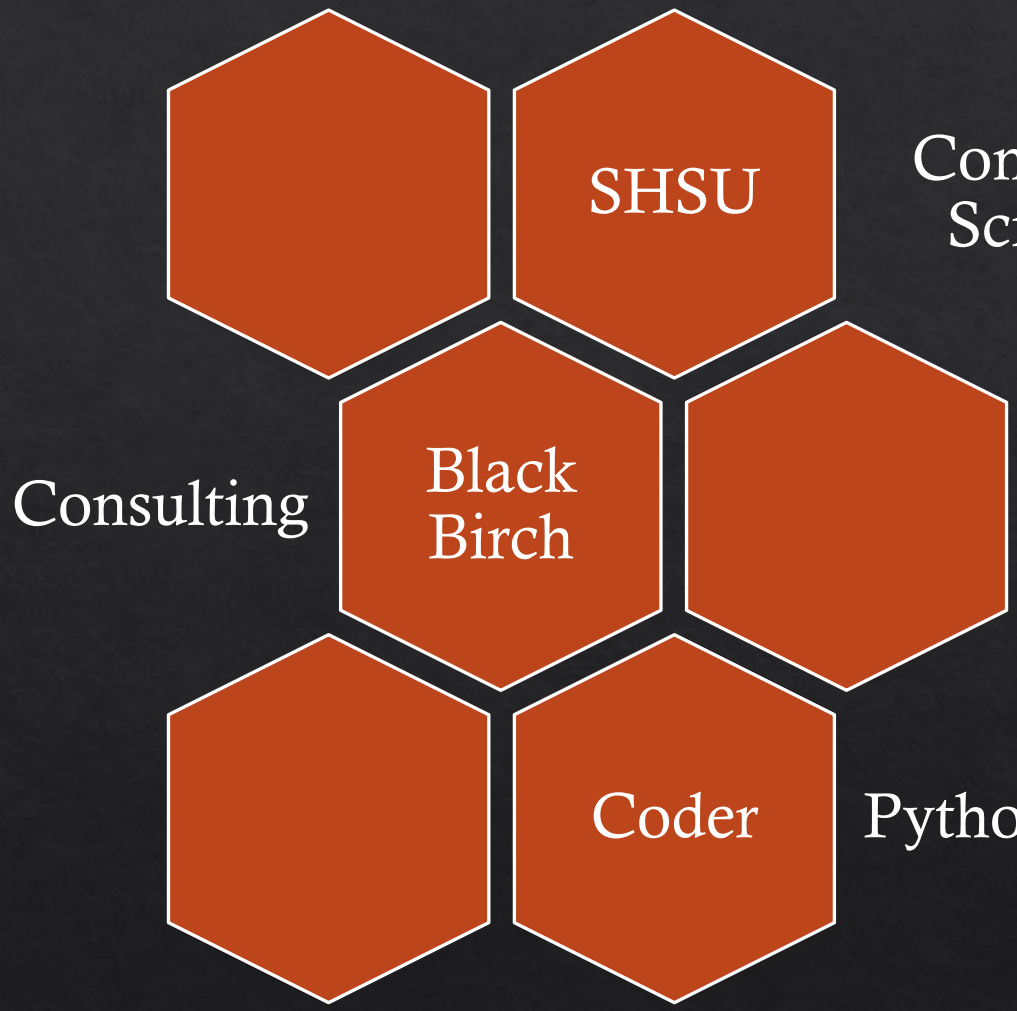

Malware Design
Abusing legacy Microsoft transports and session architecture

About Me

R.J. McDown

@BeetleChunks

A computer scientist who has made a career out of hacking into numerous fortune 500 companies through consulting red team engagements.



BLACKBIRCH

https://BlackBirchSecurity.com

Introduction

Red Team Challenges

- Compromise networks, endpoints, and data
- Remaining undetected
- Long term access

Solution Requirements

- Employ stealthy TTPs at multiple levels
- Provide enough functionality to be effective

Prerequisite High-level Overview

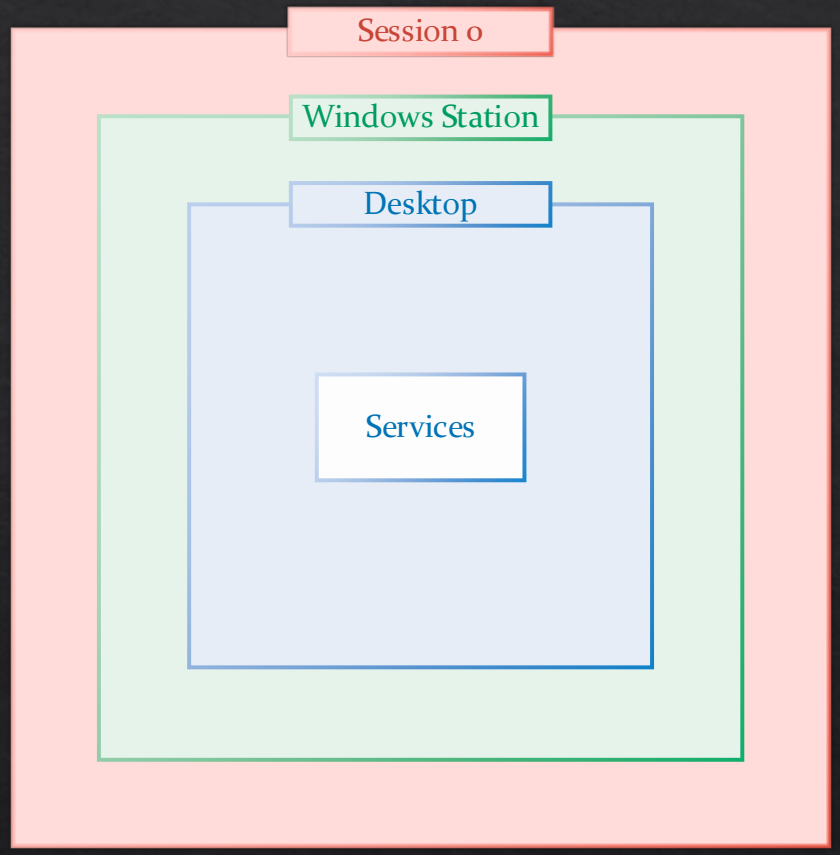
Windows Architecture

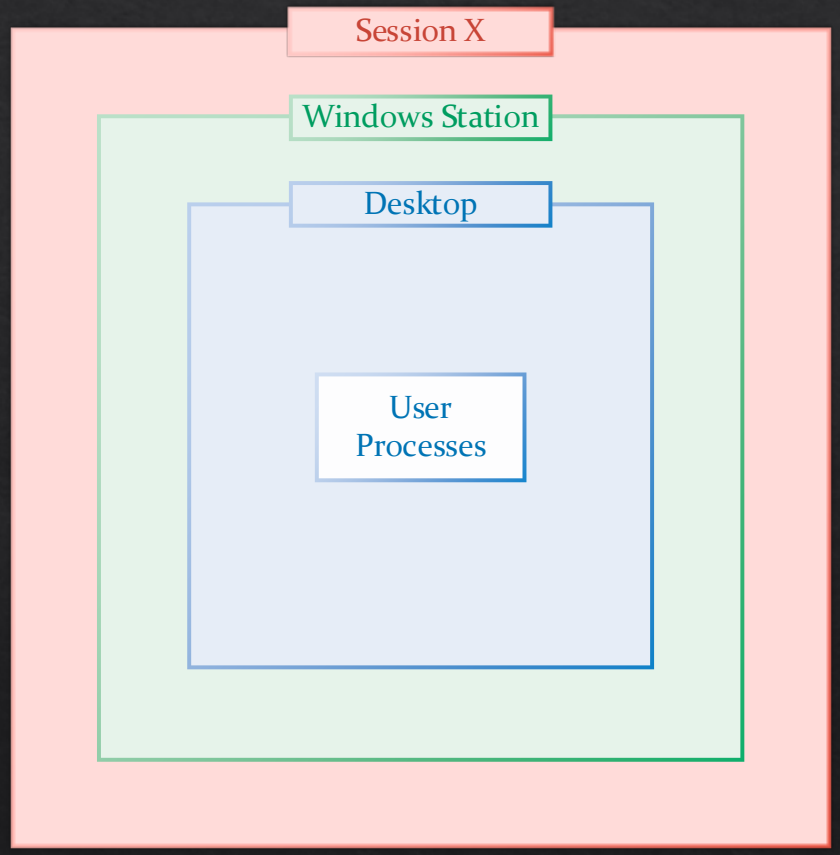
- Sessions
- Stations
- Desktops
- Processes
- Tokens

Protocols

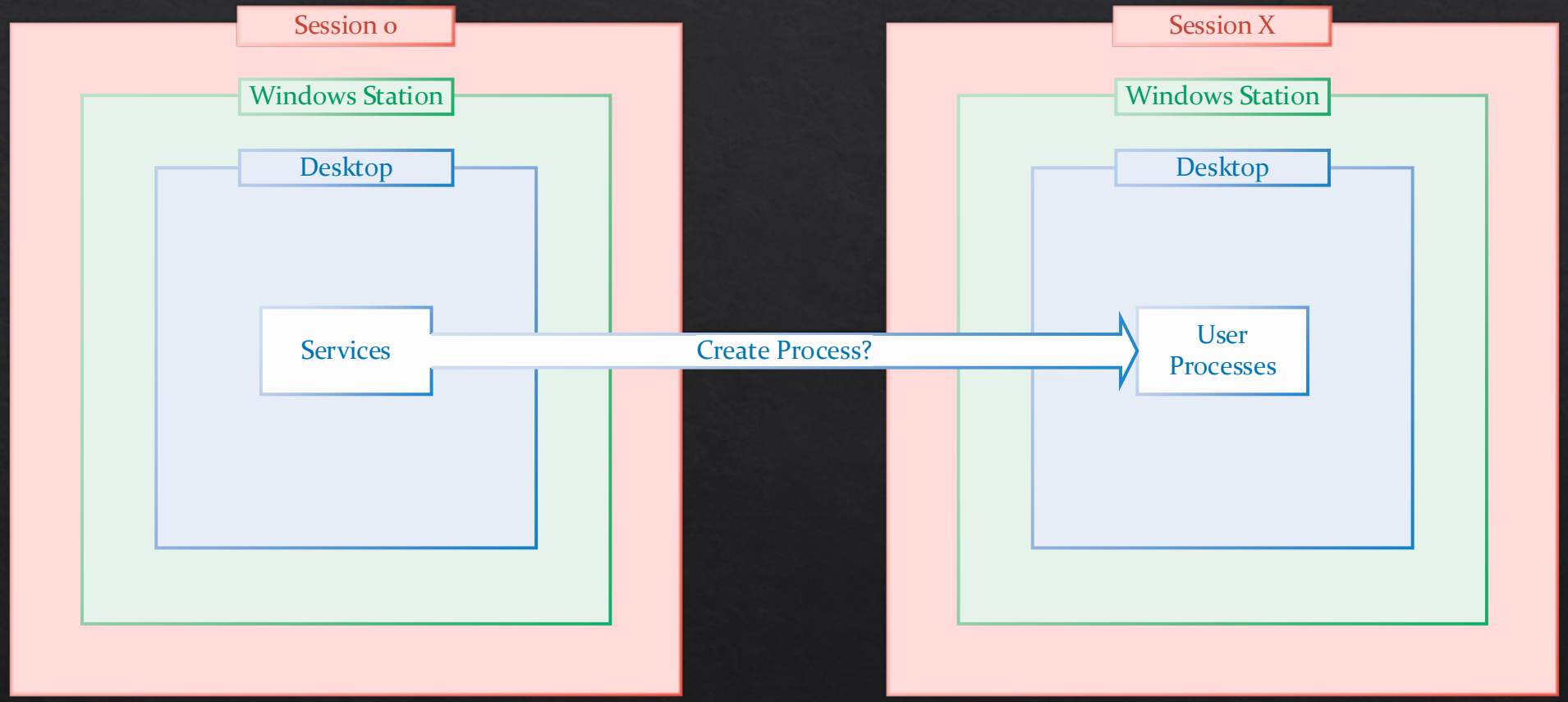
- Mailslots
- NetBIOS
- UDP

Windows Architecture



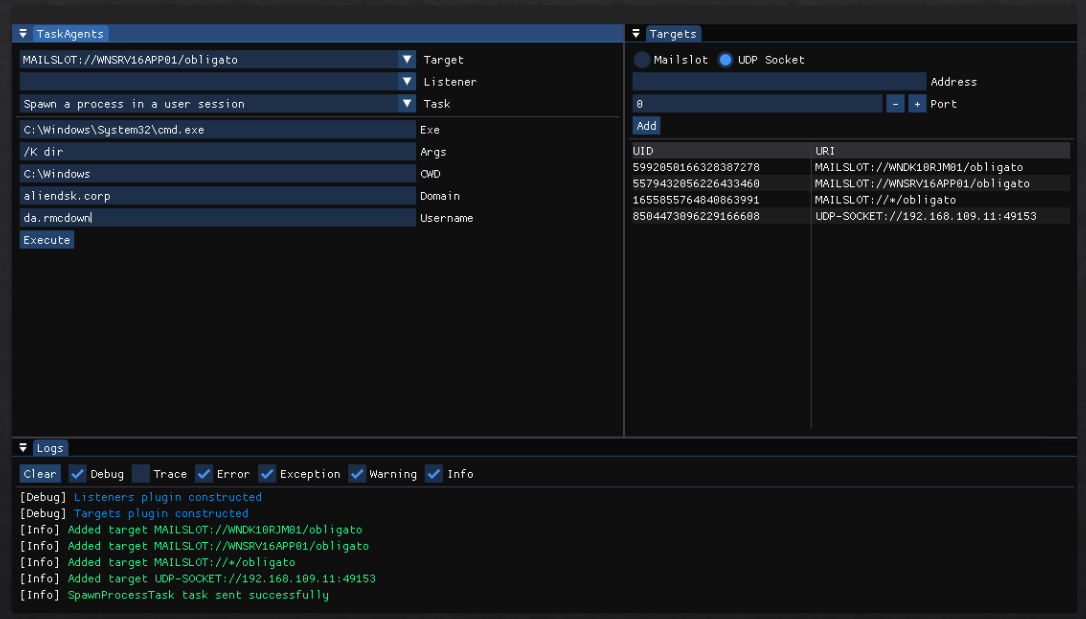


Windows Architecture



Obligato Overview

- Reliable long-term access without agent check-in requirements
- Recover access if primary interactive C2 is detected
- Custom messaging protocol
  - Operator issues Tasks
  - Messages delivered via Routes
  - Notifications (responses) are optional
- Agent runs as a Windows service in Session 0



Implant Messaging Protocol

- Obligato implements a custom messaging protocol that is transport agnostic, regardless of it being unidirectional or bidirectional
  - **Bidirectional** transports
    - Stateful protocols (i.e., TCP) or app layer protocols that use them, such as, HTTP(S)
  - **Unidirectional** transports
    - Stateless protocols (i.e., UDP) or app layer protocols that use them, such as, DNS and Mailslots over NetBIOS datagrams
- Operators leverage messages to task Obligato implants and to optionally receive responses, even over a unidirectional transport
- Accomplished through three key concepts: **Tasks**, **Routes**, and **Notifications**

Implant Messaging Protocol

Tasks

- Inbound messages requesting for some action to occur
- An operator could send a task message to an endpoint to spawn a process in a user's logon session, optionally yielding a response "notification“
- **Question**: since the messaging protocol is decoupled from the transport layer, how can it deliver a response, especially when considering a transport can be unidirectional?

Routes

- Entries within an Obligato implant describing destinations, which includes the transport type and any information needed for that type to function
- An example Mailslot route would include the NetBIOS name and Mailslot name while a UDP route would include a hostname/IP address and a port number
- Each route entry is mapped to a unique identifier that are used in route lookups

Implant Messaging Protocol

Notifications

- Response messages generated and sent by a completed task
- Only sent if the inbound task message included a valid and existing route ID

Takeaway

- Allows for tasks to be sent from an arbitrary location and transport while optionally receiving response notifications to a different location and transport

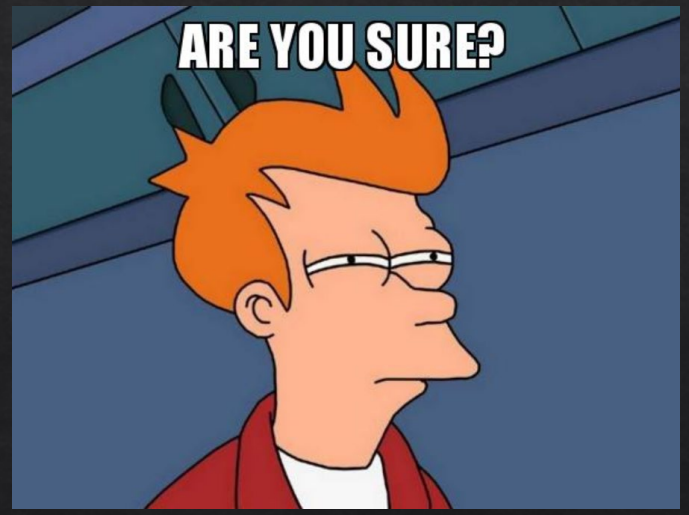
Mailslots as an Implant Messaging Transport

"The Remote Mailslot Protocol is a simple, unreliable, insecure, and unidirectional inter-process communications (IPC) protocol between a client and server … A mailslot is represented locally on the server as a file." [2]

- Why use Mailslots as an implant messaging transport?
  - network traffic not associated with the process
  - monitoring tools won't associate network activity with implant process
  - legacy APIs showing minimum support to Windows 2000 [10]
- Usage for successful Mailslot message delivery
  - Use the NetBIOS name, not the IP address
  - Broadcast message support
- Authentication is not required or supported

DEMO 1

> *"With the release of Windows 11 Insider Preview Build 25314, we have started disabling the Remote Mailslot protocol by default. This is a precursor to deprecation and eventual removal from Windows. You aren't using this extremely legacy protocol unless you're also using the deprecated and disabled-by-default SMBv1 protocol, so 99.97% of you have nothing to worry about."*[1]



- a principal program manager at Microsoft -

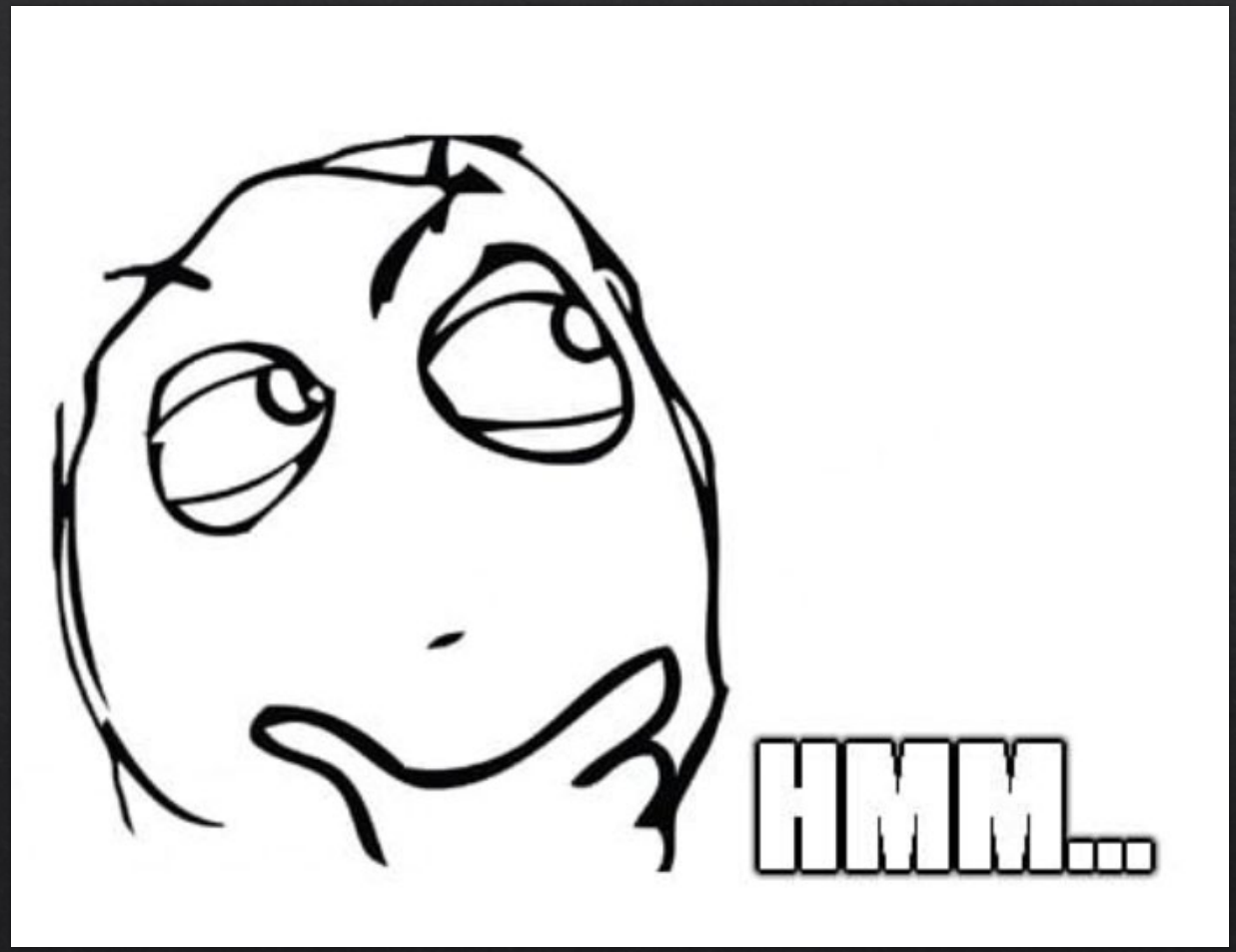
DEMO 2

Windows Logon Session Compromise

- Two key components of any engagement
  - Elevating permissions
  - Moving laterally to new endpoints within the enterprise
- Often accomplished by compromising privileged users and services
  - Credentials – LSASS process memory, LSA secrets, cached credentials, etc. (T1003) [4]
  - Logon Session – code execution in target user's Windows logon session (T1055) [4]
- Mainstream techniques for logon session compromise
  - Execute code in a logon sessions process (T1055) [4]
  - Open handle to process, duplicate token, then impersonation or process creation (T1134) [4]

Question:

What if we want to create a process, on demand, inside a target user's logon session without executing code in, or opening a handle to, another process?



Windows Task Scheduler

- Create a scheduled task to run as the target user
  - Configure to only run if user is logged-in or credentials will be required
- Process is created in user's session using WinSta0 and the default desktop
  - New windows could be displayed to the end user
- Process chain makes it obvious it stems from a scheduled task
  - taskhostw.exe and the malware have the same svchost.exe parent process
- Generates several event logs which may be scrutinized by defenders

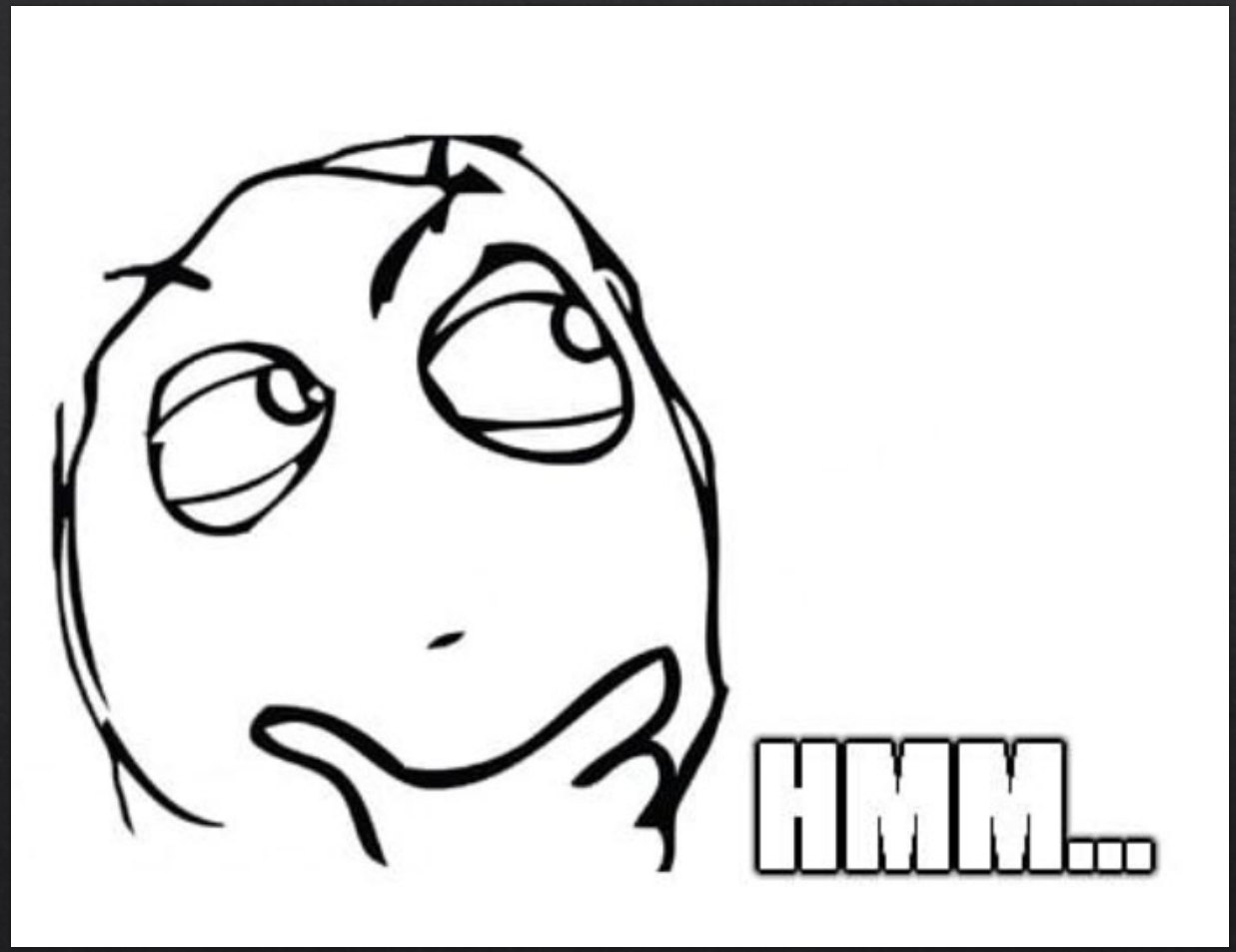
DEMO 3

Question:

What if we want to create a process, on demand, inside a target user's logon session without executing code in, or opening a handle to, another process

and…

not generate the obvious event logs, only display windows to the end user if we want, specify the parent creator process, and evade ETW?



DEMO 4

How does it work?

- Enable required privileges
  - SeDebugPrivilege will be required for spoofing the parent creator process
  - SeTcbPrivilege will be required for creating a process in a different logon session from the caller
- Spoof the Parent Process [5]
  - Open a handle to the desired parent process, not the process token, with the PROCESS_CREATE_PROCESS flag
  - Create a thread attributes list and add the parent process handle to it with UpdateProcThreadAttribute() along with the PROC_THREAD_ATTRIBUTE_PARENT_PROCESS flag

How does it work?

- CreateProcessAsUser() vs. CreateProcessWithToken()
  - *"The caller's process always runs in the caller's session, not in the session specified in the token. To run a process in the session specified in the token, use the CreateProcessAsUser function."* [12]
  - CreateProcessAsUser() is required to start a process in another user's session
- CreateProcessAsUser() requires a token to create the process, but we don't want to access/steal a token from another process, so we leverage WTSQueryUserToken()
  - gets the primary access token of the logged-on user for a specified session [13]
  - to use this function the caller must run within the context of NT Authority\SYSTEM and have "SeTcbPrivilege" enabled [13]
  - what I could not find to be documented, but is a requirement, is **the caller must be running in session 0**; performing this activity in any other session failed even if running as NT Authority\SYSTEM with all privileges enabled

Can this be detected?

- *"… is for kernel drivers that register for process creation notifications with PsSetCreateProcessNotifyRoutineEx() (or one of its variants)."* [5]

```
typedef struct _PS_CREATE_NOTIFY_INFO {
  SIZE_T               Size;
  union {
    ULONG Flags;
    struct {
      ULONG FileOpenNameAvailable : 1;
      ULONG IsSubsystemProcess : 1;
      ULONG Reserved : 30;
    };
  };
  HANDLE               ParentProcessId;
  CLIENT_ID            CreatingThreadId;
  struct _FILE_OBJECT *FileObject;
  PCUNICODE_STRING     ImageFileName;
  PCUNICODE_STRING     CommandLine;
  NTSTATUS             CreationStatus;
} PS_CREATE_NOTIFY_INFO, *PPS_CREATE_NOTIFY_INFO;
```

Can this be detected?

- *"… is for kernel drivers that register for process creation notifications with PsSetCreateProcessNotifyRoutineEx() (or one of its variants)."* [5]
- ParentProcessId member is the parent process ID set by CreateProcess()

```
typedef struct _PS_CREATE_NOTIFY_INFO {
  SIZE_T               Size;
  union {
    ULONG Flags;
    struct {
      ULONG FileOpenNameAvailable : 1;
      ULONG IsSubsystemProcess : 1;
      ULONG Reserved : 30;
    };
  };
  HANDLE               ParentProcessId;
  CLIENT_ID            CreatingThreadId;
  struct _FILE_OBJECT *FileObject;
  PCUNICODE_STRING     ImageFileName;
  PCUNICODE_STRING     CommandLine;
  NTSTATUS             CreationStatus;
} PS_CREATE_NOTIFY_INFO, *PPS_CREATE_NOTIFY_INFO;
```

Can this be detected?

- *"… is for kernel drivers that register for process creation notifications with PsSetCreateProcessNotifyRoutineEx() (or one of its variants)."* [5]
- ParentProcessId member is the parent process ID set by CreateProcess() [5]
- CreatingThreadId member is a struct containing the real creator parent process ID [5]
- *"… that information goes away, leaving only the parent ID, as it's the one stored in the EPROCESS kernel structure of the child process."* [5]

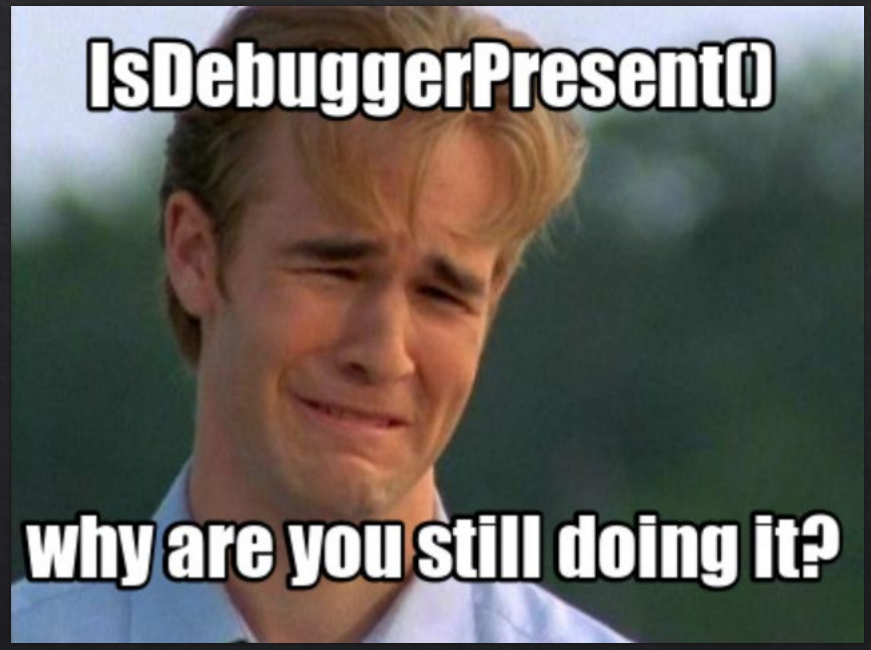
```
typedef struct _PS_CREATE_NOTIFY_INFO {
  SIZE_T               Size;
  union {
    ULONG Flags;
    struct {
      ULONG FileOpenNameAvailable : 1;
      ULONG IsSubsystemProcess : 1;
      ULONG Reserved : 30;
    };
  };
  HANDLE               ParentProcessId;
  CLIENT_ID            CreatingThreadId;
  struct _FILE_OBJECT *FileObject;
  PCUNICODE_STRING     ImageFileName;
  PCUNICODE_STRING     CommandLine;
  NTSTATUS             CreationStatus;
} PS_CREATE_NOTIFY_INFO, *PPS_CREATE_NOTIFY_INFO;
```

What about Event Tracing for Windows (ETW)?

DEMO 5

Early Execution Anti-debug & Runtime Awareness

- TLS Callbacks
- ThreadHideFromDebugger
- BeingDebugged
- Environment Variables



Thread Local Storage (TLS) Callbacks

- Allow a function to execute when a thread is created
  - Execution occurs before the main entry point
- Usage does **not** necessarily indicate malicious intent
  - Used legitimately in mainstream applications, including Chromium [7]

```
6   thread_local HANDLE hTlsCbThread;
7
8   DWORD WINAPI TlsCbThread(PVOID) {
9       // [3] Execution Trace - Race with wmain()
10      __debugbreak();
11
12      std::wstring msg = std::format(L"TLS Callback Thread - TID:{}", GetCurrentThreadId());
13      MessageBox(NULL, msg.c_str(), L"TLS CB Thread", MB_OK);
14
15      return 1;
16  }
17
18
19  typedef void(__stdcall* TLS_CALLBACK_PTR)(void* instance, int reason, void* reserved);
20
21  void __stdcall tls_callback(void* instance, int reason, void* reserved)
22  {
23      if (reason == DLL_PROCESS_ATTACH) {
24          // [1] Execution Trace
25          __debugbreak();
26
27          wchar_t msg[1024];
28          swprintf_s(msg, 1024, L"Entry TLS Callback - TID:%lu", GetCurrentThreadId());
29          MessageBox(NULL, msg, L"TLS CB", MB_OK);
30
31          hTlsCbThread = CreateThread(nullptr, 0, TlsCbThread, nullptr, 0, nullptr);
32          if (hTlsCbThread != NULL) {
33              WaitForSingleObject(hTlsCbThread, 5000);
34          }
35
36          // [2] Execution Trace
37          __debugbreak();
38      }
39  }
40
```

Anti-Debug

ThreadHideFromDebugger

- This can be set for a thread using NtSetInformationThread() and ThreadHideFromDebugger [8][9][17]
- Setting this on a thread causes it to be ignored by a debugger that attaches to the process
- The feature exists because a debugger needs to be able to ignore its own thread to function correctly
- Guess what happens when a breakpoint is set in a thread that can't be debugged... yes, it crashes the process

BeingDebugged

- We have our threads set to not allow debugging, but how do we know if one attaches so we can set a breakpoint and crash the process?
- Everyone knows about the Windows API function IsDebuggerPresent(), so maybe we don't want to use it
- We just need the address of the process environment block (PEB) then access the value at the offset of 'BeingDebugged' [14]

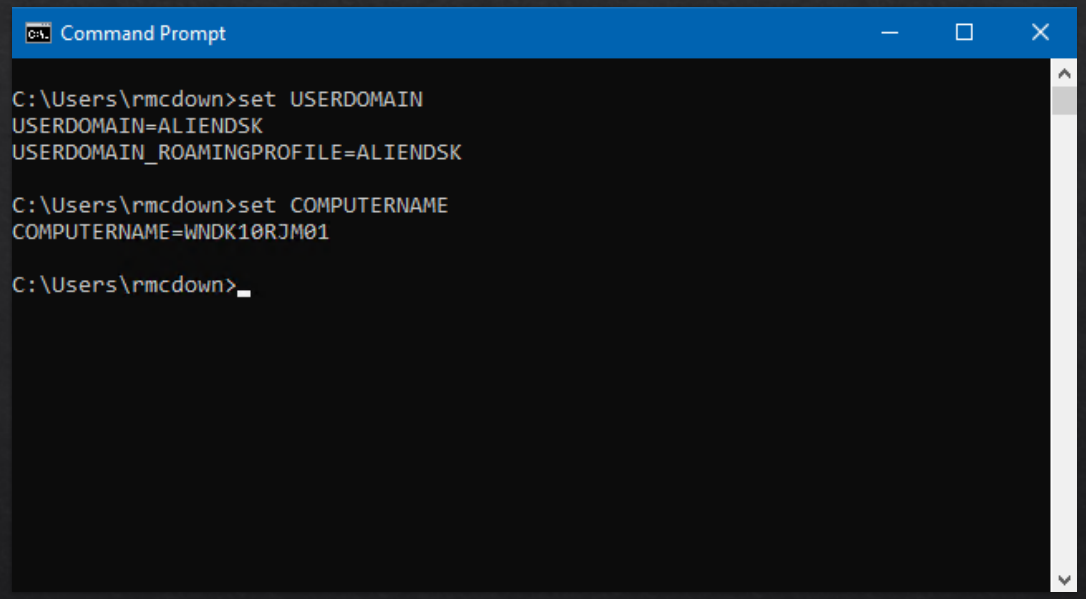
Anti-Debug

```cpp
/*
    Additional includes - Native API header files
    https://github.com/winsiderss/phnt
*/
#include "phnt_windows.h"
#include "phnt.h"

#include <iostream>
#include <Windows.h>


thread_local HANDLE hAntiDbgThread;

DWORD WINAPI AntiDbgThread(PVOID) {
    // Address of the Process Environment Block (PEB)
    uintptr_t pebAddr = __readgsqword(0x60);

    // Loop while value of 'BeingDebugged' in PEB struct
    // is 0 (not being debugged)
    while ((char)*(uintptr_t*)(pebAddr + 2) == 0) {
        pebAddr = __readgsqword(0x60);

        if (pebAddr == 0) {
            __debugbreak(); // Crash process
        }
    }

    __debugbreak(); // Crash process
    return 0;
}
```

```cpp
typedef void(__stdcall* TLS_CALLBACK_PTR)(void* instance, int reason, void* reserved);

void __stdcall tls_callback(void* instance, int reason, void* reserved) {
    if (reason == DLL_PROCESS_ATTACH) {
        unsigned char info = 5;
        hAntiDbgThread = CreateThread(nullptr, 0, AntiDbgThread, nullptr, 0, nullptr);

        if (hAntiDbgThread != NULL) {
            NTSTATUS ntSetStatus = NtSetInformationThread(
                hAntiDbgThread, THREADINFOCLASS::ThreadHideFromDebugger, &info, 0);
        }
    }
}

#pragma comment (linker, "/INCLUDE:_tls_used")
#pragma comment(linker, "/INCLUDE:p_tls_callback")
#pragma const_seg(".CRT$XLB")
EXTERN_C const TLS_CALLBACK_PTR p_tls_callback = tls_callback;
#pragma const_seg()


int wmain(int argc, wchar_t* argv[]) {
    int x, sum = 0;

    while (1) {
        std::wcout << L"Type a number: ";
        std::wcin >> x;
        sum += x;
        std::wcout << L"Sum: " << sum << std::endl;
    }

    return 0;
}
```

Environment Variables

- Detect if machine is domain joined by comparing 'USERDOMAIN' and 'COMPUTERNAME'; if they are the same the box is not domain joined
- Can help mitigate dynamic analysis as most sandboxes that I have observed do not run domain joined machines
- Have used this technique for many years successfully in many different custom payload stagers and tools



Conclusion

- Implant Messaging Protocol supports arbitrary unidirectional or bidirectional transports
  - Task response notifications are optional and can route across different transports and endpoints
  - Can extend to virtually any transport protocol
- Mailslots are available even if SMBv1 is disabled
  - Network traffic is disassociated from the server/client processes
  - Supports broadcasting a single task to all implanted endpoints in the domain network
  - Authentication not only isn't required, but not even supported

Conclusion

- User logon sessions can be compromised without touching their processes
  - Remote Desktop Services API allows primary token access given a Session ID
  - Needs special process privileges and running in Session 0
- Method for spoofing parent process is not captured by Sysinternals Process Monitor or Sysinternals Process Explorer
  - Activity can be captured by ETW, demonstrated by Process Monitor X v2
  - Process/thread callbacks capture activity, but it isn't stored in the child process kernel structure
- **Does your stack capture this activity correctly?**
- TLS Callbacks allow for early runtime code execution, providing a great place to incorporate runtime checks and anti-debugging

Questions?



BLACKBIRCH

https://BlackBirchSecurity.com

References

1. Pyle, Ned. “The Beginning of the End of Remote Mailslots.” Tech Community, Microsoft, 8 Mar. 2023, https://techcommunity.microsoft.com/t5/storage-at-microsoft/the-beginning-of-the-end-of-remote-mailslots/ba-p/3762048.
2. Corporation, Microsoft. “[MS-Mail]: Remote Mailslot Protocol.” [MS-MAIL], Microsoft, 25 June 2021, https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-MAIL/[MS-MAIL].pdf.
3. Aggarwal, Avnish. “PROTOCOL STANDARD FOR A NetBIOS SERVICE.” IETF, RFC Editor, Mar. 1987, https://datatracker.ietf.org/doc/html/rfc1001.
4. ATT&CK, MITRE. “Enterprise Techniques.” Techniques - Enterprise | MITRE ATT&CK, MITRE ATTCK, 25 Oct. 2022, https://attack.mitre.org/techniques/enterprise/.
5. Yosifovich, Pavel. “Parent Process vs. Creator Process.” Pavel Yosifovich, 10 Jan. 2021, https://scorpiosoftware.net/2021/01/10/parent-process-vs-creator-process/.
6. Schwarz, Roland. “Thread Local Storage - the C++ WAY.” CodeProject, CodeProject, 28 Aug. 2004, https://www.codeproject.com/Articles/8113/Thread-Local-Storage-The-C-Way.
7. The Chromium Authors. “Chromium/thread_local_storage_win.Cc at Main · Chromium/Chromium.” GitHub, The Chromium Project, Jan. 2012, https://github.com/chromium/chromium/blob/main/base/threading/thread_local_storage_win.cc.
8. timb3r. “How to Find Hidden Threads - Threadhidefromdebugger - Antidebug Trick.” How to Find Hidden Threads - ThreadHideFromDebugger - AntiDebug Trick, Guided Hacking, 27 Dec. 2019, https://guidedhacking.com/threads/how-to-find-hidden-threads-threadhidefromdebugger-antidebug-trick.14281/.

References

9. Chappell, Geoff. “THREADINFOCLASS.” Threadinfoclass, Jan. 1997, https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/ps/psquery/class.htm.
10. GrantMeStrength. “GetMailslotInfo Function (Winbase.h) - win32 Apps.” Win32 Apps | Microsoft Learn, 10 Oct. 2021, https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-getmailslotinfo.
11. Alvinashcraft. “Impersonation Tokens - win32 Apps.” Win32 Apps | Microsoft Learn, 1 July 2021, https://learn.microsoft.com/en-us/windows/win32/secauthz/impersonation-tokens.
12. GrantMeStrength. “CreateProcessWithTokenW Function (Winbase.h) - win32 Apps.” Win32 Apps | Microsoft Learn, 2 Jan. 2023, https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createprocesswithtokenw.
13. QuinnRadich. “WTSQUERYUSERTOKEN Function (WTSAPI32.H) - win32 Apps.” Win32 Apps | Microsoft Learn, 10 Dec. 2021, https://learn.microsoft.com/en-us/windows/win32/api/wtsapi32/nf-wtsapi32-wtsqueryusertoken.
14. Karl-Bridge-Microsoft. “PEB (Winternl.h) - win32 Apps.” PEB (Winternl.h) - Win32 Apps | Microsoft Learn, 31 Aug. 2022, https://learn.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb.
15. Yosifovich, Pavel. Windows 10 System Programming Part 1. Independently Published.
16. Yosifovich, Pavel. Windows 10 System Programming Part 2. Independently Published.
17. dmex, Winsiderss. “Native API Header Files for the System Informer Project.” GitHub, Winsiderss, Oct. 2018, https://github.com/winsiderss/phnt.