Extending Osquery's Reach

Reverse Engineering for Enhanced Windows Visibility

Marcos Oviedo

About Me

Marcos Oviedo

Hooked on Windows internals

All about defensive

and offensive endpoint security

@marcosd4h

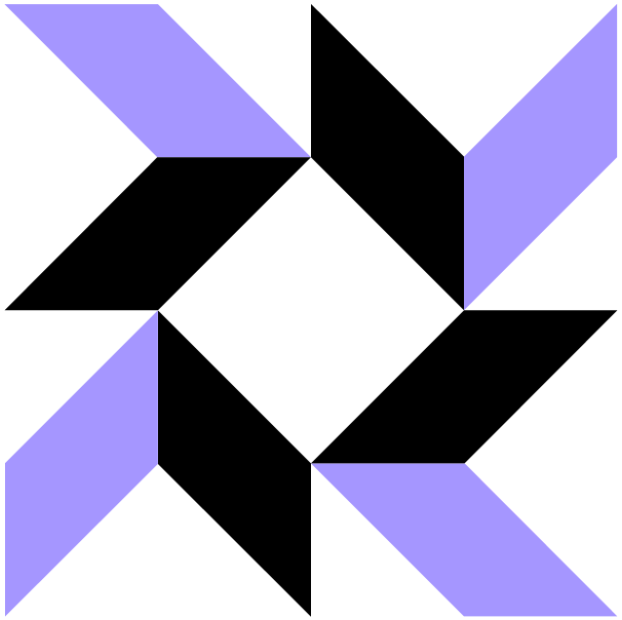
Talk Overview

- Real-world stories from the trenches
- From Idea to PR, two Osquery merge adventures
- Pushing Osquery boundaries with reverse engineering

Osquery
Overview

What is Osquery?

- SQL-powered operating system monitoring and visibility framework
- Cross-platform support (macOS, Linux, Windows)
- Originally from Facebook, now part of Linux Foundation



github.com/osquery/osquery

Why Osquery?

- Standalone agent that uses SQLite for query execution
- 300+ tables across Linux, macOS and Windows
- Widely used for real-time monitoring and endpoint security

Osquery Usage Examples

```
osquery> select username, directory from users
limit 2;
+-----------+-------------------+
| username  | directory         |
+-----------+-------------------+
| marcos    | C:\Users\marcos   |
| testuser  | C:\Users\testuser |
+-----------+-------------------+
```

```
osquery> select name, version, publisher from
programs limit 2;
+---------------+-----------+-------------+
| name          | version   | publisher   |
+---------------+-----------+-------------+
| OBS Studio    | 29.1.3    | OBS Project |
| Spotify       | 1.2.22.982| Spotify AB  |
+---------------+-----------+-------------+
```

```
osquery> select hotfix_id, description,
installed_on from patches limit 2;
+-----------+-------------+--------------+
| hotfix_id | description | installed_on |
+-----------+-------------+--------------+
| KB5029919 | Update      | 9/15/2023    |
| KB5028951 | Update      | 8/20/2023    |
+-----------+-------------+--------------+
```

```
osquery> select pid, port, address from
listening_ports limit 2;
+------+------+---------+
| pid  | port | address |
+------+------+---------+
| 1324 | 135  | 0.0.0.0 |
| 9560 | 443  | 0.0.0.0 |
+------+------+---------+
```

The Visibility Gap on Windows

- ✓ Osquery provides good endpoint visibility
- ✓ However, gaps exist on Windows
- ✓ Adversaries live in the **gaps**
- ✗ What could they be **exploiting?**

Story #1
Visibility on Windows Password Policies

include `secedit` table natively in osquery #7425

Closed zhumo opened this issue on Aug 26, 2022 · 35 comments

zhumo commented on Aug 26, 2022 • edited Contributor ···

Problem

On Windows, almost everything is configured in registry, esp. security settings. There is a registry table in osquery, and we can almost always query the registry table for the settings. HOWEVER across windows versions, the path sometimes changes, so we need to dynamically change the path for each windows version when we are writing windows policies.

Kollide has an extension for a `secedit` table that gives that path info so that we can join on it and then use that to drive the registry queries. However, this extension shells out data, which Guillaume raised as a security concern. Having something like this table, though, would make writing queries and policies for Windows much easier.

Requirements

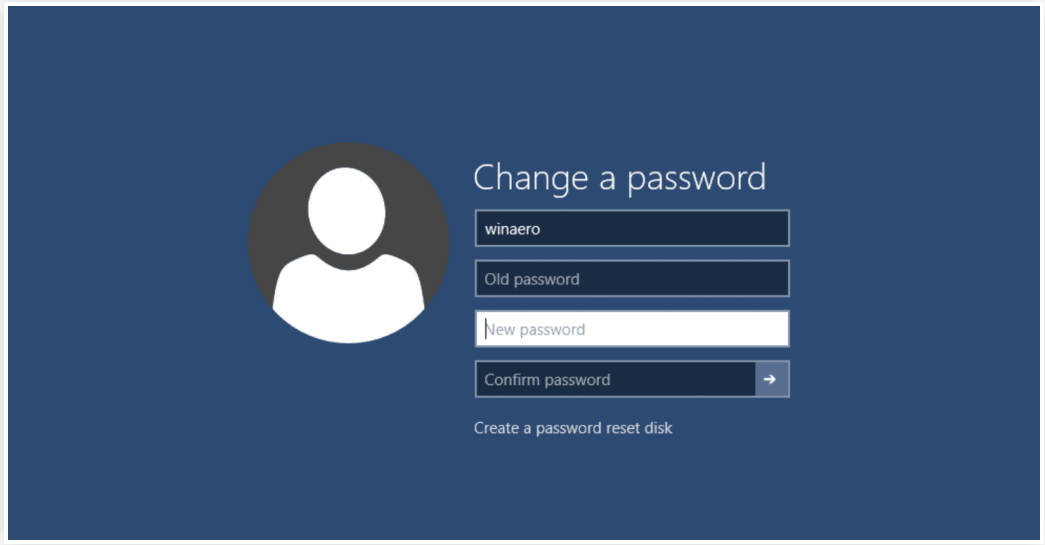
1. Incorporate secedit into osquery with, at minimum, the path names so that security users can write policy queries for windows using the `registry` table without customizing the path for each windows version.
2. If possible, collect all secedit into an osquery table.

Related

- New table to retrieve security profile information on Windows osquery/osquery#7794

Issue #7425

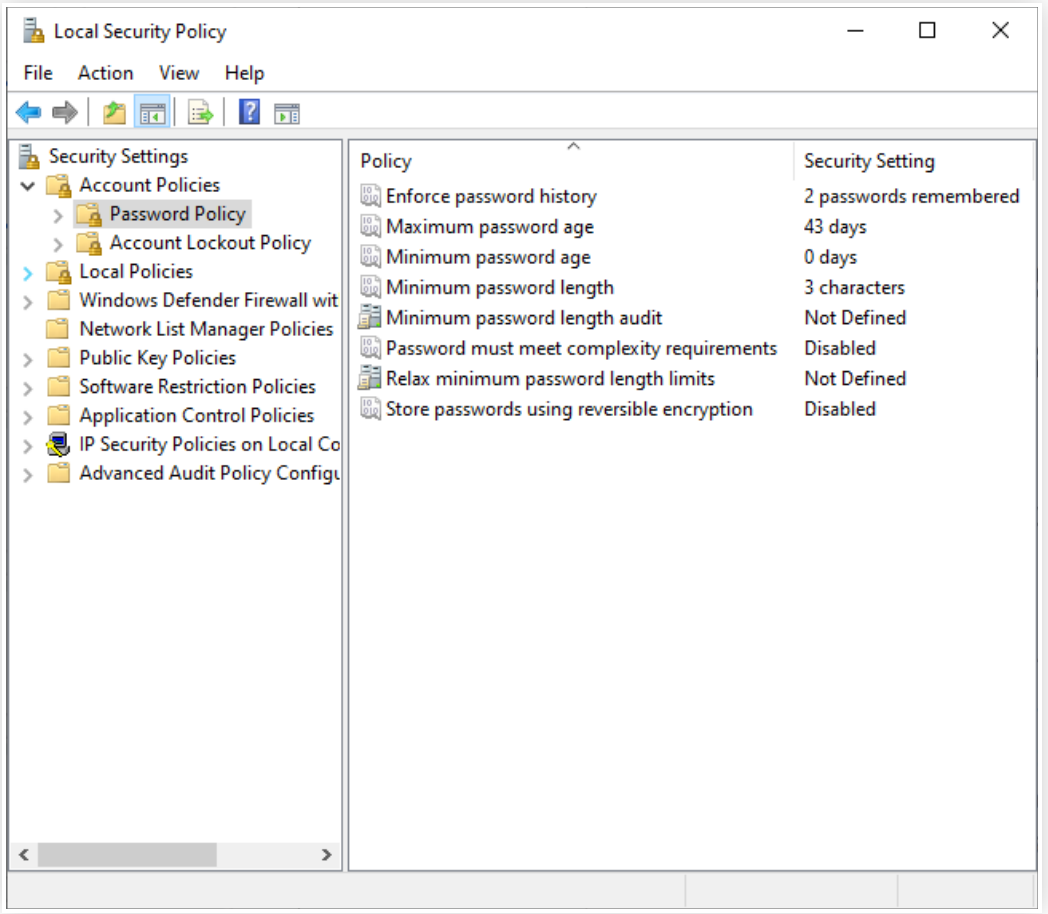
The Problem



Passwords are the most vulnerable component in an authentication implementation

Windows Password Policy Blindspot

- Password policies enforce password requirements & restrictions
- Cannot use Osquery to get password policies state
- Visibility required for Informed Defensive Decisions



Password Policies Breakdown

- Enforce password history
- Minimum password age
- Maximum password age
- Minimum password length
- Password complexity requirements
- Password reversible encryption

It would be cool to have a way to query the length and complexity of account passwords

How can these policies be retrieved?

Password Policies can be obtained via the Local Security Policy tool or via secedit.exe utility

- Both require local access
- Cannot be done at scale

Secedit Utility

- Utility to configure and analyze system security configuration policies
- Shipped on every Windows version since Windows XP
- Supports multiple command line options

```
C:\Windows\System32>secedit.exe /export /areas SECURITYPOLICY /cfg output.txt

The task has completed successfully.
See log %windir%\security\logs\scesrv.log for detail info.
```

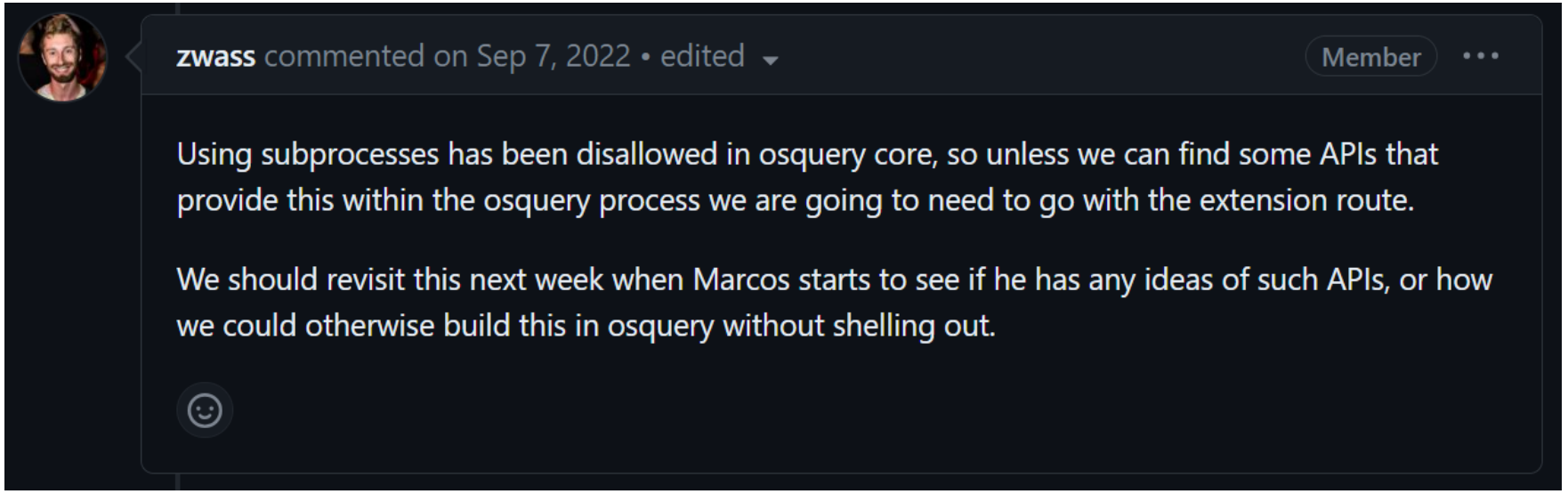
*output.txt - Notepad

File Edit Format View Help

```
[Unicode]
Unicode=yes
[System Access]
MinimumPasswordAge = 5
MaximumPasswordAge = 30
MinimumPasswordLength = 14
PasswordComplexity = 1
PasswordHistorySize = 6
LockoutBadCount = 5
RequireLogonToChangePassword = 0
ForceLogoffWhenHourExpire = 0
NewAdministratorName = "Administrator"
NewGuestName = "Guest"
ClearTextPassword = 1
LSAAnonymousNameLookup = 0
EnableAdminAccount = 0
EnableGuestAccount = 0
[Event Audit]
AuditSystemEvents = 3
AuditLogonEvents = 3
AuditObjectAccess = 3
AuditPrivilegeUse = 3
AuditPolicyChange = 3
AuditAccountManage = 3
AuditProcessTracking = 0
AuditDSAccess = 0
```

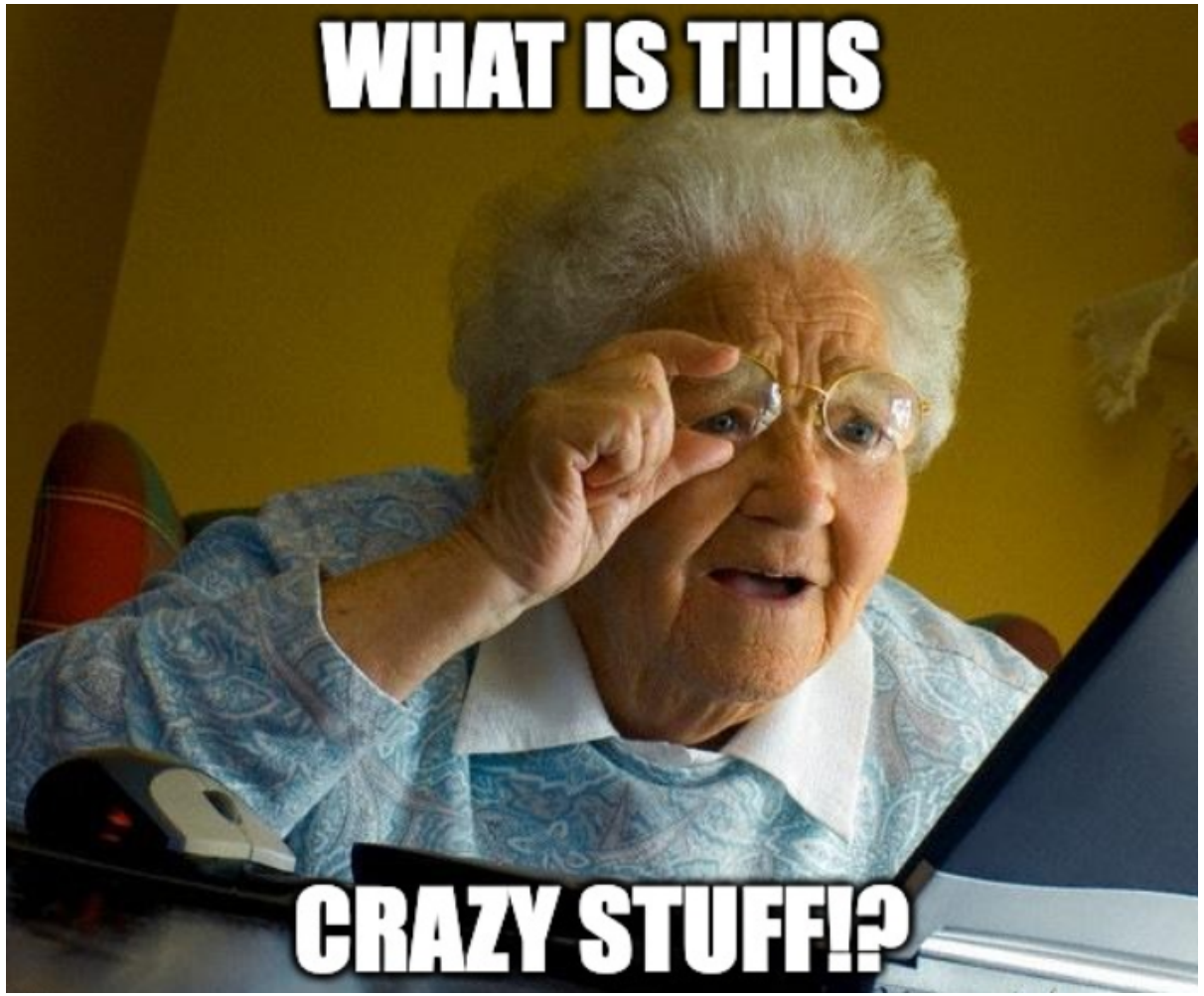
Osquery Integration Gameplan

- **secedit.exe** provides the required data
- Can Osquery call this binary and parse the results?
- Answer is **NO** - Child processes are not allowed



New Approach: Behave like Secedit

- Replicate secedit internal logic and call the same OS APIs
- The problem is that secedit internals are not documented
- **Reverse Engineering** to the rescue!

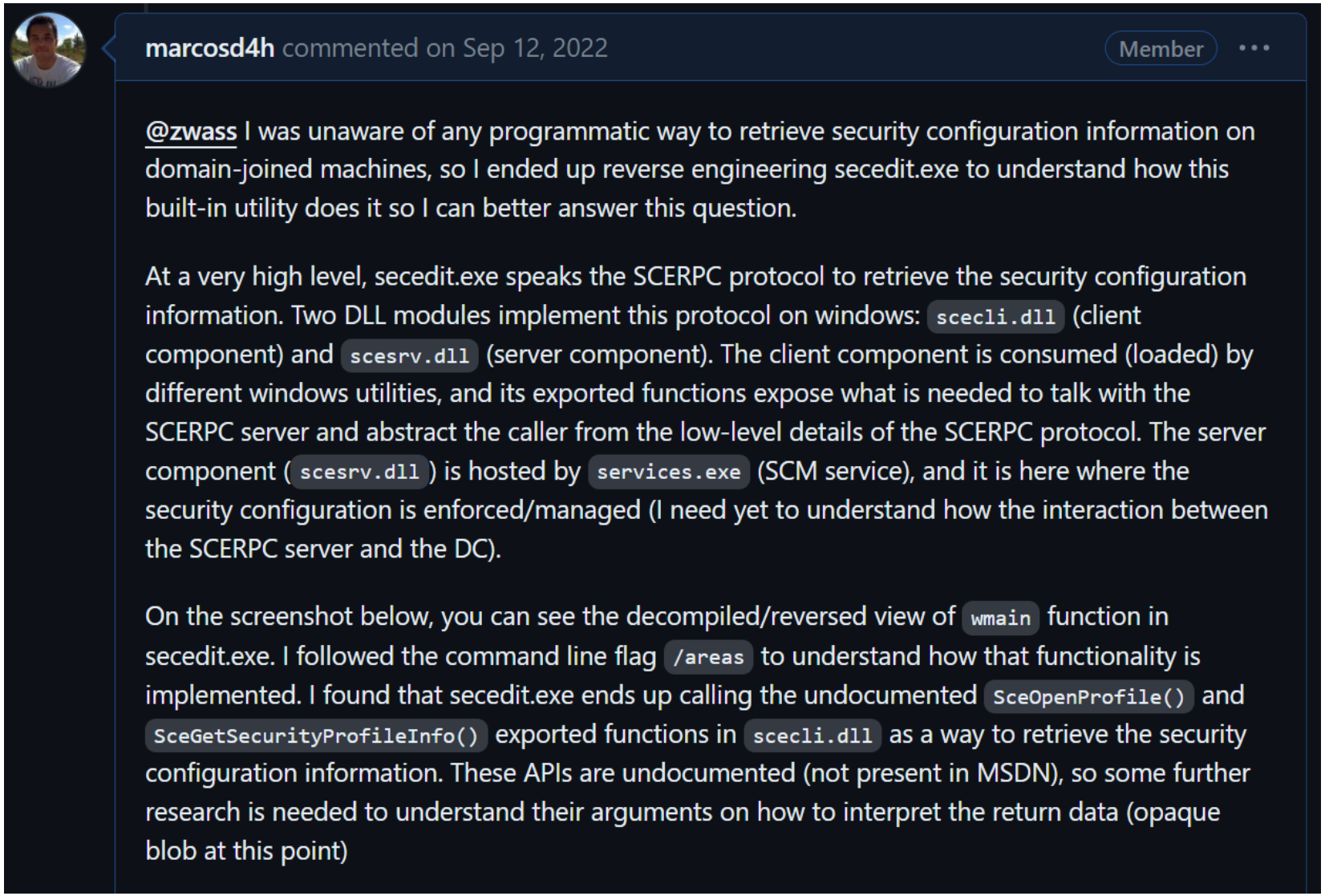


Understanding Secedit.exe

Group	Title	Information
File	Name	secedit.exe
File	Type	64-bit x64 Portable Executable (PE)
File	Size	41.00 KB (41,984 bytes)
File	Creation Time	Saturday 7 December 2019, 06:09
File	Accessed Time	Tuesday 10 October 2023, 15:09
File	Modified Time	Saturday 7 December 2019, 06:09
Hash	File MD5	FE961D8056062E047BCFBD77EBD431B7
Hash	File SHA1	2CD7718827C793F8CA53C9FE0CC11D09E450EA67
Hash	File SHA256	58348E21FB9AE1582E68AA07BF21F87D58DB03FE12123B0DF5B
Format	PE Internal Name	SeCEdit
Format	PE Product Name	Microsoft® Windows® Operating System
Format	PE Company Name	Microsoft Corporation
Format	PE Legal Copyright	© Microsoft Corporation. All rights reserved.
Format	PE Product Version	10.0.19041.1
Format	PE File Description	Windows Security Configuration Editor Command Tool
Format	PE File Version	10.0.19041.1 (WinBuild.160101.0800)
Format	PE Original Filename	SeCEdit
Security	Mitigation	Address Space Layout Randomization (ASLR) →
Security	Mitigation	Data Execution Prevention (DEP) →
Security	Mitigation	Control Flow Guard (CFG) →
Security	Mitigation	High Entropy ASLR →
Security	Mitigation	Export Suppression →
Security	Mitigation	Stack Cookies (GS) →

Type	Value	▼
Import	SceRegisterRegValues	SCECLI.dll
Import	SceAnalyzeSystem	SCECLI.dll
Import	SceIsSystemDatabase	SCECLI.dll
Import	SceFreeProfileMemory	SCECLI.dll
Import	SceGenerateRollback	SCECLI.dll
Import	SceCloseProfile	SCECLI.dll
Import	SceOpenProfile	SCECLI.dll
Import	SceConfigureSystem	SCECLI.dll
Import	SceWriteSecurityProfileInfo	SCECLI.dll
Import	SceGetSecurityProfileInfo	SCECLI.dll
Import	SceBrowseDatabaseTable	SCECLI.dll

Deep dive into secedit.exe internals



marcosd4h commented on Sep 12, 2022 — Member

@zwass I was unaware of any programmatic way to retrieve security configuration information on domain-joined machines, so I ended up reverse engineering secedit.exe to understand how this built-in utility does it so I can better answer this question.

At a very high level, secedit.exe speaks the SCERPC protocol to retrieve the security configuration information. Two DLL modules implement this protocol on windows: `scecli.dll` (client component) and `scesrv.dll` (server component). The client component is consumed (loaded) by different windows utilities, and its exported functions expose what is needed to talk with the SCERPC server and abstract the caller from the low-level details of the SCERPC protocol. The server component (`scesrv.dll`) is hosted by `services.exe` (SCM service), and it is here where the security configuration is enforced/managed (I need yet to understand how the interaction between the SCERPC server and the DC).

On the screenshot below, you can see the decompiled/reversed view of `wmain` function in secedit.exe. I followed the command line flag `/areas` to understand how that functionality is implemented. I found that secedit.exe ends up calling the undocumented `SceOpenProfile()` and `SceGetSecurityProfileInfo()` exported functions in `scecli.dll` as a way to retrieve the security configuration information. These APIs are undocumented (not present in MSDN), so some further research is needed to understand their arguments on how to interpret the return data (opaque blob at this point)

Details here

```
C:\WINDOWS\system32>secedit /export /areas SECURITYPOLICY /cfg output.log

The task has completed successfully.
See log %windir%\security\logs\scesrv.log for detail info.
```

```
if ( !_wcsicmp((const wchar_t *)v68, L"/export") )
{
  if ( v6 )
    goto LABEL_155;
  v6 = 1;
  goto LABEL_148;
}
```

```
if ( _wcsicmp((const wchar_t *)v68, L"/areas") )
{
  if ( _wcsicmp((const wchar_t *)v68, L"/log")
    || (++v21, v21 >= v73)
    || **(_WORD **)(v76 + 8 * v22 + 8) == 47 )
  {
```

```
if ( _wcsicmp((const wchar_t *)v68, L"SECURITYPOLICY") )
{
  if ( _wcsicmp((const wchar_t *)v68, L"GROUP_MGMT") )
  {
    if ( _wcsicmp((const wchar_t *)v68, L"USER_RIGHTS") )
    {
      if ( _wcsicmp((const wchar_t *)v68, L"REGKEYS") )
      {
        if ( _wcsicmp((const wchar_t *)v68, L"FILESTORE") )
        {
          if ( _wcsicmp((const wchar_t *)v68, L"SERVICES") )
            goto LABEL_155;
          v40 = 128;
        }
```

```
switch ( v6 )
{
  case 1u:
    if ( v27 )
    {
      if ( v66 || v30 )
      {
        v61 = v30;
        v15 = (void *)v66;
        Template = SceSetupGenerateTemplate(0i64, v66, v61, v27, lpFileName, v74);
      }
      else
      {
        Template = SceGetSecurityProfileInfo(0i64, 300i64, v74, &v78, 0i64);
        if ( !Template )
          Template = SceWriteSecurityProfileInfo(v27, v74, v78, 0i64);
        if ( v78 )
        {
          SceFreeProfileMemory();
          v78 = 0i64;
        }
        v15 = 0i64;
      }
```

```
                                        ; DATA XREF: wmain+10B5↑r
__int64 __fastcall SceGetSecurityProfileInfo(_QWORD, _QWORD, _QWORD, _QWORD, _QWORD)
            extrn __imp_SceGetSecurityProfileInfo:qword
                                        ; CODE XREF: wmain+11A3↑p
                                        ; wmain+130E↑p
                                        ; DATA XREF: ...
```

```
__int64 SceGetSecurityProfileInfo(void **a1, unsigned __int16 *a2,
{
  struct _SCE_PROFILE_INFO **v3; // r15
  unsigned int v4; // r14d
  int v5; // esi
  char v6; // al
  struct _SCE_ERROR_LOG_INFO **v7; // r12
  CLIENT_CALL_RETURN v8; // rax
  unsigned int Pointer; // ebx
  NTSTATUS v10; // eax
  unsigned int v11; // eax
  CLIENT_CALL_RETURN v12; // rax
  struct _SCE_PROFILE_INFO *v13; // rax
  __int64 i; // rsi
  _QWORD *v15; // rcx
  struct _SCE_PROFILE_INFO *v16; // rcx
  struct _SCE_ERROR_LOG_INFO **v17; // r9
  struct _SCE_PROFILE_INFO *v19; // [rsp+48h] [rbp-50h] BYREF
  HLOCAL hMem; // [rsp+50h] [rbp-48h] BYREF
  void *Simple; // [rsp+58h] [rbp-40h] BYREF
  CLIENT_CALL_RETURN v22; // [rsp+60h] [rbp-38h]
  CLIENT_CALL_RETURN v23; // [rsp+68h] [rbp-30h]
  RPC_BINDING_HANDLE Binding; // [rsp+B8h] [rbp+20h] BYREF
  va_list Bindinga; // [rsp+B8h] [rbp+20h]
  struct _SCE_ERROR_LOG_INFO **v26; // [rsp+C0h] [rbp+28h]
  va_list va1; // [rsp+C8h] [rbp+30h] BYREF
```

NdrClientCall3 function (rpcndr.h)

Article • 10/03/2022

In this article

- Syntax
- Parameters
- Return value
- Requirements

[NdrClientCall3 is not supported and may be altered or unavailable in the future.]

NdrClientCall3 may be altered or unavailable.

Syntax

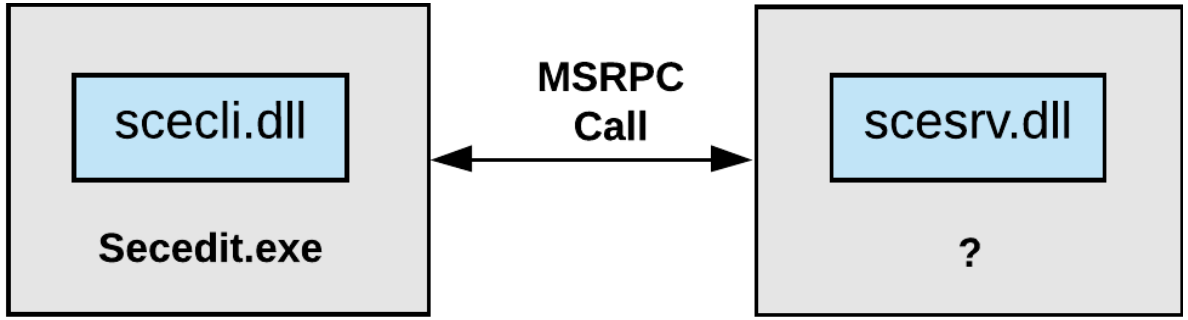
C++

```
CLIENT_CALL_RETURN RPC_VAR_ENTRY NdrClientCall3(
  MIDL_STUBLESS_PROXY_INFO *pProxyInfo,
  unsigned long            nProcNum,
  void                     *pReturnValue,
  ...
);
```

```
v10 = ScepBindSecureRpc(0i64, a2, 0i64, &Simple);
if ( v10 >= 0 )
{
  v22.Pointer = NdrClientCall3((MIDL_STUBLESS_PROXY_INFO *)&pProxyInfo, 28u,
  Pointer = (unsigned int)v22.Pointer;
  Binding = Simple;
  if ( Simple )
    RpcBindingFree((RPC_BINDING_HANDLE *)Bindinga);
  goto LABEL_21;
}
```

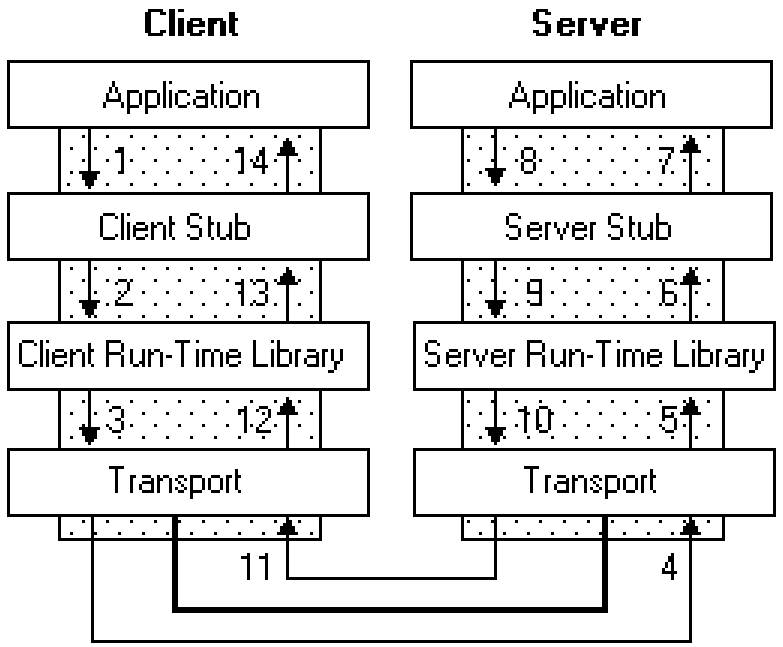
Secedit Reversing: Recap

- Our target command line options ends up calling **SceGetSecurityProfileInfo**
- This function is an export of **scecli.dll**
- **SceGetSecurityProfileInfo** calls a remote method (num 28) from an MSRPC interface
- SCE RPC Server implemented by **scesrv.dll** **93149ca2-973b-11d1-8c39-00c04fb984f9**



What is MSRPC?

- Microsoft RPC (**MSRPC**) enables clients to call functions on remote servers using a standard interface
- Multiple transport protocols supported (ALPC, Named Pipes, TCP, UDP, HTTP, etc)
- Widely used by Windows components



MSRPC Runtime lives in rpcrt4.dll

Dissecting SceSvr.dll

- Windows Security Configuration Engine (SCE) runs inside of services.exe
- Initializes SCE RPC interface at **ScepStartServerServices()**
- Named pipe transport protocol used **\\pipe\\scerpc**

```
started = RpcServerRegisterAuthInfoW(0i64, 0xAu, 0i64, 0i64);
if ( started >= 0 )
{
  v2 = (wchar_t *)LocalAlloc(0, 0x1Cui64);
  v3 = v2;
  if ( v2 )
  {
    wcscpy_s(v2, 0xEui64, L"\\PIPE\\");
    wcscat_s(v3, 0xEui64, L"scerpc");
    v4 = RpcServerUseProtseqEpW((RPC_WSTR)L"ncacn_np", 0xAu, v3, 0i64);
    v5 = v4;
    if ( !v4 || v4 == 1740 )
      v5 = RpcServerRegisterIfEx(&unk_180065000, 0i64, 0i64, 0x21u, 0x4D2u, 0i64);
    LocalFree(v3);
    v6 = I_RpcMapWin32Status(v5);
  }
```

Dissecting SceSvr.dll

```
RPC 93149ca2-973b-11d1-8c39-00c04fb984f9
C:\windows\SYSTEM32\scesrv.dll
        0 -> SceSvcRpcQueryInfo
        1 -> SceSvcRpcSetInfo
        2 -> SceRpcSetupUpdateObject
        3 -> SceRpcSetupMoveFile
        4 -> SceRpcGenerateTemplate
        5 -> SceRpcConfigureSystem
        6 -> SceRpcGetDatabaseInfo
        7 -> SceRpcGetObjectChildren
        8 -> SceRpcOpenDatabase
        9 -> SceRpcCloseDatabase
       10 -> SceRpcGetDatabaseDescription
       11 -> SceRpcGetDBTimeStamp
       12 -> SceRpcGetObjectSecurity
       13 -> SceRpcGetAnalysisSummary
       14 -> SceRpcAnalyzeSystem
       15 -> SceRpcUpdateDatabaseInfo
```

```
16 -> SceRpcUpdateObjectInfo
17 -> SceRpcStartTransaction
18 -> SceRpcCommitTransaction
19 -> SceRpcRollbackTransaction
20 -> SceRpcGetServerProductType
21 -> SceSvcRpcUpdateInfo
22 -> SceRpcCopyObjects
23 -> SceRpcSetupResetLocalPolicy
24 -> SceRpcNotifySaveChangesInGP
25 -> SceRpcControlNotificationQProcess
26 -> SceRpcBrowseDatabaseTable
27 -> SceRpcNotifySystemSecurity
28 -> SceRpcGetSystemSecurity
29 -> SceRpcNotifySystemSecurity
30 -> SceRpcSetSystemSecurity
31 -> SceRpcSetDatabaseSetting
32 -> SceRpcGetDatabaseSetting
33 -> SceRpcConfigureConvertedFileSecurity
```

Dissecting SceSvr.dll

- Proc 28 is implemented by **SceRpcGetSystemSecurity()**
- It performs access check on calling thread before retrieving policy data
- Security Policy data is obtained from SCE ESE DB
  - %windir%\security\database\secedit.sdb
- DB is locked, not possible to read the data from there

```
v10 = 0;
if ( I_RpcBindingIsClientLocal(0i64, &v10) || !v10 )
  return 9;
if ( dword_18008B4C0 )
  return 14;
result = RpcImpersonateClient(0i64);
if ( !result )
{
  IsMember[0] = 0;
  IsAdminLoggedOn = ScepIsAdminLoggedOn(IsMember);
  if ( ScepDosErrorToSceStatus(IsAdminLoggedOn) || !I
  {
    RpcRevertToSelf();
    return 23;
  }
  else
  {
    RpcRevertToSelf();
    result = ScepGetSystemSecurity(a2, a3, a4, a5);
    IsMember[1] = result;
  }
}
```

Secedit Reversing Recap

1. Secedit uses **scecli.dll** to call a remote SCE RPC interface
2. SCE RPC interface is implemented by **scesrv.dll** and runs inside services.exe
3. Called method **SceRpcGetSystemSecurity()** retrieves the policy data from locked ESE DB

Demo:
Secedit.exe
reversing

Osquery Implementation Options

RPC Client

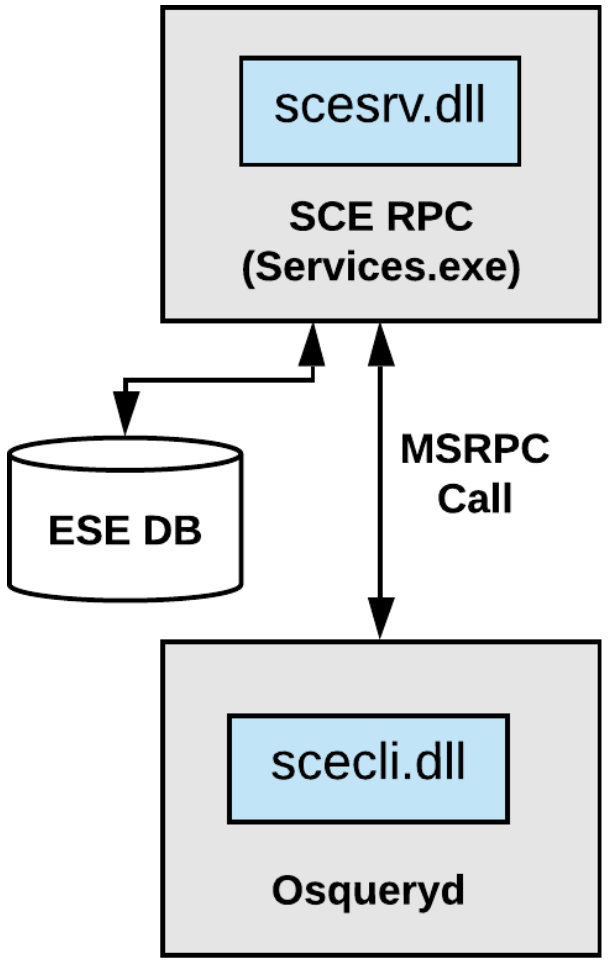
- Requires IDL maintenance and toolchain changes
- Complex to review and merge

Shadow ESE DB Read

- Modifies system state, not desired in Osquery

Calling export from scecli.dll

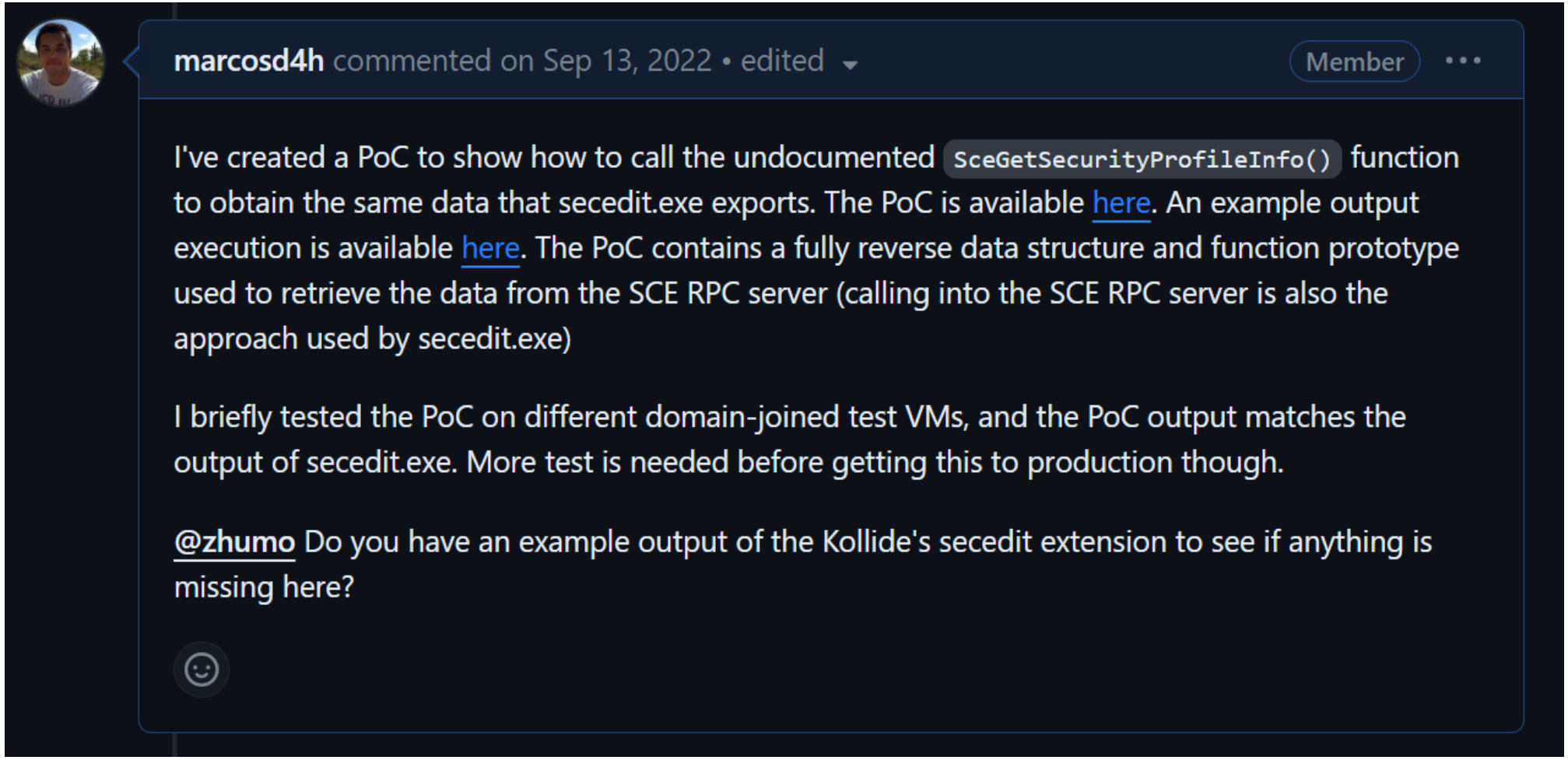
- RPC Client is always up-to-date
- Export signature extraction is possible
- No issues with consuming undocumented functions



Calling undocumented DLL exports

- Runtime Dynamic Linking can be used to consume DLL exports
- Technique relies on mapping the DLL to memory through **LoadLibrary()** or **LoadLibraryEx()**
- And then getting the address to the memory-mapped export through **GetProcAddress()**

Calling undocumented DLL exports



marcosd4h commented on Sep 13, 2022 • edited

Member

I've created a PoC to show how to call the undocumented `SceGetSecurityProfileInfo()` function to obtain the same data that secedit.exe exports. The PoC is available here. An example output execution is available here. The PoC contains a fully reverse data structure and function prototype used to retrieve the data from the SCE RPC server (calling into the SCE RPC server is also the approach used by secedit.exe)

I briefly tested the PoC on different domain-joined test VMs, and the PoC output matches the output of secedit.exe. More test is needed before getting this to production though.

@zhumo Do you have an example output of the Kollide's secedit extension to see if anything is missing here?

Details here

Calling undocumented DLL exports

```
//function pointer definition
typedef DWORD(*sce_get_security_profile_info)(
    PVOID profile_handle,
    DWORD type,
    DWORD security_area,
    PVOID profile_info,
    PVOID error_info);


int wmain()
{
    int ret = EXIT_SUCCESS;

    //loading the scecli.dll module (SCE RPC Client DLL)
    HMODULE scelib_handler = LoadLibraryExW(L"scecli.dll", 0, LOAD_LIBRARY_SEARCH_SYSTEM32);
    if (!scelib_handler)
    {
        std::wcout << L"The windows security configuration client DLL (scecli.dll) couldn't be loaded. " << std::endl;
        return EXIT_FAILURE;
    }
```

Calling undocumented DLL exports

```
//grabbing a raw pointer to one of its exported functions to perform runtime loading
PVOID func_addr = GetProcAddress(scelib_handler, "SceGetSecurityProfileInfo");
if (!func_addr)
{
    std::wcout << L"The exported function SceGetSecurityProfileInfo couldn't be found at scecli.dll." << std::endl;
    return EXIT_FAILURE;

}

//then casting the raw pointer to a known function pointer
sce_get_security_profile_info sce_getsecprofileinfo_ptr = reinterpret_cast<sce_get_security_profile_info>(func_addr);
if (!sce_getsecprofileinfo_ptr)
{
    std::wcout << L"There was a problem casting the pointer to SceGetSecurityProfileInfo function. " << std::endl;
    return EXIT_FAILURE;
}
```

Calling undocumented DLL exports

```
//and finally calling into the function passing the expected arguments
PVOID profile_data = nullptr;
PVOID error_info = nullptr;
DWORD ret_code = sce_getsecprofileinfo_ptr(nullptr, SCE_SYSTEM_FLAG, SCE_AREA_ALL_FLAG, &profile_data, &error_info);
if (ret_code != ERROR_SUCCESS)
{
    std::wcout << L"There was a problem calling into SceGetSecurityProfileInfo function - Error code is 0x" << std::h
    std::wcout << L"Did you run the poc as admin?" << ret_code << std::endl;
    return EXIT_FAILURE;
}

//sanity check before consuming the mem data
if (IsBadReadPtr(profile_data, sizeof(SCE_PROFILE_INFO)))
{
    std::wcout << L"The data returned by SceGetSecurityProfileInfo function was not right" << std::endl;
    return EXIT_FAILURE;
}

//data is ready to be consumed!
SCE_PROFILE_INFO *profile_info_data = (SCE_PROFILE_INFO*)(profile_data);
```

Calling undocumented DLL exports

```
struct SCE_PROFILE_INFO {
    DWORD unk0;
    DWORD min_passwd_age;
    DWORD max_passwd_age;
    DWORD min_passwd_len;
    DWORD passwd_complexity;
    DWORD passwd_hist_size;
    DWORD lockout_bad_count;
    DWORD reset_lockout_count;
    DWORD lockout_duration;
    DWORD req_logon_change_passwd;
    DWORD force_logoff_expire;
    PWSTR administrator_name;
    PWSTR guest_name;
    DWORD unk1;
    DWORD clear_text_passwd;
    DWORD lsa_allow_anonymous_sid_lookup;
    PVOID unk2;
    PVOID unk3;
    PVOID unk4;
    PVOID unk5;
    PVOID unk6;
    PVOID unk7;
    PVOID unk8;
    PVOID unk9;
    DWORD max_log_size[3];
```

Fun Fact:
Github has only one example of code calling into **SceGetSecurityProfileInfo**

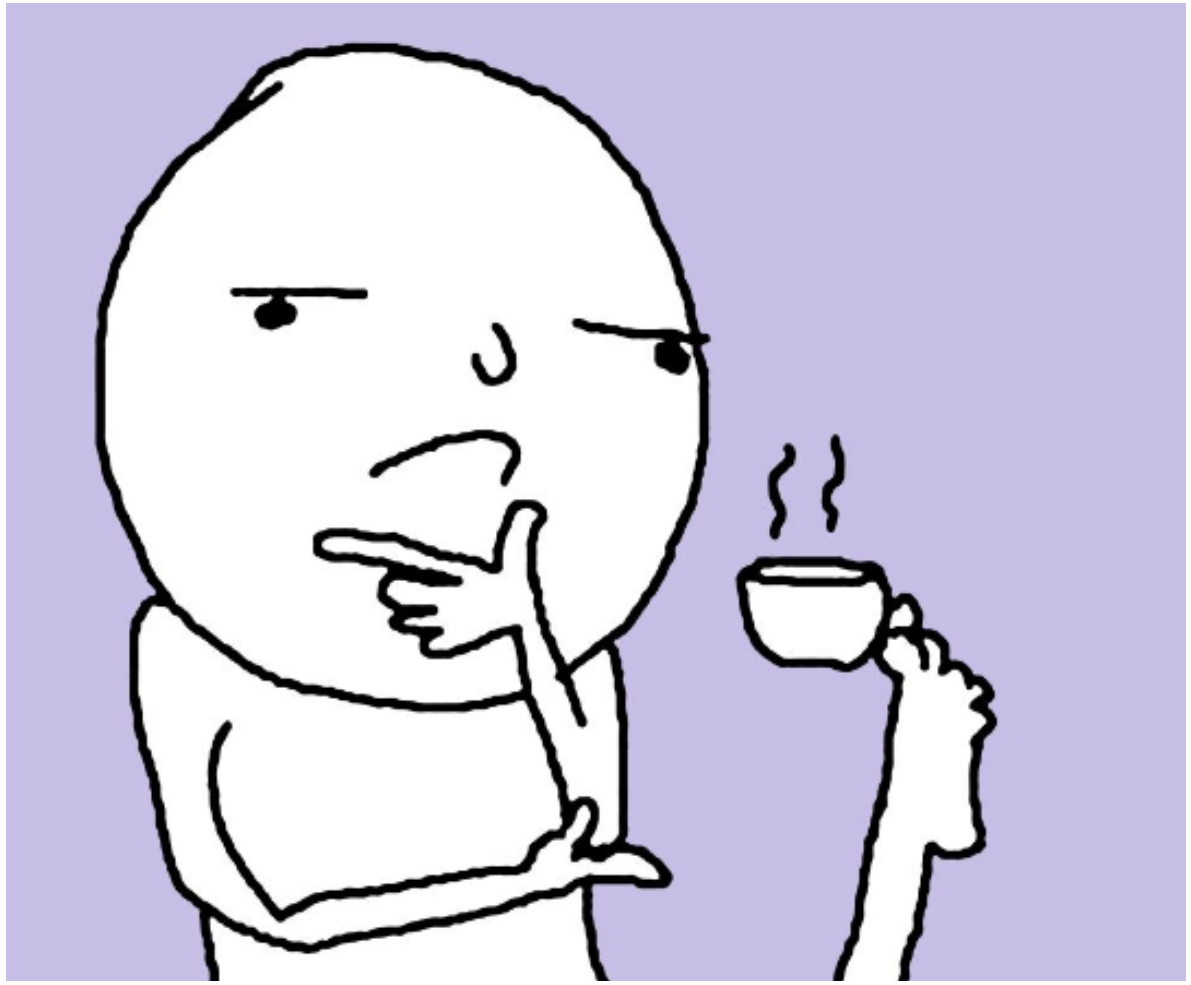
It seems that the **NSA** did the same reversing job to create a Windows Powershell Compliance Script

nsacyber / Windows-Secure-Host-Baseline

```
263         sealed class WinSceCli {
264             [DllImport("scecli.dll", CallingConvention = CallingConvention.StdCall)]
265             internal static extern uint SceGetSecurityProfileInfo(
266                 int arg1,
267                 int arg2,
268                 int arg3,
269                 out IntPtr buffer,
270                 out IntPtr optBuf
271             );
272
273             [DllImport("scecli.dll")]
274             internal static extern int SceFreeProfileMemory(IntPtr Buffer);
275         }
```

PoC is working, what now?

- Ok, we have a programmatic way to retrieve password policies data
- What should we do to get this into Osquery?



Osquery contribution guidelines

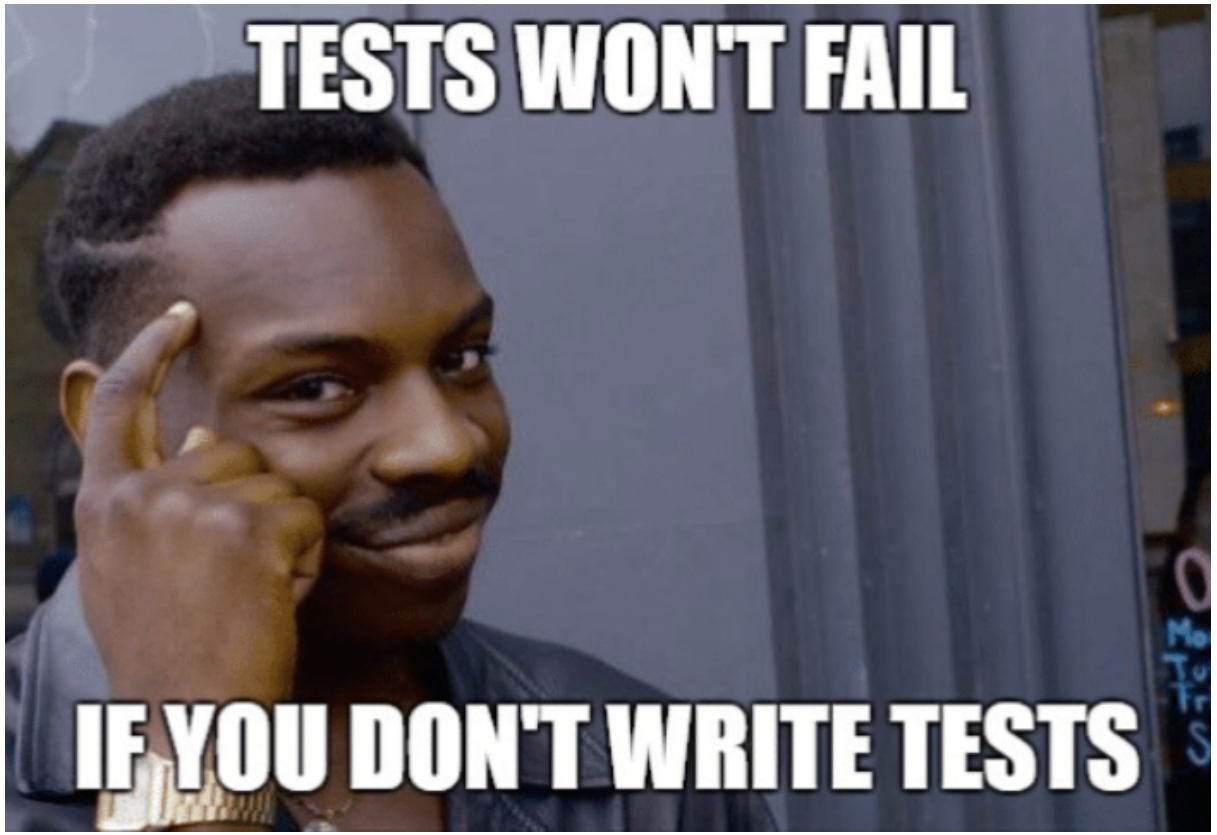
- Contributing to **Osquery** is not difficult
- There is an amazing contributing guideline here
- Osquery follows the **LLVM** Coding Standards detailed here
- There is also Osquery Slack with a lot of people willing to help

Pull Request Challenges

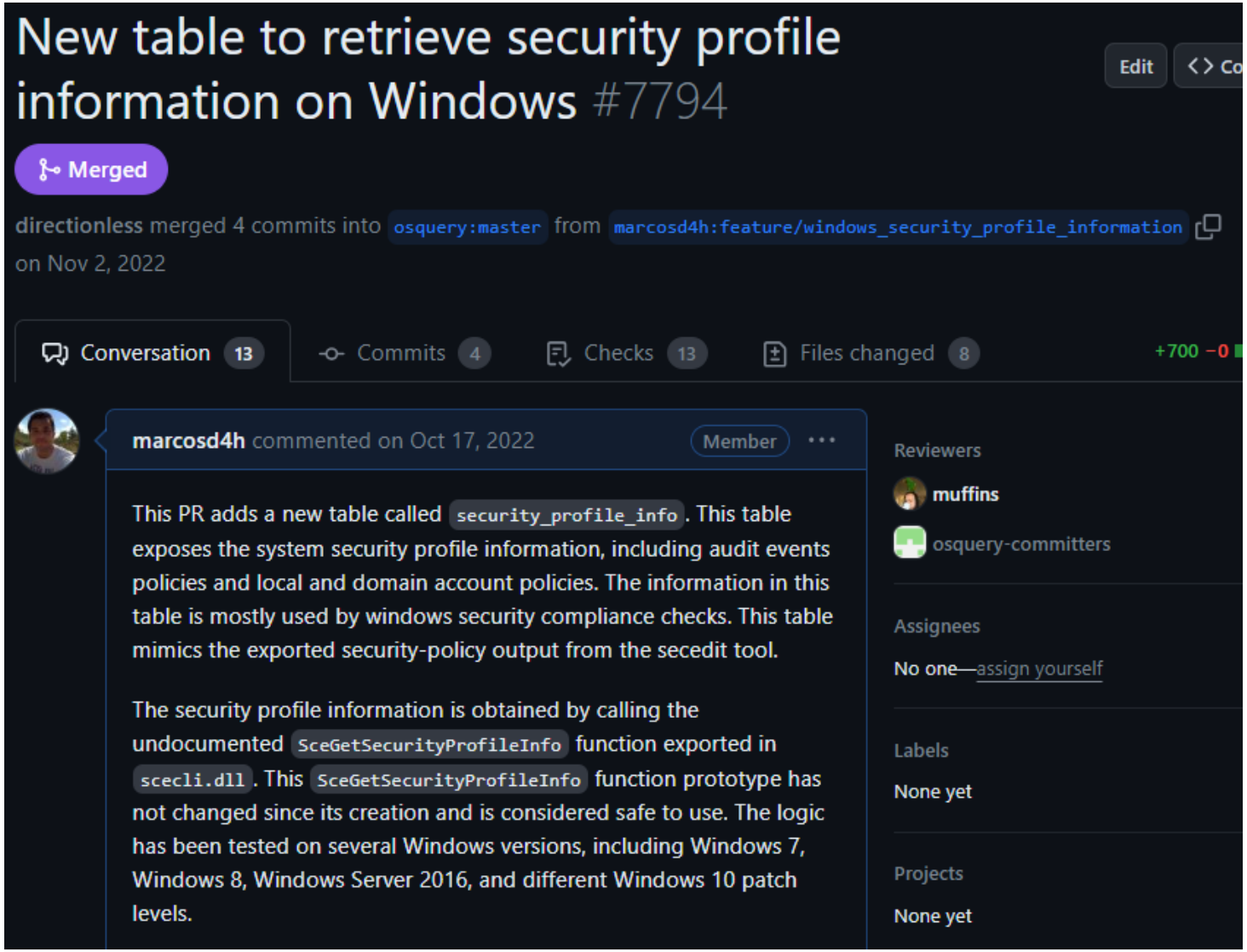
- Getting your PR code reviewed is the most difficult part
- This usually involves bothering people to look into your code changes
- ...and a few iterations until the PR gets approved and merged
- **Pro tip:** Deliver code in logical chunks for faster review

Quality Assurance

- Testing your changes is a big part of the work
- Add automated tests & combine with manual testing.
- Good OS coverage is key here!



Osquery Contribution - PR7794



Details here

Osquery Contribution - PR7794

```
table_name("security_profile_info")
description("Information on the security profile of a given system by listing the system Account and Audit
schema([
    Column("minimum_password_age", INTEGER, "Determines the minimum number of days that a password must be
    Column("maximum_password_age", INTEGER, "Determines the maximum number of days that a password can be u
    Column("minimum_password_length", INTEGER, "Determines the least number of characters that can make up
    Column("password_complexity", INTEGER, "Determines whether passwords must meet a series of strong-passw
    Column("password_history_size", INTEGER, "Number of unique new passwords that must be associated with a
    Column("lockout_bad_count", INTEGER, "Number of failed logon attempts after which a user account MUST b
    Column("logon_to_change_password", INTEGER, "Determines if logon session is required to change the pass
    Column("force_logoff_when_expire", INTEGER, "Determines whether SMB client sessions with the SMB server
    Column("new_administrator_name", TEXT, "Determines the name of the Administrator account on the local c
    Column("new_guest_name", TEXT, "Determines the name of the Guest account on the local computer"),
    Column("clear_text_password", INTEGER, "Determines whether passwords MUST be stored by using reversible
    Column("lsa_anonymous_name_lookup", INTEGER, "Determines if an anonymous user is allowed to query the l
    Column("enable_admin_account", INTEGER, "Determines whether the Administrator account on the local comp
    Column("enable_guest_account", INTEGER, "Determines whether the Guest account on the local computer is
    Column("audit_system_events", INTEGER, "Determines whether the operating system MUST audit System Chang
    Column("audit_logon_events", INTEGER, "Determines whether the operating system MUST audit each instance
    Column("audit_object_access", INTEGER, "Determines whether the operating system MUST audit each instanc
    Column("audit_privilege_use", INTEGER, "Determines whether the operating system MUST audit each instanc
    Column("audit_policy_change", INTEGER, "Determines whether the operating system MUST audit each instanc
    Column("audit_account_manage", INTEGER, "Determines whether the operating system MUST audit each event
    Column("audit_process_tracking", INTEGER, "Determines whether the operating system MUST audit process-r
    Column("audit_ds_access", INTEGER, "Determines whether the operating system MUST audit each instance of
    Column("audit_account_logon", INTEGER, "Determines whether the operating system MUST audit each time th
])
implementation("security_profile_info@genSecurityProfileInformation")
```

Details here

Osquery Contribution - PR7794

The security_profile_info table was released on Osquery version 5.7.0 (Dec 2022)


```
osquery> select maximum_password_age from security_profile_info;
+----------------------+
| maximum_password_age |
+----------------------+
| 43                   |
+----------------------+
```

Demo: Osquery table **security_profile_info**

Story #2
Windows Process Auditing

Support process auditing on Windows #7732

Closed zhumo opened this issue on Sep 13, 2022 · 5 comments

zhumo commented on Sep 13, 2022 • edited Contributor

Problem

Process auditing is when process and network connection events are tracked. Osquery currently supports process auditing for Mac and Linux, but does not for Windows via the osquery eventing framework.

Requirements

For the following tables:

- process_events
- socket_events
- process_file_events
- bpf_process_events
- es_process_file_events

create a separate table(s) which features the same set of information for Windows. Or, if possible, integrate the windows data into the existing tables.

Related

- Polyglot/elasticiq's osquery extension for windows: https://github.com/polylogyx/osq-ext-bin
- Osquery Blueprint Improving Osquery Visibility with Windows ETW osquery/osquery#7826
- Osquery PR Add ETW-based process events table for Windows osquery/osquery#7821

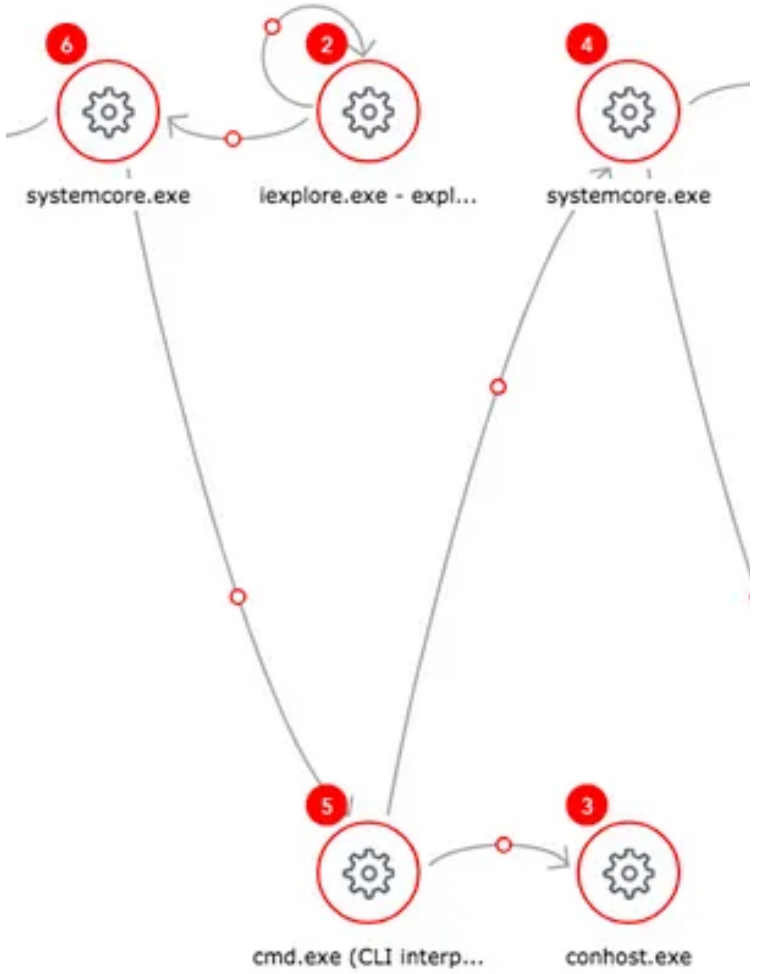
Issue #7732

The Problem

- **Evolution of Windows threat landscape**
- **Osquery's limited visibility on Windows processes**
- **Need common OS visibility framework**

Need for Windows Process Auditing

- Trace start/stop of processes over time
- Enhance threat detection and system monitoring
- Fill visibility gaps at one of the most basic levels: **process lifecycles**

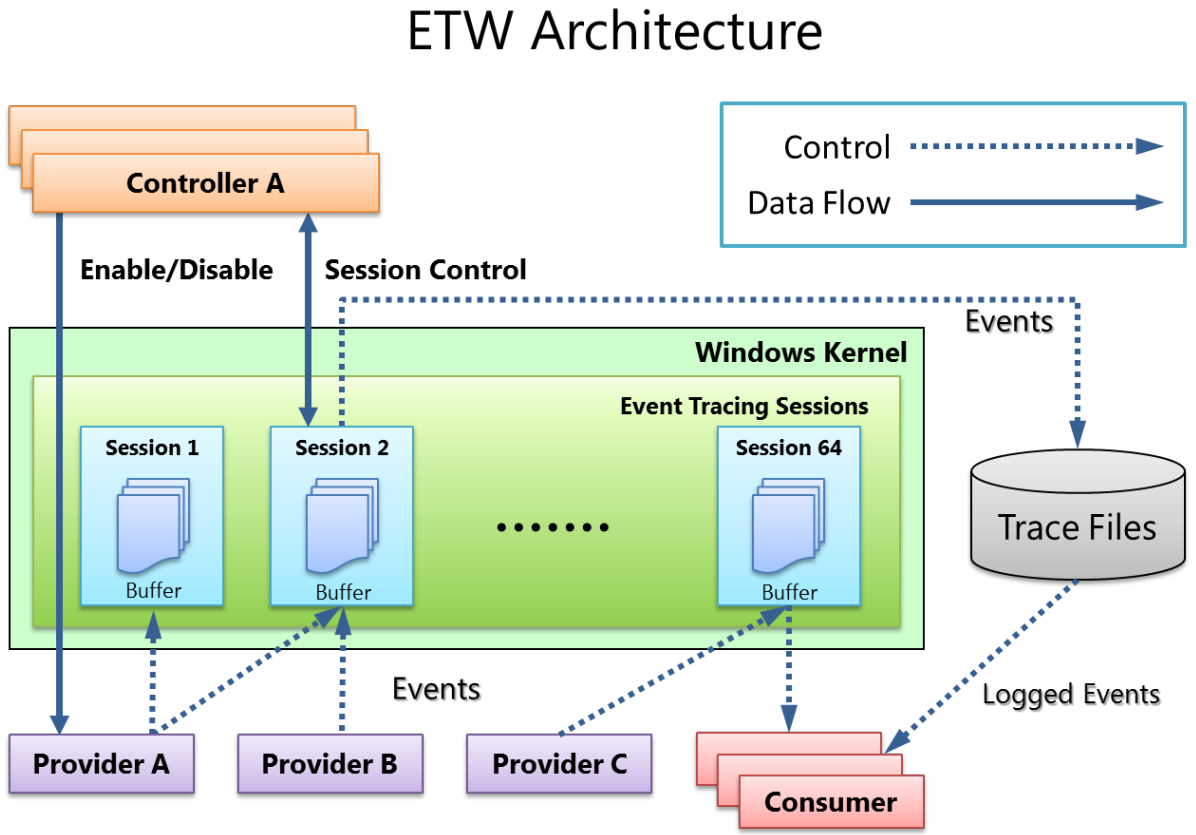


Proposed Enhancement

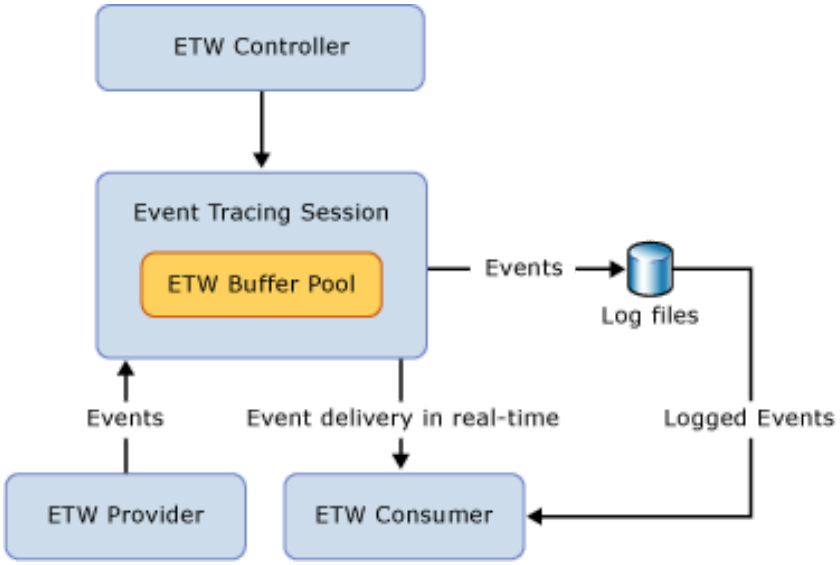
1. **Utilize ETW to enhance Osquery Windows visibility**
2. **Create an internal framework to abstract low-level ETW aspects**
3. **Develop Osquery evented table for process audit on Windows**

ETW Introduction

- High-speed, and low-overhead tracing framework
- Does not require recompilation
- Can be turned on on-demand while running
- User-Mode and Kernel-Mode tracing



ETW Introduction




```
C:\WINDOWS\system32>logman.exe query providers

Provider                                   GUID
-------------------------------------------------------------------------------
.NET Common Language Runtime               {E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}
ACPI Driver Trace Provider                 {DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: SAM      {8E598056-8993-11D2-819E-0000F875A064}
Active Directory: Kerberos Client          {BBA3ADD2-C229-4CDB-AE2B-57EB6966B0C4}
Active Directory: NetLogon                 {F33959B4-DBEC-11D2-895B-00C04F79AB69}
ADODB.1                                    {04C8A86F-3369-12F8-4769-24E484A9E725}
ADOMD.1                                    {7EA56435-3F2F-3F63-A829-F0B35B5CAD41}
Application Popup                          {47BFA2B7-BD54-4FAC-B70B-29021084CA8F}
Application-Addon-Event-Provider           {A83FA99F-C356-4DED-9FD6-5A5EB8546D68}
ATA Port Driver Tracing Provider           {D08BD885-501E-489A-BAC6-B7D24BFE6BBF}
AuthFw NetShell Plugin                     {935F4AE6-845D-41C6-97FA-380DAD429B72}
BCP.1                                      {24722B88-DF97-4FF6-E395-DB533AC42A1E}
BFE Trace Provider                         {106B464A-8043-46B1-8CB8-E92A0CD7A560}
BITS Service Trace                         {4A8AAA94-CFC4-46A7-8E4E-17BC45608F0A}
Certificate Services Client CredentialRoaming Trace {EF4109DC-68FC-45AF-B329-CA2
Certificate Services Client Trace          {F01B7774-7ED7-401E-8088-B576793D7841}
Circular Kernel Session Provider           {54DEA73A-ED1F-42A4-AF71-3E63D056F174}
Classpnp Driver Tracing Provider           {FA8DE7C4-ACDE-4443-9994-C4E2359A9EDB}
Critical Section Trace Provider            {3AC66736-CC59-4CFF-8115-8DF50E39816B}
DBNETLIB.1                                 {BD568F20-FCCD-B948-054E-DB3421115D61}
Deduplication Tracing Provider             {5EBB59D1-4739-4E45-872D-B8703956D84B}
Disk Class Driver Tracing Provider         {945186BF-3DD6-4F3F-9C8E-9EDD3FC9D558}
```

Real-time Event Delivery

- Osquery Evented tables run on callbacks
- OS callbacks use asynchronous threads
- Undisturbed applications event reception
- Ideal for **Osquery evented framework!**

ETW Events

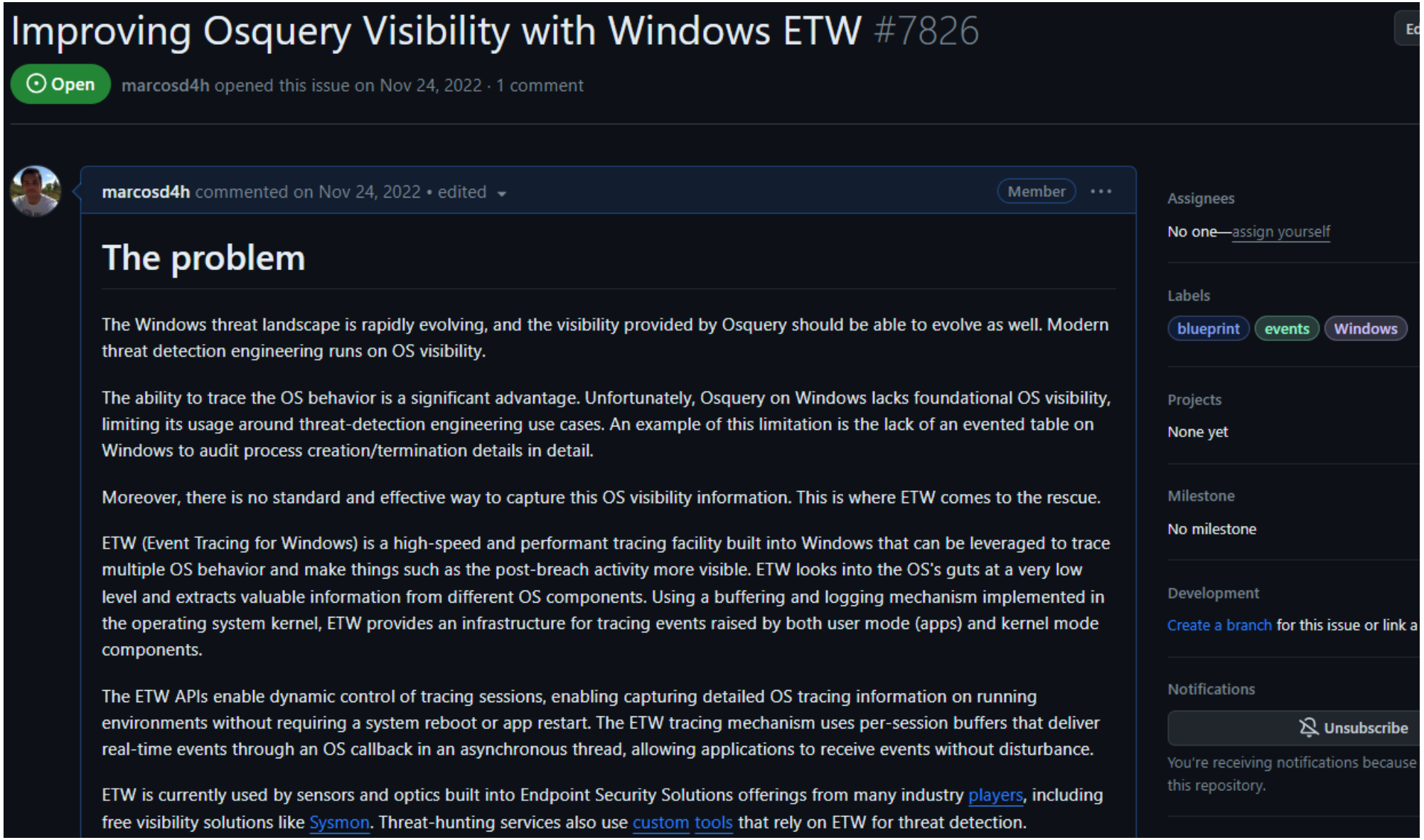
Dealing with **ETW APIs** is hard

An abstraction layer is needed

Introducing POTE

- **Programmable OS Tracing Engine (POTE)**
- **ETW layer to add tracing events as plugins**
- **Can consume Kernel and User ETW providers**
- **Event pre/post-process callbacks for filtering, enrichment & aggregation**

POTE as a Osquery Blueprint!



Improving Osquery Visibility with Windows ETW #7826

Open — marcosd4h opened this issue on Nov 24, 2022 · 1 comment

marcosd4h commented on Nov 24, 2022 • edited — Member

The problem

The Windows threat landscape is rapidly evolving, and the visibility provided by Osquery should be able to evolve as well. Modern threat detection engineering runs on OS visibility.

The ability to trace the OS behavior is a significant advantage. Unfortunately, Osquery on Windows lacks foundational OS visibility, limiting its usage around threat-detection engineering use cases. An example of this limitation is the lack of an evented table on Windows to audit process creation/termination details in detail.

Moreover, there is no standard and effective way to capture this OS visibility information. This is where ETW comes to the rescue.

ETW (Event Tracing for Windows) is a high-speed and performant tracing facility built into Windows that can be leveraged to trace multiple OS behavior and make things such as the post-breach activity more visible. ETW looks into the OS's guts at a very low level and extracts valuable information from different OS components. Using a buffering and logging mechanism implemented in the operating system kernel, ETW provides an infrastructure for tracing events raised by both user mode (apps) and kernel mode components.

The ETW APIs enable dynamic control of tracing sessions, enabling capturing detailed OS tracing information on running environments without requiring a system reboot or app restart. The ETW tracing mechanism uses per-session buffers that deliver real-time events through an OS callback in an asynchronous thread, allowing applications to receive events without disturbance.

ETW is currently used by sensors and optics built into Endpoint Security Solutions offerings from many industry players, including free visibility solutions like Sysmon. Threat-hunting services also use custom tools that rely on ETW for threat detection.

Assignees: No one—assign yourself

Labels: blueprint, events, Windows

Projects: None yet

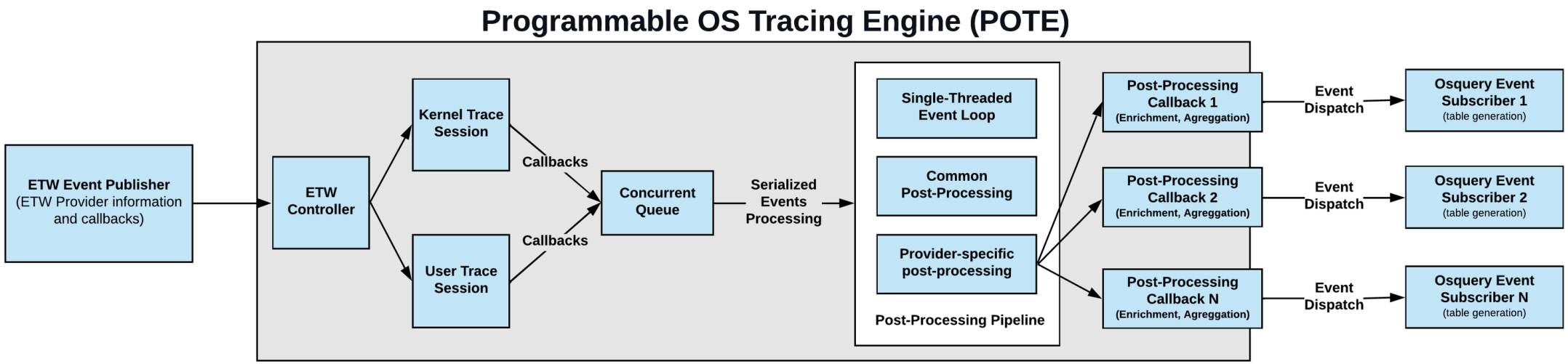
Milestone: No milestone

Development: Create a branch for this issue or link a

Notifications: Unsubscribe

You're receiving notifications because this repository.

POTE as a Osquery Blueprint!



See Osquery issue #7826 for more details

Process Audit using POTE

- **Visibility on Processes Creation/Termination**
- **Uses Kernel and Userspace Providers**
- **Data is enriched for threat-hunting usecases**

Process Audit using POTE

```
table_name("process_etw_events")
description("Windows process execution events.")
schema([
    Column("type", TEXT, "Event Type (ProcessStart, ProcessStop)"),
    Column("pid", BIGINT, "Process ID"),
    Column("ppid", BIGINT, "Parent Process ID"),
    Column("session_id", INTEGER, "Session ID"),
    Column("flags", INTEGER, "Process Flags"),
    Column("exit_code", INTEGER, "Exit Code - Present only on ProcessStop events"),
    Column("path", TEXT, "Path of executed binary"),
    Column("cmdline", TEXT, "Command Line"),
    Column("username", TEXT, "User rights - primary token username"),
    Column("token_elevation_type", TEXT, "Primary token elevation type - Present only on ProcessStart events"),
    Column("token_elevation_status", INTEGER, "Primary token elevation status - Present only on ProcessStart events"),
    Column("mandatory_label", TEXT, "Primary token mandatory label sid - Present only on ProcessStart events"),
    Column("datetime", DATETIME, "Event timestamp in DATETIME format"),
    Column("time_windows", BIGINT, "Event timestamp in Windows format", hidden=True),
    Column("time", BIGINT, "Event timestamp in Unix format", hidden=True),
    Column("eid", INTEGER, "Event ID", hidden=True),
    Column("header_pid", BIGINT, "Process ID of the process reporting the event", hidden=True),
    Column("process_sequence_number", BIGINT, "Process Sequence Number - Present only on ProcessStart events", hidden=True),
    Column("parent_process_sequence_number", BIGINT, "Parent Process Sequence Number - Present only on ProcessStart events", hidden=True),
])
attributes(event_subscriber=True)
implementation("process_etw_events@etw_process_events::genTable")
examples([
        "select * from process_etw_events;",
        "select * from process_etw_events WHERE time >= (( SELECT unix_time FROM time) - 60 );",
        "select * from process_etw_events WHERE datetime > '2022-11-18 16:48:00';",
        "select * from process_etw_events WHERE datetime BETWEEN '2022-11-18 16:40:00' AND '2022-11-18 16:50:00';"
])
```

But wait, where is the data coming from?

- **We need to ensure that data cannot be tampered**
- **We should use ETW Kernel Provider Microsoft-Windows-Kernel-Process** {22fb2cd6-0e7b-422b-a0c7-2fad1fd0e716}
- **Can we ensure this is coming from the kernel?**
- **Reverse engineering to the rescue!**

Demo: Reverse Engineering The Windows Kernel

Add ETW-based process events table for Windows #7821

Merged zwass merged 29 commits into `osquery:master` from `marcosd4h:feature/windows_process_events` on Feb 7

Conversation 139 | Commits 29 | Checks 14 | Files changed 37

zwass commented on Nov 17, 2022 • edited

This PR implements POTE (see Blueprint) and the new evented table `process_etw_events`.

Blueprint: #7826

This work done by @marcosd4h. I just opened the PR.

marcosd4h commented on Nov 24, 2022 — Member

Small clarification here: **@zwass** created this PR by mistake while we were reviewing the initial specs for this new evented table. That's why there are confusing details in the PR description.

This PR introduces POTE (Programmable OS Tracing Engine) framework + a new windows evented table called `etw_process_events` which is built on top of POTE. The main purpose of this new evented table is to audit process create/termination details.

Having POTE in place will simplify the addition of future evented tables as POTE provides a simplified mechanism to create ETW-based Event publishers.

The PR blueprint is detailed here.

Osquery Contribution - PR7821

The process_etw_events table was released on Osquery version 5.8.1 (March 2023)


```
C:\tools\osquery>osqueryi --disable-events=false --enable_process_etw_events=true
Using a [1mvirtual database[0m. Need help, type '.help'
osquery> select pid, cmdline from process_etw_events where type = "ProcessStart" limit 1;
+-------+---------------------------+
| pid   | cmdline                   |
+-------+---------------------------+
| 12976 | notepad  "Hello Ekoparty" |
+-------+---------------------------+
```

Demo: Osquery table **process_etw_events**

Thanks!
Questions?

moviedo@gmail.com

@marcosd4h