Øvelse 32 – File Integrity bypass og defense in depth

Information

Formålet med denne øvelse er at demonstrere **begrænsningerne ved udelukkende at anvende checksum-baseret file integrity monitoring (FIM)**.

I den forrige øvelse arbejdede du med checksums til at opdage ændringer i filer.
I denne øvelse skal du undersøge, hvordan denne metode kan omgås, hvis en angriber har tilstrækkelige rettigheder på systemet.

Dette er centralt for forståelsen af, hvorfor man i praksis ofte kombinerer flere sikkerhedsmekanismer, såsom:

- File Integrity Monitoring (FIM)
- auditd (event-baseret logging)

Baggrund

Checksum-baseret overvågning fungerer ved at:

1. Oprette en baseline (hashværdi)
2. Sammenligne filer med denne baseline over tid

Men hvis en trusselsaktør:

- kan ændre filen
- og samtidig opdatere checksummen

 så vil ændringen ikke blive opdaget

Dette kaldes ofte en **integrity bypass**

Defense in depth og logning

I praksis løses dette problem ikke ved at finde én perfekt metode –
men ved at kombinere flere sikkerhedsmekanismer, som supplerer hinanden. Altså **defense**

in depth.

To forskellige detektionsmetoder

Der findes grundlæggende to måder at opdage filændringer på:

1. Checksum-baseret (FIM)

- Finder ændringer ved at sammenligne filer over tid
- Svarer på: *"Er filen ændret?"*
- Opdager ændringer **efter de er sket**

2. Event-baseret (auditd)

- Registrerer handlinger i det øjeblik de sker
- Svarer på: *"Hvem – og hvordan?"*
- Logger selve ændringshandlingen (Ikke ændringen)

Hvordan de supplerer hinanden

De to metoder har forskellige styrker og svagheder:

Metode	Styrke	Svaghed
FIM (checksum)	Opdager ændringer i filer	Kan omgås hvis baseline ændres
auditd	Viser hvem og hvordan ændringen skete	Kræver korrekt konfiguration

Sammen giver de et mere komplet billede:

- FIM opdager *at noget er ændret*
- auditd forklarer *hvordan og af hvem det skete*

Detection in depth

Begrebet **detection in depth** udspringer af **defense in depth**, men fokuserer specifikt på **detektion af hændelser**.

 Med andre ord:

- Defense in depth = hele sikkerhedsstrategien
- Detection in depth = detektionslaget i denne strategi

Eksempel i denne øvelse

I denne øvelse arbejder du med to detektionsmetoder:

- Checksum (FIM)
- auditd

Disse repræsenterer forskellige perspektiver på den samme hændelse:

Perspektiv	Spørgsmål	Teknologi
FIM	"Er noget ændret?"	Checksum
auditd	"Hvad skete der?"	Syscalls
Detection in depth	"Kan vi opdage det på flere måder?"	Kombination

---

Hvorfor er det vigtigt?

En angriber omgår én detektionsmetode:

- FIM kan bypasses → hvis checksummen opdateres
- auditd kan fejle → hvis det ikke er korrekt konfigureret

 Men sandsynligheden for at omgå *begge* samtidig er langt mindre

---

 Vigtig pointe:

Sikkerhed handler ikke om én perfekt løsning
– men om flere lag, der gør det svært at skjule aktivitet

 Dette er kernen i **defense in depth**

Relation mellem begreberne

Detection in depth er en *underkategori* af **defense in depth**.

- **Defense in depth** dækker hele sikkerhedsarkitekturen
- **Detection in depth** fokuserer specifikt på at opdage angreb

Eksempel

Et system kan bestå af flere lag:

1. Firewall (forebyggelse)
2. Adgangskontrol (forebyggelse)
3. Overvågning af file integritet (detektion)
4. Monitorering af system kald (detektion)

Her er:

- Hele opsætningen = **defense in depth**
- Kombinationen af file integritet monitorering + monitorering af system kald = **detection in depth**

Kort sagt:

Ét logspor kan fejle – flere uafhængige spor er svære at undgå

Instruktioner

Husk at notere dine observationer i dit cheat sheet.

I denne øvelse skal du simulere en angriber med skriveadgang til systemet, der forsøger at skjule sine ændringer.

Bemærk: Dette er et forsimplet eksempel, som illustrerer en grundlæggende svaghed ved en ren checksum-baseret tilgang.

I praksis vil angreb ofte være mere komplekse, men øvelsen viser det centrale problem:
Hvis en angriber har tilstrækkelige rettigheder, kan både data og kontrolmekanismer manipuleres.

---

1 Opret baseline (legitim tilstand)

1. Opret en testfil:
   `echo "Dette er original data" > testfil.txt`
2. Opret en checksum:
   `sha256sum testfil.txt > checksumfil.txt`
3. Verificér filen:
   `sha256sum -c checksumfil.txt`

> Output bør vise: OK

---

2 Simulér uautoriseret ændring

1. Ændr filen:
   `echo "Ondsindet ændring" >> testfil.txt`
2. Verificér igen:
   `sha256sum -c checksumfil.txt`

> Output bør nu vise: FAILED

---

3 Simulér angriberens bypass

1. Opdatér checksummen (som en angriber kunne gøre):
   `sha256sum testfil.txt > checksumfil.txt`
2. Verificér igen:
   `sha256sum -c checksumfil.txt`

> Output viser nu: OK

---

4 (Valgfrit) Overvej auditd

Hvis auditd havde været konfigureret til at overvåge filen:

- Ville ændringen være blevet opdaget?
- Hvad ville auditd kunne vise, som checksums ikke kan?

Analyse

Undersøg:

- Hvorfor blev ændringen ikke længere opdaget?
- Hvad er det egentlige problem ved checksum-baseret overvågning?
- Hvilke forudsætninger kræver dette angreb?

Vigtig pointe

Checksums er kun pålidelige, hvis checksum-data er beskyttet

Hvis en angriber kan ændre både:

- filen
- og checksummen

så mister metoden sin effekt

Hvordan løses dette i praksis?

I virkelige systemer anvendes ofte:

- Beskyttelse af checksum-filer (fx read-only eller ekstern lagring)
- Kombination med auditd (som registrerer selve handlingen)
- Digitale signaturer i stedet for simple checksums

Refleksion

- Hvorfor er det ikke nok kun at bruge checksums?
- Hvordan kan man beskytte baseline-data?
- Hvordan kan auditd hjælpe med at opdage dette angreb?
- Hvordan ville Wazuh reagere i dette scenarie?

Læringsmål

Efter øvelsen bør du kunne:

- forklare hvordan FIM fungerer
- identificere begrænsninger ved checksum-baseret overvågning

- forstå hvorfor flere sikkerhedsmekanismer kombineres

Last update: 2026-03-20 13:58:28