Access Token Manipulation

Windows operating system မှာ လုံခြုံရေးကို ထိန်းချုပ်ဖို့ အရေးအကြီးဆုံး အယူအဆတွေထဲက တစ်ခု က Access Token ဖြစ်ပါတယ်။ User တစ်ယောက် login ဝင်လိုက်တာနဲ့ Windows က “ဒီ user က ဘယ် သူလဲ၊ ဘာလုပ်ခွင့်ရှိလဲ” ဆိုတဲ့ အချက်အလက်အားလုံးကို Access Token တစ်ခုထဲမှာ စုထားပါတယ်။ Process တစ်ခု၊ Application တစ်ခုက file တစ်ခုကို ဖွင့်ချင်တာ၊ registry ကို ပြင်ချင်တာ၊ network resource တစ်ခုကို သုံးချင်တာလို အလုပ်တိုင်းမှာ Windows က “ဒီ token နဲ့ ဒီအလုပ်လုပ်ခွင့်ရှိလား” ဆို ပြီး စစ်ဆေးပါတယ်။ အဲဒီလိုနဲ့ Access Token က Windows security model ရဲ့ အခြေခံအုတ်မြစ် ဖြစ်လာပါတယ်။

Access Token Manipulation ဆိုတာက ဒီ Access Token ကို လိမ်လည်အသုံးချခြင်း၊ ပြောင်းလဲအသုံးချ ခြင်းကို ဆိုလိုပါတယ်။ Attacker တစ်ယောက်အနေနဲ့ token ကို manipulate လုပ်နိုင်ရင် ကိုယ်မရှိသင့်တဲ့ privilege ကို ရယူနိုင်ပြီး administrator ဖြစ်သွားနိုင်ပါတယ်။ ဒါ့အပြင် အခြား computer တွေကို ဝင် ရောက်ကြည့်ရှုနိုင်ပြီး network အတွင်းမှာ တစ်စက်စက် ပြန့်သွားနိုင်ပါတယ်။ ဒါကြောင့် Access Token Manipulation ကို နားလည်ထားခြင်းက threat hunting, detection engineering နဲ့ incident response အတွက် အလွန်အရေးကြီးပါတယ်။

ဒီ module ရဲ့ အဓိကရည်ရွယ်ချက်က Access Token Manipulation ကို စစ်ဆေးလေ့လာတဲ့ အခြေခံ သဘောတရားတွေကို နားလည်အောင်လုပ်ပေးတာ ဖြစ်ပါတယ်။ အရင်ဆုံး Access Token ကို စုံစမ်း စစ်ဆေးရင် ဘာတွေကို ကြည့်ရမလဲ၊ SIEM ထဲက event တွေကို ဘယ်လို ချိတ်ဆက်စဉ်းစားရမလဲဆိုတာ ကို လေ့လာပါမယ်။ အဲဒီနောက် Access Token နဲ့ ပတ်သက်တဲ့ အရေးကြီးတဲ့ concept တွေကို တစ်ခုချင်း စီ ချိတ်ဆက်ပြီး ဆက်လက်ရှင်းပြပါမယ်။

Access Token Foundation

Windows Operating System မှာ **Access Token** ဆိုတာက လုံခြုံရေးစနစ်ရဲ့ အခြေခံအုတ်မြစ်တစ်ခု လိုပါပဲ။ User တစ်ယောက်၊ Process တစ်ခု၊ Thread တစ်ခုက system ထဲက resource တစ်ခုကို သုံးခွင့် ရှိမရှိ ဆုံးဖြတ်ပေးတာဟာ Access Token ပေါ်မူတည်ပါတယ်။ ဥပမာအားဖြင့် ဖိုင်တစ်ခုဖတ်လို့ရမလား၊ Registry ကိုပြင်လို့ရမလား၊ Administrator လုပ်ဆောင်ချက်တွေ လုပ်လို့ရမလား ဆိုတာတွေကို Windows က Access Token ကိုကြည့်ပြီးဆုံးဖြတ်ပါတယ်။

Access Token ထဲမှာ User ရဲ့ Security Identifier (SID)၊ Privileges (SeDebugPrivilege, SeImpersonatePrivilege လိုမျိုး)၊ Process Integrity Level (Low, Medium, High, System) စတဲ့ လုံခြုံရေးအချက်အလက်တွေ အများကြီးပါဝင်ပါတယ်။ ဒီအချက်အလက်တွေကို Process ကို ဖန်တီးတဲ့ အချိန်မှာ OS က သတ်မှတ်ပေးထားပြီး၊ အချို့အခြေအနေတွေမှာ အခွင့်အရေးရှိတဲ့ Process တစ်ခုက Token ကို ပြင်ဆင်နိုင်ပါတယ်။ ဒီအချက်ကပဲ Cybersecurity မှာ Token Manipulation, Privilege Escalation ဆိုတဲ့ တိုက်ခိုက်မှုတွေ ဖြစ်လာရတဲ့ အကြောင်းရင်းပါ။

Access Token အမျိုးအစား အဓိက နှစ်မျိုးရှိပါတယ်။ ပထမတစ်မျိုးက **Primary Token** ဖြစ်ပါတယ်။ User တစ်ယောက် Windows ထဲ Login ဝင်တဲ့အချိန်မှာ Primary Token တစ်ခုကို ဖန်တီးပါတယ်။ User က Program တစ်ခုကို Run လုပ်တဲ့အခါ Process အသစ်တိုင်းမှာ ဒီ Primary Token ကို Copy လုပ်ပြီး သုံးပါတယ်။ ပုံမှန်အခြေအနေမှာ Process ထဲက Thread တွေအားလုံးဟာ Process ရဲ့ Primary Token

ကို အမွေဆက်ခံပြီး Resource တွေကို Access လုပ်ပါတယ်။ ဒါကြောင့် Primary Token ကို Process ရဲ့ “အခြေခံ Identity” လို့ နားလည်နိုင်ပါတယ်။

ဒုတိယတစ်မျိုးက **Impersonation Token** ဖြစ်ပါတယ်။ ဒီ Token က Process တစ်ခုက User တစ်ယောက်ရဲ့ လုံခြုံရေးအနေအထားကို ခဏတာ ယူသုံးချင်တဲ့အခါ အသုံးပြုပါတယ်။ ဥပမာ File Server တစ်ခုကို စဉ်းစားကြည့်ပါ။ Client User တစ်ယောက်က ဖိုင်တစ်ခုကို Request လုပ်တဲ့အခါ Server က “ဒီ User မှာ ဒီဖိုင်ကို ဖတ်ခွင့်ရှိလား” ဆိုတာကို စစ်ဖို့ Client ရဲ့ Security Context ကို ခဏတာ ယူသုံးရပါတယ်။ ဒီအချိန်မှာ Server Process ထဲက Thread က Impersonation Token ကို သုံးပြီး Client အဖြစ် လုပ်ဆောင်ပါတယ်။ Impersonation Token ကို Thread တွေမှာ Assign လုပ်ပြီး လိုအပ်တဲ့အချိန်မှာသာ အသုံးပြုပါတယ်။

Primary Token နဲ့ Impersonation Token တို့ရဲ့ အဓိက ကွာခြားချက်က Primary Token က Process တစ်ခုလုံးရဲ့ အခြေခံ Security Context ဖြစ်ပြီး Logon Session နဲ့ တိုက်ရိုက်ဆက်နွယ်ပါတယ်။ Impersonation Token ကတော့ Thread အဆင့်မှာ အသုံးပြုတာဖြစ်ပြီး Logon Session နဲ့ မဖြစ်မနေ ဆက်နွယ်စရာ မလိုပါဘူး။ Thread တစ်ခုက လိုအပ်တဲ့အချိန်မှာ User အဖြစ် “သရုပ်ဆောင်” ဖို့ သုံးတာပါ။

ဒီနေရာမှာ **Security Levels** ဆိုတဲ့ အယူအဆကို နားလည်ဖို့ လိုလာပါတယ်။ Impersonation Token မှာ Security Level ဆိုတာ ပါဝင်ပါတယ်။ အရေးကြီးဆုံးနှစ်ခုက SecurityImpersonation နဲ့ SecurityDelegation ပါ။ SecurityImpersonation ဆိုတာ Local System အတွင်းမှာပဲ User အဖြစ် လုပ်ဆောင်ခွင့်ရှိတာပါ။ ဒါကို Network Logon (Type 3) နဲ့ ဆက်စပ်စဉ်းစားနိုင်ပါတယ်။ အဲဒီအခြေအနေမှာ Process က Client အဖြစ် Local Machine ပေါ်မှာပဲ Resource တွေကို Access လုပ်နိုင်ပါတယ်။

SecurityDelegation ကတော့ ပိုမို အန္တရာယ်ကြီးတဲ့ Level ဖြစ်ပါတယ်။ ဒီ Level ရှိတဲ့ Token ကို သုံးရင် Client User အဖြစ် Remote System တွေထိပါ Access လုပ်နိုင်ပါတယ်။ ဒီအရာက Interactive Logon (Type 2) နဲ့ ဆက်နွယ်ပြီး Kerberos Authentication နဲ့ အလွန်သက်ဆိုင်ပါတယ်။ ဥပမာ SharePoint Server တစ်ခုက Database Server တစ်ခုကို Client User အနေနဲ့ Access လုပ်ချင်ရင် Impersonation Token တစ်ခုတည်းနဲ့ မလုံလောက်ပါဘူး။ Delegation Token လိုအပ်ပါတယ်။

ဒီ Delegation Concept ကြောင့်ပဲ **Kerberos Delegation** ဆိုတာ အရေးကြီးလာပါတယ်။ Kerberos Delegation ဆိုတာ Server တစ်ခုကို “ဒီ User အစား Remote System တွေကို သွားလုပ်ခွင့်ပေးပါ” လို့ Domain Controller က ခွင့်ပြုပေးတဲ့ Mechanism ဖြစ်ပါတယ်။ Delegation Token တွေဟာ အမှန်တကယ် User ရဲ့ Credentials နဲ့ Backed ဖြစ်နေတာကြောင့် Attacker အတွက် အရမ်းတန်ဖိုးရှိပါတယ်။ Delegation လုပ်ခွင့်ရှိတဲ့ System တစ်ခုကို ထိန်းချုပ်နိုင်ရင် Domain Privilege Escalation နဲ့ Lateral Movement လုပ်ဖို့ အခွင့်အလမ်းအရမ်းကြီးပါတယ်။

Cybersecurity အမြင်နဲ့ ကြည့်ရင် Impersonation Token နဲ့ Delegation Token ကို သုံးတဲ့ တိုက်ခိုက်မှုတွေဟာ ရည်ရွယ်ချက်မတူပါဘူး။ Impersonation Token တွေကို အများအားဖြင့် Local Privilege Escalation အတွက် သုံးပါတယ်။ ဥပမာ SeImpersonatePrivilege ရှိတဲ့ Process တစ်ခုက SYSTEM Token ကို Impersonate လုပ်နိုင်ရင် User ကနေ SYSTEM အထိ တက်နိုင်ပါတယ်။ Delegation Token တွေကိုတော့ Domain Environment ထဲမှာ အခြား Server တွေကို ဆက်တိုက် ထိန်းချုပ်ဖို့ အသုံးပြုကြပါတယ်။

အကျဉ်းချုပ်ပြောရရင် Access Token ဆိုတာ Windows Security ရဲ့ “အထောက်အထားစာအုပ်” လိုပါပဲ။ ဘယ်သူလဲ၊ ဘာလုပ်ခွင့်ရှိလဲ၊ ဘယ်နေရာထိ သွားနိုင်လဲ ဆိုတာအားလုံးကို Token က ဆုံးဖြတ်ပါတယ်။ ဒီ Token တွေကို နားလည်လာတာနဲ့ Windows Privilege Escalation, Lateral Movement, Domain

Attacks တွေ ဘာကြောင့် ဖြစ်လာတယ်ဆိုတာကို အခြေခံအဆင့်ကနေ စပြီး သဘောပေါက်လာနိုင်ပါလိမ့်မယ်။

Double Hop Problem

Double Hop Problem ဆိုတာ Windows Network Authentication မှာ အရမ်းမကြာခဏ တွေ့ရတဲ့ ပြဿနာတစ်ခုဖြစ်ပြီး Credential တွေကို Network တစ်ဆင့်ထက်ပိုပြီး သုံးချင်တဲ့အချိန်မှာ ဖြစ်ပေါ်လာပါတယ်။ အထူးသဖြင့် Administrator တွေ၊ Red Teamers တွေ Remote Command Execution နဲ့ Lateral Movement လုပ်တဲ့အခါ ဒီပြဿနာကို မဖြစ်မနေ ကြုံရတတ်ပါတယ်။

ဥပမာအနေနဲ့ Bob ဆိုတဲ့ Administrator တစ်ယောက်ရှိတယ်လို့ စဉ်းစားကြည့်ပါ။ Bob က Charlie ရဲ့ Computer ထဲမှာရှိတဲ့ ဖိုင်တွေကို ကြည့်ချင်ပါတယ်။ ဒါပေမယ့် Network Segmentation ကြောင့် Bob ရဲ့ Computer ကနေ Charlie ရဲ့ Computer ကို တိုက်ရိုက် ချိတ်ဆက်လို့မရပါဘူး။ အဲဒီအတွက် Bob က Alice ရဲ့ Computer ကို အလယ်ခံအနေနဲ့ သုံးချင်ပါတယ်။ ဒီအခြေအနေမှာ Bob ရဲ့ Computer က ပထမဆုံး hop ဖြစ်ပြီး Alice ရဲ့ Computer က ဒုတိယ hop၊ Charlie ရဲ့ Computer က တတိယ hop (Target) ဖြစ်သွားပါတယ်။

Bob က PowerShell Remoting ကို သုံးပြီး Alice ရဲ့ Computer ပေါ်မှာ Command တစ်ခု Run လုပ်ပါတယ်။

```
icm -ComputerName alice-pc -Credential $cred -ScriptBlock {
    ls \\charlie-pc\C$\Users\Charlie\Documents
}
```

ဒီ Command ရဲ့ အဓိကရည်ရွယ်ချက်က Alice ရဲ့ Computer ပေါ်ကနေ Charlie ရဲ့ Computer ထဲရှိ File Share ကို Access လုပ်ဖို့ပါ။ အပြင်ကနေ ကြည့်ရင် Bob က Charlie ရဲ့ Computer မှာ Administrator ဖြစ်နေတဲ့အတွက် အလုပ်လုပ်သင့်သလို ထင်ရပါတယ်။

ဒါပေမယ့် အမှန်တကယ် ဖြစ်ပေါ်လာတဲ့ Event တွေကို ကြည့်ရင် ပြဿနာက ပေါ်လာပါတယ်။ ပထမဆုံး Bob ရဲ့ Computer ပေါ်မှာ PowerShell Command ကို Execute လုပ်ပါတယ်။ အဲဒီနောက် Bob ရဲ့ Credential ကို အသုံးပြုပြီး powershell.exe က Alice ရဲ့ Computer ကို Network Logon (Type 3) နဲ့ Login ဝင်ပါတယ်။ ဒီအဆင့်အထိတော့ အားလုံး အဆင်ပြေပါတယ်။

အခု Alice ရဲ့ Computer ပေါ်မှာ Inner PowerShell Process တစ်ခု Run လုပ်လာပြီး Charlie ရဲ့ Computer ကို Access လုပ်ဖို့ ကြိုးစားပါတယ်။ ဒီအချိန်မှာ Alice ရဲ့ Computer ပေါ်မှာရှိတဲ့ wsmprovhost.exe Process က Charlie ရဲ့ Computer ကို Network Logon (Type 3) နဲ့ Login ဝင်ဖို့ ကြိုးစားပါတယ်။ ဒါပေမယ့် အရေးကြီးတဲ့ ပြဿနာတစ်ခု ဖြစ်ပေါ်လာပါတယ်။ Alice ရဲ့ Computer မှာ Bob ရဲ့ Credential ကို Cached လုပ်ထားခြင်း မရှိပါဘူး။

Windows Security Design အရ Credential တွေကို Default အနေနဲ့ “နောက်တစ်ဆင့် ထပ်မသုံးနိုင်အောင်” ကာကွယ်ထားပါတယ်။ အဲဒီကြောင့် Bob ရဲ့ Credential ကို Alice က Charlie ဆီ ထပ်ပို့ခွင့် မရပါဘူး။ ဒီလိုနဲ့ wsmprovhost.exe က Bob အဖြစ် Login မဝင်နိုင်တော့ပဲ Charlie ရဲ့ Computer ဆီကို ANONYMOUS LOGON အနေနဲ့ Login ဝင်ဖို့ ကြိုးစားပါတယ်။ Charlie ရဲ့ Computer ဘက်က ကြည့်

ရင် Anonymous User တစ်ယောက်က Admin Share ကို Access လုပ်ချင်နေသလို ဖြစ်သွားတဲ့အတွက် Access Denied Error ကို ပြန်ပို့လိုက်ပါတယ်။

Event	Description

PowerShell Execution |The outer Powershell is executed on bob-pc
4624 | powershell.exe logs on to alice-pc (Type 3) with Bob’s credential
PowerShell Execution | The inner Powershell is executed on alice-pc
4624 | wsmprovhost.exe attempts to log on to charlie-pc (Type 3) as ANONYMOUS LOGON

ဒီလို Bob → Alice → Charlie ဆိုပြီး Credential ကို နှစ်ဆင့် (Double Hop) သုံးဖို့ ကြိုးစားတဲ့အခါ ပျက်သွားတဲ့ ပြဿနာကိုပဲ **Double Hop Problem** လို့ ခေါ်ပါတယ်။ အဓိက အကြောင်းရင်းက Credential Delegation မရှိဘဲ Impersonation အဆင့်နဲ့ပဲ Remote Process ကို Run လုပ်နေလို့ ဖြစ်ပါတယ်။

ဒီပြဿနာကို ဖြေရှင်းဖို့ Windows မှာ အဓိကနည်းလမ်း နှစ်မျိုးရှိပါတယ်။ ပထမနည်းလမ်းက Remote Process ထဲမှာ User ကို ပြန်ပြီး Authentication လုပ်တာပါ။ အဲဒါဆိုရင် Alice ရဲ့ Computer ပေါ်မှာ Bob ရဲ့ Credential ကို ထပ်ပြီး ထည့်ပေးရပါတယ်။ ဒုတိယနည်းလမ်းက Active Directory ထဲမှာ Alice ရဲ့ Computer ကို Delegation အတွက် Trust ပေးထားတာပါ။ ဒီလို Trust ပေးထားရင် Alice က Bob ရဲ့ Delegation Token ကို ရနိုင်ပြီး Charlie ရဲ့ Computer ကို Bob အဖြစ် Access လုပ်နိုင်ပါတယ်။

ဒီနည်းလမ်း နှစ်ခုစလုံးရဲ့ အခြေခံအချက်က အလယ်ခံ System ဖြစ်တဲ့ Alice ရဲ့ Computer က Bob ရဲ့ **Delegation Token** ကို ရရှိထားရမယ် ဆိုတာပါ။ Delegation Token မရှိဘဲ Impersonation Token သာရှိရင် Local Machine အတွင်းမှာပဲ User အဖြစ် လုပ်ဆောင်နိုင်ပြီး Remote Resource ကို Access မလုပ်နိုင်ပါဘူး။

Cybersecurity အမြင်နဲ့ ကြည့်ရင် Double Hop Problem ကို နားလည်ထားခြင်းက အရမ်းအရေးကြီးပါတယ်။ Red Teamers အတွက် Lateral Movement လုပ်တဲ့အခါ ဘာကြောင့် Access Denied ဖြစ်နေလဲ ဆိုတာကို သဘောပေါက်စေပြီး Blue Teamers အတွက် Delegation Configuration မမှန်ရင် ဘယ်လို Security Risk တွေ ဖြစ်လာနိုင်လဲ ဆိုတာကို နားလည်စေပါတယ်။ Double Hop Problem ကို နားလည်သွားရင် Kerberos Delegation, Access Token, Credential Security တို့ရဲ့ အရေးပါမှုကို အခြေခံအဆင့်ကနေ သဘောပေါက်လာနိုင်ပါလိမ့်မယ်။

Double Hop

In the SIEM scenario, Bob performed a similar command with a double hop workaround. He again used alice-pc as the intermediary. On which system did he eventually list files?

```
external_table('Winlog')
| where EventProvider == "PowerShell" and HostName startswith "alice-pc" and
EventCode == 800 and * contains "icm"
| project Message
```

Token Domain Privilege Escalation

Windows Environment ထဲမှာ **Token Theft** ဆိုတာက တိုက်ခိုက်သူတွေ အရမ်းနှစ်သက်တဲ့ နည်းလမ်းတစ်ခု ဖြစ်ပေမယ့် Defender တွေအတွက်တော့ အလွန်ခက်ခဲတဲ့ Detection Challenge တစ်ခုပါ။ အဲ့ဒီ အကြောင်းရင်းက အဓိက နှစ်ချက်ရှိပါတယ်။

ပထမအချက်က Token Theft ကို လုပ်တဲ့အခါ **Windows ရဲ့ Legitimate Functionality** ကိုပဲ အသုံးပြုတာပါ။ Attacker က Windows API တွေကို ချိုးဖောက်သုံးတာမဟုတ်ဘဲ၊ ခွင့်ပြုထားတဲ့ Function တွေကို လိုအပ်တဲ့ Privilege တွေနဲ့ သုံးပါတယ်။ Log အနေနဲ့ ကြည့်ရင် "ပုံမှန် Process တစ်ခုက ပုံမှန် API ကို ခေါ်သုံးနေသလို" ပဲ မြင်ရတတ်ပါတယ်။ ဒါကြောင့် Signature-based Detection နဲ့ ချက်ချင်းဖမ်းဖို့ ခက်ပါတယ်။

ဒုတိယအချက်က Token Theft ကို စစ်ဖို့လိုအပ်တဲ့ Log တွေကို Enable လုပ်ထားရင် **Log Volume အရမ်းကြီးလာတာ** ပါ။ Object Access, Process Access Auditing တွေကို အပြည့်ဖွင့်လိုက်ရင် Log တွေ များလွန်းပြီး Storage Cost တက်လာမယ်၊ Noise များလာမယ်၊ False Positive တွေ ပိုများလာပါလိမ့်မယ်။ ဒါကြောင့် Organization အများစုက ဒီအလုပ်ကို SIEM မလုပ်ဘဲ **EDR (Endpoint Detection & Response)** ရဲ့ Heuristic Logic တွေကိုပဲ အားထားထားကြပါတယ်။

ဒါပေမယ့် Defender ဒါမှမဟုတ် Incident Responder တစ်ယောက်အနေနဲ့ Token Theft ကို **ဘယ်လို အချိန်မှာ အသုံးချတတ်လဲ** ဆိုတာကို နားလည်ထားရင် Investigation လုပ်တဲ့အခါ အရမ်းအရေးကြီးပါတယ်။ Token Theft က ဖြစ်ပြီးသွားရင် Log Trail က အလွန်မြန်မြန်ပျောက်သွားတတ်ပါတယ်။ "ဒီ User က တကယ် Login ဝင်တာလား၊ Token ကို ခိုးပြီး သုံးနေတာလား" ဆိုတာ ခွဲခြားရခက်လာပါတယ်။

Domain Privilege Escalation အတွက် Attacker တွေက ပုံမှန်အားဖြင့် **Delegation Token** ကို Impersonate လုပ်ကြပါတယ်။ Delegation Token ဆိုတာ Remote System တွေထိ User အဖြစ် လုပ်ဆောင်နိုင်တဲ့ Token ဖြစ်တဲ့အတွက် Domain Environment ထဲမှာ အလွန်အန္တရာယ်ကြီးပါတယ်။ ဒီလို Token ကို ခိုးနိုင်ဖို့ Attacker ဟာ အနည်းဆုံး **High Integrity Context (Administrator)** သို့မဟုတ် **SYSTEM Context** ထဲမှာ ရှိရပါမယ်။ တချို့အခါ Service Account တစ်ခုကို Compromise လုပ်ထားပြီး SeImpersonatePrivilege ရှိနေရင်လည်း ဒီတိုက်ခိုက်မှုကို လုပ်နိုင်ပါတယ်။

နောက်ထပ် အရေးကြီးတဲ့ အချက်က Attacker ဟာ **Target User (Domain User)** ရဲ့ Primary Token နဲ့ Run နေတဲ့ Process တစ်ခုကို Access ရှိထားရပါမယ်။ အဲဒီ Process ထဲက Token ကို Duplicate လုပ်ပြီး Impersonate လုပ်တာက Token Theft ရဲ့ အဓိက Mechanism ပါ။

ဒီ Scenario ထဲမှာ Attacker က **bob-pc ပေါ်မှာ Run နေတဲ့ Process တစ်ခုထဲက Alice ရဲ့ Delegation Token ကို ခိုးယူခဲ့ပါတယ်**။ ဒါပေမယ့် Defender ဘက်က ကြည့်ရင် Token Theft လုပ်တယ်ဆိုတဲ့ Log ကို တိုက်ရိုက် မမြင်နိုင်ပါဘူး။ မြင်ရတဲ့ တစ်ခုတည်းသော အချက်က **bob-pc ကနေ alice-pc ကို alice အကောင့်နဲ့ Successful Logon (Event ID 4624)** ဖြစ်နေတာပါ။ Log အမြင်နဲ့ ကြည့်ရင် "Alice က bob-pc ကနေ alice-pc ကို Login ဝင်လိုက်တယ်" လို့ပဲ ထင်ရပါတယ်။

Token Theft ပြီးတဲ့နောက် Attacker က **Alice ရဲ့ PC ပေါ်မှာ ADMIN$ Share ကို Access လုပ်ပါတယ်**။ ADMIN$ ဆိုတာ Administrator Privilege မရှိရင် မဝင်နိုင်တဲ့ Share ဖြစ်တဲ့အတွက် ဒီ Access က **Event ID 5140** အဖြစ် Log ထဲမှာ ပေါ်လာပါတယ်။ ဒီ 5140 Event က Attacker ရဲ့ လှုပ်ရှားမှုကို သဲလွန်စအနေနဲ့ ပေးတဲ့ နည်းနည်းလေးသာသော Evidence ဖြစ်ပါတယ်။

ဒါပေမယ့် ဒီအရာကို မမြင်မိရင် Incident Response အနေနဲ့ "Alice က ကိုယ်တိုင် Login ဝင်ပြီး Admin Share သုံးခဲ့တာ" လို့ ထင်သွားနိုင်ပါတယ်။ ဒီလိုနဲ့ Token Theft ရဲ့ အမှန်တရားက လွယ်လွယ်နဲ့ ဖုံးကွယ်သွားပါတယ်။

Attacker တွေဟာ Token ကို Impersonate လုပ်တာတင် မကပါဘူး။ **Alice ရဲ့ Token နဲ့ Process အသစ်တွေ Spawn လုပ်နိုင်ပါတယ်**။ ဒီလို Spawn လုပ်ထားတဲ့ Child Process တွေဟာ Log ထဲမှာ `windomain\alice` အနေနဲ့ ပေါ်လာပါလိမ့်မယ်။ ဒါပေမယ့် Parent Process ကို ကြည့်လိုက်ရင် SYSTEM သို့မဟုတ် Local Administrator Account က Run နေတဲ့ Attacker Process ဖြစ်နေတာကို တွေ့နိုင်ပါတယ်။ ဒီ Parent–Child Relationship က IR အတွက် အရမ်းအရေးကြီးတဲ့ Clue ဖြစ်ပါတယ်။

အကျဉ်းချုပ်ပြောရရင် Token Domain Privilege Escalation ဆိုတာ **Credential ကို မခိုးဘဲ Identity ကို ခိုးတဲ့ တိုက်ခိုက်မှု** လို့ နားလည်နိုင်ပါတယ်။ Log တွေက ပုံမှန် User Activity လိုပဲ မြင်ရတဲ့အတွက် Detection အရမ်းခက်ပြီး Investigation မှာလည်း သဲလွန်စနည်းပါတယ်။ ဒါကြောင့် Access Token, Delegation Token, 4624 နဲ့ 5140 Event Flow တွေကို နားလည်ထားခြင်းက Blue Team, IR, DFIR အတွက် မဖြစ်မနေ လိုအပ်တဲ့ အခြေခံတစ်ခု ဖြစ်ပါတယ်။

Token Domain Privesc 2

Shortly after stealing Alice's token, the attacker stole another domain user's delegation token and accessed the same file share on alice-pc. Which user did the attacker compromise?

```
external_table('Winlog')
| where EventCode == 5140 and HostHostname contains "alice"
| project Timestamp, EventCode, EventAction, HostName, SubjectUserName,
IpAddress
```

Find the attacker IP

```
external_table('Winlog')
| where EventCode == 5140 and IpAddress == "192.168.56.110"
| project Timestamp, EventCode, EventAction, HostName, SubjectUserName,
IpAddress
```

Create Process with Token – Runas (CreateProcessWithLogon)

Attacker တစ်ယောက် System တစ်ခုအပေါ် foothold ရလာပြီးတဲ့နောက် သူ့ရဲ့နောက်ထပ် ရည်ရွယ်ချက်က “ရထားတဲ့ credential ကို ဘယ်လိုအသုံးချမလဲ” ဆိုတာပါ။ Credential ရလာနိုင်တဲ့ နည်းလမ်းတွေက Kerberoasting၊ Forced Authentication၊ Group Policy Preferences ထဲက cleartext password တွေ၊ unsecured file share တွေ၊ password spraying လိုမျိုး အမျိုးမျိုးရှိပါတယ်။ ဒါပေမယ့် Credential ကို သိနေရုံနဲ့ မလုံလောက်ပါဘူး။ Windows မှာ လုပ်ဆောင်ဖို့ **Access Token** လိုအပ်ပါတယ်။

Attacker အတွက် အကောင်းဆုံး Token က **Delegation Token** ဖြစ်ပါတယ်။ Delegation Token ရှိရင် Remote System တွေထိ User အဖြစ် Access လုပ်နိုင်လို့ Domain Privilege Escalation နဲ့ Lateral Movement အတွက် အရမ်းအရေးကြီးပါတယ်။ ဒီလို Token ကို ဖန်တီးဖို့ နည်းလမ်းတစ်ခုက

CreateProcessWithLogon ဆိုတဲ့ Windows API Function ကို သုံးတာပါ။ ဒီ Function က Windows ရဲ့ `runas` command နဲ့ Concept အတူတူပါပဲ။

CreateProcessWithLogon ကို သုံးတဲ့အခါ Username, Password, Domain, Run ချင်တဲ့ Command စတဲ့ အချက်အလက်တွေကို တိုက်ရိုက် ပေးရပါတယ်။ Windows က ဒီ Credential တွေကို အသုံးပြုပြီး **Interactive Logon (Logon Type 2)** တစ်ခုကို ဖန်တီးပေးပါတယ်။ အဲ့ဒီ Logon ကနေ Process အသစ်တစ်ခု Spawn လုပ်ပြီး အဲ့ဒီ Process က ပေးထားတဲ့ Credential ရဲ့ Security Context အောက်မှာ Run ပါတယ်။ အရေးကြီးတဲ့ အချက်က ဒီ Process က **Primary Token အသစ်တစ်ခု** နဲ့ Run နေတာပါ။

ဒီ Scenario ထဲမှာ Attacker က **bob-pc ပေါ်မှာ local administrator credential ကို သိထားပြီးသား** ဖြစ်ပါတယ်။ အဲ့ဒီ Credential ကို သုံးပြီး CreateProcessWithLogon (runas) နဲ့ **Child Process တစ်ခုကို Spawn လုပ်ခဲ့ပါတယ်**။ Parent Process က Attacker ရဲ့ Process ဖြစ်နိုင်ပြီး Child Process ကတော့ Local Administrator အဖြစ် Run နေတာဖြစ်ပါတယ်။

SIEM ထဲမှာ ဒီအရာကို ဘယ်လိုမြင်ရလဲဆိုရင် Event ID **4624 (Successful Logon)** ကို ကြည့်ရပါတယ်။

```
external_table('Winlog')
| where Timestamp between (datetime("2023-04-12T15:17:19.300Z") ..
datetime("2023-04-12T15:17:19.720Z")) and EventCode == 4624
```

ပေးထားတဲ့ Query က အချိန်အတိုင်းအတာကို ကန့်သတ်ပြီး bob-pc ပေါ်မှာ ဖြစ်ခဲ့တဲ့ Logon Event ကို ဖမ်းထားတာပါ။ ဒီ Event က ပုံမှန် Login Event လိုပဲ မြင်ရပေမယ့် အတွင်းပိုင်း Detail တွေကို ကြည့်ရင် အရေးကြီးတဲ့ သဲလွန်စတွေ ရှိပါတယ်။

ဒီ 4624 Event ထဲမှာ **Logon Process = seclogo** လို့ ပြနေပါတယ်။ seclogo ဆိုတာ Windows ရဲ့ Secondary Logon Service ဖြစ်ပြီး `runas` command နဲ့ CreateProcessWithLogon API တွေကို သုံးတဲ့အခါ မဖြစ်မနေ အသုံးပြုရပါတယ်။ ဒါကြောင့် seclogo ကို မြင်လိုက်ရင် “User က Console မှာ Password ရိုက် Login ဝင်တာ မဟုတ်ဘူး၊ Credential ကို Programmatically သုံးပြီး Process အသစ် ဖန်တီးထားတာ” ဆိုတာကို သဘောပေါက်နိုင်ပါတယ်။

နောက်ထပ် သဲလွန်စက **Source Process = svchost.exe** ဖြစ်နေတာပါ။ svchost.exe က Windows Service တွေ Run ဖို့ သုံးတဲ့ Generic Host Process ဖြစ်ပါတယ်။ Secondary Logon Service ကိုလည်း svchost.exe က Host လုပ်ထားတဲ့အတွက် CreateProcessWithLogon နဲ့ Process Spawn လုပ်တဲ့အခါ Source Process အဖြစ် svchost.exe လို့ Log ထဲမှာ ပေါ်လာပါတယ်။ ဒါက User တိုက်ရိုက် run လုပ်တာမဟုတ်ဘဲ Service တစ်ခုက ကြားခံပြီး လုပ်နေတယ်ဆိုတာကို ပြပါတယ်။

ဒီလို Event တစ်ခုကို Defender ဘက်က ကြည့်ရင် “Local Admin က runas သုံးပြီး Program တစ်ခု Run လုပ်လိုက်တယ်” လို့ ထင်ရနိုင်ပါတယ်။ ဒါပေမယ့် Incident Response အမြင်နဲ့ ကြည့်ရင် Parent Process၊ Timeline၊ နောက်ဆက်တွဲ Network Activity တွေနဲ့ ချိတ်ကြည့်ရင် **Attacker က Credential ကို အသုံးချပြီး Token အသစ် ဖန်တီးထားတာ** ဖြစ်နိုင်တယ်ဆိုတာကို သဘောပေါက်ရပါတယ်။

အကျဉ်းချုပ်ပြောရရင် CreateProcessWithLogon (runas) ဆိုတာ Attacker တွေအတွက် **Credential → Token → Process** ဆိုတဲ့ လမ်းကြောင်းကို ဖန်တီးပေးတဲ့ နည်းလမ်းပါ။ Log အနေနဲ့ ကြည့်ရင် Event 4624 တစ်ခုတည်းပဲ ဖြစ်ပေမယ့် seclogo နဲ့ svchost.exe ဆိုတဲ့ Detail နှစ်ခုက Token-based

Execution ဖြစ်နေတာကို ပြသတဲ့ အရေးကြီးတဲ့ သဲလွန်စတွေ ဖြစ်ပါတယ်။ Blue Team နဲ့ DFIR အတွက် ဒီ Pattern ကို သိထားတာက Credential Abuse နဲ့ Token Abuse ကို ခွဲခြားနိုင်ဖို့ အလွန်အရေးကြီးပါတယ်။

Compromised Process

Based on the previous query, which process did the attacker call the CreateProcessWithLogon function from?

To get this answer we just need to look at the process creation event in our timeframe. From this event, we can see the ParentImage is spoolsv.exe. The LogonId 0x145268 also matches the 4624 event we looked at earlier.

```
external_table('Winlog')
| where Timestamp between (datetime("2023-04-12T15:17:19.300Z") ..
datetime("2023-04-12T15:17:19.720Z")) and EventCode == 1
| project Timestamp, ParentCommandLine, CommandLine, LogonId
```

LogonUser Impersonation

အရင်ပိုင်းမှာ ပြောခဲ့တဲ့ CreateProcessWithLogon၊ runas၊ runas /netonly တို့က Windows မှာ Token ဖန်တီးဖို့ အသုံးများတဲ့ နည်းလမ်းတွေဖြစ်ပါတယ်။ ဒီနည်းလမ်းတွေက Logon Type 2 (Interactive) သို့မဟုတ် Logon Type 9 (New Credentials) လို ထူးခြားတဲ့ Logon Type တွေကို ဖြစ်ပေါ်စေတတ်ပါတယ်။ အချို့ Environment တွေမှာ Logon Type 9 က အလွန်ရှားပါးတဲ့အတွက် Blue Team အနေနဲ့ Detection Logic ထဲမှာ အသုံးချနိုင်ပါတယ်။ အထူးသဖြင့် Cobalt Strike ရဲ့ `make-token` command က Logon Type 9 ကို ထုတ်ပေးတဲ့အတွက် Defender တွေက ဒီ Logon Type ကို ပိုပြီး စောင့်ကြည့်တတ်ကြပါတယ်။

ဒါပေမယ့် Attacker တွေအတွက် Token ရဖို့ နည်းလမ်းက ဒီလောက်နဲ့ မကပါဘူး။ နောက်ထပ် အသုံးများတဲ့ နည်းလမ်းတစ်ခုက **LogonUser API ကို ခေါ်ပြီး Token ကို Impersonate လုပ်တာ** ဖြစ်ပါတယ်။ ဒီနည်းလမ်းမှာ Credential ကို အသုံးပြုပြီး LogonUser ကို ခေါ်လိုက်တာနဲ့ Windows က Access Token တစ်ခုကို ပြန်ပေးပါတယ်။ Attacker က အဲဒီ Token ကို Thread-level Impersonation လုပ်ပြီး User အဖြစ် လုပ်ဆောင်နိုင်ပါတယ်။

LogonUser ကို သုံးတဲ့ နည်းလမ်းက Attacker အတွက် အားသာချက် အချို့ရှိပါတယ်။ ပထမအချက်က Logon Type ကို ကိုယ်တိုင် ရွေးနိုင်တာပါ။ Interactive, Network, Batch စတဲ့ Logon Type တွေကို အခြေအနေလိုက် သုံးနိုင်ပါတယ်။ ဒုတိယအချက်က CreateProcessWithLogon လို Process အသစ်တစ်ခုကို မဖြစ်မနေ Spawn လုပ်စရာ မလိုတာပါ။ အရင်က Run နေတဲ့ Process ထဲမှာပဲ Token ကို Impersonate လုပ်ပြီး လုပ်ဆောင်နိုင်ပါတယ်။ ဒါကြောင့် Process Creation Event တွေ မပေါ်လာဘဲ လှုပ်ရှားနိုင်တဲ့ အခွင့်အလမ်း ရှိပါတယ်။

ဒီလို LogonUser Impersonation ကို သုံးတဲ့အခါ SIEM သို့မဟုတ် Security Log ထဲမှာ ပေါ်လာတဲ့ Event တွေကို ကြည့်ရင် ထူးခြားတဲ့ Pattern တစ်ခု ရှိပါတယ်။ Logon Event (4624) က ပေါ်လာပေမယ့်

Logon Type ကိုယ်တိုင်က မထူးခြားနိုင်ပါဘူး။ အမှန်တကယ် အရေးကြီးတဲ့ သဲလွန်စက **Source Process** ဖြစ်ပါတယ်။

ပုံမှန် Windows Operation မှာ Logon Type တစ်မျိုးချင်းစီအတွက် “ဘယ် Process က ဒီ Logon ကို စတင်လေ့ရှိလဲ” ဆိုတဲ့ Pattern တွေ ရှိပါတယ်။ ဥပမာ Interactive Logon (Type 2) ဆိုရင် svchost.exe သို့မဟုတ် winlogon.exe ကနေ စတင်တာကို များများတွေ့ရပါတယ်။ Workstation Unlock (Type 7) ဆိုရင် lsass.exe ကနေ လာတာက ပုံမှန်ပါ။ ဒီလို Source Process နဲ့ Logon Type တွေက Environment တစ်ခုချင်းစီမှာ Baseline လုပ်လို့ရပါတယ်။

LogonUser Impersonation ကို သုံးတဲ့အခါ ဒီ Baseline က ပျက်သွားတတ်ပါတယ်။ ဥပမာ Interactive Logon Type 2 ဖြစ်နေပေမယ့် Source Process က userland process တစ်ခု၊ သို့မဟုတ် exploit tool တစ်ခုက Run နေတာကို တွေ့ရနိုင်ပါတယ်။ ဒီလို အခြေအနေက “ဒီ Logon ကို Windows ရဲ့ ပုံမှန် Authentication Flow မဟုတ်ဘဲ Programmatically ဖန်တီးထားတာ ဖြစ်နိုင်တယ်” ဆိုတဲ့ သဲလွန်စကို ပေးပါတယ်။

Blue Team နဲ့ Incident Responder အတွက် LogonUser Impersonation ကို နားလည်ထားခြင်းက အလွန်အရေးကြီးပါတယ်။ Token Theft သို့မဟုတ် Credential Abuse လုပ်ထားတဲ့အခါ Log တွေက ပုံမှန် User Activity လိုပဲ မြင်ရတတ်ပါတယ်။ ဒါပေမယ့် Logon Type နဲ့ Source Process ကို ချိတ်ပြီး ကြည့်နိုင်ရင် “ဘယ် Logon က သဘာဝမကျဘူး” ဆိုတာကို ဖော်ထုတ်နိုင်ပါတယ်။

အကျဉ်းချုပ်ပြောရရင် LogonUser Impersonation ဆိုတာ Attacker တွေအတွက် **Process မဖန်တီးဘဲ Token ရယူနိုင်တဲ့ နည်းလမ်း** ဖြစ်ပြီး Detection ကို ပိုခက်စေပါတယ်။ Defender ဘက်ကတော့ Logon Type တစ်ခုချင်းစီအတွက် Source Process Baseline ကို သိထားပြီး အဲ့ဒီ Pattern ကနေ လွဲနေတဲ့ Logon Event တွေကို အာရုံစိုက်ကြည့်ရပါမယ်။ ဒီအယူအဆကို နားလည်ထားရင် Token-based Attack တွေကို Log အဆင့်မှာ သဲလွန်စနည်းနည်းနဲ့တောင် ဖော်ထုတ်နိုင်လာပါလိမ့်မယ်။

LogonUser Charlie

The attacker performed a LogonUser with impersonation using Charlie's credentials. Which logon type did the attacker use?

```
external_table('Winlog')
| where EventCode == 4624 and ProcessName endswith "spoolsv.exe" and
TargetUserName == "charlie"
| project Timestamp, LogonType
```

Named Pipe Impersonation

Windows မှာ Attacker တစ်ယောက်က **High Integrity (Administrator)** ကနေ **SYSTEM Integrity** ထိ တက်နိုင်ဖို့ Access Token Manipulation ကို အသုံးပြုတဲ့ နည်းလမ်း အဓိက နှစ်မျိုးရှိပါတယ်။ ပထမ နည်းလမ်းကတော့ အခြား Process တစ်ခုထဲမှာ Run နေတဲ့ SYSTEM Token ကို Handle ရယူပြီး Duplicate လုပ်တာပါ။ ဒါကို လုပ်နိုင်ဖို့ **SeDebugPrivilege** လိုအပ်ပြီး Defender ဘက်ကလည်း အန္တရာယ်ကြီးတယ်လို့ သိထားတဲ့ နည်းလမ်းပါ။

ဒီ Module မှာ အဓိက ဆွေးနွေးတာက ဒုတိယနည်းလမ်းဖြစ်တဲ့ **Named Pipe Impersonation** ဖြစ်ပါတယ်။ ဒီနည်းလမ်းရဲ့အရေးကြီးဆုံး အားသာချက်က SeDebugPrivilege မလိုဘဲ SYSTEM ကို ရနိုင်တာဖြစ်ပြီး Privilege နည်းနည်းပဲ ရထားတဲ့ Attacker တွေအတွက်တောင် အသုံးချနိုင်ပါတယ်။

အရင်ဆုံး **Named Pipe** ဆိုတာကို နားလည်ဖို့လိုပါတယ်။ Named Pipe ဆိုတာ Windows မှာ Process နှစ်ခု အချင်းချင်း ဆက်သွယ်ဖို့ အသုံးပြုတဲ့ IPC (Inter-Process Communication) Mechanism တစ်ခုပါ။ Client–Server Model နဲ့ အလုပ်လုပ်ပြီး Server က Pipe တစ်ခု ဖန်တီးထားပြီး Client က လာချိတ်ပါတယ်။ Windows Design အရ Server Process က Client ကို **Impersonate** လုပ်နိုင်ပါတယ်။ အဲ့ဒီ Impersonation ကို သုံးပြီး Server က “Client အစား” Resource တွေကို Access လုပ်နိုင်ပါတယ်။

ဒီ Design ကို Defender တွေအတွက် အဆင်ပြေအောင် ဖန်တီးထားပေမယ့် Attacker တွေအတွက်တော့ **အခွင့်အရေးကြီးတဲ့ အားနည်းချက်** ဖြစ်လာပါတယ်။ Attacker က Server ဖြစ်သွားပြီး Client ကို စောင့်နေတာပါ။ အကယ်၍ Client က SYSTEM Token နဲ့ Run နေတဲ့ Process ဖြစ်ရင် Attacker က **SYSTEM Token ကို ခိုးယူနိုင်** သွားပါတယ်။

ဒီ Scenario ထဲမှာ Attacker က SYSTEM Process ကို သူ့ Named Pipe ဆီ ချိတ်လာအောင် **အတင်းလုပ်စေတဲ့ နည်းလမ်း** ကို အသုံးပြုပါတယ်။ အရင်ဆုံး Attacker က System ပေါ်မှာ Service တစ်ခုကို ဖန်တီးပါတယ်။ ဒီအရာက Windows Log ထဲမှာ **Event ID 7045 (Service Installed)** အဖြစ် ပေါ်လာပါတယ်။ Service ဆိုတာ SYSTEM privilege နဲ့ Run နိုင်တဲ့ Mechanism ဖြစ်တဲ့အတွက် Named Pipe Impersonation မှာ မကြာခဏ အသုံးပြုကြပါတယ်။

Service ကို ဖန်တီးပြီးတဲ့နောက် Attacker က **Named Pipe တစ်ခုကို Create လုပ်ပါတယ်**။ ဒါကို Sysmon လို Log Source တွေမှာ Event ID 17 (Pipe Created) အဖြစ် တွေ့နိုင်ပါတယ်။ ဒီ Pipe က Attacker ရဲ့ “Server” ဖြစ်ပြီး SYSTEM Process ကို Client အနေနဲ့ စောင့်နေတာပါ။

နောက်တစ်ဆင့်မှာ Attacker က Service ကို Start လုပ်ပါတယ်။ ဒီအရာက Process Creation Event ဖြစ်ပြီး **Event ID 4688** အဖြစ် Log ထဲမှာ ပေါ်လာပါတယ်။ Service Run လိုက်တာနဲ့ Service Process က SYSTEM Context အောက်မှာ Run နေပြီး Attacker ပြင်ဆင်ထားတဲ့ Flow အတိုင်း Named Pipe ကို ချိတ်ဆက်ဖို့ ကြိုးစားပါတယ်။

Service က Named Pipe ကို ချိတ်တဲ့အချိန်မှာ **Event ID 18 (Pipe Connected)** ပေါ်လာပါတယ်။ ဒီ Event က အရေးကြီးဆုံး Moment ဖြစ်ပါတယ်။ ဘာကြောင့်လဲဆိုတော့ SYSTEM Process က Attacker ရဲ့ Pipe ကို Client အနေနဲ့ ချိတ်လာတဲ့အချိန်ပါ။

ဒီအချိန်မှာ Attacker က `ImpersonateNamedPipeClient` ဆိုတဲ့ Windows API ကို ခေါ်လိုက်ပါတယ်။ Windows Design အရ Server ဖြစ်တဲ့ Attacker Process က Client (SYSTEM) ကို Impersonate လုပ်ခွင့် ရရှိပါတယ်။ ဒီလိုနဲ့ Attacker က **SYSTEM Token ကို ယူပြီး ကိုယ်တိုင် SYSTEM အဖြစ် ပြောင်းလဲသွားနိုင်** ပါတယ်။

```
external_table('Winlog')
| where Timestamp between (datetime("2023-04-12T15:42:12.000Z") ..
datetime("2023-04-12T15:42:13.000Z")) and EventCode in (17, 18, 7045, 4688)
| project Timestamp, EventCode, EventAction, HostName, ServiceName,
ServiceFileName, PipeName, Image, User
```

ဒီ Event Flow ကို SIEM ထဲမှာ ကြည့်ရင် Service Create, Pipe Create, Process Start, Pipe Connect ဆိုပြီး အချိန်တိုအတွင်း ဆက်တိုက် ဖြစ်နေတာကို တွေ့နိုင်ပါတယ်။ ဒီလို Sequence ကို မြင်ရင် Named Pipe Impersonation ဖြစ်နိုင်ခြေ အရမ်းမြင့်ပါတယ်။

ဒီနည်းလမ်းရဲ့အားသာချက်က **Implant မတင်ဘဲ SYSTEM ရနိုင်တာ** ဖြစ်ပါတယ်။ ဒါပေမယ့် အများအားဖြင့် High Integrity Administrator Token ရှိပြီးသား Attacker တွေအတွက် ပိုလွယ်ပါတယ်။ ဒါဆို “Permission နည်းနည်းပဲ ရထားတဲ့ Attacker က ဘယ်လို SYSTEM ရမလဲ” ဆိုတဲ့ မေးခွန်း ထပ်ပေါ်လာပါတယ်။

ဒီနေရာမှာ **NETWORK SERVICE** Account က အရေးကြီးလာပါတယ်။ NETWORK SERVICE နဲ့ Run နေတဲ့ Service တွေမှာ Privilege မများပေမယ့် **SeImpersonatePrivilege** ရှိပါတယ်။ Named Pipe Impersonation ကို လုပ်ဖို့ အဓိကလိုအပ်တာက ဒီ Privilege ပါ။ ဒါကြောင့် Attacker က NETWORK SERVICE Process တစ်ခုကို Control ရထားရင် SYSTEM ကို တက်ဖို့ အခွင့်အလမ်း ရှိပါတယ်။

ဒီလို SYSTEM Process ကို Named Pipe ဆီ ချိတ်လာအောင် လုပ်တဲ့ နာမည်ကြီး နည်းလမ်းတစ်ခုက **Printer Bug** ဖြစ်ပါတယ်။ ဒီ Bug ကို Trigger လုပ်ဖို့ `RpcRemoteFindFirstPrinterChangeNotificationEx` ဆိုတဲ့ RPC Function ကို ခေါ်ပြီး Client Computer Argument ကို `\\localhost\pipe\<whatever>` လို Pattern နဲ့ ပေးလိုက်ပါတယ်။ Windows Printer Subsystem က ဒီ Request ကို Process လုပ်တဲ့အခါ SYSTEM Process က `\\.\pipe\spoolss` ကို ချိတ်ဆက်သွားပါတယ်။

အကယ်၍ Attacker က အဲ့ဒီ Pattern နဲ့ Named Pipe Server ကို ကြိုတင် ဖန်တီးထားရင် SYSTEM Process က သူ့ Pipe ဆီ လာချိတ်ပါတယ်။ အဲဒီအချိန်မှာ Attacker က ImpersonateNamedPipeClient ကို ခေါ်လိုက်ရင် **SYSTEM Token ကို ရယူနိုင်** သွားပါတယ်။ ဒီ Attack က Permission နည်းနည်းပဲ လိုအပ်တဲ့အတွက် NETWORK SERVICE ကနေ SYSTEM တက်ဖို့ အသုံးအများဆုံး နည်းလမ်းတစ်ခု ဖြစ်လာပါတယ်။

အကျဉ်းချုပ်ပြောရရင် Named Pipe Impersonation ဆိုတာ Windows ရဲ့ Client–Server Impersonation Design ကို အသုံးချပြီး **SYSTEM Process ကို Client အဖြစ် လာချိတ်စေကာ Token ကို ခိုးယူတဲ့ နည်းလမ်း** ဖြစ်ပါတယ်။ Log အနေနဲ့ ကြည့်ရင် Service Create, Named Pipe Create, Pipe Connect ဆိုတဲ့ Event Flow က အရေးကြီးတဲ့ သဲလွန်စတွေ ဖြစ်ပါတယ်။ Blue Team နဲ့ IR အတွက် ဒီ Concept ကို နားလည်ထားရင် “Privilege နည်းနည်းပဲ ရထားတဲ့ Process က ဘာကြောင့် SYSTEM ဖြစ်သွားလဲ” ဆိုတာကို သဘောပေါက်နိုင်ပြီး Attack Path ကို ပြန်ဆောက်လို့ ရပါလိမ့်မယ်။

PPID Spoofing (Parent Process ID Spoofing / Reparenting)

Windows operating system မှာ program တစ်ခု run လိုက်တိုင်း process တစ်ခု ဖန်တီးလာပါတယ်။ Process တစ်ခုချင်းစီမှာ “ဘယ် process က ဒီ process ကို စတင်ပေးခဲ့သလဲ” ဆိုတာကို သိမ်းထားတဲ့ အချက်အလက် ရှိပါတယ်။ ဒီအရာကို Parent–Child Process Relationship လို့ ခေါ်ပါတယ်။ ဥပမာ user တစ်ယောက်က Desktop ပေါ်က PowerShell ကို ဖွင့်လိုက်ရင် explorer.exe က powershell.exe ကို စတင်ပေးတာဖြစ်ပါတယ်။ ဒါဟာ သာမန်အသုံးပြုမှုဖြစ်ပြီး Defender တွေ၊ SOC Analyst တွေက လည်း ဒီလို relationship တွေကို ကြည့်ပြီး “ဒါက user activity လား၊ attack လား” ဆိုတာကို ခန့်မှန်းကြပါတယ်။

PPID Spoofing ဆိုတာက Attacker တစ်ယောက်က Process အသစ်တစ်ခုကို ဖန်တီးတဲ့အချိန်မှာ “Parent process ကို လိမ်ပြောင်းပြသခြင်း” ကို ဆိုလိုပါတယ်။ အမှန်တကယ် process ကို စတင်ပေးတာက Process A ဖြစ်ပေမဲ့ log ထဲမှာ Process B က စတင်ပေးထားသလို ပြထားနိုင်ပါတယ်။ ဒီလိုလုပ်နိုင်တာဟာ Windows OS ရဲ့ legitimate feature တစ်ခုကို အသုံးချတာဖြစ်ပြီး malware သီးသန့် feature မဟုတ်ပါဘူး။ ဒါကြောင့် ဒီ technique ကို detect လုပ်ဖို့ အရမ်းခက်ပါတယ်။

ဒီအရာက ဘာကြောင့် အန္တရာယ်ကြီးလဲဆိုရင် Defender တွေဟာ process relationship ကို အရမ်းယုံကြည်လို့ပါပဲ။ ဥပမာ powershell.exe ကို explorer.exe က စတင်ထားတယ်ဆိုရင် “user က PowerShell ဖွင့်တာပဲ” လို့ ထင်ရပါတယ်။ ဒါပေမဲ့ အမှန်တကယ်မှာ spoolsv.exe လို SYSTEM privilege ရှိတဲ့ process က powershell.exe ကို စတင်ပြီး Parent ကို explorer.exe လို့ လိမ်ထားနိုင်ပါတယ်။ ဒီအချိန်မှာ attacker ရဲ့ PowerShell က SYSTEM privilege နဲ့ run နေပေမဲ့ log ကြည့်ရင် user activity လိုပဲ မြင်နေရပါတယ်။ ဒီဟာကို မတွေ့မိရင် compromise တစ်ခုလုံးကို လွတ်သွားနိုင်ပါတယ်။

PPID Spoofing ကို နားလည်ဖို့ “True Parent” နဲ့ “Spoofed Parent” ဆိုတဲ့ အယူအဆကို သိထားရပါမယ်။ Windows ETW events တွေမှာ process creation အတွက် Parent ကို နှစ်နေရာမှာ သိမ်းထားနိုင်ပါတယ်။ PID field က အမှန်တကယ် process ကို စတင်ပေးတဲ့ True Parent ကို ပြပါတယ်။ ParentProcessID field ကတော့ OS ကို process ဖန်တီးချိန်မှာ ကြေညာထားတဲ့ Parent ဖြစ်ပြီး Attacker က ဒီတန်ဖိုးကို လိမ်နိုင်ပါတယ်။ ဒါကြောင့် PID နဲ့ ParentProcessID မတူနေဘူးဆိုရင် PPID Spoofing ဖြစ်နိုင်ချေ ရှိပါတယ်။

ဒီနည်းလမ်းကို detect လုပ်ရာမှာ အခက်အခဲ အများကြီး ရှိပါတယ်။ Open-source အနေနဲ့ SilkETW လို tool တွေကို သုံးပြီး ETW events ကို SIEM ထဲပို့ပြီး စစ်ဆေးနိုင်ပေမဲ့ ETW က user-mode အခြေခံ ဖြစ်တာကြောင့် attacker က အလွယ်တကူ ချိုးဖောက်နိုင်ပါတယ်။ Enterprise အဆင့်မှာတော့ driver-level မှာ process creation ကို စောင့်ကြည့်နိုင်တဲ့ EDR တွေကသာ ယုံကြည်စိတ်ချရပါတယ်။ ဒါကြောင့် SIEM တစ်ခုတည်းနဲ့ PPID Spoofing ကို အပြည့်အဝ detect လုပ်နိုင်မယ်လို့ မမျှော်လင့်သင့်ပါဘူး။

အရေးကြီးတာတစ်ခုက PPID Spoofing က malicious activity ပဲ မဟုတ်ပါဘူး။ Windows ကိုယ်တိုင် legitimate အနေနဲ့ reparenting လုပ်တဲ့ process တွေ ရှိပါတယ်။ ဥပမာ svchost.exe က service တွေအတွက် parent relationship ကို ပြောင်းပြနိုင်ပါတယ်။ consent.exe က UAC prompt အတွက် reparenting လုပ်ပါတယ်။ werfault.exe က application crash ဖြစ်တဲ့အချိန်မှာလည်း parent ကို ပြောင်းထားတာတွေ့ရပါတယ်။ ဒါကြောင့် PID မတူတာတွေ့တိုင်း attack လို့ မဆုံးဖြတ်သင့်ပါဘူး။ Context ကို အမြဲကြည့်ရပါမယ်။

ဒီ scenario မှာ SilkETW event တွေကို စစ်လိုက်တဲ့အခါ spoolsv.exe က calc.exe နဲ့ powershell.exe ကို စတင်ပေးထားပြီး Parent ကို explorer.exe လို့ လိမ်ထားတာတွေ့ရပါတယ်။ spoolsv.exe က SYSTEM privilege ရှိတဲ့ process ဖြစ်တာကြောင့် attacker က SYSTEM level ကို ရပြီးသားဖြစ်ပါတယ်။ ဒါပေမဲ့ Defender မျက်စိနဲ့ ကြည့်ရင် explorer.exe က calc.exe ဖွင့်ထားသလို မြင်နေရပါတယ်။ ဒါဟာ attacker အတွက် detection ကို ရှောင်နိုင်တဲ့ အရမ်းပြင်းထန်တဲ့ technique ဖြစ်ပါတယ်။

နိဂုံးချုပ်အနေနဲ့ PPID Spoofing ဆိုတာ Windows ရဲ့ legitimate feature ကို အသုံးချပြီး Defender တွေရဲ့ ယုံကြည်ချက်ကို လှည့်စားတဲ့ နည်းလမ်းဖြစ်ပါတယ်။ Process relationship ကိုသာ ကြည့်ပြီး ဆုံးဖြတ်တဲ့ security monitoring က ဒီ technique ကို လွယ်လွယ်နဲ့ လက်လွတ်သွားနိုင်ပါတယ်။ ဒါကြောင့် Blue Team အနေနဲ့ True Parent ကို စစ်ဆေးနိုင်တဲ့ visibility ရှိဖို့၊ SYSTEM process တွေက user-facing tools တွေကို spawn လုပ်နေရင် အထူးသတိထားဖို့၊ EDR telemetry နဲ့ SIEM hunting ကို တွဲသုံးဖို့ အရေးကြီးပါတယ်။