Intro to Auditd

Linux Audit system ကို လုံခြုံရေးလောကမှာ အများအားဖြင့် **“auditd”** လို့ ခေါ်ကြပါတယ်။ auditd ရဲ့ အဓိကတာဝန်က Linux စနစ်အတွင်းမှာ ဖြစ်ပေါ်နေတဲ့ အဖြစ်အပျက်တွေကို မှတ်တမ်းတင်ထားပေးတာပါ။ အထူးသဖြင့် လုံခြုံရေးနဲ့ ဆက်နွယ်တဲ့ အဖြစ်အပျက်တွေကို အဓိကထားပြီး မှတ်တမ်းတင်ပါတယ်။ ဥပမာ အားဖြင့် user တစ်ယောက်က system file တစ်ခုကို ဖွင့်ကြည့်တာ၊ ပြင်တာ၊ delete လုပ်တာ၊ privilege မြင့်တဲ့ command တစ်ခုကို run လုပ်တာ စတဲ့ အရာတွေကို auditd က စောင့်ကြည့်နိုင်ပါတယ်။ ဒါပေမယ့် auditd က အရာအားလုံးကို အလိုအလျောက် မမှတ်တမ်းတင်ပါဘူး။ ကျွန်တော်တို့က “audit rules” ဆိုတဲ့ စည်းမျဉ်းတွေကို သတ်မှတ်ပေးရပြီး အဲ့ဒီ rules နဲ့ ကိုက်ညီတဲ့ အဖြစ်အပျက်တွေကိုပဲ log အဖြစ် သိမ်းဆည်းပါတယ်။

Audit system က အသစ်တီထွင်ထားတာ မဟုတ်ဘဲ Linux မှာ နှစ်ပေါင်းများစွာကတည်းက ရှိလာခဲ့တာပါ။ အသုံးပြုရခက်တယ်၊ user-friendly မဖြစ်ဘူးဆိုတဲ့ အမြင်တွေရှိပေမယ့် auditd ကို အစားထိုးမယ့် standard အသစ်တစ်ခု မပေါ်လာသေးတဲ့အတွက် ယနေ့အချိန်အထိ production environment တွေမှာ ဆက်ပြီး အသုံးပြုနေကြဆဲ ဖြစ်ပါတယ်။ သင့်အနေနဲ့ ဘယ် Linux distribution ကို သုံးနေပါစေ auditd ကို အများအားဖြင့် တွေ့ရပါလိမ့်မယ်။ ထို့အပြင် လုံခြုံရေးဆိုင်ရာ product အများစုကလည်း auditd ကို အဓိက event source အဖြစ် အသုံးချနေဆဲ ဖြစ်ပါတယ်။

အခုခေတ်မှာ Linux အတွက် EDR (Endpoint Detection and Response) tool တွေ ရှိလာပြီး attack detection ကို ကူညီပေးနေပေမယ့် auditd ကို လေ့လာထားခြင်းက သင့်အတွက် အရမ်းအရေးကြီးပါ တယ်။ အကြောင်းက auditd ကို နားလည်ထားရင် EDR tool တွေက ဘယ်လို data တွေကို အခြေခံပြီး detection လုပ်နေလဲဆိုတာကို ပိုပြီး နက်နက်ရှိုင်းရှိုင်း သိလာနိုင်ပါတယ်။ တစ်နည်းအားဖြင့် auditd က Linux security telemetry ရဲ့ အခြေခံအုတ်မြစ်လို ဖြစ်ပါတယ်။

Audit system ရဲ့ အားသာချက်တစ်ခုက Linux အားလုံးမှာ အလားတူပုံစံနဲ့ ရနိုင်တာပါ။ ဒီ universality ကြောင့် ကျွန်တော်တို့အနေနဲ့ hunting နဲ့ detection အတွက် အလွန်တိကျတဲ့ telemetry တွေကို စုဆောင်း နိုင်ပါတယ်။ Production environment တွေမှာ သာမန် lab environment နဲ့ မတူတဲ့ အထူးသဘောတရား တွေ ရှိပါတယ်။ ယနေ့ခေတ်မှာ on-premise infrastructure နဲ့ cloud ကို ပေါင်းစပ်သုံးနေတဲ့ hybrid environment များလာတာကြောင့် attack surface က ပိုပြီး ကျယ်ပြန့်လာပါတယ်။

Developer တွေက private key တွေ၊ password တွေ၊ API token တွေကို security tool မစောင့်ကြည့် တဲ့ နေရာတွေမှာ သိမ်းထားတာ မကြာခဏ တွေ့ရပါတယ်။ ဒါ့အပြင် CI/CD pipeline တွေမှာ Linux system ပေါ်က data တွေရဲ့ integrity နဲ့ confidentiality က အလွန်အရေးကြီးပါတယ်။ အဲ့ဒီနေရာတွေကို attacker တစ်ယောက် ဝင်ရောက်နိုင်ခဲ့ရင် အဖွဲ့အစည်းတစ်ခုလုံးအတွက် အန္တရာယ်ကြီးပါတယ်။ auditd ကို မှန်ကန်စွာ နားလည်ပြီး သင့် environment အပေါ် အခြေခံထားတဲ့ rule တွေကို ရေးနိုင်ရင် အဲ့ဒီလို မ မြင်သာတဲ့ အန္တရာယ်တွေကို အလင်းရောင်ထိုးပြနိုင်ပါတယ်။

ဒီ module ရဲ့ ရည်ရွယ်ချက်က auditd ကို analyst perspective ကနေ နားလည်အောင် လေ့လာပေးဖို့ပါ။ အဲ့ဒါကြောင့် auditd ကို install ဘယ်လိုလုပ်ရမလဲဆိုတာထက် audit rules တွေကို ဘယ်လိုရေးရမလဲ၊ rule တစ်ခုက ဘယ်လို event တွေ generate လုပ်ပေးလဲ၊ ထွက်လာတဲ့ log တွေကို ဘယ်လိုဖတ်ပြီး ဘာ အဓိပ္ပါယ်ရလဲဆိုတာကို အဓိကထား လေ့လာပါမယ်။ နောက်ပိုင်းမှာ intrusion တစ်ခုကို ဥပမာအနေနဲ့ ယူပြီး attacker က ဘာတွေ လုပ်ခဲ့လဲဆိုတာကို audit rule တွေနဲ့ ဘယ်လို ဖော်ထုတ်နိုင်လဲဆိုတာကိုလည်း ဆက်လက် လေ့လာသွားပါမယ်။

The Audit System

ဒီအပိုဒ်က Linux Audit System ကို နည်းပညာအနည်းငယ် ပိုနက်တဲ့ အဆင့်ကနေ ရှင်းပြထားတာဖြစ်ပေမယ့် Cybersecurity အခြေခံလောက်ပဲ သိထားတဲ့သူအတွက်လည်း သဘောတရားချိတ်ဆက်ပြီး နားလည်နိုင်အောင် ရှင်းပြနိုင်ပါတယ်။

ပထမဆုံး Operating System ရဲ့အခြေခံသဘောတရားကို နားလည်ရပါမယ်။ Linux လို operating system တွေမှာ userspace လို့ ခေါ်တဲ့ နေရာမှာ application တွေ၊ command တွေ၊ user process တွေ run နေပါတယ်။ Security အတွက် အရေးကြီးတဲ့ အချက်က userspace process တစ်ခုဟာ system ရဲ့အခြားအစိတ်အပိုင်းတွေကို တိုက်ရိုက် ထိတွေ့လို့ မရအောင် OS က ခြံစည်းရိုးတစ်ခုလို ကာကွယ်ထားပါတယ်။ ဒါက isolation လို့ ခေါ်ပြီး application တစ်ခု ပျက်သွားရင် system တစ်ခုလုံး မပျက်အောင် ကာကွယ်ပေးတဲ့ အရေးကြီးတဲ့ mechanism ဖြစ်ပါတယ်။

ဒါပေမယ့် application တစ်ခုဟာ ကိုယ့်အတွင်းမှာပဲ လည်နေရင် အသုံးမဝင်ပါဘူး။ File ဖတ်ဖို့လိုတယ်၊ network နဲ့ ဆက်သွယ်ချင်တယ်၊ process အသစ်တစ်ခု စချင်တယ် စတဲ့ အလုပ်တွေ လုပ်ရပါမယ်။ အဲ့ဒီအခါ userspace process က kernel ကို တိုက်ရိုက် မထိနိုင်တဲ့အတွက် အလယ်ခံ API သို့မဟုတ် library တစ်ခုကို သုံးပြီး kernel ကို request ပို့ရပါတယ်။ Program ကို compile လုပ်ပြီး run လိုက်တဲ့အချိန်မှာ control က kernel ဆီကို ပြောင်းသွားပြီး kernel က အလုပ်ကို process အစား လုပ်ပေးပါတယ်။

Linux မှာ userspace process က kernel ကို အလုပ်ခိုင်းတဲ့ နည်းလမ်းကို system call လို့ ခေါ်ပါတယ်။ Syscall ဆိုတာ file တစ်ခုကို ဖွင့်ချင်တဲ့အချိန်၊ process အသစ်တစ်ခု ဖန်တီးချင်တဲ့အချိန်၊ process အချင်းချင်း ဆက်သွယ်ချင်တဲ့အချိန်တွေမှာ မဖြစ်မနေ အသုံးပြုရတဲ့ mechanism ဖြစ်ပါတယ်။ Security perspective ကနေ ကြည့်ရင် ဒီ syscalls တွေက အရမ်းစိတ်ဝင်စားစရာကောင်းပါတယ်။ ဘာကြောင့်လဲဆိုတော့ attacker တစ်ယောက် ဘာလုပ်လုပ် file access၊ process execution၊ privilege change တွေဟာ syscalls မပါဘဲ မဖြစ်နိုင်လို့ပါ။ Detection နဲ့ investigation အတွက် syscalls ကို စောင့်ကြည့်နိုင်ရင် attacker ရဲ့လှုပ်ရှားမှုကို အခြေခံအဆင့်ကနေ ဖမ်းနိုင်ပါတယ်။

Audit system ဟာ userspace program တွေနဲ့ kernel component တစ်ခု ပေါင်းစပ်ထားတဲ့ system ဖြစ်ပါတယ်။ Process တစ်ခု syscall run တိုင်း kernel အတွင်းက Audit component က အဲ့ဒီ syscall ကို ကြားဖြတ် ဖမ်းယူပါတယ်။ ပြီးတော့ ကျွန်တော်တို့ သတ်မှတ်ထားတဲ့ audit rules နဲ့ ကိုက်ညီမလားဆိုတာ စစ်ဆေးပါတယ်။ Rule နဲ့ ကိုက်ညီရင် အဲ့ဒီ syscall နဲ့ ဆိုင်တဲ့ အချက်အလက်တွေကို audit record အဖြစ် ပြင်ဆင်ပြီး userspace ထဲကို ပို့ပါတယ်။

အဲ့ဒီ data ကို လက်ခံပြီး disk ပေါ်ကို ရေးသိမ်းပေးတဲ့ userspace daemon ကို auditd လို့ ခေါ်ပါတယ်။ auditd က Audit system ရဲ့အဓိက userspace component ဖြစ်ပြီး log file ကို ပုံမှန်အားဖြင့် `/var/log/audit/audit.log` ထဲမှာ သိမ်းဆည်းထားပါတယ်။ Analyst တစ်ယောက်အနေနဲ့ forensic investigation လုပ်တဲ့အခါ ဒီ audit.log ဟာ အလွန်အရေးကြီးတဲ့ သက်သေမှတ်တမ်း ဖြစ်လာပါတယ်။

ဒီနေရာမှာ သတိထားရမယ့် အချက်တစ်ခုက rule တစ်ခုချင်းစီဟာ applicable ဖြစ်တဲ့ syscall တိုင်းအတွက် စစ်ဆေးခံရပါတယ်။ Rule မလိုအပ်ပဲ များလွန်းရင် syscall တိုင်းကို စစ်ရတဲ့အတွက် performance ကို ထိခိုက်စေနိုင်ပါတယ်။ Production system တွေမှာ audit rule ကို တန်းတန်းချင်း ထည့်သုံးတာ မသင့်တော်ပါဘူး။ အရင်ဆုံး lab environment မှာ စမ်းသပ်ပြီး system performance အပေါ် ဘယ်လောက် သက်ရောက်လဲဆိုတာ စစ်ဆေးရပါမယ်။ ပြီးမှ production ကို အဆင့်လိုက် ဖြန့်ချိသုံးစွဲတာက availability impact ကို လျော့ချပေးနိုင်ပါတယ်။

Auditctl

Audit system ရဲ့ နောက်ထပ် userspace tool တစ်ခုက **auditctl** ဖြစ်ပါတယ်။ auditctl ဟာ Audit system ကို ထိန်းချုပ်တဲ့ command-line utility ဖြစ်ပြီး kernel ထဲကို audit rules တွေ load လုပ်ပေးတဲ့ တာဝန်ကို ထမ်းဆောင်ပါတယ်။ `/etc/audit/audit.rules` ဖိုင်ထဲမှာ ရေးထားတဲ့ rule တစ်ကြောင်းချင်းစီဟာ တကယ်တော့ auditctl command တစ်ခုစီပါပဲ။ auditctl ကို audit daemon ရဲ့ status စစ်ဖို့၊ configuration ပြောင်းဖို့လည်း သုံးနိုင်ပေမယ့် လက်တွေ့မှာတော့ service control အတွက် `systemctl` ကို ပိုအသုံးများပါတယ်။

နောက်ဆုံး mention လုပ်ထားတဲ့ userspace component က **ausearch** ဖြစ်ပါတယ်။ ausearch ကို audit log တွေကို query လုပ်ဖို့ သုံးပါတယ်။ Raw audit.log ဖိုင်ကို တန်းဖတ်ရင် data တွေဟာ ဖတ်ရခက်ပြီး hex encoding နဲ့ timestamp တွေကြောင့် analyst အတွက် အချိန်ကုန်ပါတယ်။ ausearch က အဲ့ဒီအခက်အခဲကို လျော့ချပေးပါတယ်။ `--format interpret` သို့မဟုတ် `-i` option ကို သုံးလိုက်ရင် unix timestamp ကို လူဖတ်လို့ရတဲ့ အချိန်ပုံစံအဖြစ် ပြောင်းပေးပြီး hex encoding ဖြစ်နေတဲ့ data တွေကိုလည်း decode လုပ်ပေးပါတယ်။

ဥပမာအနေနဲ့ execve syscall ကို ရှာချင်ရင်

```
ausearch -sc execve --format interpret
```

ဒါမှမဟုတ်

```
ausearch -sc execve -i
```

လို့ run လိုက်ရင် human-readable ဖြစ်တဲ့ output ကို ရပါတယ်။

```
type=PROCTITLE msg=audit(12/10/2023 19:23:30.388:6694) : proctitle=ausearch
-sc execve --format interpret
```

ဒါကို `/var/log/audit.log` ကို ဖတ်တဲ့အခါ ရလာတဲ့ hex string

```
type=PROCTITLE msg=audit(1702257810.388:6694):
proctitle=617573656172636800 2D7363006578656376650002D2D666F726D617400696E7465
7270726574
```

နဲ့ နှိုင်းယှဉ်ကြည့်ရင် ausearch က analyst အတွက် ဘယ်လောက် အထောက်အကူပြုလဲဆိုတာကို အလွယ်တကူ မြင်နိုင်ပါတယ်။

Rules

Audit rules ဆိုတာ Linux system ထဲမှာ ဘာအဖြစ်အပျက်တွေကို စောင့်ကြည့်ပြီး log ထုတ်မလဲဆိုတာကို သတ်မှတ်ပေးတဲ့ စည်းမျဉ်းတွေ ဖြစ်ပါတယ်။ Auditd ကို install လုပ်ထားတယ်ဆိုတာနဲ့ အရာအားလုံးကို အပြည့်အဝ စောင့်ကြည့်နိုင်တာ မဟုတ်ပါဘူး။ Rule မရှိရင် auditd က ဘာကိုမှ အထူးတလည် မ

မှတ်တမ်းတင်ပါဘူး။ ဒါကြောင့် audit rules ဟာ Audit system ရဲ့ မျက်လုံးနဲ့ ဦးနှောက်လို အရေးကြီးပါတယ်။

Audit rules ကို ထည့်သွင်းနိုင်တဲ့ နည်းလမ်း နှစ်မျိုးရှိပါတယ်။ ပထမတစ်မျိုးက auditctl command ကိုသုံးပြီး runtime အတွင်းမှာ တိုက်ရိုက် ထည့်သွင်းတာပါ။ ဒီနည်းလမ်းက စမ်းသပ်မှုလုပ်တဲ့အခါ အသုံးဝင်ပေမယ့် system reboot လုပ်လိုက်ရင် rule တွေ ပျောက်သွားနိုင်ပါတယ်။ ဒုတိယနည်းလမ်းက `/etc/audit/audit.rules` ဖိုင်ထဲမှာ rule တွေကို ရေးထားတာပါ။ ဒီနည်းလမ်းနဲ့ ထည့်ထားတဲ့ rule တွေက system reboot ပြန်လုပ်လည်း မပျောက်ပါဘူး။

လက်တွေ့မှာတော့ `/etc/audit/audit.rules` ဖိုင်ထဲကို တိုက်ရိုက်ရေးတာထက် `/etc/audit/rules.d/` directory ထဲမှာ rule ဖိုင်တွေကို ခွဲပြီး သိမ်းထားတာကို ပိုပြီး အသုံးများပါတယ်။ System start ဖြစ်တဲ့အချိန် auditd က rules.d ထဲက ဖိုင်တွေကို စုစည်းပြီး audit.rules ဖိုင်ထဲကို အလိုအလျောက် ပေါင်းထည့်ပေးပါတယ်။ Analyst အနေနဲ့ လက်ရှိ system မှာ ဘယ် audit rules တွေ အလုပ်လုပ်နေတယ်ဆိုတာကို အောက်က command နဲ့ စစ်ဆေးနိုင်ပါတယ်။

```
auditctl -l
```

Audit system ဟာ default အနေနဲ့ အချို့သော event တွေကိုတော့ ထုတ်ပေးထားပါတယ်။ ဥပမာ sshd, sudo, passwd, pam လို security နဲ့ ဆိုင်တဲ့ utility တွေကို အနည်းငယ် စောင့်ကြည့်ထားပါတယ်။ ဒါပေမယ့် ဒီ default rules တွေကို လုံလောက်တယ်လို့ မထင်သင့်ပါဘူး။ ဥပမာ sudo နဲ့ command run လုပ်တဲ့အခါ log ထုတ်ပေးပေမယ့် root account နဲ့ တိုက်ရိုက် command run လုပ်တဲ့အခါတော့ log မထုတ်ပေးနိုင်တဲ့ အခြေအနေတွေ ရှိပါတယ်။ အဲ့ဒါကြောင့် default audit events တွေက security monitoring အတွက် အပြည့်အဝ မလုံလောက်ဘူးလို့ ယူဆရပါမယ်။

Auditd ရဲ့ source code အတွင်းမှာ compliance အတွက် အသုံးပြုနိုင်တဲ့ rule ဥပမာတွေ အများကြီးပါဝင်ပါတယ်။ PCI DSS လို payment security standard တွေ၊ DISA STIG လို government security guideline တွေအတွက် အသင့်သုံး rule ဖိုင်တွေ ရှိပါတယ်။ ဒါပေမယ့် အခြေခံကနေ စတင်ချင်တဲ့သူအတွက် Florian Roth က စုစည်းထားတဲ့ Best Practice Configuration ဟာ ပိုပြီး အသုံးဝင်တဲ့ baseline တစ်ခု ဖြစ်ပါတယ်။ ဒီ rule set ကို သုံးလို့ performance ထိခိုက်မလားဆိုတာကတော့ ကိုယ့် environment အပေါ်မှာ မူတည်ပါတယ်။ System ပေါ်မှာ service များရင်၊ workload များရင် rule များလွန်းတာက performance ကို ထိခိုက်နိုင်ပါတယ်။ အဲ့ဒီအခါ noise မလိုတဲ့ event တွေကို exclude rule နဲ့ ဖယ်ထုတ်တာ၊ အရေးကြီးတဲ့ file တွေအတွက်ပဲ watch ထားတာလို စနစ်တကျ ပြင်ဆင်ရပါတယ်။

Audit rules ကို သဘောတရားအရ အမျိုးအစား သုံးမျိုး ခွဲနိုင်ပါတယ်။

1. Control rules ဆိုတာ Audit system ကိုယ်တိုင်ရဲ့ configuration ကို ထိန်းချုပ်တဲ့ rule တွေ ဖြစ်ပါတယ်။ ဥပမာ rule အားလုံးကို clear လုပ်တာ၊ event rate limit သတ်မှတ်တာလို အလုပ်တွေကို control rules နဲ့ လုပ်ပါတယ်။ ဒါတွေက monitoring ထက် management ဘက်ကို ပိုဦးတည်ပါတယ်။

2. System call rules ကတော့ security analyst တွေအတွက် အရေးအကြီးဆုံး အမျိုးအစား ဖြစ်ပါတယ်။ ဒီ rule တွေကို သုံးပြီး execve, open, unlink လို syscall တစ်ခုချင်းစီကို စောင့်ကြည့်နိုင်ပါတယ်။ Process တစ်ခု command run လုပ်တာ၊ file ဖွင့်တာ၊ file ဖျက်တာ စတဲ့ attacker လှုပ်ရှားမှုအများစုဟာ syscall မပါဘဲ မဖြစ်နိုင်တဲ့အတွက် ဒီ rule မျိုးတွေက detection နဲ့ investigation

အတွက် အခြေခံအုတ်မြစ် ဖြစ်ပါတယ်။

3. File system rules ကို watch rules လို့လည်း ခေါ် ကြပါတယ်။ ဒီ rule မျိုးက file သို့မဟုတ် directory တစ်ခုကို စောင့်ကြည့်ဖို့ သုံးပါတယ်။ သတ်မှတ်ထားတဲ့ file ကို read, write, execute လုပ်တဲ့အခါ event ထုတ်ပေးပါဆိုပြီး Audit ကို ပြောထားတာပါ။ ဥပမာ `/etc/passwd`, `/etc/shadow` လို အရေးကြီးတဲ့ file တွေကို watch ထားရင် attacker တစ်ယောက် ပြင်ဆင်တာ၊ ဖတ်ကြည့်တာကို အလွယ်တကူ သိနိုင်ပါတယ်။

Syscall Rules

Syscall rule ဆိုတာက ဒီ system call တွေထဲက ကိုယ်စိတ်ဝင်စားတဲ့ syscall တွေကို စောင့်ကြည့်ပြီး log ထုတ်ပေးပါဆိုပြီး Audit system ကို ပြောထားတဲ့ စည်းမျဉ်း ဖြစ်ပါတယ်။

ဥပမာအနေနဲ့ အောက်ပါ rule ကို ကြည့်ကြရအောင်။

```
auditctl -a always,exit -S execve -F arch=b64 -k command
```

ဒီ rule တစ်ကြောင်းမှာ ပါတဲ့ option တစ်ခုချင်းစီက အရေးကြီးတဲ့ အဓိပ္ပါယ်တွေ ပါရှိပါတယ်။

ပထမဆုံး `-a` option က rule ကို rule list ထဲမှာ append လုပ်မယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ Rule list ဆိုတာ audit system က စစ်ဆေးသွားတဲ့ rule အစီအစဉ်ကို ပြောတာပါ။ -a option နဲ့ အတူ action နဲ့ list ကို သတ်မှတ်ပေးရပါတယ်။ ဒီနေရာမှာ `always` က action ဖြစ်ပြီး `exit` က list ဖြစ်ပါတယ်။ Action ဆိုတာ matching ဖြစ်တဲ့ event ကို log ထုတ်မလား၊ မထုတ်ဘူးလား ဆိုတာကို ပြောတာပါ။ `always` ဆိုရင် log ထုတ်မယ်၊ `never` ဆိုရင် log မထုတ်ဘူးလို့ kernel ကို ပြောတာပါ။

List ကို filter လို့လည်း ခေါ် ကြပါတယ်။ List အမျိုးအစားတွေက `task`, `exit`, `user`, `exclude`, `filesystem` ဆိုပြီး အမျိုးမျိုးရှိပါတယ်။ အစပိုင်းမှာတော့ `exit` နဲ့ `exclude` ကို နားလည်ထားရင် လုံလောက်ပါတယ်။ `Exit` list ဆိုတာ syscall တစ်ခု အလုပ်ပြီးပြီး kernel ထဲကနေ ပြန်ထွက်လာတဲ့ အချိန်မှာ event တစ်ခု ထုတ်ပေးတဲ့ list ဖြစ်ပါတယ်။ ဒါကြောင့် `exit` list ဟာ အသုံးအများဆုံးဖြစ်ပြီး syscall တစ်ခါ ဖြစ်တိုင်း record တစ်ခု ရလာပါတယ်။

`Exclude` list ကတော့ ထုတ်လာမယ့် event တွေထဲက မလိုချင်တဲ့ အရာတွေကို ဖယ်ထုတ်ဖို့ သုံးပါတယ်။ `Exclude` list ကို သုံးတဲ့အခါ action ကို always လို့ ရေးထားပေမယ့် အမှန်တကယ်မှာ never လို့ သဘောထားပြီး matching ဖြစ်တဲ့ event တွေကို drop လုပ်ပစ်ပါတယ်။ ဥပမာအနေနဲ့

```
-a always,exclude -F msgtype=CWD
```

ဆိုတဲ့ rule ကို execve rule အပေါ်မှာ ထားလိုက်ရင် CWD ဆိုတဲ့ auxiliary event တွေကို မထုတ်ပေးတော့ပါဘူး။ Action မှာ always လို့ ရေးထားပေမယ့် exclude list ဖြစ်တဲ့အတွက် matching event တွေကို kernel က အလွယ်တကူ လျစ်လျူရှုသွားပါတယ်။ ဒါက noise လျော့ဖို့ အရေးကြီးတဲ့ နည်းလမ်းတစ်ခုပါ။

နောက်တစ်ခုက `-S` option ဖြစ်ပါတယ်။ ဒီ option က ကိုယ်စောင့်ကြည့်ချင်တဲ့ syscall ကို သတ်မှတ်ပေးတာပါ။ ဒီဥပမာမှာ execve ကို သုံးထားပါတယ်။ Execve syscall ဟာ executable သို့မဟုတ် script တစ်ခုကို run လုပ်ပြီး process အသစ်တစ်ခု ဖန်တီးတဲ့အခါ အသုံးပြုပါတယ်။ အခြေခံအားဖြင့် Linux မှာ command run လုပ်တိုင်း execve ကို သုံးရတာဖြစ်လို့ attacker activity ကို သိချင်ရင် အလွန်အရေးကြီးတဲ့ syscall ဖြစ်ပါတယ်။

`-F` option က conditional filter ထည့်ဖို့ သုံးပါတယ်။ Field နဲ့ value ကို တွဲပြီး သတ်မှတ်ပေးတာပါ။ ဥပမာ arch=b64 ဆိုတာ 64-bit architecture နဲ့ run နေတဲ့ process တွေကိုပဲ match လုပ်မယ်လို့ အဓိပ္ပါယ်ရပါတယ်။ arch!=b64 ဆိုရင် 64-bit မဟုတ်တဲ့ process တွေအားလုံးကို ဆိုလိုပါတယ်။ Numeric value တွေအတွက် <, >, <=, >=, &, &= လို operator တွေလည်း သုံးနိုင်ပါတယ်။ ဒါပေမယ့် အရေးကြီးတဲ့ အချက်က field အားလုံးကို filter အနေနဲ့ သုံးလို့ မရပါဘူး။ ဥပမာ tty field က interactive command ကို ခွဲခြားဖို့ အသုံးဝင်ပေမယ့် filter အနေနဲ့တော့ မသုံးနိုင်ပါဘူး။ ဒါကြောင့် audit rule ရေးတဲ့အခါ ဘယ် field တွေ filter လို့ရလဲဆိုတာကို သိထားရပါမယ်။

နောက်ဆုံး `-k` option က key ကို သတ်မှတ်ပေးတာပါ။ Key ဆိုတာ rule ကို နာမည်ပေးလိုက်သလို ဖြစ်ပါတယ်။ ဒီဥပမာမှာ key=command လို့ ထည့်ထားပါတယ်။ Event ထွက်လာတဲ့အခါ audit record ထဲမှာ

```
key=command
```

ဆိုတဲ့ field ပါလာပြီး analyst အနေနဲ့ ausearch -k command လို command နဲ့ လွယ်လွယ်ကူကူ ရှာဖွေနိုင်ပါတယ်။ Rule များလာတဲ့ environment တွေမှာ key ဟာ rule management နဲ့ analysis အတွက် အလွန်အရေးကြီးပါတယ်။

Syscall Events

Linux system ပေါ်မှာ command တစ်ခု run လုပ်လိုက်တဲ့အချိန် execve syscall rule ကို ထားထားရင် kernel အတွင်းမှာ audit record အများကြီးကို တစ်ပြိုင်နက် ဖန်တီးပါတယ်။ အဲ့ဒီ record တွေကို နောက်ဆုံးမှာ auditd က စုစည်းပြီး `/var/log/audit/audit.log` ထဲကို ရေးသွားပါတယ်။ ဒီ event တွေအားလုံးထဲမှာ အဓိကအဖြစ်ဆုံးက SYSCALL event ဖြစ်ပါတယ်။ SYSCALL ဆိုတာ “ဒီမှာ system call တစ်ခု ဖြစ်သွားပြီ” ဆိုတာကို ကိုယ်စားပြုတဲ့ အဓိက record ဖြစ်ပါတယ်။

```
type=SYSCALL msg=audit(1701803185.584:5746): arch=c000003e syscall=59
success=yes exit=0 a0=55baa8552ec0 a1=55baa85533d0 a2=55baa8552540 a3=8
items=2 ppid=8327 pid=8337 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0
egid=0 sgid=0 fsgid=0 tty=pts1 ses=2 comm="cat" exe="/usr/bin/cat"
subj=unconfined key=636F6D6D616E640DARCH=x86_64 SYSCALL=execve AUID="ace"
UID="root" GID="root" EUID="root" SUID="root" FSUID="root" EGID="root"
SGID="root" FSGID="root"
```

SYSCALL event ကို ကြည့်ရင် field/value pair အများကြီးကို whitespace နဲ့ ခွဲပြီး ရေးထားတာကို တွေ့ရပါလိမ့်မယ်။ အစပိုင်းမှာတော့ အားလုံးကို နားလည်ဖို့ မလိုပါဘူး။ အရေးကြီးတာက ဒီ record ဟာ syscall တစ်ခုအတွက် kernel က မှတ်တမ်းတင်ထားတဲ့ အချက်အလက် အစုတစ်ခုဆိုတာကို သိထားရပါမယ်။

SYSCALL record ထဲက msg field က အရမ်းအရေးကြီးပါတယ်။ msg ထဲမှာ timestamp နဲ့ correlation ID ပါဝင်ပါတယ်။ Timestamp က event ဖြစ်သွားတဲ့ အချိန်ကို ပြသပြီး correlation ID ကတော့ ဒီ event နဲ့ ဆက်စပ်တဲ့ auxiliary event တွေကို ချိတ်ဆက်ဖို့ အသုံးပြုပါတယ်။ Execve တစ်ခါ run လိုက်ရင် SYSCALL တစ်ခုတည်းမဟုတ်ဘဲ EXECVE, CWD, PATH, PROCTITLE လို auxiliary event တွေလည်း ထွက်လာပါတယ်။ ဒီ correlation ID တူနေတဲ့ event တွေအားလုံးကို စုလိုက်ရင် “command တစ်ခု run လိုက်တဲ့အခါ ဖြစ်သွားတဲ့ အရာအားလုံး” ကို တစ်စုတစ်စည်းတည်း မြင်နိုင်ပါတယ်။ Incident investigation လုပ်တဲ့အခါ ဒီ correlation ID က အရမ်းအသုံးဝင်ပါတယ်။

syscall field က ဒီ event ဟာ ဘယ် syscall အတွက်လဲဆိုတာကို ပြပါတယ်။ ဥပမာ 59 ဆိုတာ execve syscall ကို ကိုယ်စားပြုတဲ့ numeric value ဖြစ်ပါတယ်။ Kernel အတွင်းမှာ syscall တွေကို နံပါတ်နဲ့ ခေါ်တာဖြစ်လို့ ဒီလို တွေ့ရတာပါ။ ausearch သုံးရင် ဒီနံပါတ်တွေကို လူဖတ်လို့ရတဲ့ execve လို စာသားအဖြစ် ပြန်ပြောင်းပေးနိုင်ပါတယ်။ Logging pipeline ကောင်းကောင်းရှိရင် ဒီ field ကို human-readable အဖြစ် enrich လုပ်ထားတာက analyst အတွက် အရမ်းအဆင်ပြေပါတယ်။

a0 ကနေ **a3** ထိ field တွေက syscall ကို ပေးလိုက်တဲ့ argument ပထမလေးခုကို ပြတာပါ။ ဒါပေမယ့် execve ရဲ့ argument တွေက pointer တွေဖြစ်တဲ့အတွက် SYSCALL event ထဲမှာတော့ အသုံးမဝင်ပါဘူး။ တကယ့် command နဲ့ argument အပြည့်အစုံကို သိချင်ရင် EXECVE ဆိုတဲ့ auxiliary event ကို ကြည့်ရပါတယ်။ Audit design အရ SYSCALL က kernel-level အချက်အလက်ကို ပိုဦးစားပေးပြီး auxiliary event တွေက လူနားလည်လွယ်တဲ့ context ကို ပေးတာဖြစ်ပါတယ်။

auid, uid, gid စတဲ့ field တွေက POSIX UID/GID အချက်အလက်တွေကို ပြပါတယ်။ ဒီနေရာမှာ အရေးကြီးဆုံး သဘောတရားက auid (Audit User ID) ဖြစ်ပါတယ်။ auid က “ဒီ process ကို စတင်ခဲ့တဲ့ မူလ user က ဘယ်သူလဲ” ဆိုတာကို ကိုယ်စားပြုပါတယ်။ ဥပမာ user ace က login ဝင်ပြီး su root လုပ်လိုက်ရင် uid, euid စတဲ့ field တွေက root ဖြစ်သွားပေမယ့် auid နဲ့ AUID ကတော့ ace အဖြစ်ပဲ ဆက်ရှိနေပါတယ်။ ဒါကြောင့် forensic investigation လုပ်တဲ့အခါ “root နဲ့ run လုပ်ထားတဲ့ command ကို တကယ်တမ်း ဘယ် user က စတင်ခဲ့လဲ” ဆိုတာကို သိနိုင်ပါတယ်။ ဒီအချက်က attacker attribution အတွက် အရမ်းအရေးကြီးပါတယ်။

tty, pid, ppid လို field တွေက process context ကို ပြပါတယ်။ pid က ဒီ command ရဲ့ process ID ဖြစ်ပြီး ppid က ဒီ process ကို ခေါ်ထားတဲ့ parent process ဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် shell က command ကို run လုပ်တာဖြစ်လို့ ppid က bash ဖြစ်တတ်ပါတယ်။ tty field က command ကို ဘယ် terminal ကနေ run လုပ်လဲဆိုတာ ပြပါတယ်။ pts ဆိုတာ pseudo-terminal ဖြစ်ပြီး ssh လို remote login ကနေ လာတာကို ညွှန်ပြတတ်ပါတယ်။ tty အဖြစ် ပုံမှန် tty နံပါတ်ပါရင် physical console ဖြစ်နိုင်ပါတယ်။ tty=(none) ဖြစ်ရင် terminal မပါဘဲ background process သို့မဟုတ် service တစ်ခုက run လုပ်တာ ဖြစ်နိုင်ပါတယ်။ ဒီအချက်တွေက lateral movement နဲ့ persistence စစ်တဲ့အခါ အရမ်းအသုံးဝင်ပါတယ်။

```
type=EXECVE msg=audit(1701803185.584:5746): argc=2 a0="cat"
a1="/var/log/audit/audit.log"
```

Execve syscall အတွက် SYSCALL event တစ်ခုတည်း မဟုတ်ဘဲ auxiliary event တွေကိုလည်း တွေ့ရပါမယ်။ EXECVE event က argument အရေအတွက်နဲ့ argument အပြည့်အစုံကို ပြပါတယ်။ ဒီဥပမာမှာ `cat` command ကို `/var/log/audit/audit.log` ဖတ်ဖို့ run လုပ်ထားတာကို တိတိကျကျ မြင်နိုင်ပါတယ်။

```
type=CWD msg=audit(1701803185.584:5746): cwd="/home/ace"
```

CWD event က command ကို ဘယ် working directory ကနေ run လုပ်လဲဆိုတာ ပြပါတယ်။ Attacker တစ်ယောက် home directory ထဲက run လုပ်တာလား၊ `/tmp` ထဲက run လုပ်တာလားဆိုတာ သိရင် context ပိုမိုနားလည်နိုင်ပါတယ်။

```
type=PATH msg=audit(1701803185.584:5746): item=0 name="/usr/bin/cat"
inode=1180 dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL
cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0OUID="root" OGID="root"

type=PATH msg=audit(1701803185.584:5746): item=1 name="/lib64/ld-linux-x86-
64.so.2" inode=3140 dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00
nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0
cap_frootid=0OUID="root" OGID="root"
```

PATH event က execve ကို ပေးလိုက်တဲ့ binary သို့မဟုတ် script ကို ပြပါတယ်။ ဒီဥပမာမှာ `/usr/bin/cat` နဲ့ dynamic loader ဖြစ်တဲ့ /lib64/ld-linux-x86-64.so.2 ကို auditd က မှတ်တမ်းတင်ထားပါတယ်။ Linux မှာ executable ကို run လိုက်တဲ့အခါ shared library တွေ load ဖို့ dynamic linker ကို အသုံးပြုရတဲ့အတွက် auditd က ဒီ path နှစ်ခုစလုံးကို မှတ်တမ်းတင်ထားတာပါ။ Malware analysis လုပ်တဲ့အခါ PATH event တွေက အလွန်အရေးကြီးပါတယ်။

```
type=PROCTITLE msg=audit(1701803185.584:5746):
proctitle=636174002F7661722F6C6F672F617564697742F617564697742E6C6F67
```

PROCTITLE event ကတော့ command အပြည့်အစုံကို hex-encoded ASCII အဖြစ် သိမ်းထားတာပါ။

```
cat /var/log/audit/audit.log
```

Decode လုပ်လိုက်ရင် cat /var/log/audit/audit.log လို့ တန်းမြင်ရပါတယ်။ အလွန်အဆင်ပြေတဲ့ field ဖြစ်ပေမယ့် PROCTITLE က truncation ဖြစ်နိုင်တာ၊ argument အချို့ကို မှန်မှန်မပြနိုင်တာလို limitation တွေရှိပါတယ်။ ဒါကြောင့် critical investigation လုပ်တဲ့အခါ PROCTITLE တစ်ခုတည်းကို မယုံဘဲ EXECVE နဲ့ SYSCALL event တွေကို အဓိကထားသုံးသင့်ပါတယ်။

စုစည်းပြီး ပြောရရင် Syscall Events ဆိုတာ Linux system ပေါ်မှာ command တစ်ခု run လိုက်တဲ့အချိန် ဖြစ်သွားတဲ့ အရာအားလုံးကို kernel အဆင့်ကနေ အသေးစိတ် မှတ်တမ်းတင်ထားတဲ့ security telemetry ဖြစ်ပါတယ်။ Correlation ID ကို အခြေခံပြီး SYSCALL, EXECVE, CWD, PATH, PROCTITLE တွေကို ချိတ်ဆက်ဖတ်နိုင်ရင် attacker က ဘယ်သူလဲ၊ ဘယ်နေရာကနေ၊ ဘာကို run လုပ်ခဲ့လဲဆိုတာကို အချိန်လိုင်းနဲ့ တိတိကျကျ ပြန်လည်တည်ဆောက်နိုင်ပါတယ်။

Auditbeat

Windows မှာ event log ကို စစ်တဲ့အခါ အများအားဖြင့် event ID သို့မဟုတ် event.code ကို အဓိကထားကြည့်ကြပါတယ်။ Event ID တစ်ခုချင်းစီက ဖြစ်ပေါ် ခဲ့တဲ့ အဖြစ်အပျက်ကို ကိုယ်စားပြုနေတာကြောင့် detection နဲ့ hunting အတွက် အရမ်းအသုံးဝင်ပါတယ်။ ဒါပေမယ့် Linux auditd နဲ့ Auditbeat ကို သုံးတဲ့အခါမှာတော့ event ID တစ်ခုတည်းကို မကြည့်ဘဲ context ကို ပိုပြီး အလေးထားရပါတယ်။

Auditd event တွေအတွက် analyst တစ်ယောက် အဓိက စိတ်ဝင်စားရမယ့် အချက်သုံးချက်ရှိပါတယ်။ ပထမအချက်က

- ဒီ event ဟာ file system watch rule ကနေ ထွက်လာတာလား၊ syscall rule ကနေ ထွက်လာတာလားဆိုတာပါ။
- ဒုတိယအချက်က syscall rule ဖြစ်တယ်ဆိုရင် ဘယ် syscall ကြောင့် ဒီ event ထွက်လာတာလဲဆိုတာပါ။
- တတိယအချက်က file watch ဖြစ်တယ်ဆိုရင် file ပေါ်မှာ ဘယ် operation ကို လုပ်ခဲ့လဲဆိုတာပါ။

ဒီသုံးချက်ကို နားလည်နိုင်ရင် auditd event တွေကို အစုလိုက်အပြုံလိုက် ခွဲခြားပြီး analysis လုပ်နိုင်ပါတယ်။

Auditbeat က auditd log တွေကို ingest လုပ်တဲ့အခါ Elastic Common Schema (ECS) နဲ့ ကိုက်ညီအောင် field တွေကို map လုပ်ပေးပါတယ်။ ဒီအတွက် analyst အနေနဲ့ `event.category`, `auditd.data.syscall` နဲ့ `event.action` ဆိုတဲ့ field တွေကို သုံးပြီး event တွေကို aggregate လုပ်နိုင်ပါတယ်။

Event Category

Event.category ဆိုတာ ဒီ event ဟာ ဘယ်အမျိုးအစားထဲ ဝင်လဲဆိုတာကို ပြပါတယ်။ File system နဲ့ ဆိုင်တဲ့ event တွေကို file category ထဲ ထားပြီး syscall နဲ့ ဆိုင်တဲ့ process execution တွေကို process category ထဲ ထားပါတယ်။

event.category	Description
file	File system events
process	System call events
authentication	Authentication events like USER_LOGIN and CRED_DISP
session	Session events like USER_START and USER_END

ဒီလို category ခွဲထားတာကြောင့် hunting query ရေးတဲ့အခါ “process activity ပဲကြည့်မယ်”၊ “file modification ပဲကြည့်မယ်” လို့ အလွယ်တကူ filter လုပ်နိုင်ပါတယ်။

Event Action

Event.action field ကတော့ ဒီ event မှာ ဘာလုပ်ဆောင်ချက် ဖြစ်သွားလဲဆိုတာကို ပြပါတယ်။

event.action	syscall
deleted	unlink

event.action	syscall
opened-file	openat, open
renamed	rename
executed	execve

Auditd ရဲ့ syscall တွေနဲ့ သဘောတရားတူအောင် map လုပ်ထားတာဖြစ်လို့ deleted ဆိုရင် unlink syscall ကြောင့် file ဖျက်လိုက်တာကို ဆိုလိုပါတယ်။ Opened-file ဆိုရင် open သို့မဟုတ် openat syscall ကြောင့် file ဖွင့်ထားတာကို ပြပါတယ်။ Renamed ဆိုရင် rename syscall ဖြစ်ပြီး executed ဆိုရင် execve syscall ဖြစ်ပါတယ်။ ဒီ mapping က အလွန် intuitive ဖြစ်တဲ့အတွက် analyst အနေနဲ့ syscall အမည်ကို တိုက်ရိုက် မသိဘဲ action ကို ကြည့်ပြီးတောင် ဘာဖြစ်သွားလဲဆိုတာကို ခန့်မှန်းနိုင်ပါတယ်။

Field Mapping

Field mapping အပိုင်းကို ကြည့်ရင် Auditbeat က analyst-friendly ဖြစ်အောင် data ကို ပြန်လည် စီစဉ်ပေးထားတာကို မြင်နိုင်ပါတယ်။ Process execution event တွေအတွက် procname ကို ProcessName အဖြစ် map လုပ်ထားပြီး EXECVE event ထဲက argument တွေကို ProcessArgs array အဖြစ် သိမ်းထားပါတယ်။ ဒါကြောင့် Windows process creation event တွေနဲ့ ဆင်တူတဲ့ ပုံစံနဲ့ Linux process execution ကို စစ်ဆေးနိုင်ပါတယ်။

ဥပမာအနေနဲ့ execve syscall ကြောင့် ဖြစ်တဲ့ process execution event အချို့ကို ကြည့်ချင်ရင် အောက်ပါ query ကို သုံးနိုင်ပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall == "execve"
| extend Args = strcat_array(ProcessArgs, " ")
| project Timestamp, ProcessName, ProcessPid, Args
| take 10
```

ဒီ query က auditd data ထဲက execve syscall ဖြစ်တဲ့ event တွေကို ရွေးထုတ်ပြီး ProcessArgs array ကို string အဖြစ် ပြောင်းပေးပါတယ်။ နောက်ဆုံးမှာ timestamp, process name, pid နဲ့ argument အပြည့်အစုံကို မြင်နိုင်ပါတယ်။ Windows event log နဲ့ ပုံစံတူတဲ့ output ရတဲ့အတွက် cross-platform hunting လုပ်တဲ့အခါ အရမ်းအဆင်ပြေပါတယ်။

Auditd ရဲ့ correlation ID ကို Auditbeat မှာ auditd.sequence ဆိုတဲ့ field အဖြစ် map လုပ်ထားပါတယ်။ ဒါက အရင်ရှင်းခဲ့တဲ့ correlation ID နဲ့ အတူတူပါပဲ။ Syscall event တစ်ခုနဲ့ ဆက်စပ်တဲ့ SYSCALL, EXECVE, CWD, PATH, PROCTITLE event တွေအားလုံးမှာ auditd.sequence တူနေပါတယ်။ SIEM ထဲမှာ suspicious activity တွေ့လိုက်ရင် “ဒီ event ကို host ပေါ်မှာ raw audit log အနေနဲ့ ပြန်ကြည့်ချင်တယ်” ဆိုတဲ့အခါ correlation ID ကို သုံးပြီး ausearch နဲ့ query လုပ်နိုင်ပါတယ်။

ဥပမာ correlation ID တစ်ခုကို အသုံးပြုပြီး host ပေါ်မှာ event အားလုံးကို ကြည့်ချင်ရင်

```
ausearch -i -a <correlation ID>
```

လို့ run လိုက်ရင် human-readable ပုံစံနဲ့ audit event တွေအားလုံးကို ပြန်မြင်နိုင်ပါတယ်။ SIEM level analysis နဲ့ host level forensic analysis ကို ဒီ correlation ID တစ်ခုတည်းနဲ့ ချိတ်ဆက်နိုင်တာက auditd နဲ့ Auditbeat ရဲ့ အားသာချက်ကြီးတစ်ခုပါ။

Beacon

The attacker downloaded a beacon from their C2 server. What IP address did they download the beacon from?

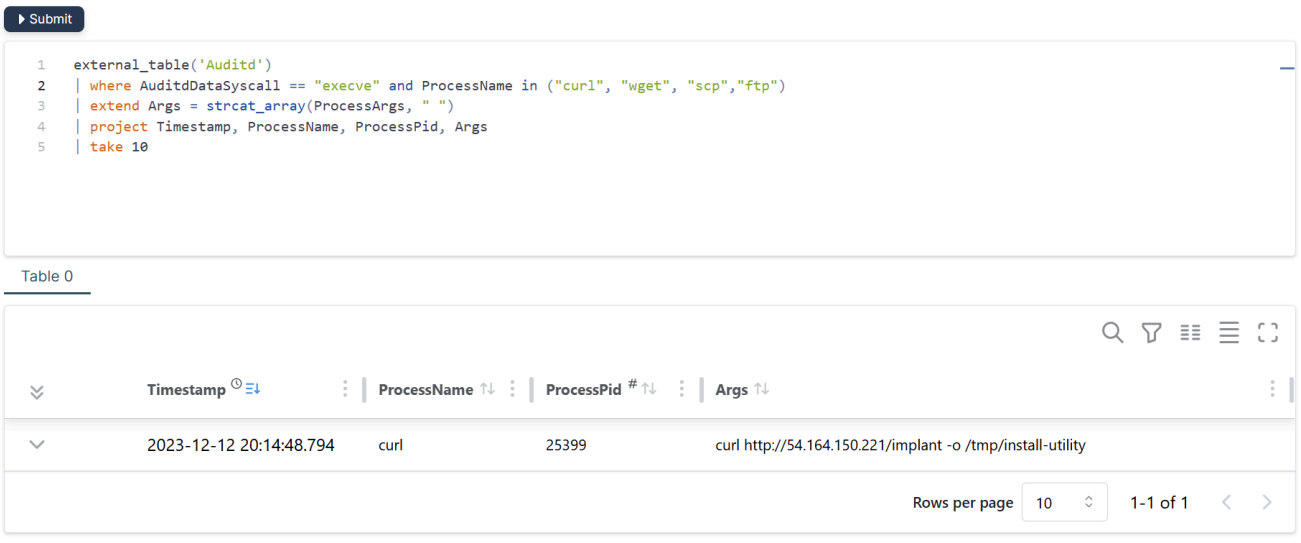
Hint: Think of some common Linux utilities that download files.

ဒီ scenario မှာ attacker က C2 server ကနေ **beacon ကို download** လုပ်ခဲ့တယ်လို့ ပြောထားပါတယ်။ Linux ပေါ်မှာ file download လုပ်တဲ့အခါ attacker အများဆုံး အသုံးပြုတဲ့ tool တွေက **curl, wget, scp, ftp** လို utilities တွေ ဖြစ်ပါတယ်။ ဒီလို tool တွေအားလုံးဟာ **command execution** ဖြစ်တဲ့အတွက် execve syscall မပါဘဲ မဖြစ်နိုင်ပါဘူး။ အဲ့ဒါကြောင့် auditd / Auditbeat data ထဲမှာ execve syscall ကို အခြေခံပြီး စစ်ဆေးရပါတယ်။

ပေးထားတဲ့ query က ဒီသဘောတရားကို မှန်မှန်ကန်ကန် အသုံးချထားပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall == "execve"
  and ProcessName in ("curl", "wget", "scp","ftp")
| extend Args = strcat_array(ProcessArgs, " ")
| project Timestamp, ProcessName, ProcessPid, Args
| take 10
```

ဒီ query ရဲ့ အဓိက ရည်ရွယ်ချက်က curl, wget စတဲ့ download utility တွေကို run လုပ်ထားတဲ့ event တွေကို ဆွဲထုတ်တာပါ။ အရေးကြီးဆုံး field က **Args** ဖြစ်ပါတယ်။ Linux မှာ IP address ကို auditd ရဲ့ network event ထဲမှာ မဟုတ်ဘဲ **command argument ထဲမှာပဲ အများအားဖြင့် တွေ့ရပါတယ်**။



Test : Initial Access

What IP did the attacker initially access Ubuntu from?

Hint: Look at other event types that might be helpful:

```
external_table('Auditd')
| summarize count() by tostring(EventCategory)
```

The results:

["file"]	40164
["process"]	3299
["configuration","network"]	2
["session"]	3
["authentication"]	10
["network"]	4
	1

You can access the events in a category with:

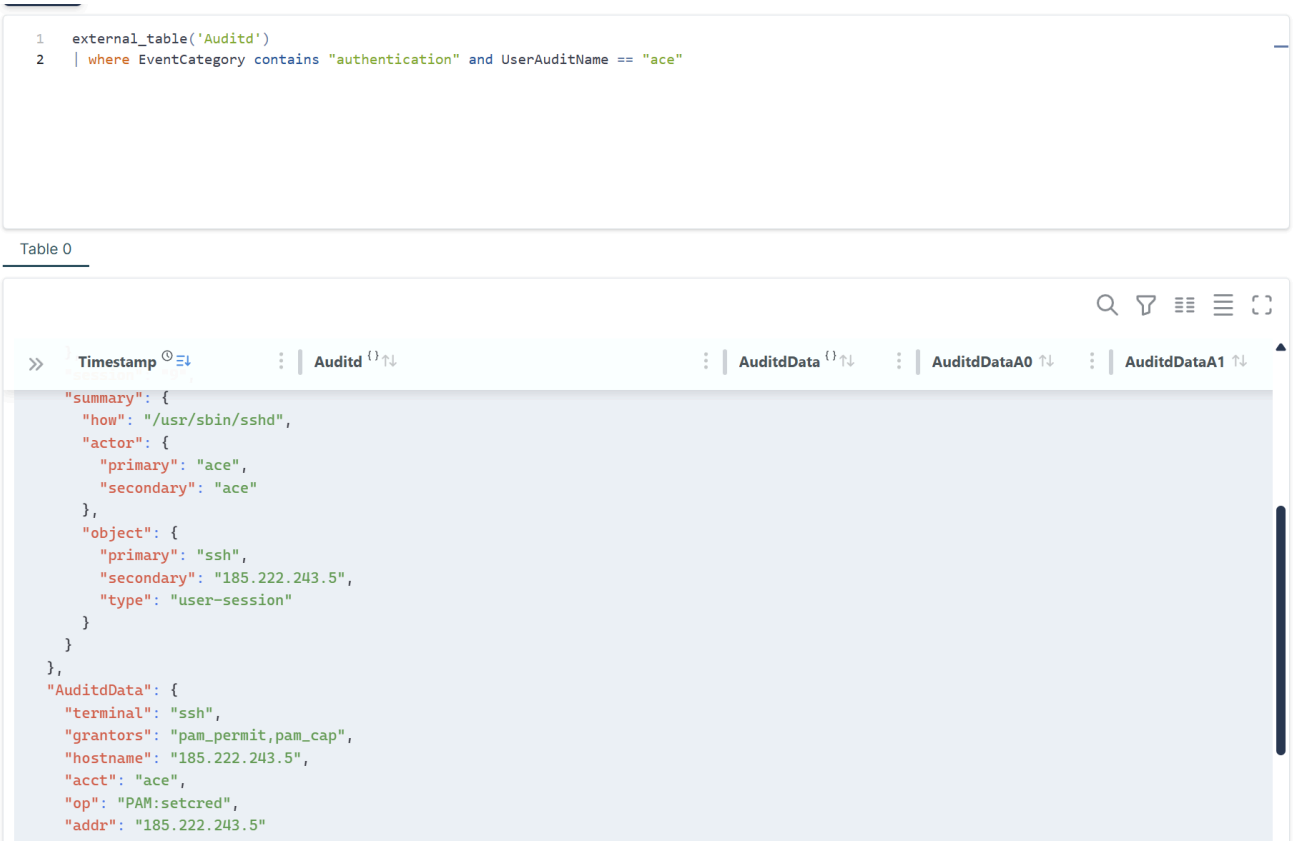
```
external_table('Auditd')
| where EventCategory contains "<category>"
```

You want to look for activity from the user that ran the command.

Initial Access ဆိုတဲ့အခါ **process event (execve)** ကို မကြည့်ဘဲ **authentication category** ကို အဓိကကြည့်ရပါတယ်။ ဘာကြောင့်လဲဆိုတော့ system ထဲကို ဝင်လာတဲ့အချိန်ဟာ login / authentication ဖြစ်တဲ့အချိန်ပါ။

```
external_table('Auditd')
| where EventCategory contains "authentication" and UserAuditName == "ace"
| project Timestamp, AuditdMessageType, UserAuditName, EventAction
```

IP address က **authentication event ရဲ့ detail field ထဲမှာပဲ ပါလာတာ** ဖြစ်ပါတယ်။ အထူးသဖြင့် cred_acq (credential acquisition) နဲ့ user_acct event တွေကို အသေးစိတ်ကြည့်ရင် `addr=` သို့မဟုတ် `src=` လို field ထဲမှာ originating IP address ပါလာတတ်ပါတယ်။ SSH ကနေ login ဝင်လာရင် auditd က remote IP ကို ဒီနေရာမှာ မှတ်တမ်းတင်ထားပါတယ်။



Test : Private Keys

From which user other than ace did the attacker steal an SSH private key from?

attacker က system ထဲဝင်ပြီးနောက် ace အပြင် အခြား user တွေရဲ့အရေးကြီးဖိုင်ကို စူးစမ်းပြီး ဖတ်ယူခဲ့သလားဆိုတာကို forensic နည်းလမ်းနဲ့ သက်သေပြရပါတယ်။

Linux မှာ SSH private key တွေက user တစ်ယောက်ချင်းစီရဲ့ home directory အောက်က `.ssh/id_rsa` ဆိုတဲ့ဖိုင်အဖြစ် သိမ်းထားတာများပါတယ်။ ဒီဖိုင်ကို ရသွားရင် attacker က အဲဒီ user အနေနဲ့ အခြား server တွေကို password မလိုဘဲ login ဝင်နိုင်သွားလို့ အရမ်းအန္တရာယ်ရှိပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall contains "execve" and UserAuditName == "ace" and
ProcessArgs contains "ssh"
| extend Args = strcat_array(ProcessArgs, " ")
| project Timestamp, ProcessName, Args
```

ဒီ query မှာ attacker ဖြစ်တဲ့ **ace user** က ssh နဲ့ ဆိုင်တဲ့ argument တွေပါတဲ့ command တွေ run ခဲ့သလားဆိုတာကို ရှာထားပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall contains "execve" and UserAuditName == "ace" and ProcessArgs contains "ssh"
| extend Args = strcat_array(ProcessArgs, " ")
| project Timestamp, ProcessName, Args
```

Table 0

Timestamp	ProcessName	Args
2023-12-12 20:25:56.451	cat	cat /home/myuser/.ssh/
2023-12-12 20:25:56.451	cat	cat /home/myuser/.ssh/id_rsa
2023-12-12 20:25:56.447	cat	cat /home/ace/.ssh/authorized_keys
2023-12-12 20:25:56.447	find	find /home/myuser/.ssh/ -not -iname *.pub
2023-12-12 20:25:56.443	cat	cat /home/ace/.ssh/id_rsa
2023-12-12 20:25:56.439	find	find /home/ace/.ssh/ -not -iname *.pub
2023-12-12 20:25:56.439	cat	cat /home/ace/.ssh/
2023-12-12 20:25:43.843	ls	ls --color=auto /home/myuser/.ssh
2023-12-12 20:25:43.839	ls	ls --color=auto /home/ace/.ssh

Rows per page 10 1-9 of 9

အရေးကြီးဆုံးတွေ့ရှိချက်က
attacker က `cat` command ကို သုံးပြီး

```
/home/myuser/.ssh/id_rsa
```

ကို ဖတ်ခဲ့တာပါ။

ဒီနေရာမှာ အရေးကြီးတဲ့ အချက်က ဖိုင်ရဲ့ path ထဲမှာ **myuser** ဆိုတဲ့ username ပါနေပါတယ်။
ဒါကြောင့် attacker က myuser ရဲ့ SSH private key ကို ခိုးခဲ့တာ ဆိုတာ သေချာသွားပါတယ်။ အချိန်ခဏလေးအတွင်းဖြစ်သွားတာမလို့ attacker က key တစ်ဖိုင်တည်း မခိုးဘဲ **script တစ်ခုနဲ့ user အားလုံးရဲ့ key တွေကို အစုလိုက် ဖတ်ဖို့ ကြိုးစားထားတာ** ဖြစ်နိုင်ပါတယ်။

```
for user in `ls /home`; do for file in `find /home/$user/.ssh/ -not -iname
*.pub`; do echo $file; cat $file; echo ;done;done
```

ဒါက automated credential harvesting လုပ်နေတဲ့ classic attacker behavior ဖြစ်ပါတယ်။
Auditd မှာတော့ bash one-liner အပြည့်အစုံ မမြင်ရပေမယ့်
execve syscall ထဲက argument အပိုင်းအစတွေကို ဆက်စပ်ကြည့်ရင်
ဒီလို loop လုပ်ထားတယ်ဆိုတာကို ခန့်မှန်းနိုင်ပါတယ်။

Tip

real incident တစ်ခုမှာ attacker ရဲ့ bash session ကို **PPID (Parent Process ID)** နဲ့ ဆက်စပ်ပြီး timeline ဆွဲရပါတယ်။ ဒီ scenario မှာတော့ အရင်ကတွေ့ထားတဲ့ curl command (beacon download)

ရဲ့ PPID ကို အခြေခံပြီး အဲ့ဒီ PPID နဲ့ ဆက်နွယ်တဲ့ process တွေကို filter လုပ်ရင် attacker က beacon ကို run ခဲ့တဲ့ install-utility လို indicator တွေကိုပါ တွေ့နိုင်ပါတယ်။

```
| where ProcessPpid in(25448,25393)
```

File System Rules

Auditd မှာ file system rules ဆိုတာက **"ဒီဖိုင်ကို ဘယ်သူ ဘာလုပ်လဲ"** ဆိုတာကို စောင့်ကြည့်ဖို့ သုံးတဲ့ rule အမျိုးအစားပါ။ ဥပမာအနေနဲ့ `/etc/passwd` ဆိုတဲ့ဖိုင်ကို စဉ်းစားကြည့်ရအောင်။ ဒီဖိုင်က Linux system မှာ user account အချက်အလက်တွေ ပါတဲ့ အရေးကြီးဖိုင်ဖြစ်လို့ attacker တွေ စူးစမ်းကြည့်ချင်တဲ့ file တစ်ခုဖြစ်ပါတယ်။

```
-w /etc/passwd -p wa -k etcpasswd
```

ဒီ rule မှာ `-w` ဆိုတာ watch လုပ်မယ်ဆိုတဲ့အဓိပ္ပါယ်ပါ။ `/etc/passwd` ကို watch လုပ်ထားတဲ့အတွက် ဘယ် process မဆို ဒီဖိုင်နဲ့ ပတ်သက်ပြီး တစ်ခုခုလုပ်လိုက်တာနဲ့ Auditd က log ထုတ်ပေးမှာဖြစ်ပါတယ်။ ဖိုင်တစ်ဖိုင်တည်းမက directory ကိုပါ watch လုပ်လို့ရပေမယ့် directory watch ဖြစ်ရင် file အသစ်ထည့်တာ၊ ဖျက်တာပဲ log ထွက်ပြီး ဖိုင်အတွင်းဘာလုပ်လဲဆိုတာကိုတော့ မသိနိုင်ပါဘူး။ အဲ့ဒါကြောင့် အရေးကြီးတဲ့ ဖိုင်တွေကို သီးသန့် rule နဲ့ watch လုပ်တာက ပိုကောင်းပါတယ်။

`-p` option ကတော့ **ဘယ်လို access မျိုးကို စောင့်ကြည့်မလဲ** ဆိုတာကို သတ်မှတ်ပေးတာပါ။ ဒီ rule မှာ `wa` ဆိုပြီး သုံးထားတာက write access နဲ့ attribute ပြောင်းလဲမှုတွေကို စောင့်ကြည့်မယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ Linux မှာ

- r - file ကို ဖတ်တာ၊
- w - ရေးတာ၊
- e - execute လုပ်တာ၊
- a - permission ပြောင်းတာ

တွေဟာ forensic အရ အရေးကြီးတဲ့ signal တွေဖြစ်ပါတယ်။

`-k etcpasswd` ဆိုတဲ့ key ကတော့ ဒီ rule ကနေ ထွက်လာတဲ့ event တွေကို နောက်ပိုင်း analysis လုပ်တဲ့အခါ အလွယ်တကူ ခွဲထုတ်နိုင်ဖို့ label တပ်ပေးထားတာပါ။ forensic လုပ်တဲ့အခါ rule key တွေက အလွန်အသုံးဝင်ပါတယ်။

Detection

အခု ace user က `/etc/passwd` ကို `cat` နဲ့ ဖတ်လိုက်တဲ့အခါ Auditd က event တွေ ထုတ်ပေးပါတယ်။ အရေးကြီးဆုံးက `type=SYSCALL` ဖြစ်ပါတယ်။ ဒီ event ထဲမှာ syscall အနေနဲ့ `openat` ဆိုတာကို မြင်ရပါတယ်။

> openat ဆိုတာ Linux မှာ ဖိုင်ဖွင့်တဲ့ syscall ဖြစ်ပြီး file system rule တွေမှာ အများဆုံးတွေ့ရပါတယ်။

```
type=SYSCALL msg=audit(1702334836.691:2616): arch=c000003e syscall=257
success=yes exit=3 a0=ffffff9c a1=7ffd4ff34776 a2=0 a3=0 items=1 ppid=2165
pid=9241 auid=1000 uid=1000 gid=1000 euid=1000 suid=1000 fsuid=1000
egid=1000 sgid=1000 fsgid=1000 tty=pts0 ses=1 comm="cat" exe="/usr/bin/cat"
subj=unconfined key="etcpasswd"ARCH=x86_64 SYSCALL=openat AUID="ace"
UID="ace" GID="ace" EUID="ace" SUID="ace" FSUID="ace" EGID="ace" SGID="ace"
FSGID="ace"

type=CWD msg=audit(1702334836.691:2616): cwd="/home/ace"

type=PATH msg=audit(1702334836.691:2616): item=0 name="/etc/passwd"
inode=48945 dev=08:01 mode=0100644 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL
cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0OUID="root" OGID="root"

type=PROCTITLE msg=audit(1702334836.691:2616):
proctitle=636174002F6574632F706173737764
```

ဒီ log ကိုကြည့်ရင် `comm="cat"` နဲ့ `exe="/usr/bin/cat"` ဆိုပြီး ဘယ် command က ဒီဖိုင်ကို access လုပ်တယ်ဆိုတာကို တိတိကျကျ သိနိုင်ပါတယ်။ user information တွေကြည့်ရင် ace user ကပဲ ဒီလုပ်ရပ်ကို လုပ်ခဲ့တာလည်း သေချာပါတယ်။

SYSCALL event တစ်ခုတည်းနဲ့ မပြီးသေးပါဘူး။ Auditd က auxiliary event တွေကိုပါ ထုတ်ပေးပါတယ်။ `CWD` event က command ကို ဘယ် directory ကနေ run လဲဆိုတာ ပြပါတယ်။ ဒီမှာ `/home/ace` ဆိုတာကြောင့် ace ရဲ့ home directory ကနေ command run ခဲ့တာဆိုတာကို သိရပါတယ်။

`PATH` event ကတော့ access လုပ်ခဲ့တဲ့ ဖိုင်ရဲ့ full path ကို ပြပါတယ်။ ဒီနေရာမှာ `/etc/passwd` ကို access လုပ်ထားတာကို တိတိကျကျ မြင်ရပါတယ်။ owner က root ဖြစ်ပြီး permission က 644 ဖြစ်တယ်ဆိုတာကိုပါ forensic အနေနဲ့ သိနိုင်ပါတယ်။

`PROCTITLE` event ကတော့ command အပြည့်အစုံကို hex encoding နဲ့ သိမ်းထားတာပါ။ decode လုပ်လိုက်ရင် `cat /etc/passwd` ဆိုတာကို တိုက်ရိုက်မြင်နိုင်ပါတယ်။

ဒီ log တွေကို raw format နဲ့ ဖတ်ရတာ ခက်လို့

```
ausearch -i -a 2616
```

ကို သုံးပါတယ်။ correlation ID ဖြစ်တဲ့ 2616 ကို သုံးပြီး query လုပ်လိုက်ရင် event အားလုံးကို လူဖတ်လို့ရတဲ့ format နဲ့ ပြန်ပေးပါတယ်။ ဒီမှာ syscall=openat, O_RDONLY ဆိုတဲ့ flag တွေကို မြင်ရပါတယ်။

```
type=SYSCALL msg=audit(12/11/23 22:47:16.691:2616) : arch=x86_64
syscall=openat success=yes exit=3 a0=AT_FDCWD a1=0x7ffd4ff34776 a2=O_RDONLY
a3=0x0 items=1 ppid=2165 pid=9241 auid=ace uid=ace gid=ace euid=ace suid=ace
fsuid=ace egid=ace sgid=ace fsgid=ace tty=pts0 ses=1 comm=cat
exe=/usr/bin/cat subj=unconfined key=etcpasswd
```

Auditd log ထဲမှာ `a0`, `a1`, `a2` လို field တွေကို မြင်ရတာက ဒီ syscall ကို ခေါ်တဲ့အချိန် ပေးခဲ့တဲ့ argument တွေကို ကိုယ်စားပြုနေတာပါ။

ဒီဥပမာမှာ `a0` က `AT_FDCWD` ဖြစ်ပါတယ်။ ဒါက special file descriptor တစ်ခုဖြစ်ပြီး "လက်ရှိ working directory ကို အခြေခံပြီး ဖိုင်ကို ရှာပါ" ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ဆိုလိုတာက cat command ကို run ခဲ့တဲ့ directory ကို အခြေခံပြီး `/etc/passwd` ကို ဖွင့်လိုက်တာဖြစ်ပါတယ်။

`a1` ကတော့ ဖိုင် path ရဲ့ memory address ကို ပြထားတာပါ။ ဒါက kernel အတွင်းပိုင်းအတွက် အသုံးဝင်ပေမယ့် analyst အနေနဲ့တော့ ဒီ address တန်ဖိုးကို တိုက်ရိုက်ကြည့်ပြီး ဘာမှ အထူးသိနိုင်တာ မရှိပါဘူး။ ဖိုင် path အမှန်ကိုတော့ PATH event ထဲမှာ သီးသန့် ရရှိပါတယ်။

`a2` ကတော့ အရေးကြီးဆုံး argument ဖြစ်ပါတယ်။ ဒီမှာ `O_RDONLY` ဆိုတဲ့ flag ကို သုံးထားတာကို မြင်ရပါတယ်။ ဒါရဲ့ အဓိပ္ပါယ်က **"ဖိုင်ကို ဖတ်ချင်တယ်၊ မရေးဘူး၊ မပြောင်းဘူး"** ဆိုတာပါ။ ဒီအချက်ကြောင့် forensic analyst အနေနဲ့ cat command က `/etc/passwd` ကို ဖတ်ရုံသာ လုပ်ပြီး modify မလုပ်ထားဘူးဆိုတာကို ယုံကြည်နိုင်ပါတယ်။

Conclusion

ဒီလို syscall argument တွေကို နားလည်ထားရင် အကျိုးကျေးဇူးကြီးပါတယ်။ ဥပမာအားဖြင့် `/etc/passwd` ကို access လုပ်တာကို မြင်လိုက်ရုံနဲ့ မကောင်းဘူးလို့ ချက်ချင်း ဆုံးဖြတ်လို့ မရပါဘူး။ ဖတ်နေတာလား၊ ရေးနေတာလား၊ overwrite လုပ်နေတာလားဆိုတာကို argument တွေကြည့်မှသာ သိနိုင်ပါတယ်။ attacker ဖြစ်ရင် `O_WRONLY` ဒါမှမဟုတ် `O_RDWR` လို flag တွေကို သုံးနိုင်ပြီး အဲ့ဒီအခါမှာ alert ဖြစ်သင့်ပါတယ်။

တချို့ incident တွေမှာ log တစ်ခုက သံသယဖြစ်ပေမယ့် syscall argument ကို နားလည်ပြီး ပြန်ကြည့်လိုက်တဲ့အခါ legitimate behavior ဖြစ်နေတာကိုလည်း တွေ့ရနိုင်ပါတယ်။ ဒါကြောင့် argument တွေကို သိထားတာက false positive လျော့စေနိုင်သလို true attack ကိုလည်း ပိုတိကျစွာ ခွဲခြားနိုင်ပါတယ်။

အဲ့ဒီအပြင် syscall argument အချက်အလက်တွေကို audit rule design မှာပါ အသုံးချနိုင်ပါတယ်။ ဥပမာအားဖြင့် read-only access တွေကို exclude လုပ်ပြီး write access ဖြစ်မှ log ထုတ်မယ်ဆိုတဲ့ rule တွေကို ဆောက်နိုင်ပါတယ်။ ဒါက performance ကောင်းစေပြီး noise ကိုလည်း လျော့စေပါတယ်။

တကယ်လို့ syscall တစ်ခုရဲ့ argument တွေကို မသေချာဘူး၊ နားမလည်ဘူးဆိုရင် အကောင်းဆုံးနည်းလမ်းက **syscall ရဲ့ man page ကို ဖတ်ခြင်း** ဖြစ်ပါတယ်။ `man openat` လို့ ရိုက်လိုက်ရင် openat syscall က ဘယ် argument ကို ဘယ်အဓိပ္ပါယ်နဲ့ သုံးလဲဆိုတာကို တိတိကျကျ ဖော်ပြထားပါတယ်။ အဲ့ဒီ documentation ကို ausearch က ပြတဲ့ `a0`, `a1`, `a2` နဲ့ တိုက်စပ်ပြီး ကြည့်လိုက်ရင် syscall behavior ကို နက်နက်ရှိုင်းရှိုင်း နားလည်နိုင်ပါတယ်။

Test : Credential Access

Which file did the attacker steal user hashes from?

Linux system မှာ user တွေရဲ့ password hash တွေကို တိုက်ရိုက် `/etc/passwd` မှာ မသိမ်းတော့ပါဘူး။ လုံခြုံရေးအတွက် `/etc/shadow` ဆိုတဲ့ ဖိုင်ထဲမှာ သိမ်းထားပါတယ်။ ဒီဖိုင်ကို root user သာ ဖတ်နိုင်

အောင် permission ကန့်သတ်ထားတဲ့အတွက် attacker တစ်ယောက်က ဒီဖိုင်ကို ရနိုင်ပြီဆိုရင် **credential access အဆင့်ကို ရောက်သွားပြီ** လို့ ဆိုနိုင်ပါတယ်။

အခုဒီ incident မှာ attacker ဖြစ်တဲ့ **ace user** က system ထဲမှာ compromise ဖြစ်ထားပြီး audit logs ထဲမှာ ဘယ်ဖိုင်တွေကို access လုပ်ခဲ့လဲဆိုတာကို စစ်ကြည့်ရပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall == "openat" and UserAuditName == "ace"
| summarize count() by FilePath
```

အရင်ဆုံး auditd watch rule တွေကို အသုံးပြုပြီး ace user က **ဘယ်ဖိုင်တွေကို openat syscall နဲ့ ဖွင့်ခဲ့လဲ** ဆိုတာကို စုစည်းကြည့်ပါတယ်။ openat ဆိုတာ ဖိုင်ကို ဖွင့်တဲ့ syscall ဖြစ်လို့ file access ကို သိနိုင်ပါတယ်။ audit logs ကို summarize လုပ်လိုက်တဲ့အခါ ace user က access လုပ်ခဲ့တဲ့ file path တွေကို မြင်ရပါတယ်။

```
1  external_table('Auditd')
2  | where AuditdDataSyscall in ("openat","execve") and UserAuditName == "ace" and FilePath == "/etc/shadow" or ProcessArgs contains "/etc/shadow"
3  | extend Args = strcat_array(ProcessArgs, " ")
4  | project Timestamp, AuditdDataSyscall, Args
```

Table 0

Timestamp	AuditdDataSyscall	Args
2023-12-12 20:28:19.191	openat	
2023-12-12 20:28:19.191	execve	cat /etc/shadow
2023-12-12 20:14:42.058	openat	

Audit results ကို ကြည့်လိုက်တဲ့အခါ ace user က **/etc/shadow** ကို openat syscall နဲ့ access လုပ်ထားတာကို တွေ့ရပါတယ်။ ဒီအချက်တစ်ခုတည်းနဲ့ပဲ attacker က user hashes တွေကို ခိုးယူဖို့ ကြိုးစားထားတာဆိုတာကို ခန့်မှန်းနိုင်ပါတယ်။ ဘာကြောင့်လဲဆိုတော့ `/etc/shadow` ထဲမှာ password hash တွေကို သိမ်းထားတာဖြစ်ပြီး legitimate user တစ်ယောက်က နေ့စဉ်အသုံးပြုဖို့ မလိုအပ်တဲ့ ဖိုင်ဖြစ်ပါတယ်။

နောက်ထပ် အရေးကြီးတဲ့အချက်က ဒီ action က **watch rule (openat)** နဲ့ **syscall rule (execve)** နှစ်မျိုးလုံးကို trigger လုပ်ထားတာပါ။ Syscall rule မှာ root user အတွက် execve ကို log လုပ်ထားတဲ့ rule ရှိတဲ့အတွက် attacker က `cat` command ကို သုံးပြီး `/etc/shadow` ကို ဖတ်ခဲ့တာကို တိတိကျကျ မြင်နိုင်ပါတယ်။

Test : Persistence

What file did the attacker modify for persistence?

Cyber attack lifecycle မှာ **Persistence** ဆိုတာက attacker က system ကို တစ်ခါ compromise လုပ်ပြီးရင် reboot ဖြစ်သွားတောင်၊ user logout လုပ်သွားတောင် သူ့ malicious code ကို ပြန်လည် run နိုင်

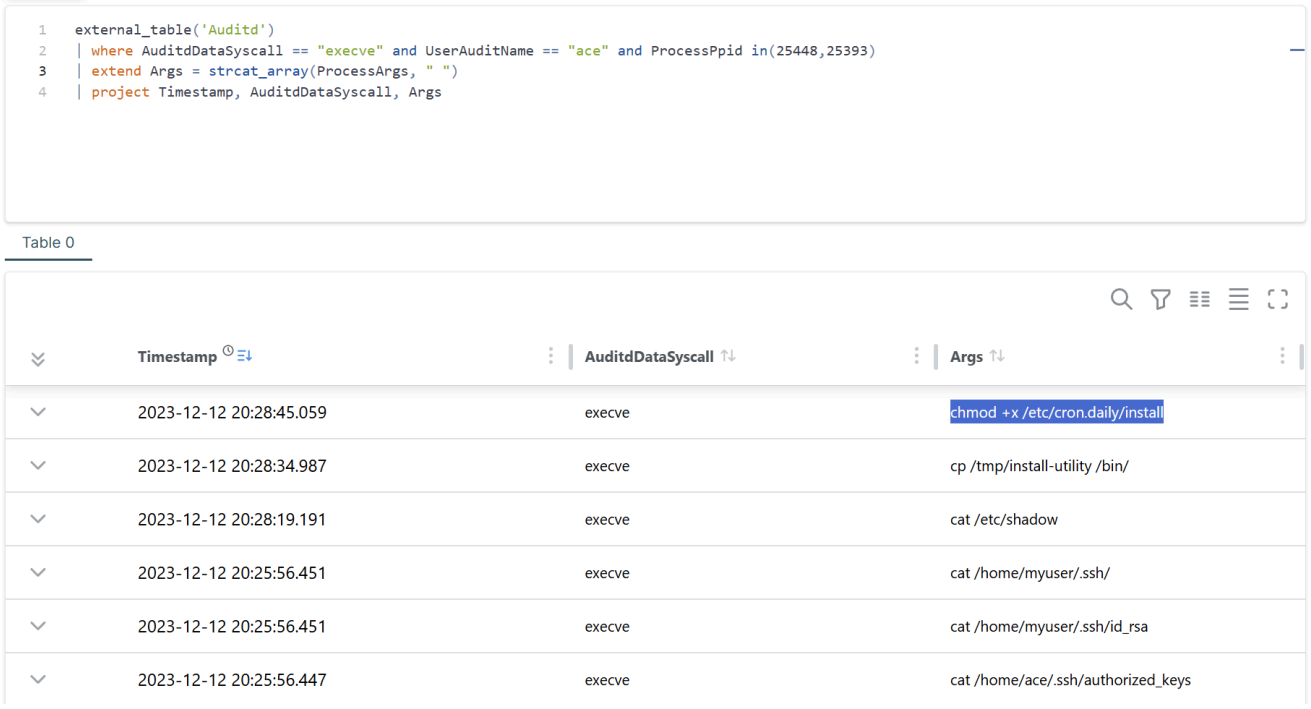
အောင် စနစ်တကျ ပြင်ဆင်ထားခြင်းကို ဆိုလိုပါတယ်။ Linux မှာ persistence ရဖို့ အသုံးများတဲ့နည်းလမ်းတွေထဲက တစ်ခုက **cron job** တွေကို အသုံးချတာပါ။

cron ဆိုတာ Linux မှာ အချိန်ဇယားနဲ့ command တွေကို အလိုအလျောက် run ပေးတဲ့ service ဖြစ်ပါတယ်။ အထူးသဖြင့် `/etc/cron.daily` directory ထဲမှာ ထည့်ထားတဲ့ script တွေက နေ့တိုင်း အလိုအလျောက် run ပါတယ်။ ဒါကြောင့် attacker တစ်ယောက်အနေနဲ့ ဒီနေရာထဲကို malicious file တစ်ခု ထည့်နိုင်ပြီဆိုရင် persistence ရပြီးသား ဖြစ်သွားပါတယ်။

ဒီ incident မှာ forensic analysis လုပ်တဲ့အခါ အရင်က သေချာထားပြီးသား **bash timeline** ကို အသုံးချပါတယ်။ အဲဒီ timeline က attacker ရဲ့ bash session နဲ့ ဆက်နွယ်နေတဲ့ PPID တွေဖြစ်တဲ့ 25448 နဲ့ 25393 ကို အခြေခံထားတာပါ။ အဲဒီ PPID တွေကနေ run ထွက်လာတဲ့ command တွေကို စုစည်းကြည့်လိုက်ရင် attacker ရဲ့ လှုပ်ရှားမှုတွေကို အစဉ်လိုက် မြင်နိုင်ပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall == "execve" and UserAuditName == "ace" and
ProcessPpid in(25448,25393)
| extend Args = strcat_array(ProcessArgs, " ")
| project Timestamp, AuditdDataSyscall, Args
```

အဲဒီ result တွေကို ကြည့်လိုက်တဲ့အခါ attacker က **cron.daily directory ထဲက ဖိုင်တစ်ခုကို ပြင်ဆင်ထားတာ** ကို တွေ့ရပါတယ်။ ဒီအချက်က အလွန်အရေးကြီးပါတယ်။ cron.daily ထဲက ဖိုင်ကို modify လုပ်တယ်ဆိုတာက system က နေ့တိုင်း အလိုအလျောက် run ပေးမယ့် code ထဲကို attacker ရဲ့ instruction ကို ထည့်လိုက်တယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။



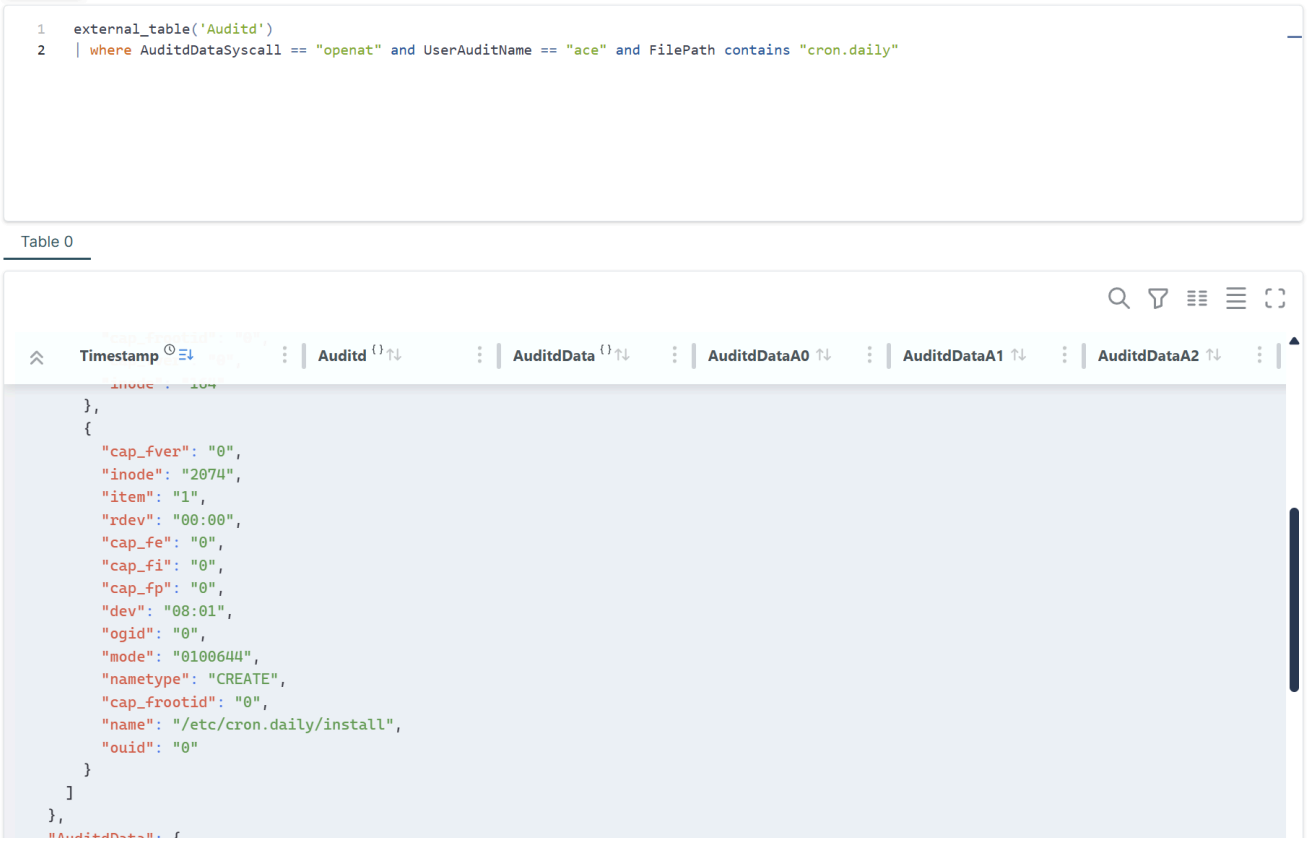
Test : File System Event

Did the attacker create `/etc/cron.daily/install` or just modify it?

Auditd watch rule ကနေ ထွက်လာတဲ့ event ကို ကြည့်ရင်
`openat` syscall နဲ့ cron.daily directory ထဲမှာ file access လုပ်ထားတာကို တွေ့ရပြီး
အဲ့ဒီ event က `chmod +x` မလုပ်ခင် အချိန်မှာ ဖြစ်ပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall == "openat" and UserAuditName == "ace" and
FilePath contains "cron.daily"
```

PATH event ထဲက data ကို ကြည့်တဲ့အခါ
file ရဲ့ nametype နဲ့ inode အချက်အလက်တွေက
ရှိပြီးသားဖိုင်ကို ပြင်တာမဟုတ်ဘဲ **file အသစ် create လုပ်ထားတာ** ကို အတည်ပြုပေးပါတယ်။



ဒါကြောင့် အဖြေက **attacker က `/etc/cron.daily/install` ကို အသစ် create လုပ်ခဲ့တာပါ။**

Beacon Frequency

Which of the following best describes the attacker's beacon interval (the amount of time that passes between check-ins)?

Hint: Look at socket syscalls.

ဒီ **Beacon Frequency** မေးခွန်းရဲ့ အဓိကအကြောင်းအရာက **attacker ရဲ့ beacon (C2 ကို check-in လုပ်တာ) က ဘယ်လောက်အချိန်ခြားပြီး တစ်ခါလုပ်လဲ** ဆိုတာကို သိချင်တာပါ။

```
external_table('Auditd')
| where AuditdDataSyscall == "socket" and UserAuditName == "ace" and
ProcessName == "install-utility"
```

Auditd မှာ beacon behavior ကို ကြည့်ဖို့ အကောင်းဆုံး syscall က **socket** ဖြစ်ပါတယ်။ beacon တစ်ခါလုပ်တိုင်း C2 server နဲ့ ဆက်သွယ်ရတာကြောင့် socket syscall တစ်ခုခု မဖြစ်မနေ ထွက်ပါတယ်။ ဒီ incident မှာ beacon ကို run နေတာက `install-utility` process ဖြစ်ပြီး ace user နဲ့ run နေတာကို အခြေခံပြီး filter လုပ်ပါတယ်။

```
external_table('Auditd')
| where AuditdDataSyscall == "socket" and UserAuditName == "ace" and
ProcessName == "install-utility"
| summarize count() by bin(Timestamp,1m)
| render columnchart
```

socket syscall event တွေကို အချိန်လိုက် စုစည်းကြည့်လိုက်တဲ့အခါ
event တွေက **တစ်မိနစ်ခြား တစ်ကြိမ်** လောက် ထွက်နေတာကို မြင်ရပါတယ်။ bar chart နဲ့ ကြည့်လိုက်ရင်လည်း minute တစ်ခုချင်းစီမှာ socket event တစ်ခုစီ စနစ်တကျ ထွက်နေတဲ့ pattern ကို တွေ့နိုင်ပါတယ်။

ဒါကြောင့် ဒီ attacker ရဲ့ beacon interval ကို ဖော်ပြရင်
တစ်မိနစ်တစ်ကြိမ် (approximately every 60 seconds) C2 ကို check-in လုပ်နေပါတယ် ဆိုတာက အမှန်ဆုံးဖြစ်ပါတယ်။
