UAC Bypasses

Windows Operating System မှာ **User Account Control (UAC)** ဆိုတာက Windows Vista ကစပြီး Windows 7, 8, 10, 11 အထိ ဆက်လက်အသုံးပြုလာတဲ့ လုံခြုံရေးစနစ်တစ်ခုဖြစ်ပါတယ်။ UAC ရဲ့ အဓိကရည်ရွယ်ချက်ကတော့ ကွန်ပျူတာစနစ်ကို မလိုလားအပ်တဲ့ ပြောင်းလဲမှုတွေ မဖြစ်အောင် ကာကွယ်ဖို့ပါ။ အထူးသဖြင့် Administrator အခွင့်အရေးလိုအပ်တဲ့ အလုပ်တွေကို မည်သူမဆို အလိုအလျောက်လုပ်လို့မရအောင် တားဆီးထားပါတယ်။

ပုံမှန်အားဖြင့် Windows ကို သုံးနေတဲ့ User တစ်ယောက်ဟာ Administrator ဖြစ်နေရင်တောင် အစမှာတော့ **Standard User အခွင့်အရေး**နဲ့ပဲ လည်ပတ်နေပါတယ်။ တစ်ခုခုကို စနစ်အဆင့်မှာ ပြောင်းလဲမယ်ဆိုရင်၊ ဥပမာ Software install လုပ်တာ၊ Registry ပြောင်းတာ၊ System File ပြင်တာမျိုးလုပ်ချင်ရင် Windows က **UAC Prompt** လို့ခေါ်တဲ့ အတည်ပြု Dialog Box တစ်ခု ပြပါတယ်။ ဒီ Prompt မှာ User ကို “ဒီအလုပ်ကို လုပ်ခွင့်ပေးမလား” လို့ မေးပြီး User က ကိုယ်တိုင် Yes ကို နှိပ်မှသာ အလုပ်ဆက်လုပ်နိုင်ပါတယ်။ ဒီလိုနည်းနဲ့ Malware သို့မဟုတ် မလိုလားအပ်တဲ့ Program တွေက User မသိဘဲ စနစ်ကို ပြောင်းလဲသွားတာကို ကာကွယ်ပေးပါတယ်။

အလွယ်ပြောရရင် UAC ကို **လုံခြုံရေးအလွှာတစ်ခု** လို့စဉ်းစားနိုင်ပါတယ်။ ဒီအလွှာက Automated Malware တွေကို တားဆီးပေးသလို Administrator ကိုယ်တိုင်တောင် မတော်တဆ စနစ်အရေးကြီးအရာတွေ ပြောင်းမိတာကိုလည်း ကာကွယ်ပေးပါတယ်။ ဒါကြောင့် UAC က Windows လုံခြုံရေးအတွက် အရေးကြီးတဲ့ အခန်းကဏ္ဍတစ်ခုဖြစ်ပါတယ်။

ဒါပေမယ့် လက်တွေ့မှာတော့ Windows ကို ပုံမှန်အလုပ်လုပ်နိုင်အောင် ပြုလုပ်ဖို့အတွက် UAC ကို ကျော်လွှားနိုင်တဲ့ နည်းလမ်းတွေကို Microsoft ကိုယ်တိုင်က ထည့်သွင်းထားရပါတယ်။ ဒီနည်းလမ်းတွေကို **UAC Bypass** လို့ခေါ်ပါတယ်။ အဓိကအားဖြင့် Windows ရဲ့ ဒီဇိုင်းအရ ယုံကြည်ထားတဲ့ Component တွေကို Administrator အခွင့်အရေးနဲ့ အလိုအလျောက် လုပ်ခွင့်ပေးထားတာကြောင့် ဒီလို Bypass ဖြစ်နိုင်တာပါ။ Attackers တွေကတော့ ဒီသဘောတရားကို အသုံးချပြီး Administrator Prompt မပေါ်ဘဲ Privilege မြှင့်နိုင်အောင် ကြိုးစားကြပါတယ်။

UAC Bypass နည်းလမ်းတွေမှာ အများဆုံးတွေ့ရတာက

- DLL Hijacking,
- Elevated COM Interface အသုံးပြုခြင်း,
- File သို့မဟုတ် Registry ကို ပြောင်းလဲခြင်း,
- Scheduled Task တွေကို အသုံးချခြင်း စတဲ့ နည်းလမ်းတွေဖြစ်ပါတယ်။

ဒီနည်းလမ်းတွေက Windows ရဲ့ယုံကြည်ထားတဲ့ လုပ်ဆောင်ချက်တွေကို အသုံးချတာဖြစ်လို့ UAC Prompt မပေါ်ဘဲ Administrator အခွင့်အရေးနဲ့ Code ကို ပြေးနိုင်ပါတယ်။

UAC ကို ဘယ်လိုအမြင်နဲ့ နားလည်သင့်လဲ

UAC အကြောင်းကို စပြီး သိထားရမယ့် အရေးကြီးဆုံးအချက်ကတော့ **UAC ကို လုံခြုံရေးအတွက် အပြည့်အဝ အားထားလို့မရဘူး** ဆိုတာပါ။ တကယ်လို့ Attacker တစ်ယောက်က စနစ်တစ်ခုထဲကို ဝင်

ရောက်နိုင်ပြီး၊ အဲ့ဒီ User က Local Administrator အခွင့်အရေးရှိနေပြီဆိုရင် Defender ဘက်ကနေ “သူ System Privilege ရပြီးသားလို့” စဉ်းစားထားသင့်ပါတယ်။ ဘာကြောင့်လဲဆိုတော့ Attacker က UAC Bypass နည်းလမ်းတစ်ခုခု သုံးနိုင်တယ်၊ User ကို လှည့်စားပြီး UAC Prompt ကို ကိုယ်တိုင် Yes နှိပ်အောင်လုပ်နိုင်တယ်၊ ဒါမှမဟုတ် Vulnerability တစ်ခုကို Exploit လုပ်ပြီး Privilege မြှင့်နိုင်ပါတယ်။

ဒါပေမယ့် UAC က ဘယ်အချိန်မှာ အဓိက အရေးပါလာလဲဆိုရင် Attacker ဟာ **Medium Integrity Process** တစ်ခုထဲမှာပဲ ရှိနေပြီး Administrator Credential တစ်ခုမှ မရှိသေးတဲ့အချိန်ပါ။ Medium Integrity ဆိုတာက ပုံမှန် User Program တစ်ခုလိုပဲ အခွင့်အရေးကန့်သတ်ထားတဲ့ Process ဖြစ်ပါတယ်။ အဲ့ဒီအခြေအနေမှာပဲ UAC Prompt က Attacker ကို တားဆီးပေးနိုင်ပါတယ်။

ဒီနေရာမှာ Remote Management ဆိုတဲ့ လက်တွေ့အသုံးချမှုကို စဉ်းစားကြည့်ရပါမယ်။ Administrator တွေအနေနဲ့ ကွန်ပျူတာ အများကြီးကို တစ်ခါတည်း စီမံခန့်ခွဲရပါတယ်။ အဲ့ဒီအချိန်တိုင်း UAC Prompt ကို Workstation တစ်လုံးချင်းစီမှာ သွားပြီး Confirm လုပ်နေရရင် အလုပ်မဖြစ်နိုင်ပါဘူး။ ဒါကြောင့် Remote Management Tool တွေက UAC Prompt မလိုဘဲ High Integrity သို့မဟုတ် System Integrity Process ကို တိုက်ရိုက်ဖန်တီးနိုင်အောင် ဒီဇိုင်းလုပ်ထားပါတယ်။ Attacker က User ရဲ့ Credential ကို ရရှိသွားရင် အဲ့ဒီ Mechanism တွေကို အသုံးချပြီး Privilege မြင့်တဲ့ Process ကို ရနိုင်ပါတယ်။

ဥပမာအနေနဲ့

- WMI ကို သုံးပြီး Lateral Movement လုပ်ရင် WmiPrvSE.exe ကနေ High Integrity Process တစ်ခု ဖန်တီးနိုင်ပါတယ်။
- Remote Service Execution ကို သုံးရင် services.exe ကနေ System Integrity Process တစ်ခု ထွက်လာပါတယ်။ ဒီအခြေအနေတွေမှာတော့ UAC က လုံးဝ အလုပ်မလုပ်တော့ပါဘူး။

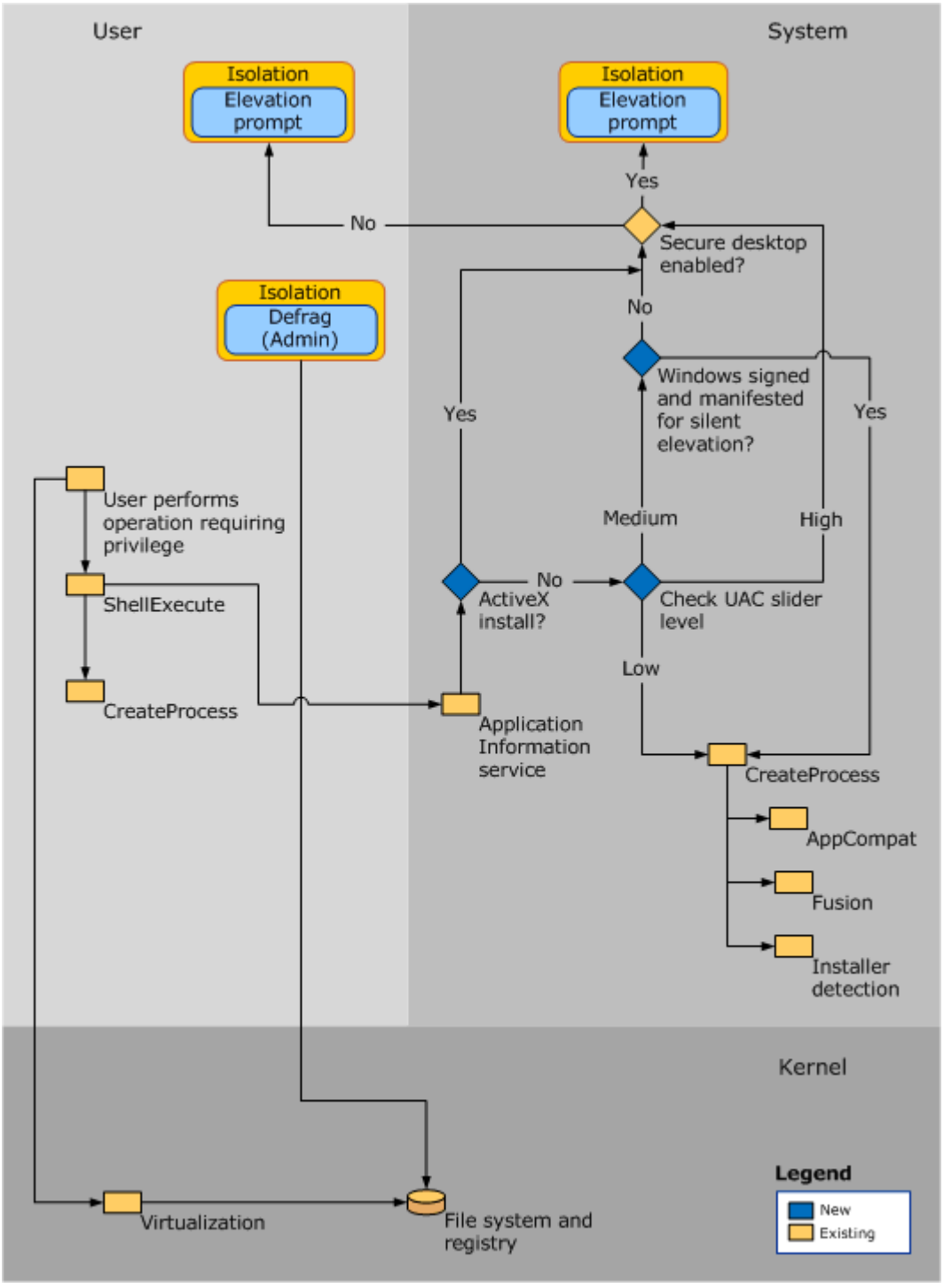
levels of protection

UAC မှာ **Protection Level မတူတဲ့ Setting မျိုးစုံ** ရှိပြီး Control Panel သို့မဟုတ် Group Policy ကနေ ပြင်ဆင်နိုင်ပါတယ်။ ဒီ Level တွေက UAC Prompt ဘယ်လောက်ခဏခဏပေါ်မလဲ၊ ဘယ်လောက်တင်းကြပ်လဲ ဆိုတာကို ဆုံးဖြတ်ပေးပါတယ်။ User အနေနဲ့ လုံခြုံရေးနဲ့ အသုံးအဆောင်မှုကြား Balance လုပ်ဖို့ ရည်ရွယ်ထားတာပါ။ ဒါပေမယ့် အချို့ UAC Bypass နည်းလမ်းတွေက Always Notify လို အမြင့်ဆုံး Setting မှာတောင် အလုပ်လုပ်နိုင်ပါတယ်။ အဲ့ဒီလိုဆိုပေမယ့် Always Notify က တခြား Setting တွေထက် အများကြီး လုံခြုံပါတယ်။

- **Always Notify** ဆိုတဲ့ Setting မှာတော့ Program တစ်ခုခုက စနစ်ကို ပြောင်းချင်တဲ့အချိန်တိုင်း၊ ဒါမှမဟုတ် Windows Setting အရေးကြီးတွေ ပြောင်းချင်တဲ့အချိန်တိုင်း UAC Prompt မဖြစ်မနေ ပေါ်လာပါတယ်။ User က ကိုယ်တိုင် ခွင့်ပြုမှသာ ဆက်လုပ်နိုင်ပါတယ်။ Prompt ပေါ်လာချိန်မှာ Desktop ကို Dim လုပ်ထားတဲ့အတွက် အခြား Program တွေက ကြားဝင် မလုပ်နိုင်ပါဘူး။
- **Default Setting** ဖြစ်တဲ့ Notify me only when programs try to make changes မှာတော့ Program က ပြောင်းလဲမှုလုပ်တဲ့အခါပဲ Prompt ပေါ် ပြီး User ကိုယ်တိုင် Windows Setting ပြောင်းတဲ့အခါ Prompt မပေါ်ပါဘူး။ ဒါပေမယ့် Desktop Dim လုပ်ထားသေးတဲ့အတွက် လုံခြုံရေးအဆင့်က မဆိုးသေးပါဘူး။
- **Do not dim Desktop** ဆိုတဲ့ Dim မလုပ်တဲ့ Setting ကတော့ ပိုမိုအန္တရာယ်ရှိပါတယ်။ ဘာကြောင့်လဲဆိုတော့ UAC Prompt ပေါ်နေတဲ့အချိန်မှာ အခြား Application တွေက Interaction လုပ်နိုင်ပြီး Fake Prompt တင်ပြတဲ့ Spoofing Attack ဖြစ်နိုင်လို့ပါ။

- **Never Notify** ကတော့ လုံးဝ မလုံခြုံတဲ့ Setting ဖြစ်ပါတယ်။ Program တစ်ခုခုက Administrator အခွင့်အရေးနဲ့ အလုပ်လုပ်ချင်ရင် Prompt မပေါ်ဘဲ တိုက်ရိုက် ဆက်လုပ်နိုင်ပါတယ်။ ဒီ Setting ကို အသုံးပြုတာဟာ UAC ကို ပိတ်ထားသလိုပဲ ဖြစ်ပြီး လုံခြုံရေးအန္တရာယ် အရမ်းမြင့်ပါတယ်။

UAC က ပေးနိုင်တဲ့ လုံခြုံရေးအကျိုးကျေးဇူးကို အနည်းဆုံးရချင်ရင် **Always Notify ကို မဖြစ်မနေ သုံးသင့်ပါတယ်**။ အကြောင်းကတော့ UAC Bypass အများစုဟာ Silent Elevation ခွင့်ပြုထားတဲ့ Feature တွေကို အသုံးချတာဖြစ်လို့ပါ။ Attacker တွေက Process ရဲ့ Process Environment Block (PEB) ကို ပြင်ပြီး ကိုယ့် Process ကို explorer.exe လို ယုံကြည်ထားတဲ့ Process တစ်ခုလို့ စနစ်ကို လှည့်စားနိုင်ပါတယ်။ အဲဒီလိုလုပ်ရင် UAC Prompt မပေါ်ဘဲ Privilege မြင့်သွားနိုင်ပါတယ်။



Microsoft ရဲ့ UAC Architecture Diagram ကို ကြည့်မယ်ဆိုရင် User က Privilege လိုတဲ့ Operation တစ်ခု လုပ်တဲ့အချိန်ကစပြီး System အထိ လမ်းကြောင်းကို မြင်နိုင်ပါတယ်။ အဲဒီလမ်းကြောင်းထဲမှာ UAC Slider Level ကို စစ်တဲ့ အဆင့်တစ်ခု ရှိပါတယ်။ Slider ကို Always Notify လို့ထားထားရင် UAC

Prompt ကို မဖြစ်မနေ မြင်ရပါမယ်။ ဒါပေမယ့် အဲ့ဒီအောက် Level တွေမှာတော့ System က Source Process ဟာ Signed ဖြစ်လား၊ Silent Elevation ခွင့်ပြုထားတဲ့ Process လားဆိုတာပဲ စစ်ပြီး Prompt မပေါ်ဘဲ ဆက်လုပ်သွားပါတယ်။

Bypass Methodology

အများဆုံးတွေ့ရတဲ့ UAC Bypass ဖြစ်လာတဲ့ အခြေအနေတစ်ခုကို စဉ်းစားကြည့်ပါမယ်။ Attacker တစ်ယောက်က Phishing Email သို့မဟုတ် လှည့်စားတဲ့ နည်းလမ်းတစ်ခုခုနဲ့ User ကို မိမိရဲ့ Payload ကို ဖွင့်အောင် လုပ်နိုင်ခဲ့တယ်ဆိုပါစို့။ အဲ့ဒီ User ဟာ Local Administrator အခွင့်အရေးရှိတဲ့သူ ဖြစ်နိုင်ပါတယ်။ ဒါပေမယ့် User က Payload ကို Right Click လုပ်ပြီး “Run as administrator” လုပ်မထားဘူးဆိုရင် Windows က အဲ့ဒီ Program ကို **Medium Integrity Process** အနေနဲ့ပဲ ပြေးစေပါတယ်။ ဒီအချိန်မှာ Attacker က Code Execution ရထားပေမယ့် စနစ်အဆင့်မှာတော့ Standard User လိုပဲ ကန့်သတ်ထားပါတယ်။

Medium Integrity Process ထဲမှာ ရှိနေရင် Attacker က Network ပေါ်မှာ Domain User အနေနဲ့ အလုပ်အတော်များများ လုပ်နိုင်ပါတယ်။ ဒါပေမယ့် Local Workstation ထဲမှာတော့ Administrator လို အခွင့်အရေးအပြည့်မရှိသေးပါဘူး။ ဒီကန့်သတ်ချက်ကြောင့် Credential Dumping လို အရေးကြီးတဲ့ Attack နည်းလမ်းအများစုကို မလုပ်နိုင်ပါဘူး။ Windows မှာ Credential တွေကို ခိုးယူဖို့ဆိုရင် High Integrity သို့မဟုတ် System Integrity Process ထဲမှာ Code ကို run နိုင်ဖို့ လိုအပ်ပါတယ်။

- ဒီလို Privilege မြှင့်ဖို့အတွက် Attacker တွေက UAC ကို ကျော်လွှားရပါမယ်။ အဲ့ဒီအတွက် အသုံးများတဲ့ နည်းလမ်းတွေဟာ Windows ရဲ့ ယုံကြည်ထားတဲ့ Mechanism တွေကို အသုံးချတာဖြစ်ပါတယ်။ ဥပမာအနေနဲ့ High သို့မဟုတ် System Integrity နဲ့ ပြေးနေတဲ့ Process တစ်ခုက အသုံးပြုနေတဲ့ **File, Registry Key သို့မဟုတ် DLL ကို ပြောင်းလဲခြင်း**ဖြစ်နိုင်ပါတယ်။ ဒါမှမဟုတ် Windows ကိုယ်တိုင် အမြင့်ဆုံး Privilege နဲ့ အလုပ်လုပ်ရမယ့် Feature တစ်ခုခုကို အလွဲသုံးစားလုပ်နိုင်ပါတယ်။ အများဆုံးတွေ့ရတာက **COM Interface တွေကို အသုံးချခြင်း**ဖြစ်ပါတယ်။

DLL Hijacking

DLL Hijacking ဆိုတာကတော့ Windows ရဲ့ DLL Load လုပ်တဲ့ အစဉ်အလာကြောင့် ဖြစ်လာတဲ့ အားနည်းချက်ကို အသုံးချတာပါ။ Windows Program တစ်ခုက DLL ကို Load လုပ်တဲ့အခါ သတ်မှတ်ထားတဲ့ Search Order တစ်ခုအတိုင်း ဖိုင်တွေကို လိုက်ရှာပါတယ်။ အဲ့ဒီ Search Path ထဲက နေရာတစ်ခုခုမှာ Attacker က မိမိရဲ့ Malicious DLL ကို ထားနိုင်ခဲ့ရင် Program က အမှန်တကယ် အသုံးပြုသင့်တဲ့ DLL အစား အဲ့ဒီ Malicious DLL ကို Load လုပ်ပြီး Execute လုပ်သွားပါတယ်။ အဲ့ဒီ Program က Auto-Elevation ခွင့်ရှိတဲ့ Process ဖြစ်နေခဲ့ရင် Attacker ရဲ့ Code က High သို့မဟုတ် System Integrity နဲ့ ပြေးသွားနိုင်ပါတယ်။

Some examples of DLL hijacking opportunities for auto-elevation processes:

- DismCore.dll
- atl.dll
- iscsiexe.dll
- BluetoothDiagnosticUtil.dll

- GdiPlus.dll

ဥပမာတချို့ကို ကြည့်ရင် dism.exe က အသုံးပြုတဲ့ DLL တွေဟာ မူလရှိသင့်တဲ့ Folder အပြင်ကနေ DLL ကို ရှာတာကြောင့် DismCore.dll ကို မိမိရှိနေတဲ့ Folder ထဲမှာ ထားနိုင်ရင် Load လုပ်သွားနိုင်ပါတယ်။ WMI MMC က atl.dll ကို မတွေ့တဲ့အခါ Attacker ထားထားတဲ့ DLL ကို Load လုပ်နိုင်ပါတယ်။ iscsicpl.exe နဲ့ sdiagnhost.exe တို့ကလည်း User ရဲ့ Path ထဲကို ကြည့်ပြီး DLL ကို Load လုပ်တဲ့အတွက် DLL Hijacking ဖြစ်နိုင်ပါတယ်။ dccw.exe မှာလည်း GdiPlus.dll ကို အသုံးချနိုင်ပါတယ်။ ဒီဥပမာတွေဟာ အနည်းငယ်သာ ဖြစ်ပြီး တခြား သိပြီးသား၊ မသိသေးတဲ့ DLL Hijacking အခွင့်အလမ်းတွေ အများကြီး ရှိနေပါသေးတယ်။

ဒီနေရာမှာ မေးစရာရှိလာတာက **Attacker က ဘယ်လိုလုပ်ပြီး Administrator ကာကွယ်ထားတဲ့ Folder တွေထဲကို DLL ကို ထည့်နိုင်တာလဲ**ဆိုတာပါ။ အများအားဖြင့် ဒီအလုပ်ကို **Elevated COM Interface** တွေကို အသုံးချပြီး လုပ်ပါတယ်။ ဥပမာ IFileOperation လို COM Object တွေဟာ Administrator အခွင့်အရေးနဲ့ File Copy လုပ်နိုင်ပါတယ်။ UAC ကို Always Notify မထားထားဘူးဆိုရင် ဒီ COM Interface ကို သုံးပြီး Protected Location ထဲကို File တွေ ရွှေ့လို့ရပါတယ်။

COM Abuse

COM Abuse ဆိုတာကတော့ File Copy တင်မကပါဘူး။ COM Object အချို့ဟာ ကိုယ်တိုင်ပဲ Elevated Context ထဲမှာ Code ကို Execute လုပ်နိုင်ပါတယ်။ အချို့ COM Interface တွေမှာ Attacker က ကိုယ်တိုင် မိမိရဲ့ Malicious Executable ကို Parameter အနေနဲ့ ပေးပြီး Execute လုပ်နိုင်ပါတယ်။ တချို့ COM Interface တွေမှာတော့ User ရဲ့ Path ထဲမှာ သို့မဟုတ် သတ်မှတ်ထားတဲ့ Location တစ်ခုမှာ Executable ကို ထားထားရပါတယ်။

- ICMLuaUtil,
- IColorDataProxy,
- IEditionUpgradeManager,
- IElevatedFactoryServer တို့လို COM Interface တွေဟာ ဒီလို UAC Bypass အတွက် အသုံးများတဲ့ ဥပမာတွေပါ။

Registry Key Modifications

Registry Key Modification ကတော့ Windows Component တွေက ယုံကြည်ထားတဲ့ Registry Setting တွေကို လှည့်စားတဲ့ နည်းလမ်းပါ။ Auto-Elevation ခွင့်ရှိတဲ့ Program တစ်ခုက Registry ထဲမှာ သတ်မှတ်ထားတဲ့ Path ကို ဖတ်ပြီး Program ကို Run လုပ်တဲ့အခါ Attacker က အဲ့ဒီ Key ကို ပြောင်းပြီး မိမိရဲ့ Malicious Executable ကို ပြေးအောင် လုပ်နိုင်ပါတယ်။ ဒီလိုလုပ်ရင် UAC Prompt မပေါ်ဘဲ High Integrity Process ထဲမှာ Code Execution ရနိုင်ပါတယ်။

Scheduled Tasks

Scheduled Task ကို အသုံးချတဲ့ နည်းလမ်းမှာတော့ “Run with highest privileges” လို့ သတ်မှတ်ထားတဲ့ Task တွေကို Target လုပ်ပါတယ်။ အချို့ Task တွေမှာ Executable Path ထဲမှာ Environment Variable တွေကို အသုံးပြုထားပါတယ်။ Attacker က အဲ့ဒီ Variable ကို ပြောင်းနိုင်ခဲ့ရင် မိမိရဲ့ Malicious Executable ကို မကာကွယ်ထားတဲ့ Location ကနေ ပြေးစေနိုင်ပါတယ်။ ဒါအပြင် Elevated COM Object တွေကို အသုံးချပြီး Scheduled Task အသစ်တွေကို Register လုပ်ပြီး Execute လုပ်တာလည်း ဖြစ်နိုင်ပါတယ်။

IFileOperation + DLL Hijacking

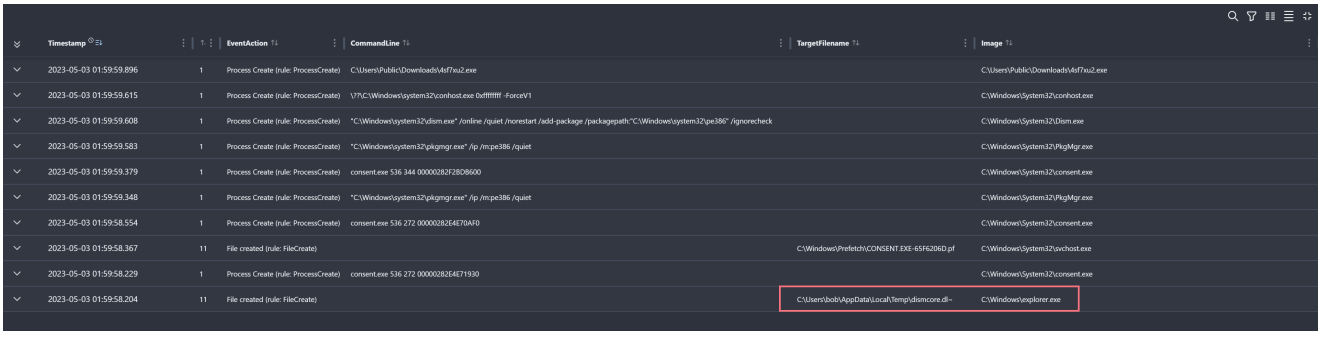
IFileOperation ဆိုတာက Windows Shell ထဲမှာ File တွေကို Copy, Move, Delete လုပ်ဖို့ အသုံးပြုတဲ့ COM Interface တစ်ခုဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် Administrator အခွင့်အရေးလိုတဲ့ Folder တွေဖြစ်တဲ့ System32 လိုနေရာတွေကို File Copy လုပ်မယ်ဆိုရင် UAC Prompt ပေါ်သင့်ပါတယ်။ ဒါပေမယ့် UAC ကို Always Notify မထားထားတဲ့ အခြေအနေမှာ Administrator ဖြစ်တဲ့ User က Medium Integrity Process ထဲမှာပဲ ရှိနေသော်လည်း IFileOperation ကို အသုံးပြုပြီး ကန့်သတ်ထားတဲ့ Folder တွေထဲကို File တွေကို UAC Prompt မပေါ်ဘဲ ရွှေ့နိုင်ပါတယ်။ ဒီအချက်ကို Attacker တွေက အလွန်အသုံးများပါတယ်။

IFileOperation ကို သီးသန့်သုံးတာထက် DLL Hijacking နဲ့ တွဲသုံးတဲ့အခါ အန္တရာယ်ပိုကြီးလာပါတယ်။ DLL Hijacking ဆိုတာက Auto-Elevation ခွင့်ရှိတဲ့ Windows Program တစ်ခုက DLL ကို Load လုပ်တဲ့ အချိန်မှာ Search Order ကို အသုံးချပြီး Malicious DLL ကို Load လုပ်စေတဲ့ နည်းလမ်းဖြစ်ပါတယ်။ Windows System တစ်ခုထဲမှာ DLL Hijacking လုပ်လို့ရနိုင်တဲ့ Program တွေ အများကြီးရှိပြီး Attacker က IFileOperation ကို သုံးနိုင်ရင် အဲ့ဒီ Hijacking Opportunity အားလုံးကို အသုံးချနိုင်ပါတယ်။

အခု Log Analysis ကို ကြည့်ကြပါစို့။

```
external_table("Winlog")
| where Timestamp > datetime(2023-05-03T01:59:58.100Z)
    and Timestamp < datetime(2023-05-03T01:59:59.900Z)
    and (EventCode == 1 or EventCode == 11)
| project Timestamp, EventCode, EventAction, CommandLine, TargetFilename,
Image
```

Winlog Table ကို Query လုပ်ပြီး သတ်မှတ်ထားတဲ့ အချိန်အတွင်း Process Creation Event နဲ့ File Creation Event တွေကို စစ်ထားပါတယ်။



Timestamp	EventCode	EventAction	CommandLine	TargetFilename	Image
2023-05-03 01:59:59.896	1	Process Create (rule: ProcessCreate)	C:\Users\Public\Downloads\4sf7xu2.exe		C:\Users\Public\Downloads\4sf7xu2.exe
2023-05-03 01:59:59.615	1	Process Create (rule: ProcessCreate)	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1		C:\Windows\System32\conhost.exe
2023-05-03 01:59:59.608	1	Process Create (rule: ProcessCreate)	"C:\Windows\system32\dism.exe" /online /quiet /norestart /add-package /packagepath:"C:\Windows\system32\pe386" /ignorecheck		C:\Windows\System32\Dism.exe
2023-05-03 01:59:59.583	1	Process Create (rule: ProcessCreate)	"C:\Windows\system32\pkgmgr.exe" /ip /m:pe386 /quiet		C:\Windows\System32\PkgMgr.exe
2023-05-03 01:59:59.379	1	Process Create (rule: ProcessCreate)	consent.exe 536 344 00000282F28DB600		C:\Windows\System32\consent.exe
2023-05-03 01:59:59.348	1	Process Create (rule: ProcessCreate)	"C:\Windows\system32\pkgmgr.exe" /ip /m:pe386 /quiet		C:\Windows\System32\PkgMgr.exe
2023-05-03 01:59:58.554	1	Process Create (rule: ProcessCreate)	consent.exe 536 272 00000282E4E70AF0		C:\Windows\System32\consent.exe
2023-05-03 01:59:58.367	11	File created (rule: FileCreate)		C:\Windows\Prefetch\CONSENT.EXE-65F6206D.pf	C:\Windows\System32\svchost.exe
2023-05-03 01:59:58.229	1	Process Create (rule: ProcessCreate)	consent.exe 536 272 00000282E4E71930		C:\Windows\System32\consent.exe
2023-05-03 01:59:58.204	11	File created (rule: FileCreate)		C:\Users\bob\AppData\Local\Temp\dismcore.dl~	C:\Windows\explorer.exe

ဒီ Log ထဲမှာ အရင်ဆုံး တွေ့ရတဲ့ Process Creation Event က Akagi64.exe ဖြစ်ပြီး ဒါဟာ Attacker ရဲ့ Payload ဖြစ်ပါတယ်။ ဒီ Process ဟာ Medium Integrity နဲ့ Run နေတဲ့ Attacker ရဲ့ Control Process ဖြစ်ပါတယ်။

နောက်ထပ် တွေ့ရတဲ့ Event ကတော့ File Creation Event ဖြစ်ပြီး Attacker က Malicious DLL ကို Disk ထဲကို Drop လုပ်ထားတာပါ။ IFileOperation ကို သုံးဖို့အတွက် Source File နဲ့ Destination File လိုအပ်လို့ ဒီ DLL ကို အရင်ဆုံး Create လုပ်ရတာပါ။ ထူးခြားတာက Log ကို ကြည့်မယ်ဆိုရင် ဒီ File ကို explorer.exe က Create လုပ်သလို ပေါ်နေပါတယ်။ ဒါပေမယ့် အမှန်တကယ်တော့ explorer.exe မဟုတ်ပါဘူး။

Winlog Event Data ထဲက ProcessId ကို ကြည့်မယ်ဆိုရင် PID 1328 ဖြစ်ပြီး ဒီ PID က Akagi64.exe ရဲ့ PID နဲ့ ကိုက်ညီပါတယ်။ ဒီလို ဖြစ်လာရတဲ့ အကြောင်းရင်းကတော့ Attacker က Elevated COM Object ကို Invoke လုပ်နိုင်ဖို့ Windows ကို လှည့်စားထားလို့ပါ။ Windows က Silent Elevation ခွင့်ပေးမပေး စစ်တဲ့အချိန်မှာ Source Process ရဲ့ Digital Signature ကို အပြည့်အစုံ မစစ်ဘဲ Process Environment Block (PEB) ကိုပဲ စစ်ပါတယ်။ Attacker က DLL Drop မလုပ်ခင်မှာ PEB ကို Patch လုပ်ပြီး ကိုယ့် Process ကို explorer.exe လို ယုံကြည်ထားတဲ့ Process တစ်ခုလို ပြုလုပ်ထားပါတယ်။ အဲ့ဒီကြောင့် Log မှာ explorer.exe က File Create လုပ်သလို ထင်ရတဲ့ ထူးခြားတဲ့ Artifact တစ်ခု ကျန်ခဲ့တာပါ။

အဲ့ဒီနောက် Malicious dismcore.dll ကို IFileOperation ကို အသုံးပြုပြီး C:\Windows\System32 ထဲကို Copy လုပ်ပါတယ်။ ဒီအဆင့်ကို ဒီ Case မှာ Log မတွေ့ရပေမယ့် ပုံမှန်အားဖြင့် ဒီလိုလုပ်တဲ့အခါ DllHost.exe ဆိုတဲ့ Process တစ်ခု Run တာကို တွေ့ရတတ်ပါတယ်။ Command Line ကတော့ DllHost.exe နဲ့ COM CLSID တစ်ခုပါတဲ့ ပုံစံဖြစ်ပါတယ်။

> DllHost.exe /Processid:{3AD05575-8857-4850-9277-11B85BDB8E09}

ဒီ Process က Administrator အခွင့်အရေးနဲ့ Run ပြီး System32 ထဲကို DLL ကို Copy လုပ်ပေးပါတယ်။ နောက်တစ်ဆင့်အနေနဲ့ DllHost.exe က `C:\Windows\System32\dismcore.dll` ကို Create လုပ်တဲ့ File Creation Event တစ်ခု ထပ်ပေါ်တတ်ပါတယ်။

နောက်ဆုံးအဆင့်မှာ Attacker က dism.exe ကို Run အောင် လုပ်ရပါတယ်။ ဒါကို တိုက်ရိုက် Run မလုပ်ဘဲ pkgmgr.exe ကို Start လုပ်ပါတယ်။ pkgmgr.exe က dism.exe ကို ခေါ် ပြီး Run ပေးတဲ့ Windows Component တစ်ခုဖြစ်ပါတယ်။ dism.exe ဟာ Auto-Elevation ခွင့်ရှိတဲ့ Program ဖြစ်တဲ့အတွက် Start လုပ်တဲ့အချိန်မှာ System32 ထဲက dismcore.dll ကို Load လုပ်ပါတယ်။ ဒီအချိန်မှာ Attacker ထားထားတဲ့ Malicious dismcore.dll ကို Load လုပ်သွားပြီး High Integrity Context ထဲမှာ Execute ဖြစ်သွားပါတယ်။

ဒီ Malicious DLL ထဲမှာတော့ Attacker ရဲ့ Implant ဖြစ်တဲ့ 4sf7xu2.exe ကို Unprotected Location တစ်ခုကနေ Execute လုပ်တဲ့ Code ပါဝင်ပါတယ်။ အဲ့ဒီအချိန်ကစပြီး Attacker ဟာ Medium Integrity ကနေ High Integrity သို့မဟုတ် System Level အထိ တက်သွားနိုင်ပြီး Credential Dumping လို အန္တရာယ်ကြီးတဲ့ Attack တွေကို ဆက်လုပ်နိုင်သွားပါတယ်။

Test : Spoofed explorer.exe

The attacker also spoofed explorer.exe when creating the file ATL.dl~. What is the process ID of the attacker's process that created the file?

အရင်ဆုံး နားလည်ထားရမယ့် အချက်က Attacker ဟာ အမှန်တကယ် explorer.exe ကို အသုံးမပြုဘဲ၊ **ကိုယ့် Process ကို explorer.exe လို လှည့်စားထားတာ** ဖြစ်ပါတယ်။ Windows က Elevated COM Object တွေကို ခေါ်ခွင့်ပေးမပေး စစ်တဲ့အခါ Source Process ရဲ့ Signature ကို အပြည့်အစုံမစစ်ဘဲ Process Environment Block (PEB) ထဲက Image Name ကို အခြေခံပြီး စစ်တာကို Attacker က အသုံးချထားတာပါ။ ဒါကြောင့် Log ထဲမှာ explorer.exe က လုပ်သလို ပေါ်နေပေမယ့် အမှန်တကယ်တော့ တခြား Malware Process က လုပ်ထားတာဖြစ်ပါတယ်။

```
external_table('Winlog')
| where EventCode == 11
| where Message contains "ATL.dl"
| project ProcessId
```

ဒီအရာကို အတည်ပြုဖို့အတွက် Log ထဲမှာ **ATL.dll (သို့) ATL.dl~** ဖိုင် Create ဖြစ်တဲ့ Event ကို စစ်ပါတယ်။ ATL.dll ဆိုတာ DLL Hijacking အတွက် မကြာခဏ အသုံးချခံရတဲ့ DLL ဖြစ်ပြီး၊ explorer.exe က Create လုပ်သလို ပေါ်နေတဲ့ File Creation Event တွေဟာ အထူးသတိထားစရာ ဖြစ်ပါတယ်။ Query ထဲမှာ EventCode 11 ကို သုံးထားတာက File Creation Event ကို ရှာဖို့ဖြစ်ပြီး Image ထဲမှာ explorer.exe ပါတာကို Filter လုပ်ထားပါတယ်။ TargetFilename ထဲမှာ ATL.dl ပါတာကို စစ်တာက Hijacking ဆိုင်ရာ Artifact ကို ရှာဖို့ ဖြစ်ပါတယ်။

```
external_table("Winlog")
| where EventCode == 11
    and Image contains "explorer.exe"
    and TargetFilename contains "ATL.dl"
| project Timestamp, EventCode, Image, TargetFilename, ProcessId
```

ဒီ Query ရဲ့ Result ကို ကြည့်လိုက်ရင် winlog.event_data.ProcessId က `6960` ဖြစ်ပါတယ်။ အရေးကြီးတာက ဒီ Process ID ကို တခြား Process Creation Event တွေနဲ့ ပြန်ချိတ်ကြည့်လိုက်တဲ့အခါ၊ PID 6960 ဟာ explorer.exe မဟုတ်ဘဲ `"C:\Users\bob\Desktop\Akagi64.exe"` ဖြစ်နေတာကို တွေ့ရပါတယ်။ ဒါက Attacker ရဲ့ Malware Process ဖြစ်ပြီး explorer.exe လို ဟန်ဆောင်ထားတာ ဖြစ်ပါတယ်။ ဒီအချက်တစ်ခုတည်းနဲ့တင် explorer.exe ကို Spoof လုပ်ထားတယ်ဆိုတာကို သက်သေပြနိုင်ပါတယ်။

ဒါပေမယ့် Process ID ကိုပဲ အားထားရင် အချိန်ကြာတဲ့ Log တွေမှာ **PID Collision** ဖြစ်နိုင်ပါတယ်။ PID Collision ဆိုတာက Process တစ်ခု ပိတ်သွားပြီး နောက်ထပ် Process တစ်ခုက အဲ့ဒီ PID ကို ပြန်အသုံးပြုတာမျိုးပါ။ ဒီလိုအခြေအနေကို ရှောင်ရှားဖို့ ပိုမိုယုံကြည်စိတ်ချရတဲ့ နည်းလမ်းက **Process GUID** ကို အသုံးပြုတာပါ။

File Creation Event ထဲမှာ ပါတဲ့ ProcessGuid ကို ယူပြီး Process Creation Event (EventCode 1) တွေနဲ့ ပြန်ရှာကြည့်ရင် အဲ့ဒီ GUID နဲ့ ဆက်နွယ်နေတဲ့ Process ကို တိတိကျကျ ဖော်ထုတ်နိုင်ပါတယ်။ Query ထဲမှာ ProcessGuid ကို Filter လုပ်ထားတာက အဲ့ဒီ Process ကို တစ်ခုတည်းတိတိကျကျ Identify လုပ်ဖို့ ဖြစ်ပါတယ်။ ဒီလို စစ်ကြည့်လိုက်တဲ့အခါ Image နဲ့ CommandLine ထဲမှာ Akagi64.exe နဲ့ Attacker ရဲ့ Payload တွေ ပါနေတာကို တွေ့ရပါလိမ့်မယ်။

```
external_table("Winlog")
| where EventCode == 1
    and ProcessGuid contains "2fe0b2e9-c5fb-6451-9805-000000000c00"
| project Timestamp, EventCode, Image, CommandLine, ProcessId
```

ဒီအရာက ဘာကို ပြသလဲဆိုရင် Log ထဲမှာ explorer.exe လို့ ပေါ်နေတဲ့ File Creation Event ကို တစ်ခါတည်း ယုံကြည်လို့မရဘူးဆိုတာပါ။ Attacker က PEB ကို Patch လုပ်ပြီး explorer.exe လို လှည့်စားထားနိုင်ပါတယ်။ Forensics သို့မဟုတ် Detection လုပ်တဲ့အခါ File Creation Event ထဲက

Image Name တစ်ခုတည်းကို မကြည့်ဘဲ Process ID, Process GUID နဲ့ Process Creation Event တွေကို ချိတ်ဆက်စစ်ဆေးရင် Spoofed explorer.exe လို UAC Bypass Artifact တွေကို ဖော်ထုတ်နိုင်ပါတယ်။

အကျဉ်းချုပ်ပြောရရင် ဒီ Test က **UAC Bypass + DLL Hijacking + COM Abuse** တိုက်ခိုက်မှုတွေမှာ Attacker တွေက explorer.exe ကို ဘယ်လို လှည့်စားတတ်သလဲ၊ Defender ဘက်က ဘယ်လို Log Correlation လုပ်ပြီး အမှန်တရားကို ဖော်ထုတ်နိုင်လဲဖြစ်ပါတယ်။

Registry Manipulation

အရင်ဆုံး Winlog ကို Query လုပ်ပြီး သတ်မှတ်ထားတဲ့ အချိန်အတွင်း ဖြစ်ပေါ် ခဲ့တဲ့ Event တွေကို ကြည့်ပါတယ်။

```
external_table("Winlog")
| where Timestamp > datetime(2023-05-03T01:57:42.044Z)
    and Timestamp < datetime(2023-05-03T01:57:42.710Z)
| project Timestamp, EventCode, EventAction, CommandLine, TargetObject,
Details
```

Timestamp	EventCode	EventAction	CommandLine	TargetObject	Details
2023-05-03 01:57:42.706	1	Process Create (rule: ProcessCreate)	"C:\Users\bob\Desktop\mtyqs.exe"		
2023-05-03 01:57:42.705	4688	Process Creation	"C:\Users\bob\Desktop\mtyqs.exe"		
2023-05-03 01:57:42.673	13	Registry value set (rule: RegistryEvent)		HKU\S-1-5-21-1072563919-790901262-2159826448-4195\Software\Microsoft\Windows\Curre	Visited:
2023-05-03 01:57:42.672	13	Registry value set (rule: RegistryEvent)		HKU\S-1-5-21-1072563919-790901262-2159826448-4195\Software\Microsoft\Windows\Curre	Cookie:
2023-05-03 01:57:42.672	13	Registry value set (rule: RegistryEvent)		HKU\S-1-5-21-1072563919-790901262-2159826448-4195\Software\Microsoft\Windows\Curre	(Empty)
2023-05-03 01:57:42.637	13	Registry value set (rule: RegistryEvent)		HKU\S-1-5-21-1072563919-790901262-2159826448-4195\Software\Microsoft\Windows\Curre	Binary Data
2023-05-03 01:57:42.457	13	Registry value set (rule: RegistryEvent)		HKU\S-1-5-21-1072563919-790901262-2159826448-4195_Classes\Local Settings\MuiCache\1f	Binary Data
2023-05-03 01:57:42.407	1	Process Create (rule: ProcessCreate)	"C:\Windows\system32\fodhelper.exe"		
2023-05-03 01:57:42.406	4688	Process Creation	"C:\Windows\system32\fodhelper.exe"		
2023-05-03 01:57:42.402	4673	Sensitive Privilege Use			

ဒီ Result ထဲမှာ အစောဆုံး တွေ့ရတဲ့ Event ကတော့ attacker ရဲ့ executable ဖြစ်တဲ့ **regmanipulation.exe** ကို Run လုပ်ထားတာပါ။ ဒါက Registry ကို ပြောင်းလဲဖို့ အသုံးပြုတဲ့ Attacker ရဲ့ Tool ဖြစ်ပါတယ်။ ဒီ Process ဟာ အရင်က မြင်ခဲ့ဖူးသလိုပဲ explorer.exe လို ဟန်ဆောင်ထားတဲ့ Spoofed Process ဖြစ်နိုင်ပါတယ်။

regmanipulation.exe ကို Run လုပ်ပြီး မကြာခင်မှာပဲ Registry Key သုံးခုကို Create လုပ်ထားတာကို တွေ့ရပါတယ်။ ပထမဆုံး Key က CLSID တစ်ခုအတွက် `(Default)` Value ဖြစ်ပြီး ဒီ CLSID ဟာ Windows မှာ မူလမရှိတဲ့ Arbitrary CLSID တစ်ခု ဖြစ်ပါတယ်။ ဒုတိယ Key ကတော့ အဲ့ဒီ CLSID အောက်မှာ `DelegateExecute` ဆိုတဲ့ Value ဖြစ်ပြီး ဒီ Value ထဲမှာ Attacker ရဲ့ Implant ဖြစ်တဲ့ Executable ကို Point လုပ်ထားပါတယ်။ ဒီ DelegateExecute က COM Object တွေကို Execute လုပ်တဲ့အချိန် Windows က အထူးအခွင့်အရေးနဲ့ ခေါ်သုံးတတ်တဲ့ Mechanism တစ်ခုဖြစ်ပါတယ်။

တတိယ Registry Key ကတော့ `ms-settings\shell\open\command\SymbolicLinkValue` ဖြစ်ပါတယ်။ ဒီ Key ကို Attacker က အထက်မှာ ဖန်တီးထားတဲ့ Arbitrary CLSID ကို Point လုပ်အောင် ပြုလုပ်ထားပါတယ်။ ဒီ Key ဟာ Default အနေနဲ့ Windows Registry ထဲမှာ မရှိပါဘူး။ ဒါကြောင့် User တစ်

ယောက်က ကိုယ်တိုင် Create လုပ်နိုင်ပြီး Windows Component တွေရဲ့ လုပ်ဆောင်ပုံကို လှည့်စားနိုင်ပါတယ်။

Log ကို ဆက်ကြည့်လိုက်ရင် ဒီ Registry Key တွေကို Create လုပ်ထားတဲ့ Process ID ဟာ regmanipulation.exe နဲ့ ကိုက်ညီနေတာကို တွေ့ရပါတယ်။ ဒါပေမယ့် Image Name ကို ကြည့်ရင် explorer.exe လို ပေါ်နေတာတွေ့ရနိုင်ပါတယ်။ ဒီဟာက Attacker ရဲ့ ပုံမှန် လုပ်ထုံးလုပ်နည်းဖြစ်ပြီး Process Environment Block (PEB) ကို Patch လုပ်ပြီး explorer.exe လို လှည့်စားထားတာနဲ့ ကိုက်ညီပါတယ်။

Registry ကို ပြင်ဆင်ပြီးပြီးချင်း Attacker က **fodhelper.exe** ကို Execute လုပ်ပါတယ်။ fodhelper.exe ဆိုတာက Windows မှာ Auto-Elevation ခွင့်ရှိတဲ့ Trusted Binary တစ်ခုဖြစ်ပြီး UAC Prompt မပေါ်ဘဲ Administrator အခွင့်အရေးနဲ့ Run နိုင်ပါတယ်။ fodhelper.exe Run လုပ်တဲ့အချိန်မှာ ms-settings ဆိုင်ရာ Registry Key တွေကို ကြည့်ပြီး Command ကို ဆုံးဖြတ်ပါတယ်။ Attacker က အဲ့ဒီ Registry Key တွေကို ပြင်ထားတဲ့အတွက် fodhelper.exe က အမှန်တကယ် Windows Setting ကို မဖွင့်ဘဲ Attacker ရဲ့ Implant ဖြစ်တဲ့ **mtyqs.exe** ကို Unprotected Location တစ်ခုကနေ Execute လုပ်သွားပါတယ်။

ဒီအချိန်မှာ mtyqs.exe ဟာ **High Integrity Process** အနေနဲ့ Run ဖြစ်သွားပါတယ်။ ဆိုလိုတာက Attacker ဟာ Medium Integrity ကနေ UAC Prompt မပေါ်ဘဲ Administrator အဆင့်ကို တက်သွားနိုင်တာဖြစ်ပါတယ်။ ဒီနောက်ပိုင်းမှာ Credential Dumping, System Modification စတဲ့ အန္တရာယ်ကြီးတဲ့ Attack တွေကို ဆက်လုပ်နိုင်ပါတယ်။

ဒီ Attack အလုပ်လုပ်နိုင်တဲ့ အဓိကအကြောင်းရင်းက `SymbolicLinkValue` Registry Key ဟာ Windows မှာ Default မရှိတဲ့ Key ဖြစ်လို့ပါ။ Default မရှိတဲ့အတွက် User က ကိုယ်တိုင် Create လုပ်နိုင်ပြီး fodhelper.exe လို Auto-Elevation Program တစ်ခုရဲ့ အလုပ်လုပ်ပုံကို လှည့်စားနိုင်ပါတယ်။ Windows က ဒီ Key ရဲ့ တရားဝင်မှုကို တင်းကြပ်စွာ မစစ်ဆေးတဲ့အတွက် UAC Bypass ဖြစ်သွားပါတယ်။

Conclusion

ဒီ Registry Manipulation Technique ဟာ Windows Registry ထဲက မူလမရှိတဲ့ Key တစ်ခုကို အသုံးချပြီး Auto-Elevation ခွင့်ရှိတဲ့ fodhelper.exe ကို လှည့်စားကာ Attacker ရဲ့ Code ကို High Integrity နဲ့ Execute လုပ်စေတဲ့ နည်းလမ်းပါ။ Defender ဘက်ကနေ ကြည့်ရင် ms-settings ဆိုင်ရာ Registry Key အသစ်တွေ ပေါ်လာတာ၊ fodhelper.exe က မထင်မှတ်တဲ့ Executable ကို Spawn လုပ်တာတွေကို တွေ့ရရင် UAC Bypass တစ်ခု ဖြစ်နိုင်တယ်လို့ သံသယထားပြီး စစ်ဆေးသင့်ပါတယ်။

Test : More Spoofed explorer.exe

The attacker performed a similar UAC bypass by creating a (Default) key with an executable filename in it. Which key did they create?

အရင်ဆုံး နားလည်ထားရမယ့်အချက်က Attacker ဟာ Registry ကို ပြောင်းတဲ့အချိန်မှာ **explorer.exe လို့ ဟန်ဆောင်ထားတဲ့ Process** ကို ထပ်မံအသုံးပြုထားတယ်ဆိုတာပါ။ ဒါကို သေချာဖော်ထုတ်ဖို့ အကောင်းဆုံးနည်းလမ်းက **Sysmon Event ID 13** ကို စစ်တာပါ။ **Event ID 13 က Registry Value ကို Set လုပ်တဲ့အခါ Generate ဖြစ်ပါတယ်။** Registry Value အသစ်ကို Set လုပ်တဲ့ Event တွေထဲက

Value ရဲ့အဆုံးမှာ .exe နဲ့ဆုံးတာတွေကို စစ်ပါတယ်။ ဘာကြောင့်လဲဆိုတော့ .exe ကို Point လုပ်တဲ့ Registry Change တွေဟာ Code Execution နဲ့ တိုက်ရိုက်ဆက်နွယ်နေတတ်လို့ပါ။

နောက်ထပ်အရေးကြီးတဲ့ Filter က Image ကို explorer.exe လို့ ပေါ်နေတာကို စစ်တာပါ။ Registry Change ကို explorer.exe က လုပ်တယ်လို့ မြင်ရရင် သံသယထားရပါမယ်။ အထူးသဖြင့် Modified Key ဟာ `(Default)` ဖြစ်နေရင် UAC Bypass နဲ့ COM Hijack ဆိုင်ရာ Pattern ဖြစ်နိုင်ပါတယ်။

```
where EventCode == 13
    and Details endswith "exe"
    and Image endswith "explorer.exe"
    and TargetObject endswith "(Default)"
```

ဒါပေမယ့် explorer.exe ကို တကယ်ဟုတ်မဟုတ် ခွဲဖို့အတွက် **Process GUID** ကို အသုံးပြုရပါတယ်။ Attacker က PEB Masquerading လုပ်နိုင်တာက File နဲ့ Registry Operation တွေပဲ ဖြစ်တဲ့အတွက် Process Creation Event တွေမှာတော့ အမှန်တကယ် explorer.exe ရဲ့ GUID ကို ဖမ်းလို့ရပါတယ်။ ဒါကြောင့် အရင်ဆုံး Process Execution Event တွေထဲက explorer.exe ရဲ့ ParentProcessGuid အားလုံးကို စုထားပြီး explorerGuids လို့ သတ်မှတ်ထားပါတယ်။

```
let explorerGuids = external_table("Winlog")
    | where EventCode == 1 and ParentImage endswith "explorer.exe"
    | distinct ParentProcessGuid;
```

အဲ့ဒီနောက် Registry Value Set Event တွေကို ရှာပြီး explorer.exe လို့ ပေါ်နေသော်လည်း ProcessGuid က explorerGuids ထဲ မပါဘူးဆိုရင် အဲ့ဒါဟာ **Spoofed explorer.exe** ဖြစ်တယ်လို့ သတ်မှတ်နိုင်ပါတယ်။ `ProcessGuid !in (explorerGuids)` ဒီလိုနည်းနဲ့ စစ်လိုက်တဲ့အခါ အရင်က စစ်ခဲ့တဲ့ Registry Modification တစ်ခုကို ပြန်တွေ့ရသလို နောက်ထပ် အသစ်တစ်ခုကိုပါ တွေ့လာပါတယ်။

> C:\Users\bob\Downloads\oxbpwcav1.exe

တွေ့လာတဲ့ Registry Key က User Hive အောက်က CLSID တစ်ခုရဲ့ `(Default)` Value ဖြစ်ပြီး အဲ့ဒီ Value က Attacker ရဲ့ Implant ဖြစ်တဲ့ executable ကို Point လုပ်ထားပါတယ်။ အရေးကြီးတာက ဒီ Executable ဟာ System32 မဟုတ်ဘဲ User ရဲ့ Downloads Folder လို **Unprotected Location** ထဲမှာ ရှိနေတာပါ။ ဒါဟာ UAC Bypass ဖြစ်နေတယ်ဆိုတဲ့ သေချာတဲ့ သဲလွန်စတစ်ခုဖြစ်ပါတယ်။

Full Query

```
let explorerGuids = external_table("Winlog")
    | where EventCode == 1 and ParentImage endswith "explorer.exe"
    | distinct ParentProcessGuid;
external_table("Winlog")
| where EventCode == 13
    and Details endswith "exe"
    and Image endswith "explorer.exe"
    and TargetObject endswith "(Default)"
```

```
    and ProcessGuid !in (explorerGuids)
| project Timestamp, EventCode, Details, TargetObject
```

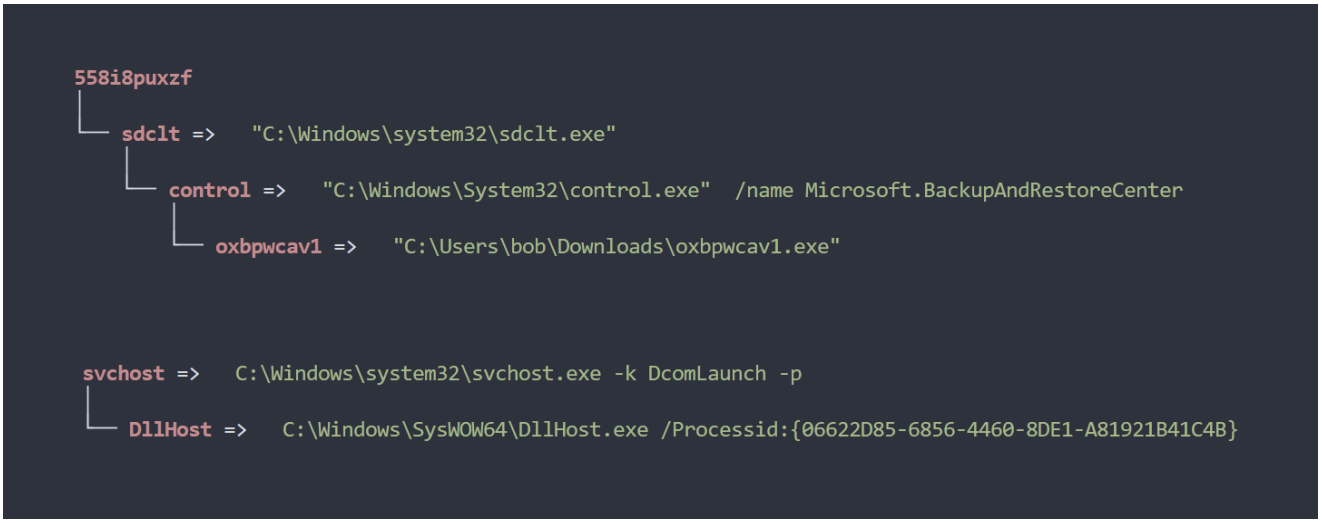
Attack Explanation

အခု Attack Explanation ကို ကြည့်ရင် Registry Key တွေကို ဖန်တီးထားတဲ့ ပုံစံက အရင် fodhelper.exe Attack နဲ့ ဆင်တူပါတယ်။ ဒါပေမယ့် Bypass Mechanism က မတူပါဘူး။ ဒီတစ်ခါမှာ Attacker က ms-settings ကို မသုံးဘဲ Folder Shell ဆိုင်ရာ Registry Path ကို အသုံးပြုထားပါတယ်။ အထူးသဖြင့်

```
HKU\S-1-5-21-1072563919-790901262-2159826448-
4195_Classes\Folder\shell\open\command\SymbolicLinkValue
```

ကို ဖန်တီးပြီး Arbitrary CLSID ကို Point လုပ်ထားပါတယ်။

Registry ကို ပြင်ပြီးတဲ့နောက် Attacker က **sdclt.exe** ကို Execute လုပ်ပါတယ်။ sdclt.exe ကလည်း Windows ရဲ့ Auto-Elevation ခွင့်ရှိတဲ့ Trusted Binary တစ်ခုဖြစ်ပါတယ်။ sdclt.exe Run ဖြစ်တဲ့အချိန်မှာ Folder Shell ဆိုင်ရာ Registry Key တွေကို ဖတ်ပြီး Command ကို ဆုံးဖြတ်ပါတယ်။ Attacker က အဲ့ဒီ Registry ကို ပြင်ထားတဲ့အတွက် sdclt.exe က Windows Function ကို မလုပ်တော့ဘဲ Attacker ရဲ့ Implant ကို High Integrity နဲ့ Execute လုပ်သွားပါတယ်။



အဲ့ဒီနောက် Process Tree ကို ကြည့်ရင် Trusted Windows Binary ကနေ User Folder ထဲက Executable ကို Spawn လုပ်ထားတဲ့ ပုံစံကို တွေ့ရပါတယ်။ ဒါဟာ UAC Bypass ဖြစ်နေတယ်ဆိုတဲ့ အထူးသဘောတရားကို ပြသတဲ့ Artifact ဖြစ်ပါတယ်။

Conclusion*

အကျဉ်းချုပ်ပြောရရင် ဒီဟာက Registry Manipulation + PEB Masquerading + Auto-Elevation Binary Abuse တို့ကို ပေါင်းပြီး Attacker က explorer.exe လို ဟန်ဆောင်ထားတဲ့ Process နဲ့ Registry ကို ပြောင်း၊ sdclt.exe ကို အသုံးပြုပြီး UAC Prompt မပေါ်ဘဲ High Integrity Process ကို ရယူနိုင်တယ် ဆိုတာကို ပြသထားတာပါ။ Defender ဘက်ကနေ ကြည့်ရင် explorer.exe လို့ ပေါ်နေတဲ့ Registry Change တွေကို Process GUID နဲ့ Correlate လုပ်ပြီး စစ်ဆေးနိုင်ရင် ဒီလို UAC Bypass Attack တွေကို ထိရောက်စွာ ဖော်ထုတ်နိုင်ပါတယ်။

Environment Variable

Windows မှာ Scheduled Task တချို့ကို **HighestAvailable** ဆိုတဲ့ Setting နဲ့ Run အောင် သတ်မှတ်ထားပါတယ်။ ဒါဟာ Task ကို Run လုပ်တဲ့အခါ User က Administrator ဖြစ်နေရင် UAC Prompt မပေါ်ဘဲ High Integrity နဲ့ Run အောင် ခွင့်ပြုထားတာပါ။ DiskCleanup ဆိုတဲ့ Scheduled Task ဟာ ဒီလို HighestAvailable နဲ့ Run အောင် သတ်မှတ်ထားတဲ့ Task တစ်ခုဖြစ်ပါတယ်။

```
PS C:\> schtasks.exe /query /TN \Microsoft\Windows\DiskCleanup\SilentCleanup /xml
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.6" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <SecurityDescriptor>D:AI(A;;FA;;;BA)(A;;FA;;;SY)(A;;FRFX;;;AU)</SecurityDescriptor>
    <Source>$(@%systemroot%\system32\cleanmgr.exe,-1300)</Source>
    <Author>$(@%systemroot%\system32\cleanmgr.exe,-1300)</Author>
    <Description>$(@%systemroot%\system32\cleanmgr.exe,-1301)</Description>
    <URI>\Microsoft\Windows\DiskCleanup\SilentCleanup</URI>
  </RegistrationInfo>
  <Principals>
    <Principal id="Authenticated Users">
      <GroupId>S-1-5-32-545</GroupId>
      <RunLevel>HighestAvailable</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <ExecutionTimeLimit>PT15M</ExecutionTimeLimit>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <StartWhenAvailable>true</StartWhenAvailable>
    <RunOnlyIfIdle>true</RunOnlyIfIdle>
    <IdleSettings>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>true</RestartOnIdle>
    </IdleSettings>
    <UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>
    <MaintenanceSettings>
      <Period>P1D</Period>
      <Deadline>P1M</Deadline>
    </MaintenanceSettings>
  </Settings>
  <Triggers />
  <Actions Context="Authenticated Users">
    <Exec>
      <Command>%windir%\system32\cleanmgr.exe</Command>
      <Arguments>/autocleanstoragesense /d %systemdrive%</Arguments>
    </Exec>
  </Actions>
</Task>
```

DiskCleanup Task ရဲ့ထူးခြားချက်တစ်ခုက Command Path ထဲမှာ **Environment Variable** ကို အသုံးပြုထားတာပါ။ အထူးသဖြင့် `%windir%` ဆိုတဲ့ Variable ကို သုံးထားပါတယ်။ `%windir%` ဆိုတာ ပုံမှန်အားဖြင့် `C:\Windows` ကို ရည်ညွှန်းတဲ့ System Environment Variable ဖြစ်ပါတယ်။ DiskCleanup Task ကို Run လုပ်တဲ့အခါ Windows က `%windir%\system32\cleanmgr.exe` ဆိုတဲ့ Command ကို Execute လုပ်ပါတယ်။

ဒီနေရာမှာ Attackers တွေက Windows ရဲ့အားနည်းချက်ကို တွေ့ရှိထားပါတယ်။ Environment Variable တွေကို User Context ထဲမှာ ပြောင်းလဲနိုင်တဲ့အတွက် Attacker က `%windir%` ကို မူလ

`C:\Windows` မဟုတ်ဘဲ မိမိရဲ့ Malicious Executable ရှိနေတဲ့ Path ကို Point လုပ်အောင် ပြောင်းနိုင်ပါတယ်။ ဥပမာ `%windir%` ကို `C:\Users\bob\Desktop\gjwcr94ucn.exe` လို့ Set လုပ်လိုက်ပါမယ်။

အဲ့ဒီနောက် Attacker က **SilentCleanup** ဆိုတဲ့ Scheduled Task ကို Execute လုပ်ပါတယ်။ SilentCleanup ဟာ DiskCleanup Task ကို Background မှာ Silent အနေနဲ့ Run ပေးတဲ့ Task ဖြစ်ပြီး HighestAvailable Privilege နဲ့ Run ပါတယ်။ Task Run ဖြစ်တဲ့အချိန်မှာ Windows က `%windir%` ကို Command Path ရဲ့ အရှေ့မှာ ထည့်လိုက်တဲ့အတွက် Command Line ဟာ ပုံမှန်မဟုတ်တဲ့ ပုံစံတစ်ခု ဖြစ်လာပါတယ်။

အဲ့ဒီ Command Line က

> `"%windir%"\system32\cleanmgr.exe /autoclean /d C:`

ဖြစ်ပြီး `%windir%` ကို Attacker ပြောင်းထားတာကြောင့် အမှန်တကယ် Run သွားတာက `C:\Users\bob\Desktop\gjwcr94ucn.exe` ဖြစ်သွားပါတယ်။ Windows က cleanmgr.exe ကို Run လိုက်တယ်လို့ ထင်ပေမယ့် အမှန်တကယ်တော့ Attacker ရဲ့ Implant ကို High Integrity နဲ့ Execute လုပ်သွားတာပါ။

```
external_table("Winlog")
| where Timestamp > datetime(2023-05-03T02:01:05.714Z)
    and Timestamp < datetime(2023-05-03T02:01:07.044Z)
    and (EventCode == 1 or EventCode == 11)
| project Timestamp, EventCode, EventAction, ParentImage, CommandLine
```

Log Analysis ကို ကြည့်ရင် Attacker က အရင်ဆုံး Environment Variable ကို Set လုပ်ထားတဲ့ ခြေရာတွေကို တွေ့နိုင်ပါတယ်။ နောက်တစ်ဆင့်မှာ SilentCleanup Scheduled Task ကို Execute လုပ်ထားတာကို Process Creation Event တွေနဲ့ မြင်နိုင်ပါတယ်။ Parent Image နဲ့ CommandLine ကို ကြည့်မယ်ဆိုရင် Scheduled Task ကနေ မထင်မှတ်တဲ့ Executable ကို Spawn လုပ်ထားတာကို တွေ့ရပါလိမ့်မယ်။ အဲ့ဒီ Executable က User Desktop လို Unprotected Location ထဲက ဖြစ်နေရင် UAC Bypass ဖြစ်နိုင်ခြေ အလွန်မြင့်ပါတယ်။

ဒီ Technique ရဲ့ အရေးကြီးဆုံးအချက်ကတော့ **UAC ကို Always Notify လို့ထားထားတောင် အလုပ်လုပ်နိုင်တယ်** ဆိုတာပါ။ အကြောင်းက UAC Prompt က User Interaction လိုတဲ့ Program Run တွေမှာပဲ အဓိက သက်ရောက်ပြီး Scheduled Task လို System Mechanism တွေကိုတော့ မတားဆီးနိုင်လို့ပါ။ Windows ကိုယ်တိုင် ယုံကြည်ထားတဲ့ Task ကို Run လုပ်နေတာဖြစ်တဲ့အတွက် UAC Prompt မပေါ်ဘဲ High Integrity Process ကို ရရှိသွားပါတယ်။

Test : Environment Variables Hunt

The attacker performed another, nearly identical, UAC bypass with the SilentCleanup scheduled task. What was the name and path of the implant?

အရင်ဆုံး နားလည်ရမယ့် အချက်က ဒီ Attack မှာ Attacker က **SilentCleanup Scheduled Task** ကို အသုံးပြုထားပြီး၊ အဲ့ဒီ Task က Run ဖြစ်တဲ့အခါ `\system32\cleanmgr.exe /autoclean /d C:` ဆို

တဲ့ Command ကို မဖြစ်မနေ သုံးတယ်ဆိုတာပါ။ ဒီအချက်က Hunt လုပ်တဲ့အခါ အလွန်အသုံးဝင်တဲ့ သဲလွန်စတစ်ခု ဖြစ်ပါတယ်။

```
external_table("Winlog")
| where EventCode == 1 and CommandLine contains "\\system32\\cleanmgr.exe
/autoclean /d C:"
| project Timestamp, EventCode, EventAction, ParentImage, CommandLine
```

ဒါကြောင့် ပထမနည်းလမ်းအနေနဲ့ Process Creation Event ဖြစ်တဲ့ EventCode 1 ကို စစ်ပြီး CommandLine ထဲမှာ `\system32\cleanmgr.exe /autoclean /d C:` ပါနေတာတွေကို ရှာပါတယ်။ ပုံမှန်အားဖြင့် ဒီ Command ကို Scheduled Task ကနေ cleanmgr.exe ကို Run တာအတွက်ပဲ သုံးသင့်ပါတယ်။ ဒါပေမယ့် Attacker က `%windir%` ကို လှည့်စားထားရင် CommandLine ရဲ့အရှေ့ပိုင်းမှာ cleanmgr.exe မဟုတ်ဘဲ တခြား Executable Path တစ်ခု ပါလာနိုင်ပါတယ်။ ဒီလို Event တွေကို တွေ့ရင် Environment Variable Based UAC Bypass ဖြစ်နိုင်ခြေ ရှိပါတယ်။

False Positive ကို လျှော့ချဖို့ နောက်တစ်ဆင့်အနေနဲ့ CommandLine က `C:\WINDOWS` နဲ့ မစတင်တဲ့ Event တွေကိုပဲ Filter လုပ်ပါတယ်။ ပုံမှန် cleanmgr.exe Run ဖြစ်ရင် CommandLine က `C:\Windows\System32\cleanmgr.exe` နဲ့ စရပါမယ်။ အဲ့ဒီအစား User Folder, Temp Folder လို နေရာက Path နဲ့ စနေတာကို တွေ့ရင် သံသယထားရပါမယ်။ ဒီလို Filter လုပ်လိုက်တဲ့အခါ Attacker ရဲ့ Implant ဖြစ်တဲ့ `C:\temp\cx12.exe` ကို Point လုပ်ထားတဲ့ Event ကို ဖော်ထုတ်နိုင်ပါတယ်။

```
external_table("Winlog")
| where EventCode == 1 and CommandLine contains "\\system32\\cleanmgr.exe
/autoclean /d C:" and CommandLine !startswith "C:\\WINDOWS"
| project Timestamp, EventCode, EventAction, ParentImage, CommandLine
```

နောက်တစ်နည်းလမ်းကတော့ Attack ရဲ့အရင်းအမြစ်ကို တိုက်ရိုက်ဖမ်းတဲ့ နည်းလမ်းဖြစ်ပါတယ်။ ဒီ Attack အလုပ်လုပ်ဖို့ `%windir%` Environment Variable ကို မဖြစ်မနေ ပြောင်းရပါတယ်။ Environment Variable ပြောင်းတဲ့အခါ Sysmon Event ID 13 ဖြစ်တဲ့ Registry Value Set Event ပေါ်လာပါတယ်။ `%windir%` ဟာ Registry ထဲမှာ `Environment\windir` ဆိုတဲ့ Key အောက်မှာ သိမ်းထားတာကြောင့် အဲ့ဒီ TargetObject ကို စစ်ပြီး Modification Event တွေကို ရှာနိုင်ပါတယ်။

```
external_table("Winlog")
| where EventCode == 13 and TargetObject contains "\\Environment\\windir"
| project Timestamp, EventCode, EventAction, TargetObject, Details
```

ဒီ Query ကို သုံးလိုက်တဲ့အခါ `%windir%` ကို ပြောင်းထားတဲ့ Registry Event ကို တိုက်ရိုက်တွေ့နိုင်ပါတယ်။ Details ထဲမှာ `%windir%` ကို ဘယ် Path ကို Point လုပ်ထားလဲဆိုတာပါလာပြီး ဒီ Case မှာတော့ `C:\temp\cx12.exe` ကို Set လုပ်ထားတာကို တွေ့ရပါတယ်။ ဒီအချက်တစ်ခုတည်းနဲ့တင် Scheduled Task UAC Bypass ဖြစ်နေတယ်ဆိုတာကို ယုံကြည်စိတ်ချရအောင် ပြောနိုင်ပါတယ်။

Elevated COM Interface

ဒီ Technique မှာ အသုံးပြုထားတာက **ICMLuaUtil COM interface** ဖြစ်ပါတယ်။ Windows ထဲမှာ COM Object ဆိုတာ စနစ်ပိုင်းဆိုင်ရာ လုပ်ဆောင်ချက်တွေကို Program တွေက ခေါ်သုံးနိုင်အောင် ပြုလုပ်ထားတဲ့ Component တွေပါ။ ICMLuaUtil ကတော့ UAC နဲ့ ဆက်စပ်နေပြီး၊ Windows ကို အလုပ်လုပ်အောင်ထားဖို့ အတွက် **အမြင့်အခွင့်အရေး (High Integrity)** နဲ့ Run ရမယ့် လုပ်ဆောင်ချက်တွေကို ခေါ်ပေးနိုင်တဲ့ COM Object တစ်ခု ဖြစ်ပါတယ်။

ဒီ Attack မှာ Attacker က ICMLuaUtil ထဲက **ShellExec method** ကို အသုံးပြုပါတယ်။ ဒီ Method ရဲ့ အရေးကြီးဆုံးအချက်က Attacker က ကိုယ်လိုချင်တဲ့ executable ကို တိုက်ရိုက်ပေးပြီး **High Integrity Process အဖြစ် Run လိုက်နိုင်တာ** ဖြစ်ပါတယ်။ IFileOperation လို ဖိုင်ရွှေ့တာ၊ DLL ထည့်တာလို အလှည့်အပြောင်းတွေ မလိုတော့ဘဲ COM Object ကို ခေါ်လိုက်တာနဲ့ payload ကို အမြင့်အခွင့်အရေးနဲ့ Run လိုက်နိုင်ပါတယ်။

Sample

```
1   |   "C:\Users\bob\desktop\Akagi64.exe" 41 C:\Windows\System32\calc.exe
2   |   C:\Windows\system32\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-
A120244FBEC7}
3   |   "C:\Windows\System32\calc.exe"
```

Log တွေကို ကြည့်လိုက်ရင် Artifact တွေက အရမ်းရှင်းပါတယ်။ အရင်ဆုံး Windows Service ဖြစ်တဲ့ **svchost.exe** က **DllHost.exe** ကို Spawn လုပ်ပါတယ်။ ဒီ DllHost.exe က ICMLuaUtil CLSID ကို အသုံးပြုပြီး Run ဖြစ်လာတာဖြစ်ပါတယ်။ ဒီအဆင့်က “Elevated COM Object ကို ခေါ်လိုက်ပြီ” ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ အဲ့ဒီနောက်မှာ DllHost.exe ကနေ **Attacker ရဲ့ executable ကို Spawn လုပ်လိုက်ပါတယ်**။ ဒီ Spawn ဖြစ်တဲ့ Process က High Integrity Process ဖြစ်ပြီး UAC Prompt မပြဘဲ Admin Level အခွင့်အရေး ရသွားပါတယ်။

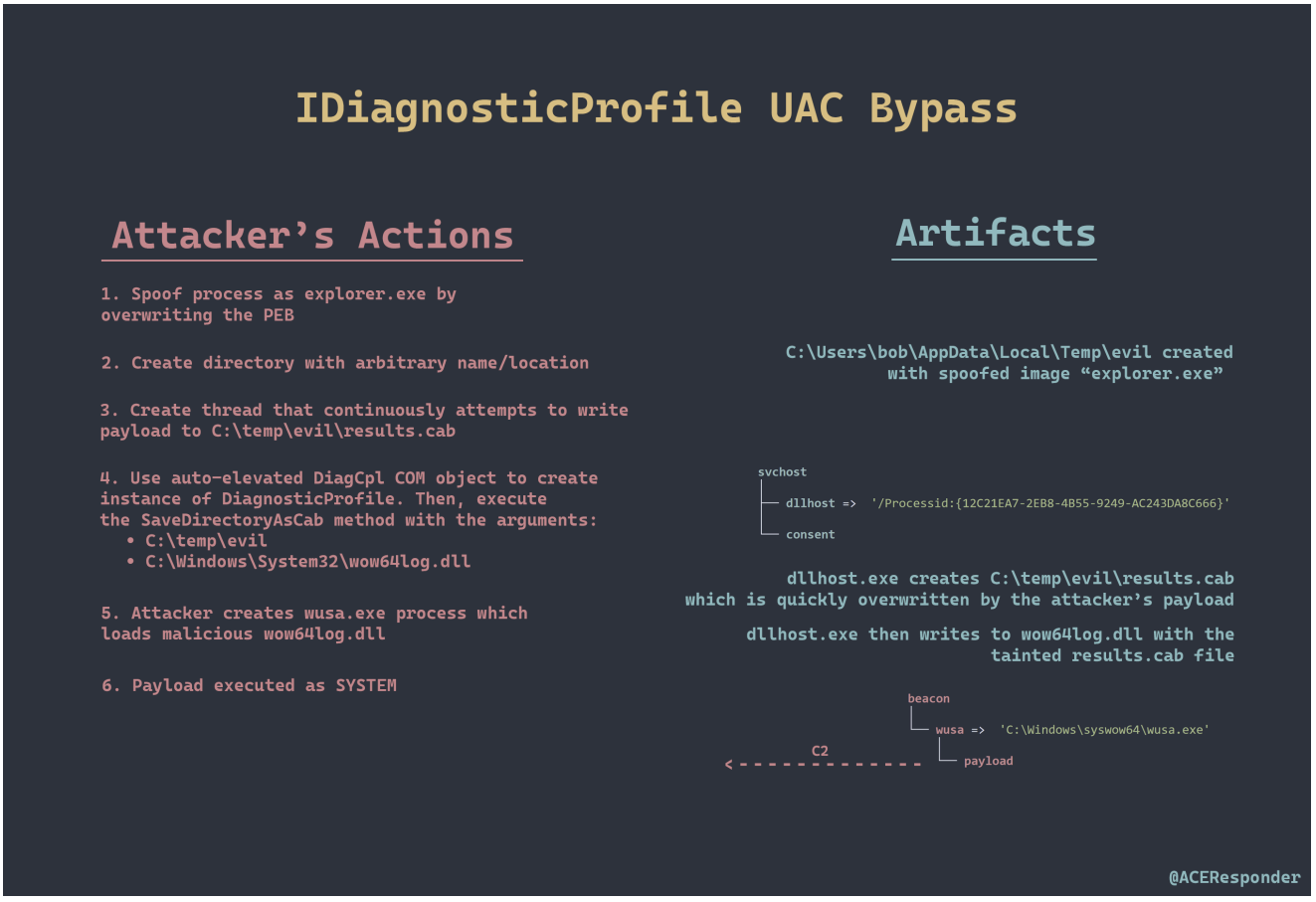
ဒီနေရာမှာ Defender ဒါမှမဟုတ် Blue Team အနေနဲ့ အထူးသတိထားရမယ့် အချက်က **High Integrity Process တွေကို ဘယ်သူ Spawn လုပ်လဲ** ဆိုတာပါ။ ပုံမှန်မဟုတ်တဲ့အခြေအနေမှာ `svchost.exe → DllHost.exe → User Folder` ထဲက exe တစ်ခု ဆိုတဲ့ Process Chain ကို တွေ့ရင် အလွန်သံသယထားသင့်ပါတယ်။ အထူးသဖြင့် Program က **Unprotected Location** ဖြစ်တဲ့ Desktop, Downloads, Temp Folder လိုနေရာတွေက Run နေတာဆိုရင် ပိုပြီး အန္တရာယ်ကြီးပါတယ်။

အကယ်လို့ System တွေမှာ UAC ကို **Always notify မထားဘူး** ဆိုရင် ဒီလို Elevated COM Interface Technique တွေက အလွန်အသုံးဝင်ပြီး အန္တရာယ်လည်းကြီးပါတယ်။ ဒါကြောင့် High Integrity Process အသစ်တွေကို အမြဲစောင့်ကြည့်ဖို့၊ Elevated COM Object ကနေ Spawn ဖြစ်လာတဲ့ Process တွေကို Log နဲ့ Hunt လုပ်ဖို့ အရေးကြီးပါတယ်။ ဒီလိုလုပ်နိုင်ရင် ICMLuaUtil လို COM Abuse ကို အသုံးပြုပြီး UAC Bypass လုပ်ထားတဲ့ Attack တွေကို အချိန်မီ ဖော်ထုတ်နိုင်ပါလိမ့်မယ်။

IDiagnosticProfileUAC (DLL hijack + auto-elevated COM Interface)

ဒီ Scenario မှာလည်း Attacker ရဲ့ ရည်ရွယ်ချက်က အတူတူပါပဲ။ အနိမ့်အခွင့်အရေး (Medium Integrity) ကနေ **High Integrity Process** ကို ရဖို့ ဖြစ်ပါတယ်။ ဒါပေမယ့် ဒီတစ်ခါတော့ IFileOperation ကို မသုံး

တော့ဘဲ **IDiagnosticProfile COM Interface** ထဲက **SaveDirectoryAsCab** ဆိုတဲ့ Method ကို အသုံးချပါတယ်။ ဒီ COM Object က Windows မှာ Auto-Elevated ဖြစ်တဲ့အတွက် UAC Prompt မပြဘဲ High Integrity နဲ့ Run နိုင်ပါတယ်။



SaveDirectoryAsCab Method ရဲ့အလုပ်လုပ်ပုံကို နားလည်ရင် Attack ကို ပိုပြီး နားလည်နိုင်ပါတယ်။ ဒီ Method က Folder တစ်ခုကို ယူပြီး **.cab ဖိုင်တစ်ခုအဖြစ် ပြောင်းပြီး** User သတ်မှတ်ပေးတဲ့ Location ကို Copy လုပ်ပေးပါတယ်။ ဒီအလုပ်စဉ်အတွင်းမှာ အရေးကြီးတဲ့ အချက်တစ်ခုရှိပါတယ်။ အဲဒါက .cab ဖိုင်ကို **အရင် Create လုပ်ပြီးမှ Destination ကို Copy လုပ်တာ** ဖြစ်ပါတယ်။ ဒီအချိန်အတိုအတွင်းကို Race Condition လို့ ခေါ်ပါတယ်။

Attacker က ဒီ Race Condition ကို အသုံးချပါတယ်။ SaveDirectoryAsCab က results.cab ဖိုင်ကို Create လုပ်လိုက်တဲ့ အချိန်နဲ့ System32 ကို Copy မလုပ်ခင်အချိန်အတွင်းမှာ Attacker က results.cab ကို **မကောင်းတဲ့ DLL နဲ့ အစားထိုးရေးသားလိုက်ပါတယ်**။ ဒီလိုလုပ်နိုင်ဖို့ Attacker က Background Thread တစ်ခုထားပြီး results.cab ကို အမြဲ overwrite လုပ်နေပါတယ်။ အချိန်ကို မှန်မှန်မိသွားတဲ့အခါ results.cab အစား **DLL ဖိုင်တစ်ခုကို Windows က ယုံကြည်ပြီး System32 ထဲကို Copy လုပ်သွားပါလိမ့်မယ်**။

Log တွေကို ကြည့်လိုက်ရင် ပထမဆုံးတွေ့ရတာက **DllHost.exe** Process ဖြစ်ပါတယ်။ Command Line ထဲမှာ DiagnosticProfile COM Interface ရဲ့ CLSID ပါဝင်နေပြီး Auto-Elevated COM Invocation ဖြစ်ကြောင်း သိနိုင်ပါတယ်။ ဒီ DllHost.exe က Attacker ဖန်တီးထားတဲ့ Folder ထဲမှာ results.cab ကို Create လုပ်ပါတယ်။ ဒီ Folder ကို ဖန်တီးတဲ့အချိန်မှာလည်း Attacker က PEB Spoofing လုပ်ထားတဲ့ explorer.exe အဖြစ် ပေါ်နေတတ်ပါတယ်။

results.cab ဖိုင် Create ဖြစ်ပြီး မကြာခင်မှာပဲ **C:\Windows\System32\wow64log.dll** ဆိုတဲ့ ဖိုင် Create ဖြစ်လာတာကို တွေ့ရပါလိမ့်မယ်။ ပုံမှန် Windows အလုပ်လုပ်ပုံအရ Diagnostic Profile က ဒီလို

DLL ကို System32 ထဲရေးတာ မဖြစ်သင့်ပါဘူး။ ဒီအချက်က Attack ဖြစ်နေကြောင်းကို အလွန်ရှင်းလင်းစွာ ပြသနေတဲ့ Artifact ဖြစ်ပါတယ်။

နောက်ဆုံးအဆင့်မှာ Attacker က wow64log.dll ကို Load ဖြစ်အောင် **wusa.exe** ကို အသုံးပြုပြီး Trigger လုပ်ပါတယ်။ wusa.exe က Windows Update Standalone Installer ဖြစ်ပြီး DLL Load လုပ်တဲ့ အခါ System32 ကို ယုံကြည်ထားတာကြောင့် **Malicious DLL ကို High Integrity Context နဲ့ Execute လုပ်သွားပါလိမ့်မယ်**။ ဒီအချိန်မှာ Attacker က အပြည့်အဝ Administrator Level အခွင့်အရေး ရသွားပါတယ်။

UAC Bypass ဆိုတာ ဒီလောက်နဲ့ပဲ မပြီးသေးဘူး

ဒီအထိ လေ့လာပြီးတဲ့အချိန်မှာ သင် နားလည်ထားသင့်တဲ့ အရေးကြီးဆုံးအချက်က **Always notify နဲ့ အခြား UAC Setting တွေရဲ့ ကွာခြားချက်** ဖြစ်ပါတယ်။ Always notify ကို မထားထားတဲ့ System တွေမှာ Silent Elevation ကို အခြေခံထားတဲ့ UAC Bypass နည်းလမ်းတွေ အလုပ်လုပ်နိုင်တာကို သေချာမြင်လာရပါပြီ။ UAC ကို “ရှိလို့ လုံခြုံတယ်” လို့ ယူဆတာက အန္တရာယ်ရှိပြီး Setting မမှန်ရင် Attacker အတွက် အခွင့်အရေးပေးထားသလို ဖြစ်သွားနိုင်ပါတယ်။

ထို့အပြင် **Environment Variable နဲ့ DLL Hijacking** တွေက Security အတွက် ဘယ်လောက် ခက်ခဲတဲ့ ပြဿနာတွေ ဖြစ်နိုင်လဲ ဆိုတာကိုလည်း နားလည်လာသင့်ပါတယ်။ Program တစ်ခုက ယုံကြည်ထားတဲ့ Path, DLL Name, Variable တစ်ခုခုကို အသုံးချနေရင် Attacker က အဲ့ဒီအရာကို ပြောင်းလဲနိုင်တဲ့ အခါ System က မကောင်းတဲ့ Code ကို ယုံကြည်ပြီး Run လုပ်ပေးသွားနိုင်ပါတယ်။ ဒါဟာ Windows ရဲ့ ပုံမှန် Feature တွေကိုပဲ အသုံးချထားတာ ဖြစ်တဲ့အတွက် Detect လုပ်ရခက်တာလည်း ဖြစ်ပါတယ်။

UAC Bypass ကို Hunt လုပ်တဲ့အခါ သတိထားစရာ Pattern တချို့ကို မှတ်ထားဖို့ လိုပါတယ်။ System32 လို Windows System Folder နာမည်တွေနဲ့ ဆင်တူတဲ့ Folder တွေကို User Folder လို Unprotected Location တွေထဲမှာ ဖန်တီးထားတာတွေကို တွေ့ရင် သံသယထားသင့်ပါတယ်။ Medium Integrity Process ကနေ High သို့မဟုတ် System Integrity Process ကို Spawn လုပ်ထားတဲ့ Process Chain တွေကလည်း အရေးကြီးတဲ့ Indicator တွေ ဖြစ်ပါတယ်။

ထို့အပြင် ပုံမှန်အားဖြင့် မရှိသင့်တဲ့ DLL Name တွေကို Create လုပ်ထားတဲ့ File Creation Event တွေ၊ explorer.exe လို့ ပြထားပေမယ့် အမှန်တကယ် explorer.exe မဟုတ်တဲ့ Process က File သို့မဟုတ် Registry ကို ပြင်ဆင်ထားတာတွေကို တွေ့ရင် UAC Bypass ဖြစ်နိုင်ခြေ အလွန်မြင့်ပါတယ်။ HighestAvailable Run Level နဲ့ Run လုပ်မယ့် Scheduled Task တွေကို User Location ထဲက Executable နဲ့ ချိတ်ဆက်ထားတာတွေ၊ Environment Variable တွေကို User Executable ဆီ ပြောင်းညွှန်ထားတာတွေကလည်း သတိထားစစ်ဆေးသင့်တဲ့ အချက်တွေပါ။

အကယ်လို့ ကိုယ်တိုင် လက်တွေ့ စမ်းကြည့်ချင်တယ်ဆိုရင် [UACMe Project](#) ကို လေ့လာကြည့်တာက အလွန်ကောင်းပါတယ်။ ဒီ Project က လူသိများပြီးသား UAC Bypass Technique တွေကို တစ်နေရာတည်းမှာ စုစည်းပေးထားတဲ့ Resource ဖြစ်လို့ Red Team အနေနဲ့ နားလည်ဖို့ကောင်းသလို Blue Team အနေနဲ့ Detection Idea တွေ တည်ဆောက်ဖို့လည်း အလွန်အသုံးဝင်ပါတယ်။
