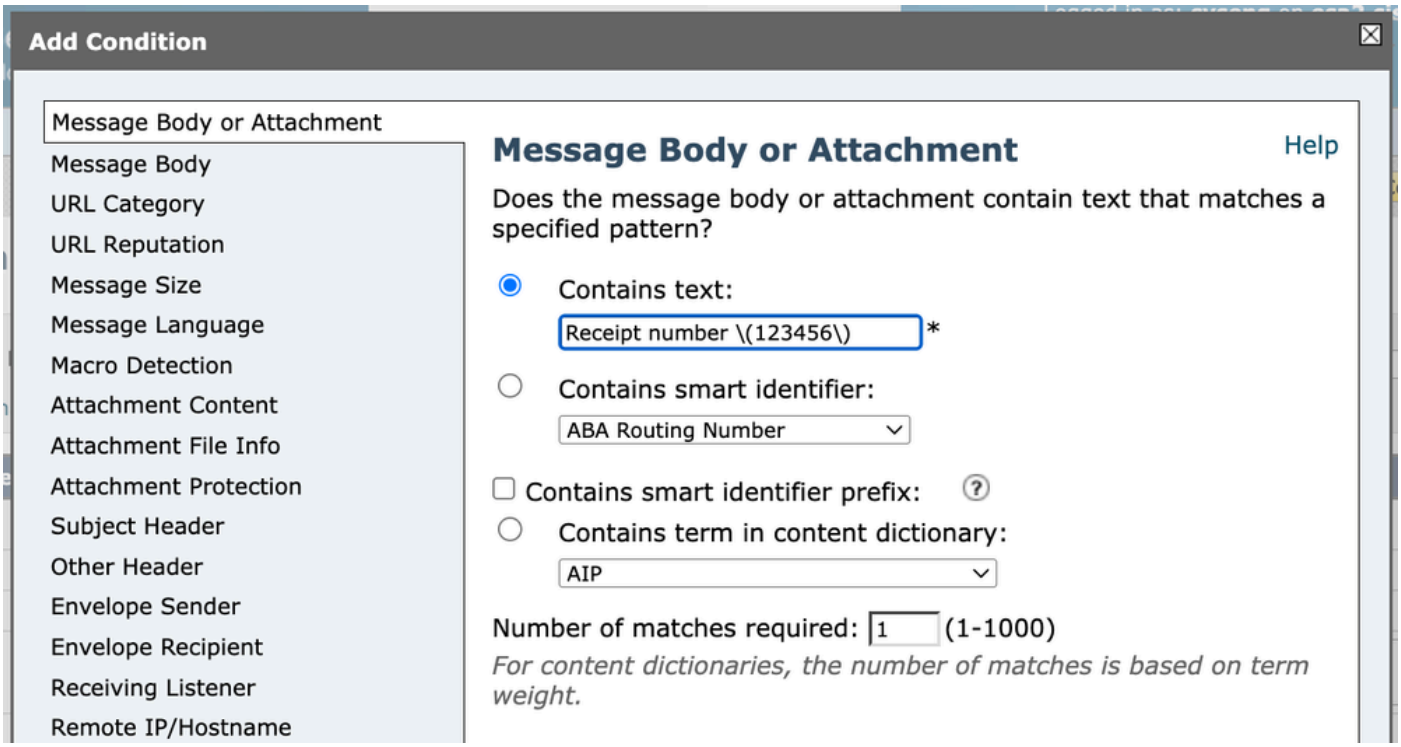
Bij het configureren van een Content Filter voorwaarde (bijvoorbeeld overeenkomend met de Subject header), moet de reguliere expressie worden ingevoerd in het condition veld. Als we de letterlijke tekst willen vergelijken:

```
Receipt number (123456)
```

We moeten aan de haakjes ontsnappen omdat het speciale karakters zijn in reguliere expressies.

Daarom moet de regex zelf worden geschreven als:

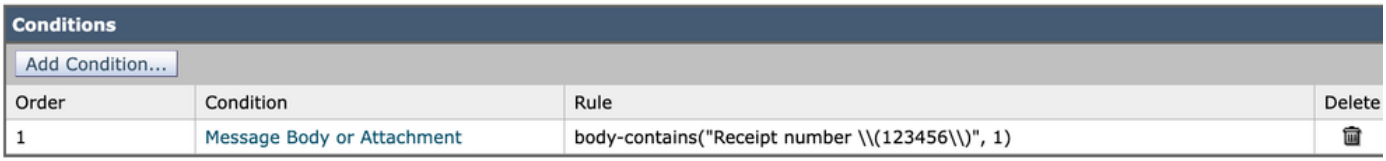
```
Receipt number \(123456\)
```



Inhoudsfilter 1

Wanneer u echter de volledige filterconditie in de GUI of de uitvoer voor geavanceerde configuratie bekijkt, kan deze worden weergegeven als:

```
subject == "Receipt number \\(123456\\)"
```



Inhoudsfilter 2

Dit kan op het eerste gezicht verwarrend zijn. De reden voor de dubbele backslashes (\\) is dat de backslash zelf ook een speciaal teken is binnen geciteerde tekenreeksen. In deze context wordt één backslash gebruikt om te ontsnappen aan de haakjes voor de regex-engine, en de tweede backslash wordt gebruikt om te ontsnappen aan de backslash in de aangehaalde string.

In de praktijk:

\(123456\) is de eigenlijke reguliere expressie.

\\(is hoe het systeem \(vertegenwoordigt in een opgegeven configuratie string.

Hoewel het anders lijkt wanneer het wordt weergegeven, blijft de logische reguliere expressie die wordt geëvalueerd:

Ontvangstnummer \(123456\)

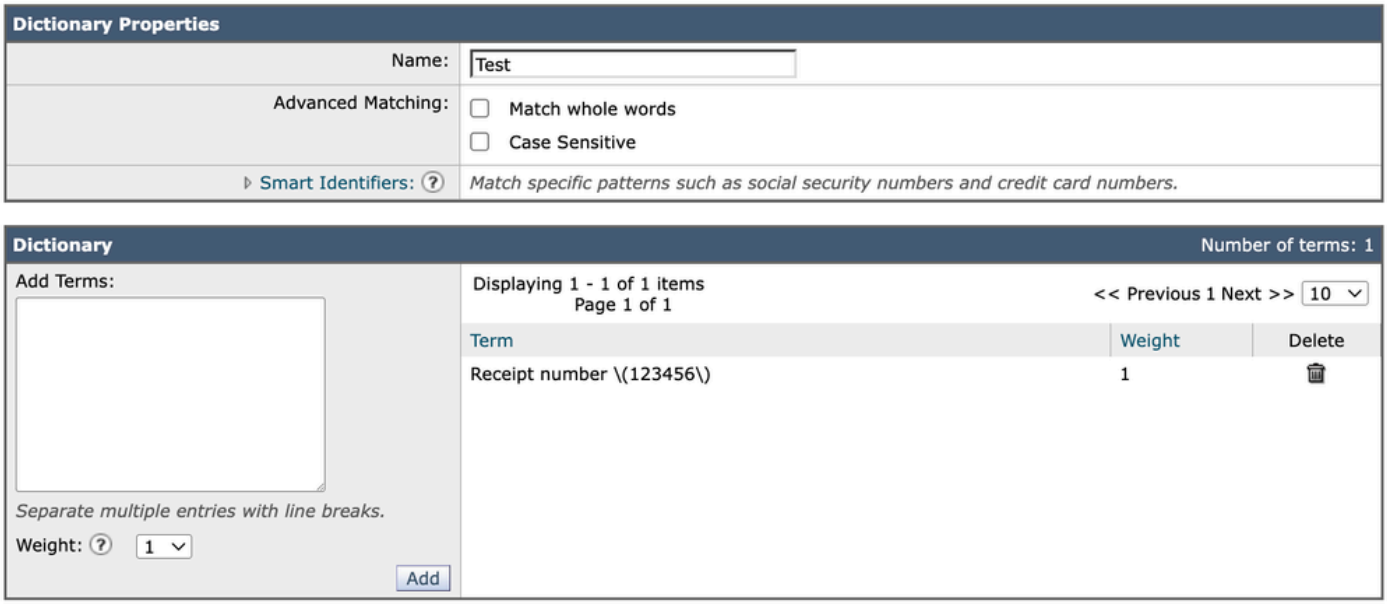
Dit is gewoon een kwestie van string ontsnappen in configuratie-uitvoer.

De uitdrukking in een woordenboek

Wanneer u dezelfde expressie aan een woordenboek toevoegt, wordt de vermelding direct ingevoerd als:

```
Receipt number \(123456\)
```

In dit geval wordt het nog steeds weergegeven precies zoals geschreven. In tegenstelling tot de GUI Inhoudsfilter hebben woordenboeken geen extra ontsnappingslagen nodig in hun visuele configuratieformaat.



woordenboek

Elk woordenboekitem wordt geëvalueerd als platte tekst of als een reguliere expressie, afhankelijk van de structuur. Als er speciale tekens zijn opgenomen (zoals haakjes in dit geval), moet de expressie al goed zijn ontsnapt wanneer deze wordt ingevoerd.

Informatie over "Match Whole Words"

Bij het configureren van een woordenboek is er een optie met de naam "Match Whole Words." In veel gevallen is het raadzaam om niet te vertrouwen op deze instelling bij het werken met reguliere expressies.

De reden hiervoor is dat woordgrenzend gedrag nauwkeuriger kan worden gecontroleerd met behulp van regex-ankers.

Voorbeeld:

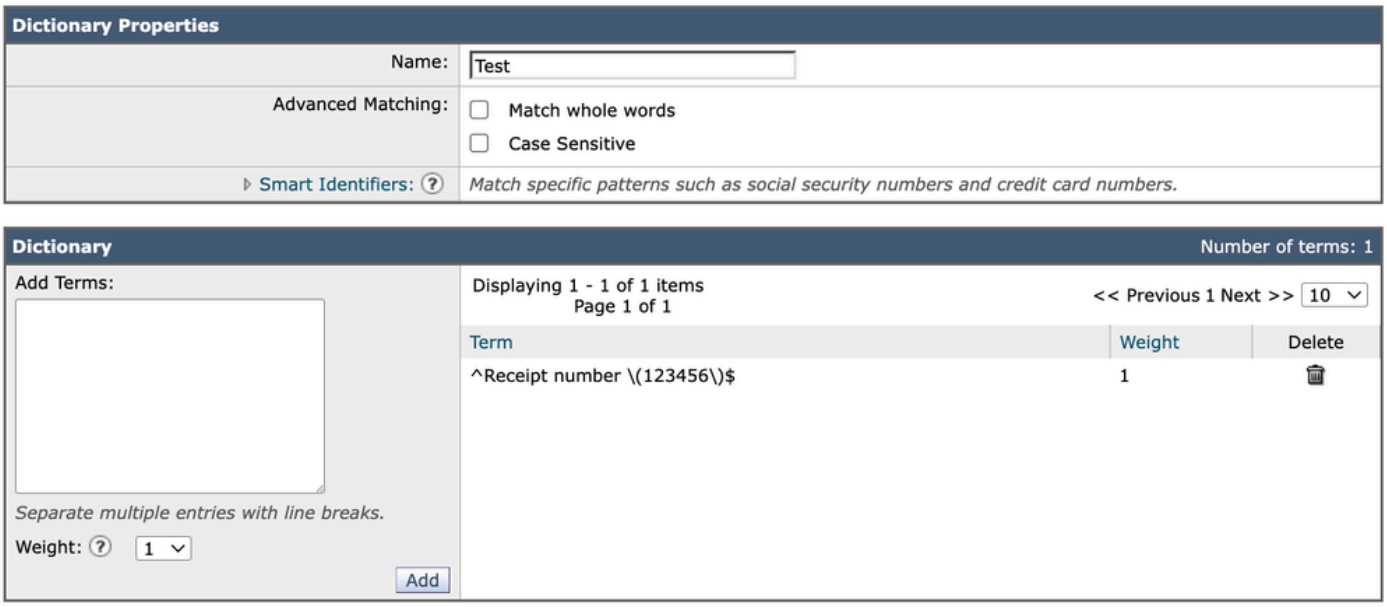
^ zorgt ervoor dat de wedstrijd begint bij het begin.

$ zorgt ervoor dat de wedstrijd aan het einde eindigt.

Het gebruik van ankers zoals:

```
^Receipt number \(123456\)$
```

Biedt expliciete en voorspelbare controle over exact matchinggedrag. Deze benadering vermijdt mogelijke dubbelzinnigheid met betrekking tot de interpretatie van woordgrenzen, vooral in meertalige of niet-ASCII-omgevingen.



Woordenboek 2

Om deze reden is het over het algemeen beter om match precision direct binnen de reguliere expressie te beheren in plaats van te vertrouwen op de optie "Match Whole Words".

Het begrijpen van deze subtiele verschillen tussen inhoudsfilters en woordenboeken zorgt ervoor dat expressies zich consistent gedragen en vermindert het risico op configuratiefouten tijdens de implementatie.

Regex-kostenrangschikking in Cisco ESA

Bij het werken met reguliere expressies in Cisco ESA hangt de impact op de prestaties grotendeels af van hoeveel tekst de engine moet scannen en hoeveel backtracking deze moet uitvoeren. Aangezien ESA hele berichtlichamen, MIME-onderdelen en zelfs gedecodeerde bijlagen moet evalueren, kunnen inefficiënte patronen het CPU-gebruik aanzienlijk verhogen.

Het is een praktische rangschikking van de hoogste computerkosten tot de laagste.

Duurste — patronen met een hoog risico

Deze expressies kunnen een dramatische invloed hebben op de prestaties, vooral op grote berichten.

Geneste kwantificeerders (slechtste geval)

Voorbeelden:

```
(.*)+
(.+)+
(\S+)+
```

Deze zijn extreem gevaarlijk omdat ze exponentiële backtracking-scenario's creëren.

Een kwantor in een andere kwantor dwingt de regex-engine om veel combinaties te proberen voordat deze faalt.

In het echte verkeer kan dit ernstige CPU-pieken veroorzaken.

Aanbeveling: Vermijd onbegrensde en dubbelzinnige geneste kwantificeerders.

Greedy .* Gevolgd door een verplicht patroon

Voorbeeld:

```
.*text
.*\/\?text
```

Dit patroon neemt eerst het hele bericht in beslag en maakt vervolgens een back-up van teken voor teken totdat het de vereiste subtekenreeks vindt.

Als het patroon niet aanwezig is - of in de buurt van het einde verschijnt - trekt de motor het vereiste token terug en test het op veel posities, wat de CPU-kosten verhoogt.

In ESA, waar lichamen groot kunnen zijn en MIME-inhoud kunnen bevatten, wordt dit heel snel duur.

Aanbeveling: Niet voorbereiden.* om subtekenreeksen te detecteren. ESA doorzoekt al de geëvalueerde inhoud en toonaangevende jokertekens verhogen alleen het backtracking- en CPU-gebruik.

```
text$
\/\?text$
```

Grote alternatieven met gedeelde voorvoegsels

Voorbeeld:

```
(a.*b|a.*c|a.*d)
```

Wanneer meerdere alternatieven een structuur delen, evalueert de engine elke tak sequentieel.

Als vroege takken bijna overeenkomen, maar laat falen, probeert de motor het uitgebreid opnieuw.

Hierdoor neemt de evaluatietijd aanzienlijk toe.

Middelhoge kosten — voorzichtig gebruiken

Deze patronen zijn niet catastrofaal, maar kunnen nog steeds inefficiënt zijn.

Breed .* Gebruik

Voorbeeld:

```
https://.*\?text
```

Hoewel niet exponentieel, .* staat nog steeds onbeperkte matching toe. Als de verwachte subtekenreeks niet snel wordt weergegeven, scant de engine grote delen van het bericht.

In ESA komt dit vaak voor bij het scannen van e-mailinstanties op phishing-URL's.

Luie kwantificeerders (+?, *?)

Voorbeeld:

```
\S+?
.*?
```

Luie kwantificeerders veranderen de matching strategie (kortste-eerste). Ze kunnen overmatching in sommige patronen verminderen, maar in grote 'zoek'-werklasten kunnen ze pogingen verhogen wanneer de afsluitende token te laat is of ontbreekt.

In veel ESA-gebruiksgevallen bieden ze geen echt voordeel en kunnen ze onnodige interne herhalingen introduceren.

Zeer generieke tekenklassen

Voorbeelden:

```
\S+
.+
```

Deze maken een breed wedstrijdbereik mogelijk, waardoor het aantal potentiële backtrackingpaden toeneemt.

Meer specifieke karakterklassen hebben altijd de voorkeur.

Lage kosten — veilige en efficiënte patronen

Deze worden aanbevolen voor ESA-productieomgevingen.

vaste lettertekens

Voorbeelden:

```
text
iw\.adc
```

Letterlijke strings zijn de meest efficiënte matches. De motor voert eenvoudige vergelijkingen uit met minimale overhead.

Gebruik ankers om het zoekbereik te beperken

Wanneer de wedstrijd op een specifieke positie wordt verwacht, overweeg dan om het patroon te verankeren met behulp van ^ of $. Ankers beperken de evaluatie tot vaste posities en voorkomen dat de engine de volledige inhoud onnodig scant. Dit kan backtracking verminderen en de

prestaties verbeteren, met name in grote berichtinstanties of gestructureerde headers.

```
^Invoice$
```

Specifieke tekenklassen

```
[A-Za-z0-9.-]+
[^/\s]+
```

Deze beperken wat kan overeenkomen, drastisch verminderen van de zoekruimte en het beperken van backtracking.

Gestructureerde en beperkte patronen

Voorbeeld:

```
https?:\/\/[A-Za-z0-9.-]+(?:\/[^?\s]*)*\/\?text
```

- Het domein is gefixeerd.
- Geen gebruik van.*.
- bevat geen catastrofale geneste patronen (bijvoorbeeld (.*)+)
- Geen onnodige luie operatoren.
- Elke sectie is beperkt.

Dit vermindert de CPU-impact aanzienlijk in vergelijking met brede wildcard-matching.

Praktische richtlijnen voor Cisco ESA

Bij het ontwerpen van regex voor berichten- of inhoudsfilters:

1. Hoe specifieker het patroon, hoe beter de prestaties.
2. Vermijd .* tenzij het echt nodig is - en vermijd vooral het plaatsen van vereiste tokens erna.
3. Gebruik nooit geneste kwantificeerders.
4. Geef de voorkeur aan expliciete tekenklassen boven jokertekens.
5. Test altijd nieuwe expressies in de monitormodus voordat u deze afdwingt.

Regex-prestatievergelijking (Cisco ESA Context)

Patroon	aanbevolen	backtrackingrisico	ESA-impact	Aanbevolen alternatief
HTTPS?:\/\/.*/\? Tekst.*	Nee	Hoog	hoger	^https?:\/\/[A-Za-z0-9.-]+(?:\/[^?\s]*)*\/\?text
HTTPS?:\/\/.*\? Tekst	⚠ Met voorzichtigheid	middelhoog	middelhoog	^https?:\/\/[^?\s]+\?text$
HTTPS?:\/\/.*	Nee	middelhoog	Gemiddeld	^https?:\/\/[A-Za-z0-9.-]+(?:\/[^\s]*)
.*wachtwoord	Nee	Hoog	hoger	Wachtwoord$
.*tekst.*	Nee	Hoog	hoger	tekst
.*(factuur|betaling|overschrijving)	Nee	Hoog	hoger	(factuur|betaling|overschr
(.+)+	Nooit	Zeer hoog (exponentieel)	Ernstig	Herstructurering zonder g kwantificeerders (voorbee
.*@.*	Nee	Hoog	hoger	[A-Za-z0-9._%+-]+@[A-Z 9.-]+\.[A-Za-z]{2,}
\S+?	Niet ideaal	Gemiddeld	Gemiddeld	\S+ of meer specifieke kla zoals [A-Za-z0-9.-]+
.*\/admin	Nee	Hoog	hoger	\/admin$
.*(inloggen|verifiëren).*	Nee	Hoog	hoger	(aanmelden|verifiëren)
^.*tekst	Nee	Hoog	hoger	tekst$ (of ^tekst als positi belang is)

Conclusie

Regelmatige expressies zijn een krachtig en flexibel hulpmiddel binnen Cisco ESA, waardoor nauwkeurige inhoudsinspectie en geavanceerde beleidshandhaving in zowel berichtenfilters als inhoudfilters mogelijk zijn. Maar met die flexibiliteit komt verantwoordelijkheid. Slecht ontworpen of onvoldoende geteste expressies kunnen leiden tot fout-positieve resultaten, gemiste detecties, prestatievermindering of onbedoelde verstoring van legitiem e-mailverkeer.

Daarom moet het gebruik van reguliere uitdrukkingen in het ESR altijd gestructureerd en gedisciplineerd zijn. De creatiefase moet ervoor zorgen dat de expressie syntactisch correct is, goed ontsnapt, efficiënt en logisch afgestemd op het beoogde doel. Externe tools en AI-ondersteunde generatie kunnen dit proces aanzienlijk versnellen, maar ze mogen nooit zorgvuldige validatie vervangen.

Even belangrijk is de valideringsfase binnen de ESA-omgeving zelf. Omdat ESA berichten verwerkt door middel van MIME-parsing, header-normalisatie en het decoderen van inhoud, kan het gedrag in de echte wereld afwijken van de theoretische verwachtingen. Met tools zoals Trace en het implementeren van filters in de aanvankelijke logboekregistratie- of monitoringmodus kunnen beheerders correct gedrag bevestigen zonder operationeel risico.

Kortom, reguliere expressies moeten zo eenvoudig mogelijk worden gehouden, grondig worden getest en voorzichtig worden ingezet. Een goed ontworpen en goed gevalideerd filter handhaaft niet alleen het beleid effectief, maar beschermt ook de stabiliteit van het systeem en zorgt voor voorspelbaar gedrag in productieomgevingen.

documentatie

Voor aanvullende technische details en officiële richtlijnen over hoe reguliere expressies worden geïmplementeerd en gebruikt binnen Cisco ESA, moeten beheerders de productdocumentatie van Cisco raadplegen

De sectie "Reguliere expressies in regels" biedt een overzicht van hoe reguliere expressies worden geëvalueerd in berichtenfilters en inhoudsfilters, inclusief syntaxisoverwegingen en gebruik binnen regelvoorwaarden.

https://www.cisco.com/c/en/us/td/docs/security/ces/ces_16-0-3/user_guide/b_ESA_Admin_Guide_ces_16-0-3/b_ESA_Admin_Guide_12_1_chapter_01000.html#con_1192755

De sectie "Richtlijnen voor het gebruik van reguliere expressies" biedt praktische aanbevelingen over de juiste syntaxis, het verankeren van expressies, het omgaan met speciale tekens en het vermijden van veelvoorkomende fouten die de prestaties of de nauwkeurigheid van de matching kunnen beïnvloeden.

https://www.cisco.com/c/en/us/td/docs/security/ces/ces_16-0-3/user_guide/b_ESA_Admin_Guide_ces_16-0-3/b_ESA_Admin_Guide_12_1_chapter_01000.html#con_1125696

Het beoordelen van deze officiële bronnen wordt sterk aanbevolen bij het ontwerpen of oplossen van problemen met filters die afhankelijk zijn van reguliere expressies, omdat ze gezaghebbende

richtlijnen bieden die zijn afgestemd op de specifieke AsyncOS-versie die wordt gebruikt.

Over deze vertaling

Cisco heeft dit document vertaald via een combinatie van machine- en menselijke technologie om onze gebruikers wereldwijd ondersteuningscontent te bieden in hun eigen taal. Houd er rekening mee dat zelfs de beste machinevertaling niet net zo nauwkeurig is als die van een professionele vertaler. Cisco Systems, Inc. is niet aansprakelijk voor de nauwkeurigheid van deze vertalingen en raadt aan altijd het oorspronkelijke Engelstalige document (link) te raadplegen.