

The Pool Party You  
Will Never Forget:  
**New Process Injection  
Techniques Using  
Windows Thread Pools**



# Alon Leviev

---

Security Researcher at SafeBreach

---

21 years old

---

Self-taught

---

OS internals, reverse engineering and vulnerability research

---

Former BJJ world and european champion

---

 SafeBreach



# Agenda

---

Process Injection Background

---

Research Motivation & Questions

---

Detection Approach

---

Research Goals

---

User-mode Thread Pool Deep Dive

---

Introducing PoolParty

---

Process Injection Implications

---

Takeaways

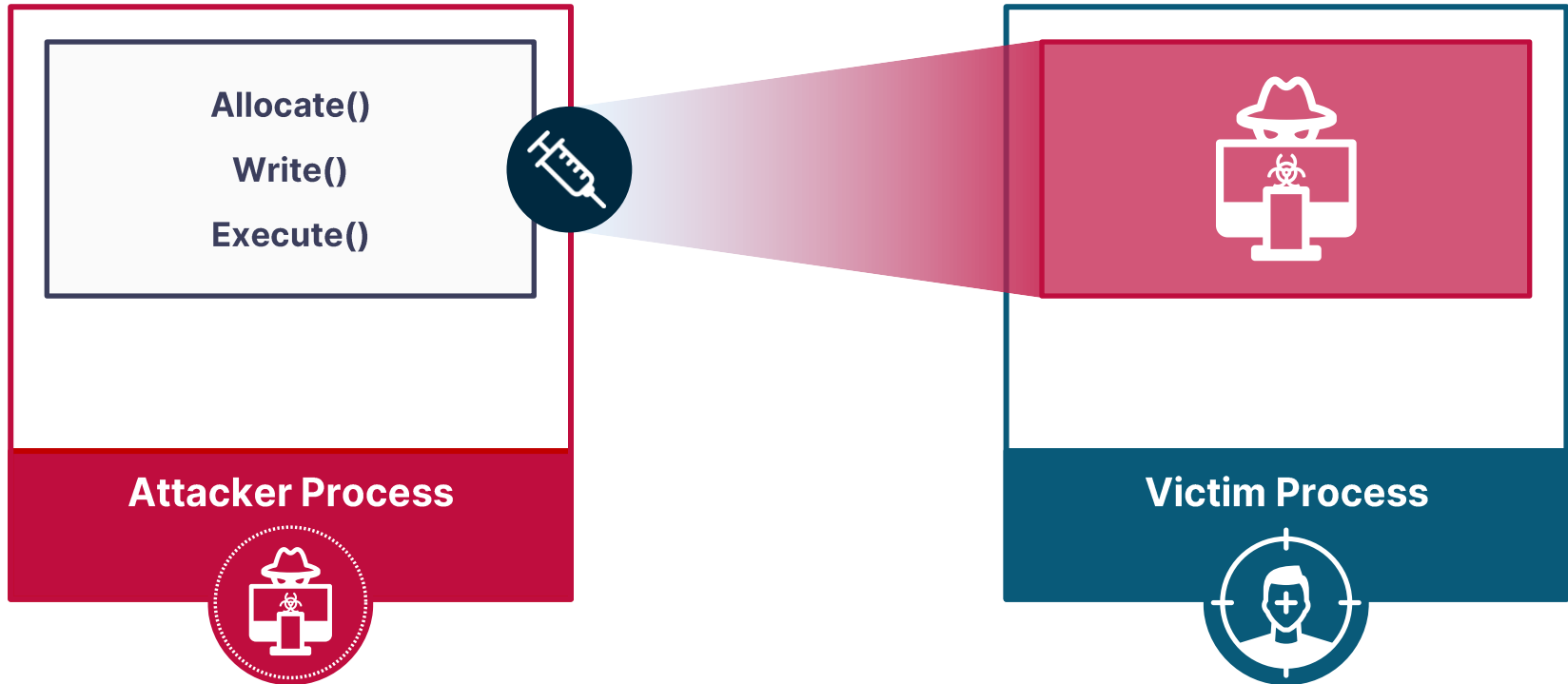
---



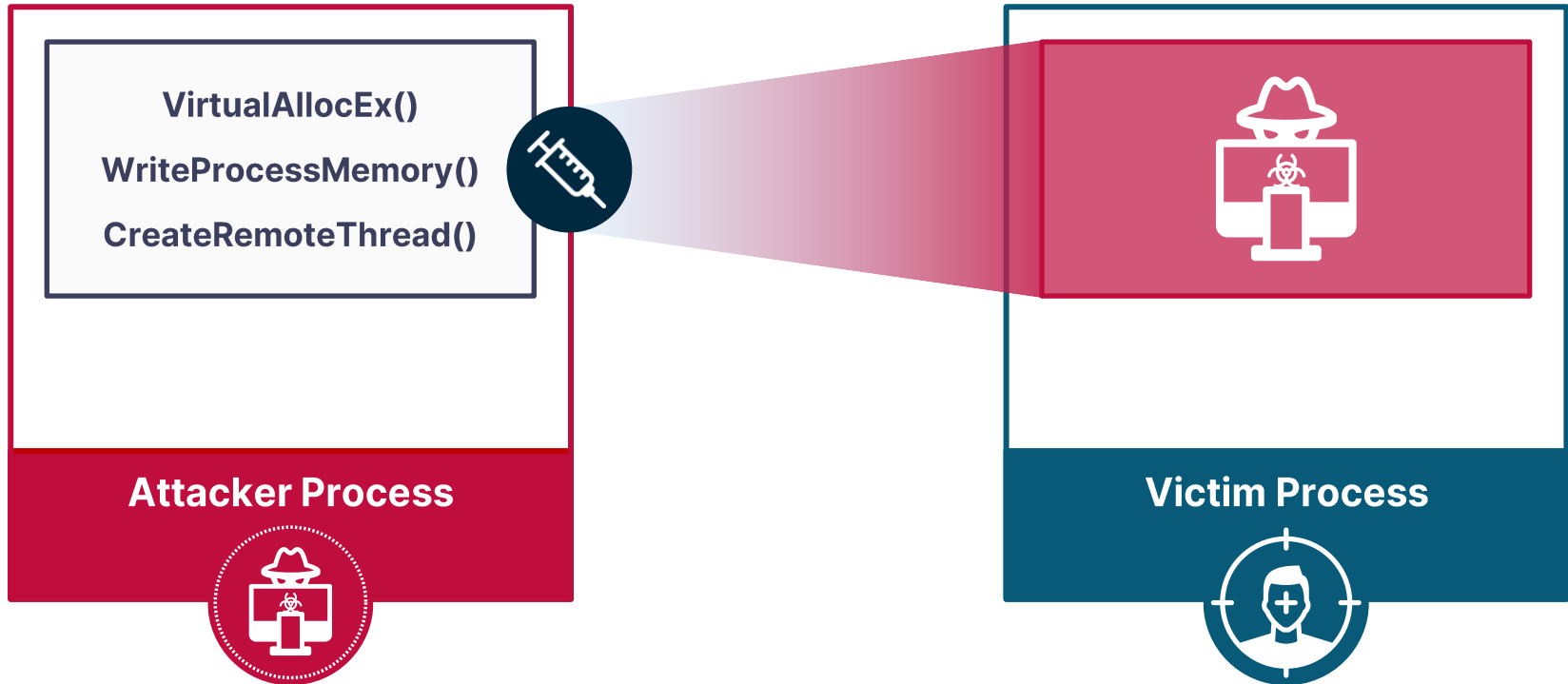
# Process Injection Background



# Process Injection Background



# Process Injection Background



# Motivation



# Motivation

---

Process injection techniques abuses legitimate features of the OS

---

Can an EDR effectively distinguish a legitimate versus a malicious use of a feature?

---

Is the current detection approach generic enough?

---

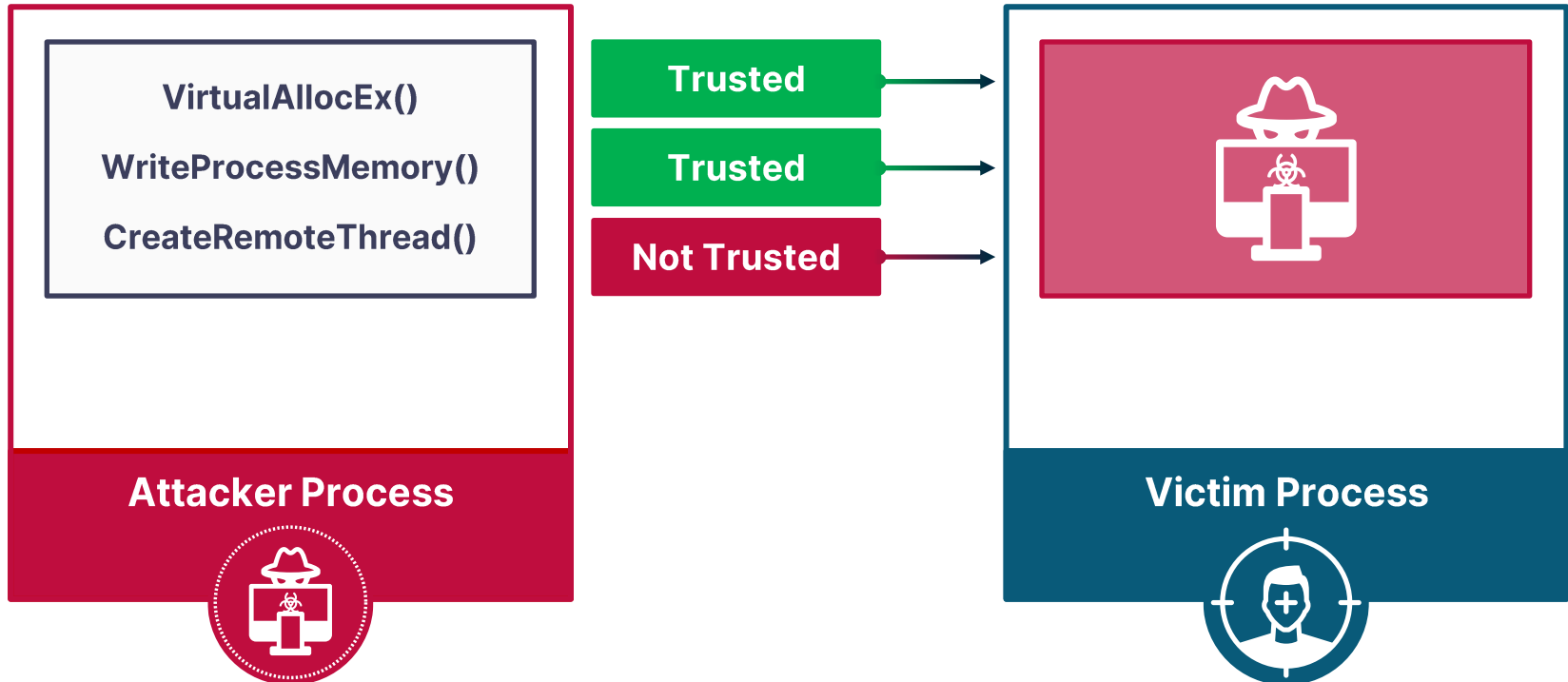




# Detection Approach



# Detection Approach – Spotting Detection Focus



# Detection Approach – CreateRemoteThread Injection



NtCreateThreadEx(**Remote Process**)



NtCreateThreadEx(**Current Process**)

# Detection Approach – APC Injection



NtQueueApcThread(**Remote Thread**)



NtQueueApcThread(**Local Thread**)

# Detection Approach – Summary

---

Allocate and write primitives are not detected

---

Detection is based on execution primitives

---

Execution primitives gets flag by inspection of initiator and creator

---



# Research Goals



# Research Goals

---

Fully undetectable process  
injection techniques

- Applicable against all Windows processes
- 



# What Ifs

---

What if the execute primitive is built with write and allocate primitives?

---

What if the execution is triggered by a legitimate action?

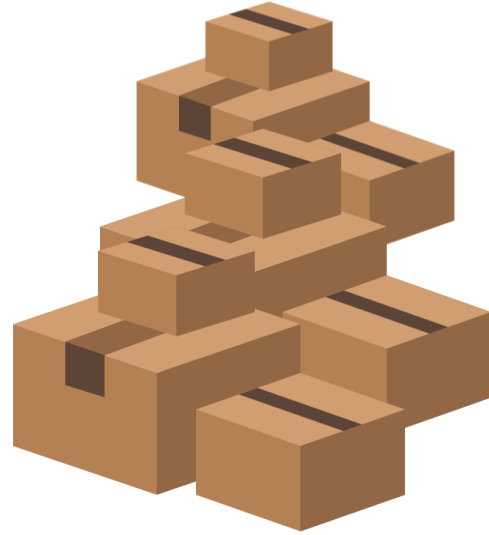
---





# What Is a Thread Pool?

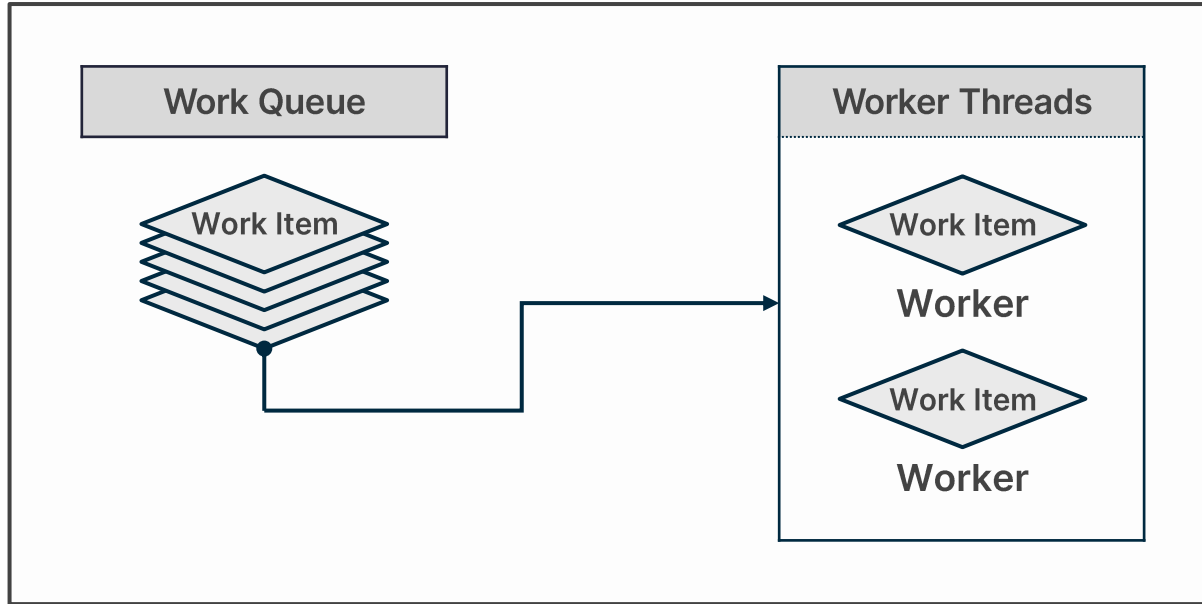
I wish these boxes could be sent in parallel



# What Is a Thread Pool?



# How a Thread Pool Works?



# Why Thread Pool?

---

All processes have a thread pool by default

---

Work items and thread pools are represented by structures

---

Multiple work item types are supported

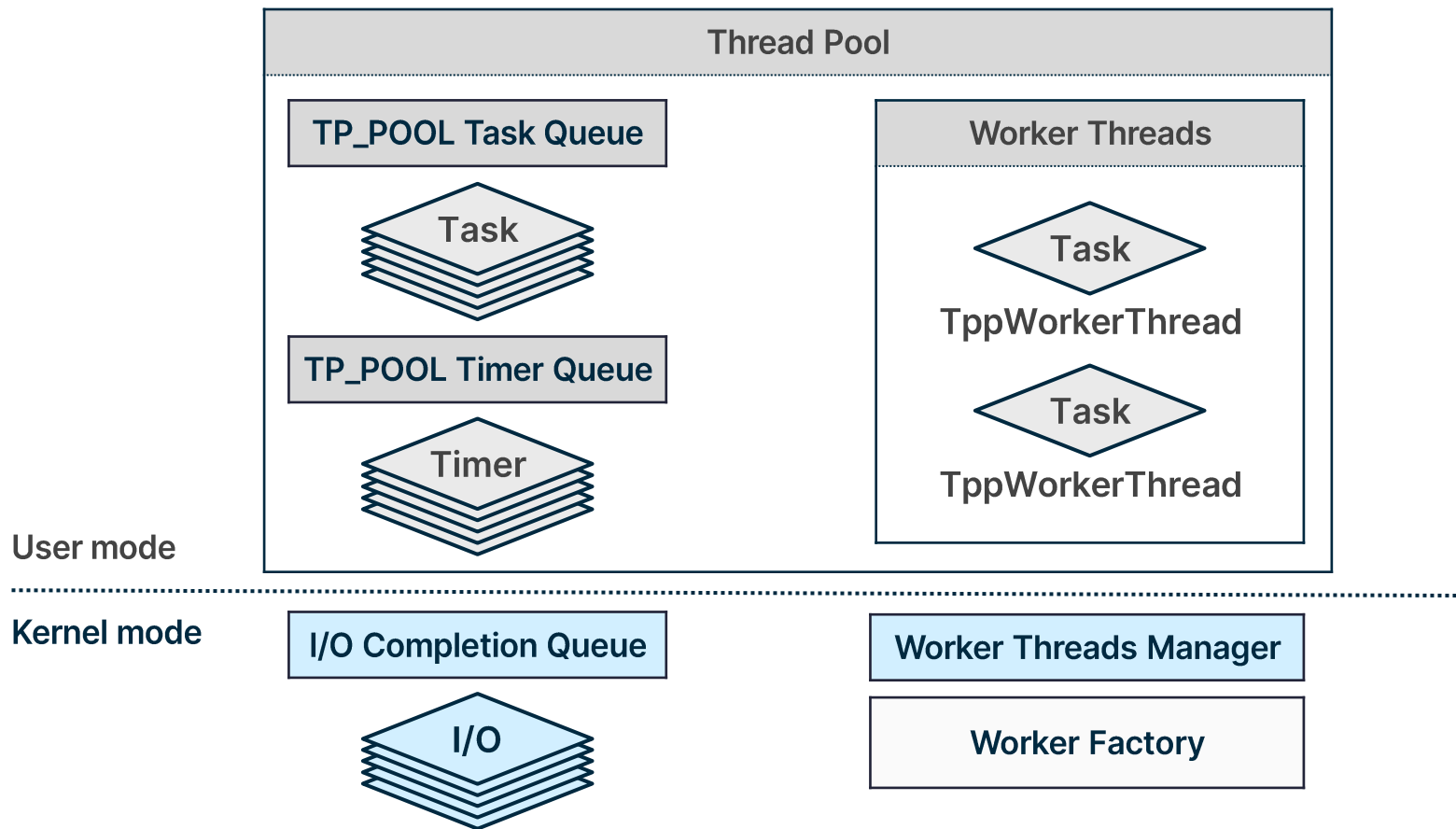
---



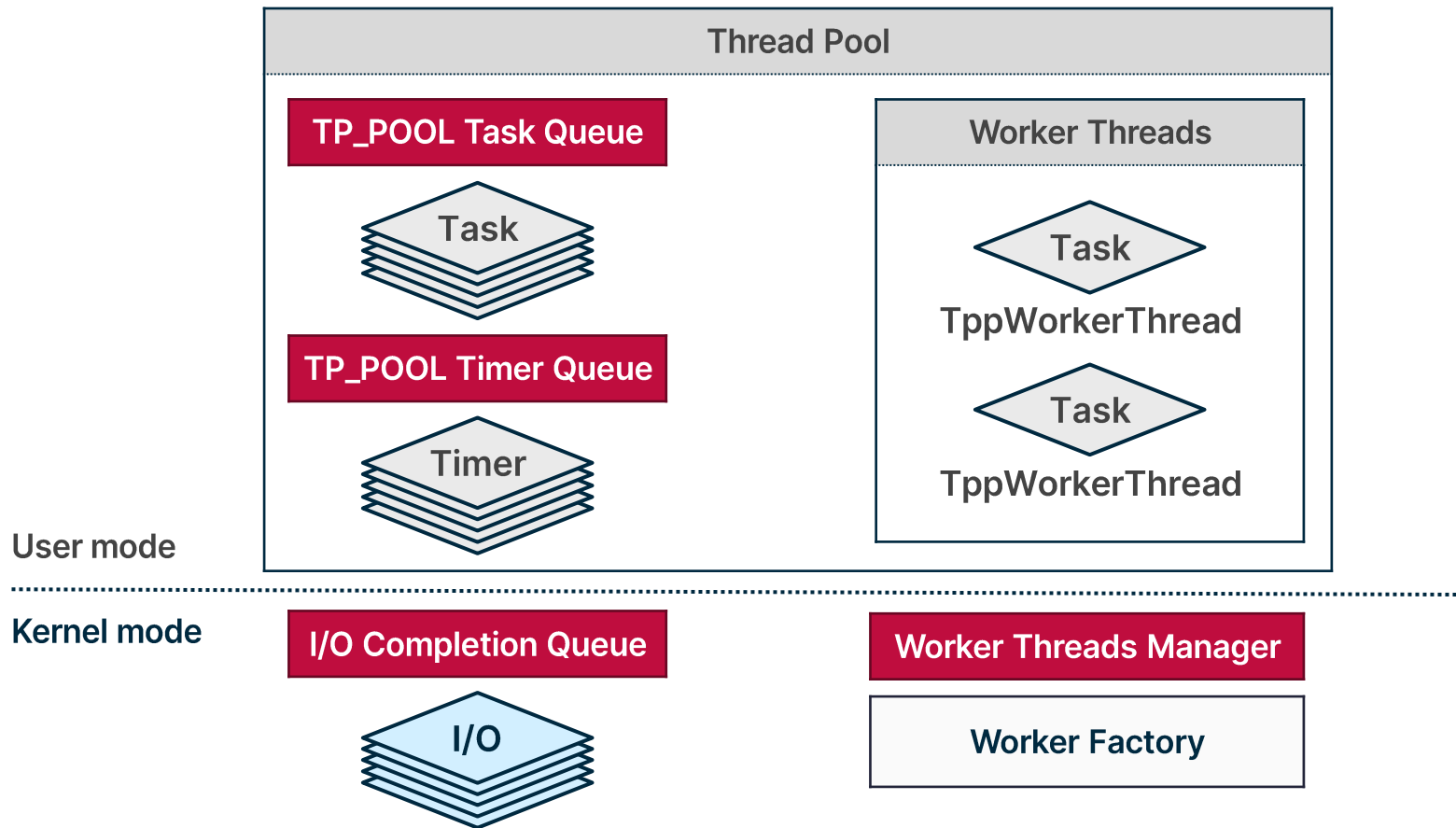
# User-Mode Thread Pool Deep Dive



# User-Mode Thread Pool Architecture



# Defining Attack Surface



PoolParty State

No friends in the pool

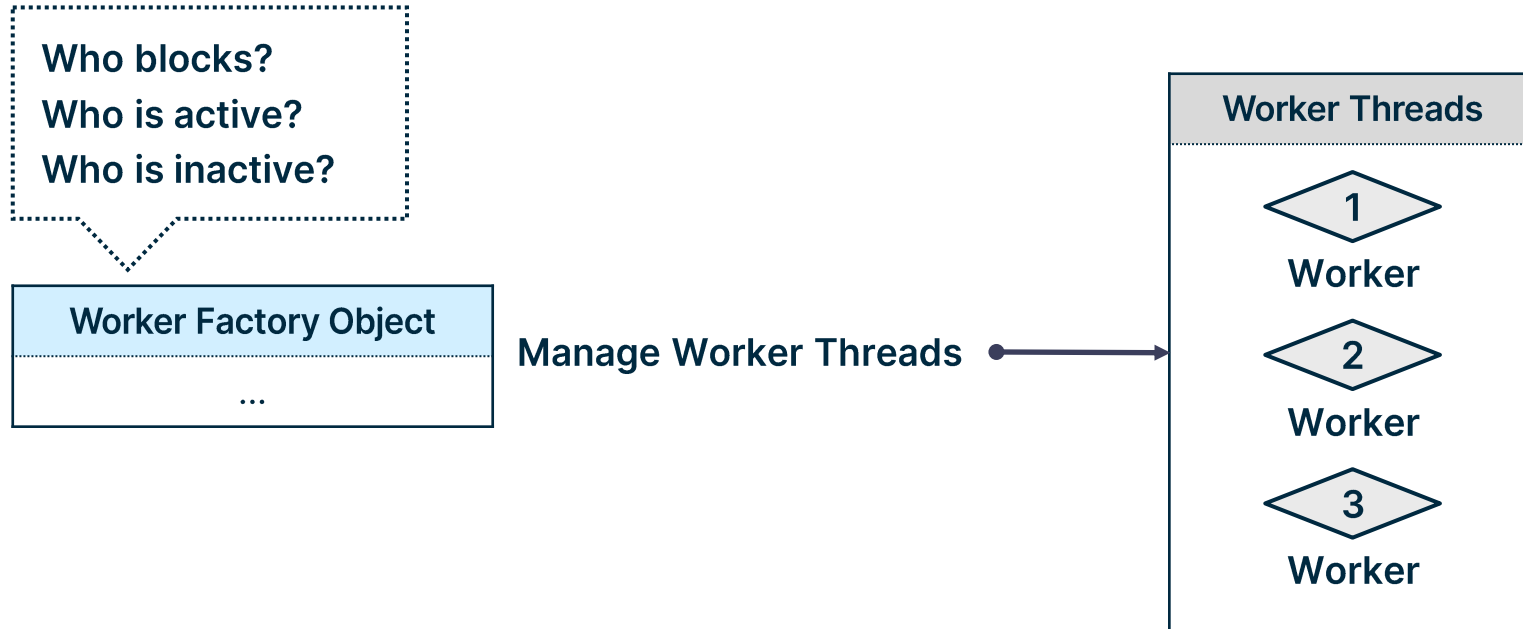




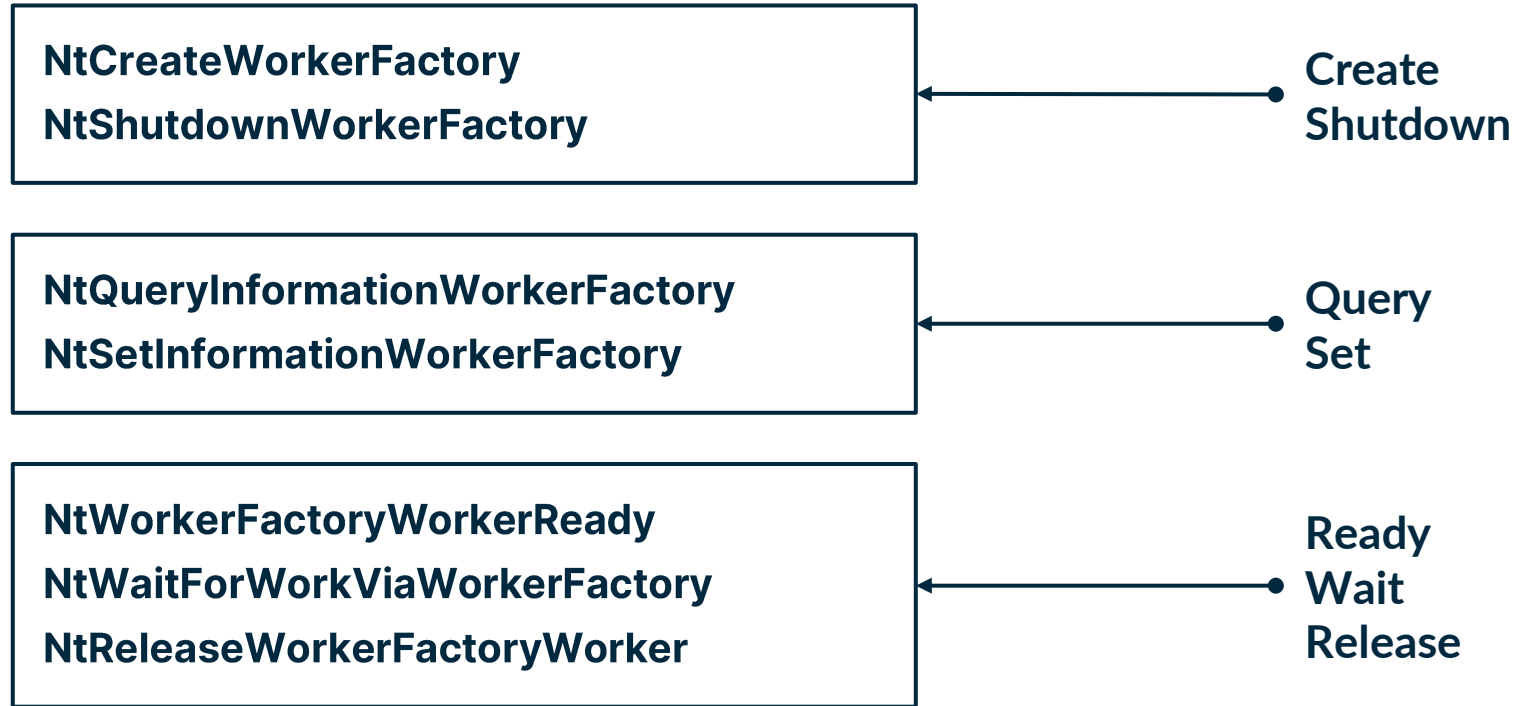
# Attacking Worker Factories



# Worker Factories Introduction



# Worker Factories System Calls



# Attacking Worker Factories

```
NTSTATUS NTAPI NtCreateWorkerFactory(
    _Out_ PHANDLE WorkerFactoryHandleReturn,
    _In_ ACCESS_MASK DesiredAccess,
    _In_opt_ POBJECT_ATTRIBUTES ObjectAttributes,
    _In_ HANDLE CompletionPortHandle,
    _In_ HANDLE WorkerProcessHandle,
    _In_ PVOID StartRoutine,
    _In_opt_ PVOID StartParameter,
    _In_opt_ ULONG MaxThreadCount,
    _In_opt_ SIZE_T StackReserve,
    _In_opt_ SIZE_T StackCommit
);
```

# Attacking Worker Factories

```
C:\Users\User\Desktop\PoolParty>CreateWorkerFactoryByProcessName.exe explorer.exe
[+] target Process ID: 4656
[+] Retrieved handle to the target process: 0xd0
[+] Allocated shellcode memory in the target process: 0000000003010000
[+] Written shellcode to the target process
[+] Created Worker Factory I/O completion port: 0xc4
[-] NtCreateWorkerFactory failed: The parameter is incorrect.
```

# Attacking Worker Factories

## Ntoskrnl:: NtCreateWorkerFactory

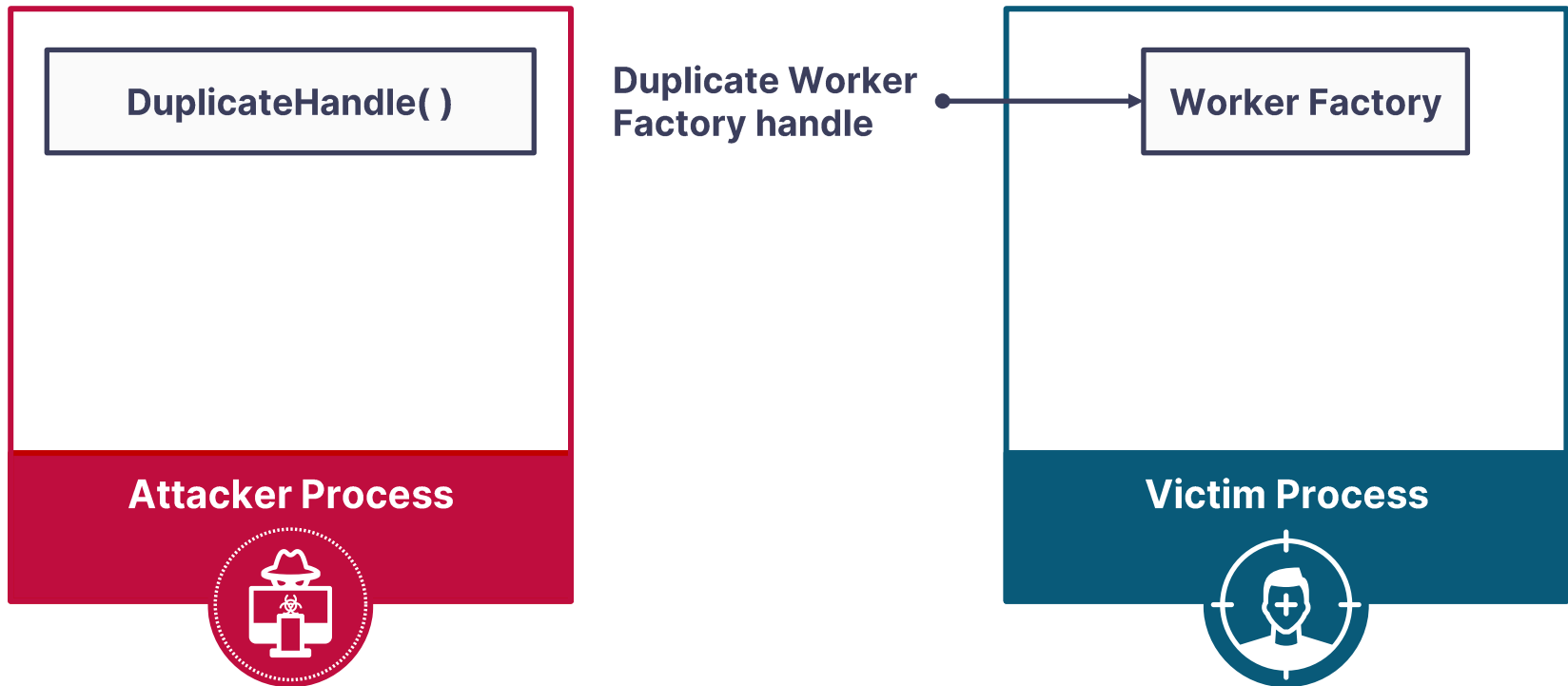
```
NTSTATUS NTAPI NtCreateWorkerFactory(..., HANDLE WorkerProcessHandle, ...)
{
    [snip]

    KPROCESS * pWorkerProcessObject;
    ObpReferenceObjectByHandleWithTag(WorkerProcessHandle, ..., &pWorkerProcessObject);

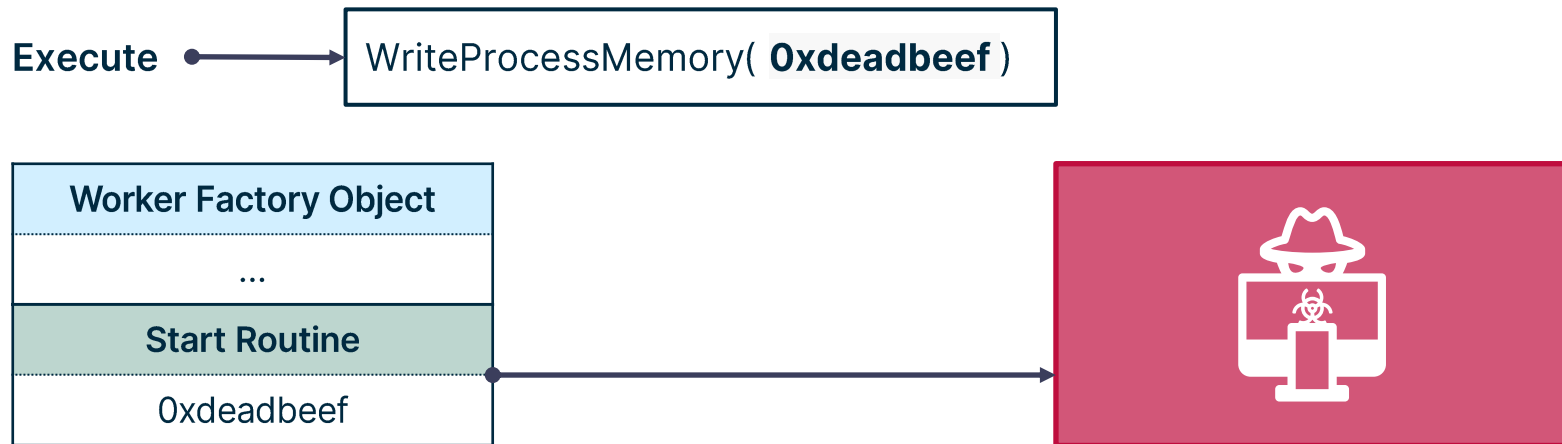
    if ( KeGetCurrentThread()->ApcState.Process != pWorkerProcessObject)
    {
        return STATUS_INVALID_PARAMETER;
    }

    [snip]
}
```

# Attacking Worker Factories



# Attacking Worker Factories





# Attacking Worker Factories

```
NTSTATUS NTAPI NtQueryInformationWorkerFactory(  
    _In_ HANDLE WorkerFactoryHandle,  
    _In_ QUERY_WORKERFACTORYINFOCLASS WorkerFactoryInformationClass,  
    _In_reads_bytes_(WorkerFactoryInformationLength) PVOID WorkerFactoryInformation,  
    _In_ ULONG WorkerFactoryInformationLength,  
    _Out_opt_ PULONG ReturnLength  
);
```

# Attacking Worker Factories

```
typedef enum _QUERY_WORKERFACTORYINFOCLASS
{
    WorkerFactoryBasicInformation = 7,
} QUERY_WORKERFACTORYINFOCLASS, * PQUERY_WORKERFACTORYINFOCLASS;
```

# Attacking Worker Factories

```
typedef struct _WORKER_FACTORY_BASIC_INFORMATION
{
    [snip]
    PVOID StartRoutine;
    [snip]
} WORKER_FACTORY_BASIC_INFORMATION, * PWORKER_FACTORY_BASIC_INFORMATION;
```

# Attacking Worker Factories

```
NTSTATUS NTAPI NtSetInformationWorkerFactory(  
    _In_ HANDLE WorkerFactoryHandle,  
    _In_ SET_WORKERFACTORYINFOCLASS WorkerFactoryInformationClass,  
    _In_reads_bytes_(WorkerFactoryInformationLength) PVOID WorkerFactoryInformation,  
    _In_ ULONG WorkerFactoryInformationLength,  
);
```

# Attacking Worker Factories

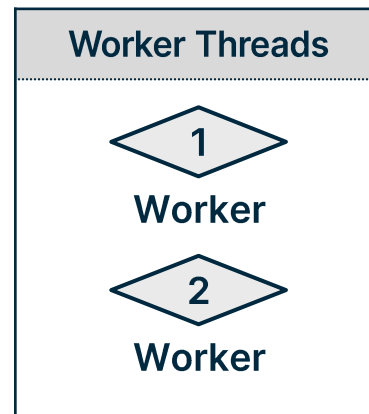
```
typedef enum _SET_WORKERFACTORYINFOCLASS
{
    WorkerFactoryTimeout = 0,
    WorkerFactoryRetryTimeout = 1,
    WorkerFactoryIdleTimeout = 2,
    WorkerFactoryBindingCount = 3,
    WorkerFactoryThreadMinimum = 4,
    WorkerFactoryThreadMaximum = 5,
    WorkerFactoryPaused = 6,
    WorkerFactoryAdjustThreadGoal = 8,
    WorkerFactoryCallbackType = 9,
    WorkerFactoryStackInformation = 10,
    WorkerFactoryThreadBasePriority = 11,
    WorkerFactoryTimeoutWaiters = 12,
    WorkerFactoryFlags = 13,
    WorkerFactoryThreadSoftMaximum = 14
} SET_WORKERFACTORYINFOCLASS, * PSET_WORKERFACTORYINFOCLASS;
```

# Attacking Worker Factories

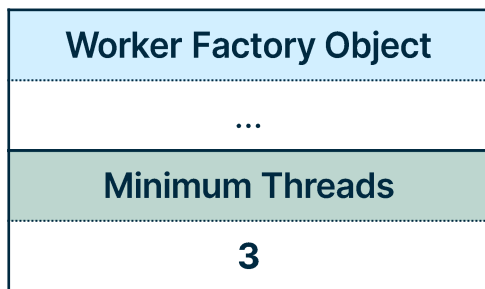
Execute

→ NtSetInformationWorkerFactory(Running Threads Num + 1)

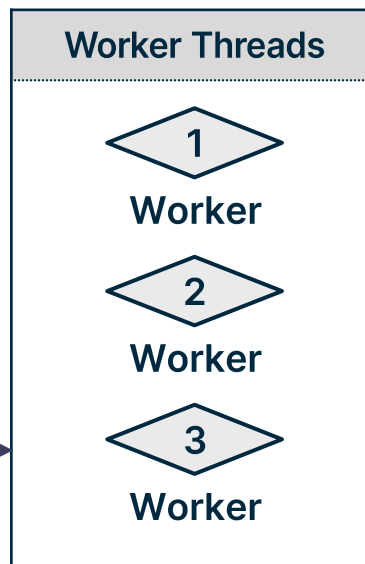
| Worker Factory Object |
|-----------------------|
| ...                   |
| Minimum Threads       |
| 2                     |



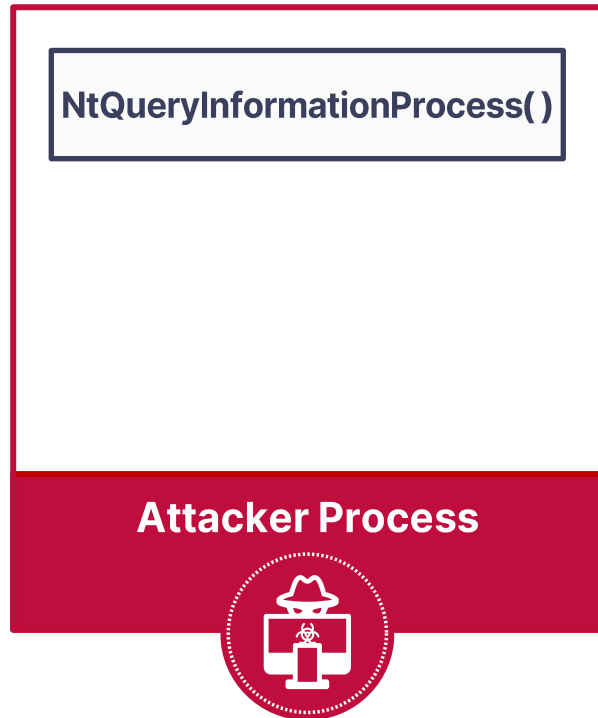
# Attacking Worker Factories



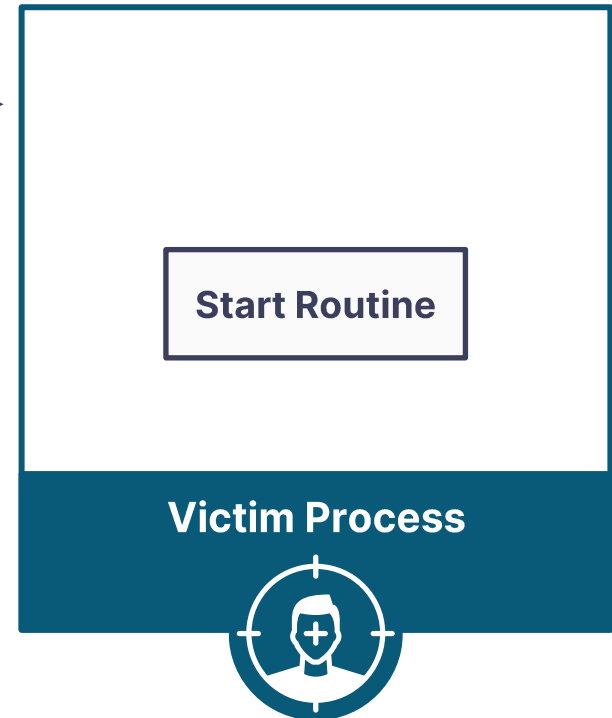
Create new worker thread →



# Attacking Worker Factories

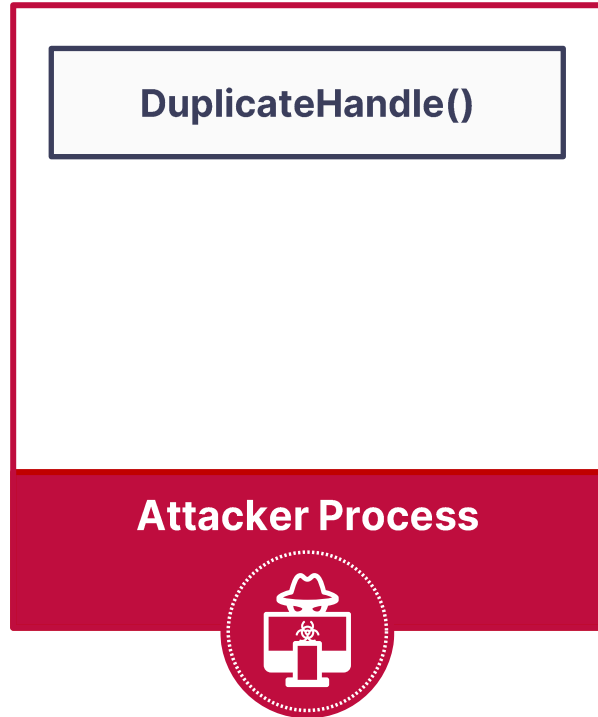


Get handle table →

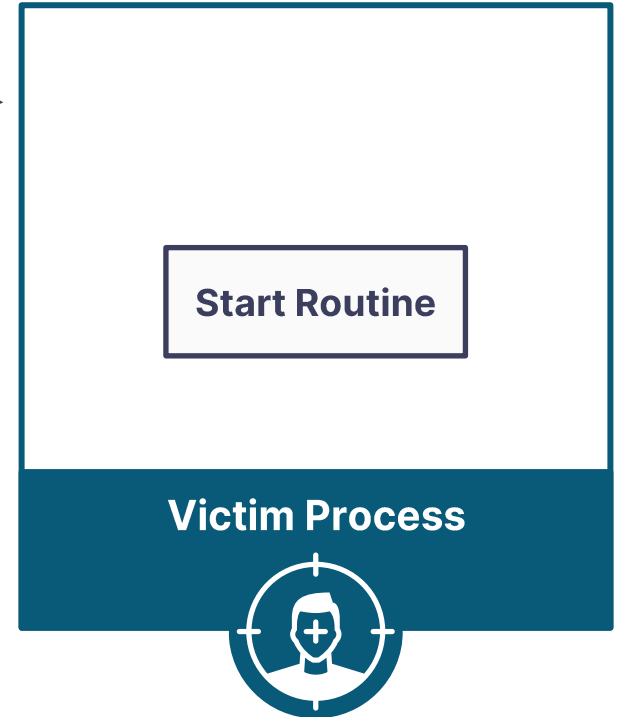




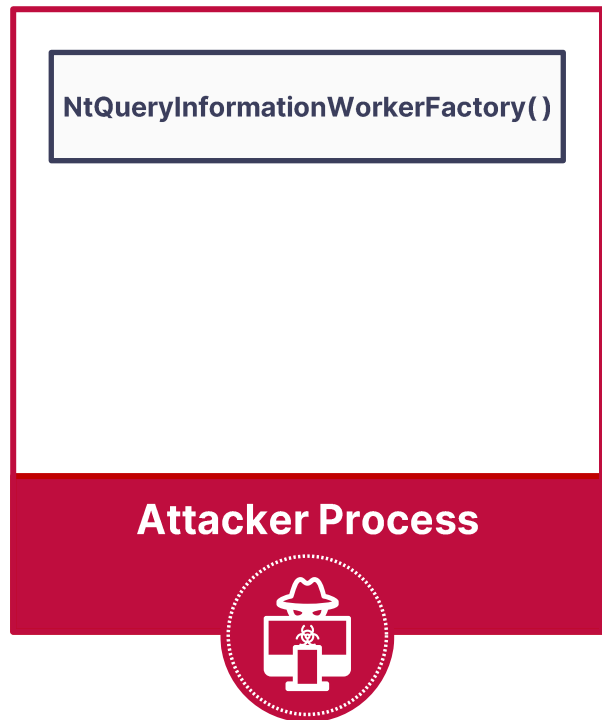
# Attacking Worker Factories



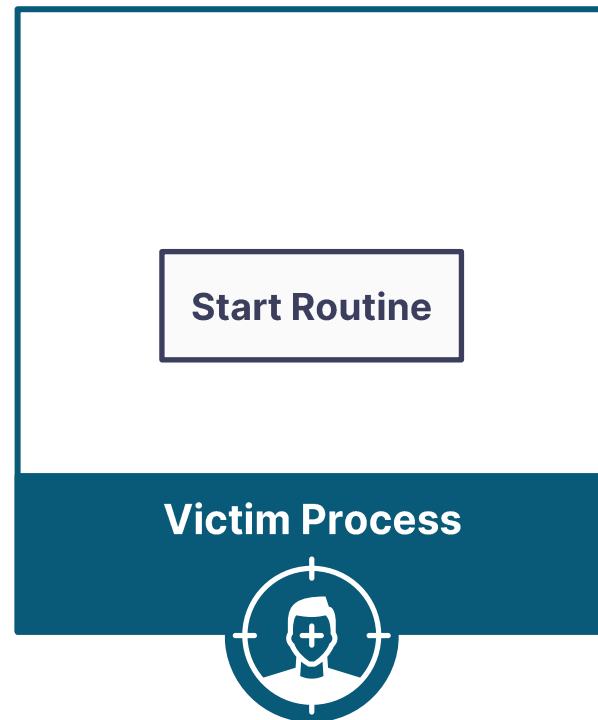
Duplicate  
Worker Factory  
handle →



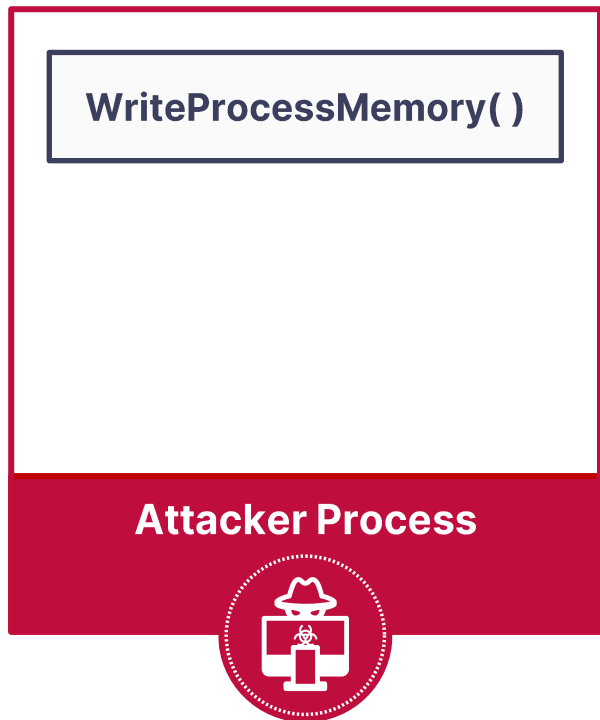
# Attacking Worker Factories



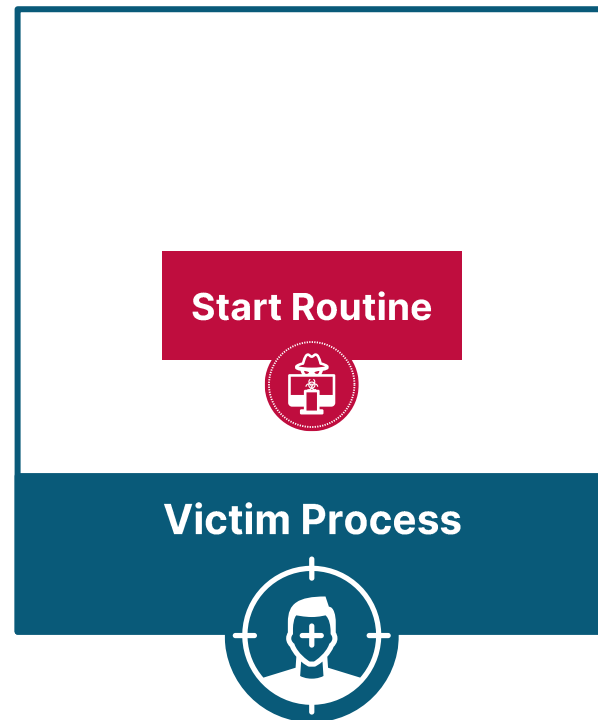
Get Worker  
Factory info



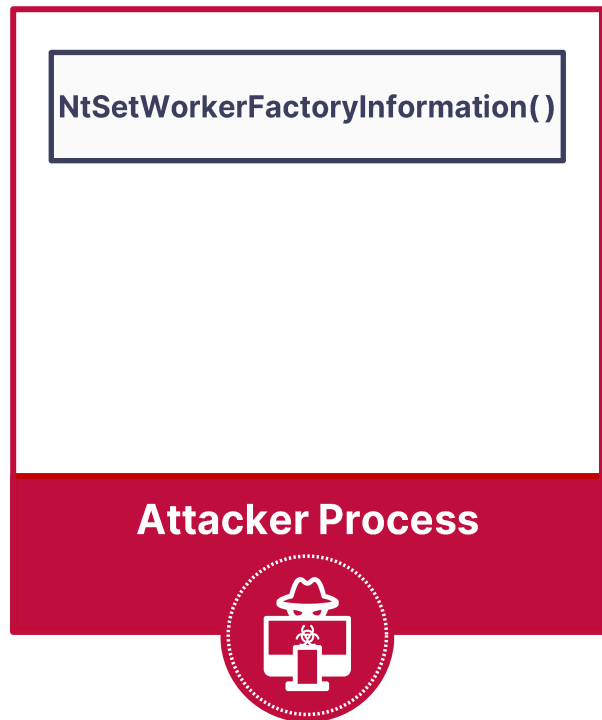
# Attacking Worker Factories



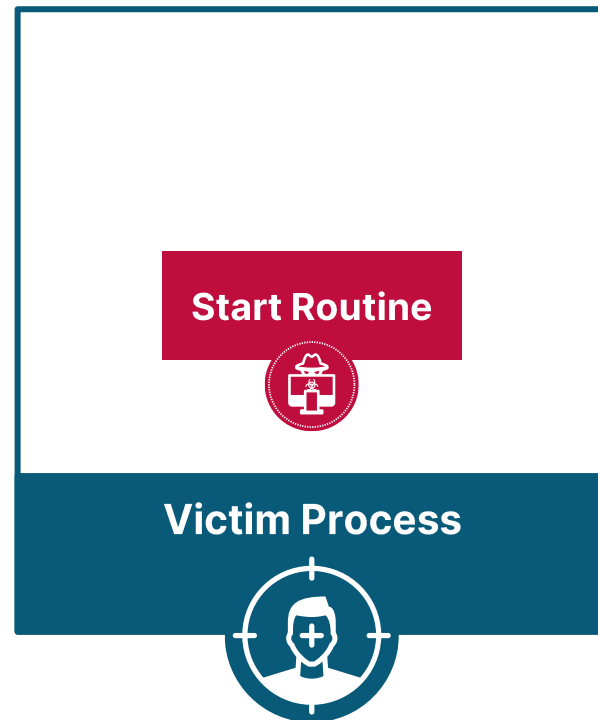
Write shellcode  
to start routine



# Attacking Worker Factories



Increase worker  
factory minimum threads →



PoolParty State

First friend in the pool



# Attacking Thread Pools



# Attacking Thread Pools

---

## Goal

---

Insert work items to a target process

---

---

## Focus of analysis

---

How work items are inserted thread pools

---

---

## Assumptions

---

Access to the worker factory of the thread pool

---



# Attacking Thread Pools - Work Item Types

Regular Work Items

**TP\_WORK**

Asynchronous Work Items

**TP\_IO**

**TP\_WAIT**

**TP\_JOB**

**TP\_ALPC**

Timer Work Items

**TP\_TIMER**



# Attacking Thread Pools - Queue Types

**Regular work items  
are queued here**



**TP\_POOL Task Queue**

**Asynchronous work  
items are queued here**



**I/O Completion Queue**

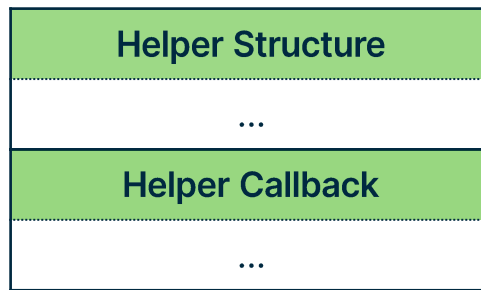
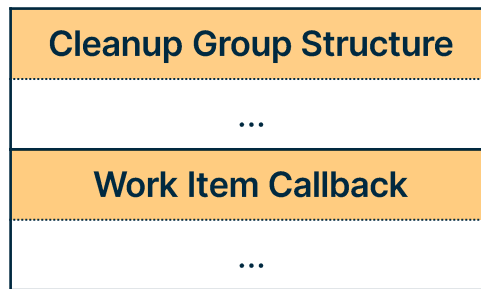
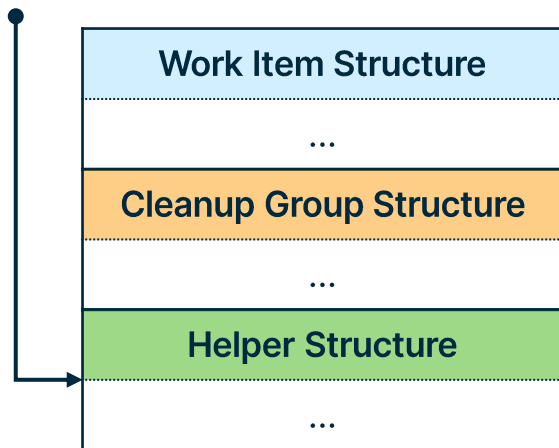
**Timer work items are  
queued here**



**TP\_POOL Timer Queue**

# User-Mode Thread Pool - Helper Structures

**Queue Helper Structure**



**Helper Executes Callback**



# Attacking Thread Pools

Regular Work Items

**TP\_WORK**

Asynchronous Work Items

**TP\_IO**

**TP\_WAIT**

**TP\_JOB**

**TP\_ALPC**

Timer Work Items

**TP\_TIMER**

# Attacking Thread Pools - TP\_WORK

```
typedef struct _TP_WORK
{
    _TPP_CLEANUP_GROUP_MEMBER CleanupGroupMember;
    TP_TASK Task;
    TPP_WORK_STATE WorkState;
    INT32 __PADDING__[1];
} TP_WORK, * PTP_WORK;
```

Helper  
Structure



# Attacking Thread Pools - TP\_WORK

## Ntdll::TpPostTask

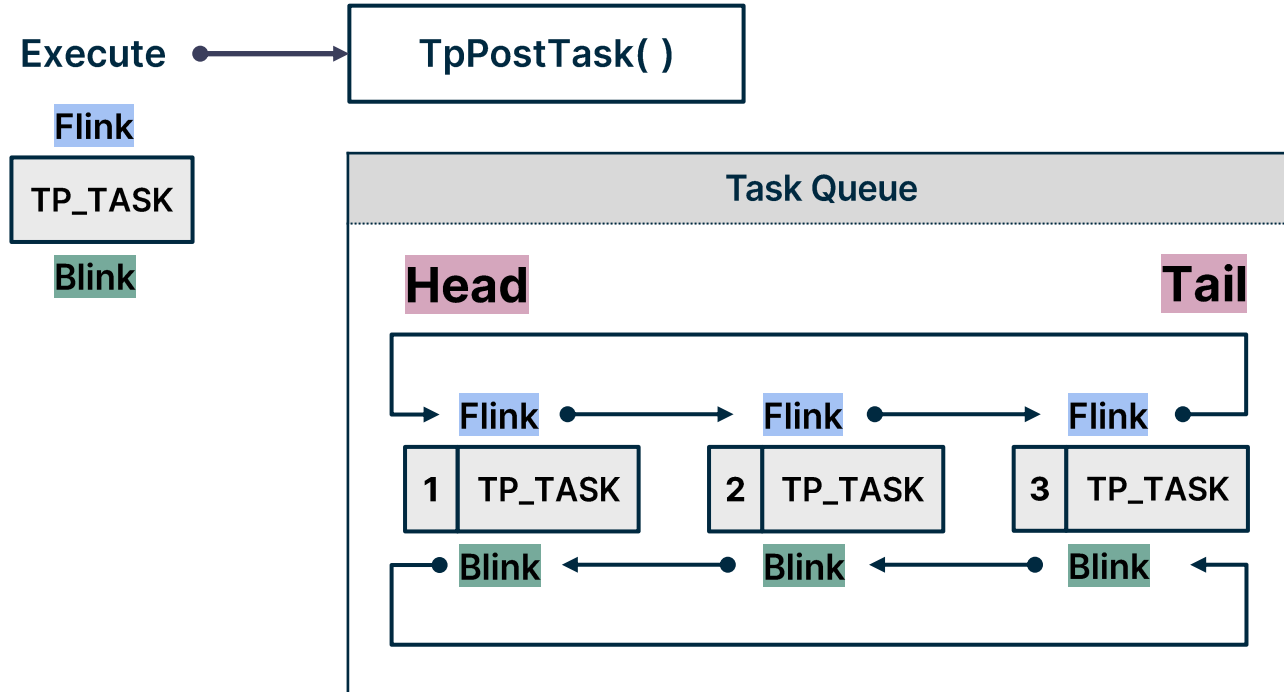
```
NTSTATUS NTAPI TpPostTask(TP_TASK* TpTask, TP_POOL* TpPool, int CallbackPriority, ...)
{
    [snip]

    TPP_QUEUE* TaskQueue = &TpPool->TaskQueue[CallbackPriority];

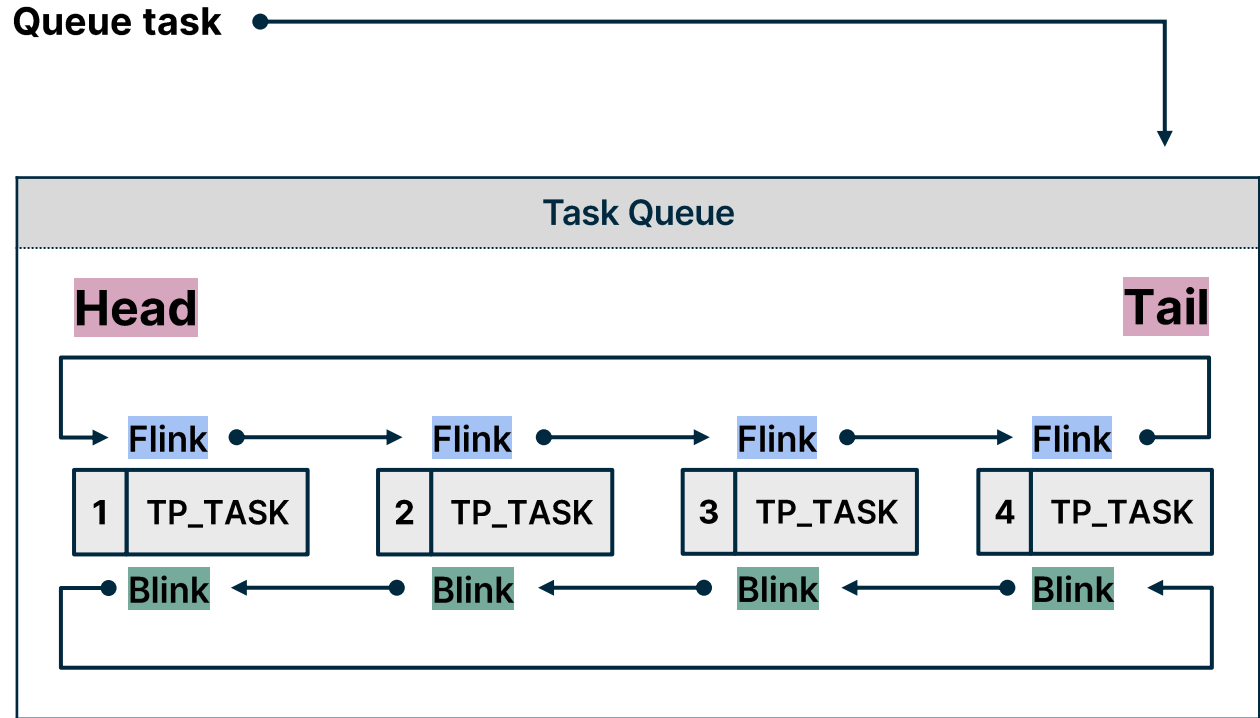
    InsertTailList(&TaskQueue->Queue, &TpTask->ListEntry);

    [snip]
}
```

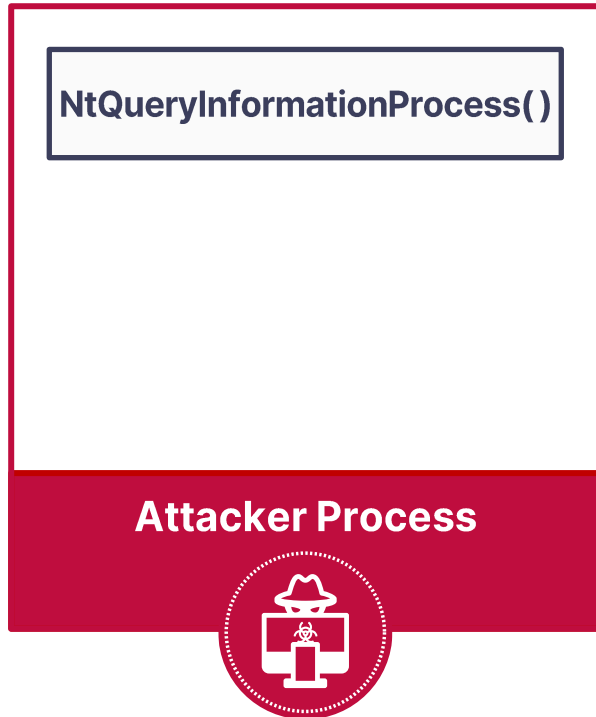
# Attacking Thread Pools - TP\_WORK



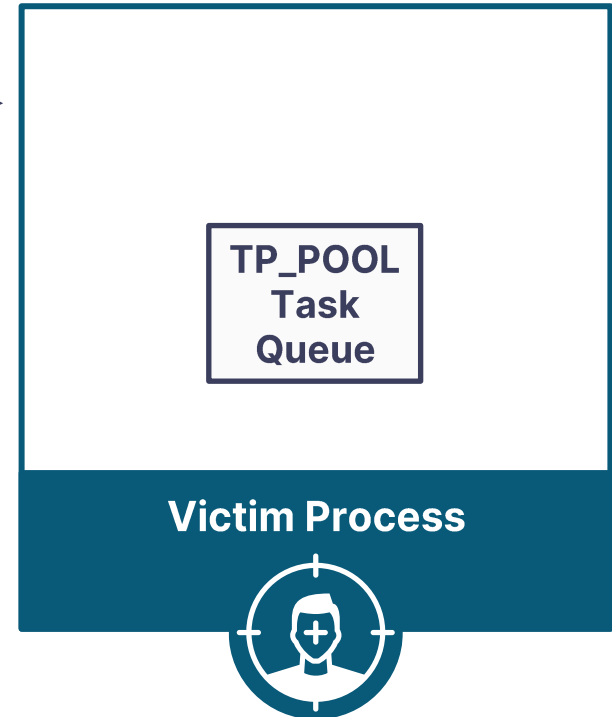
# Attacking Thread Pools - TP\_WORK



# Attacking Thread Pools – TP\_WORK



Get handle table →

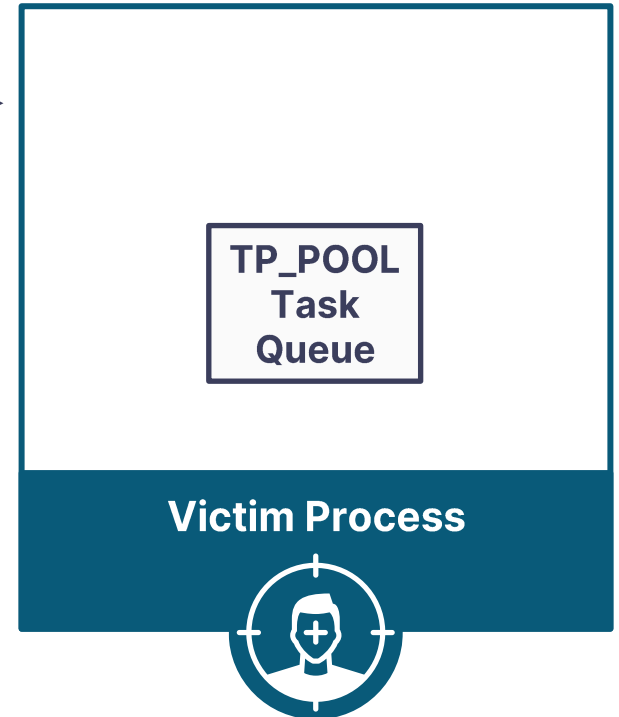




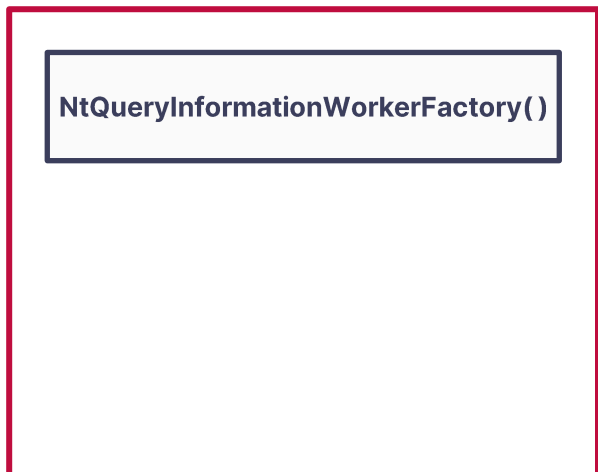
# Attacking Thread Pools – TP\_WORK



Duplicate  
Worker Factory  
handle →



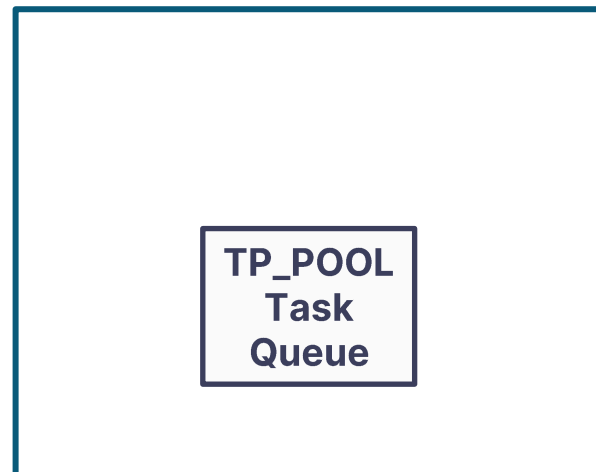
# Attacking Thread Pools – TP\_WORK



**Attacker Process**



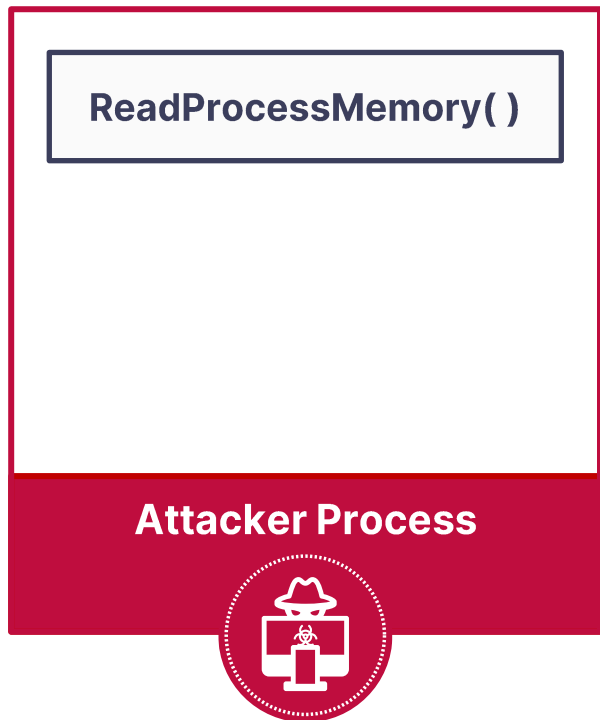
Get Worker  
Factory info



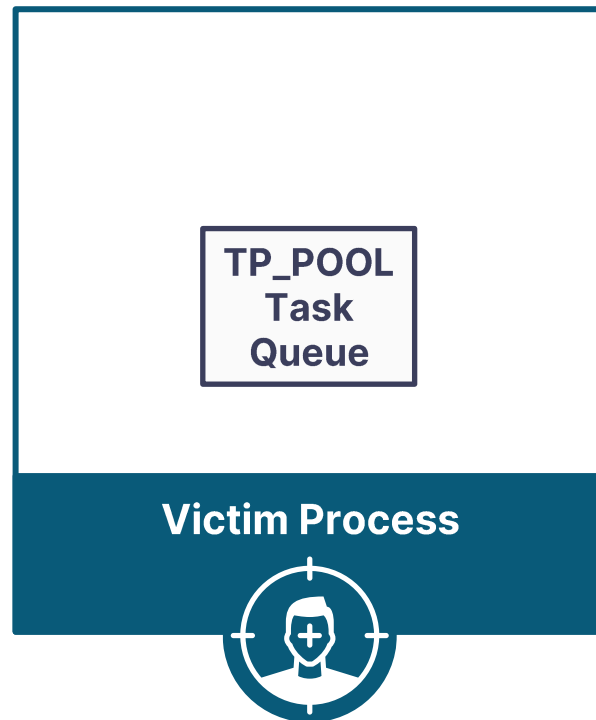
**Victim Process**



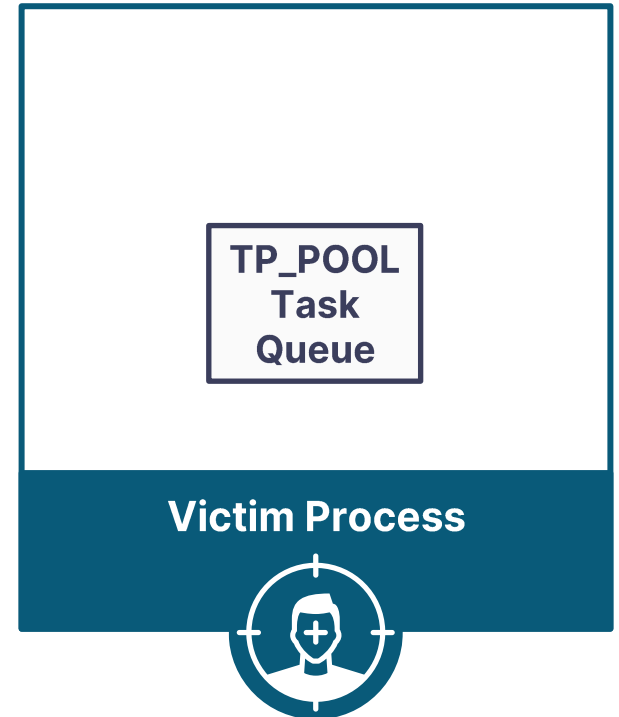
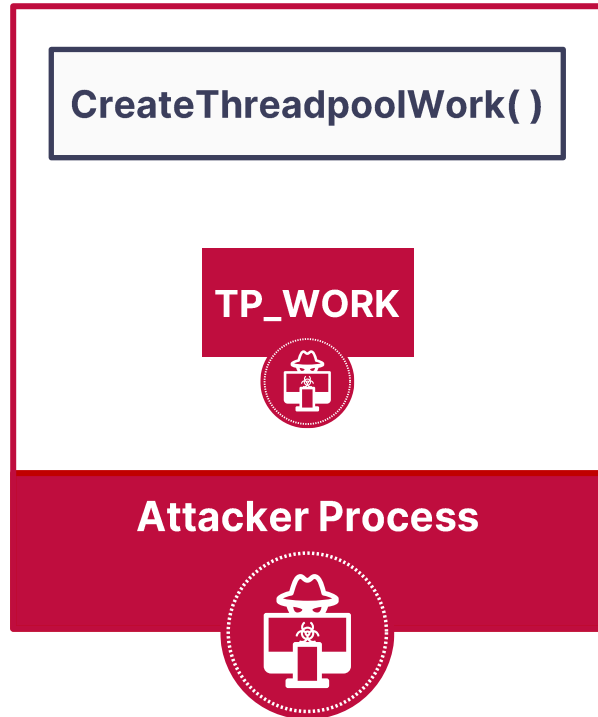
# Attacking Thread Pools – TP\_WORK



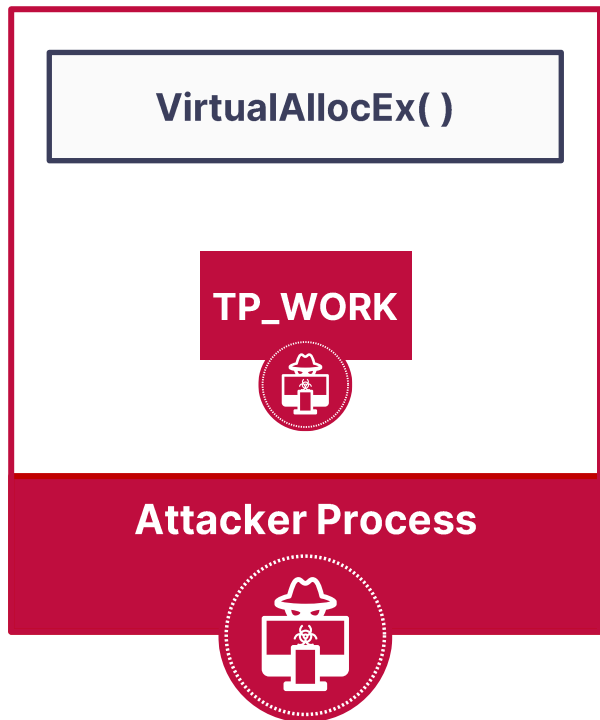
Read TP\_POOL →



# Attacking Thread Pools – TP\_WORK



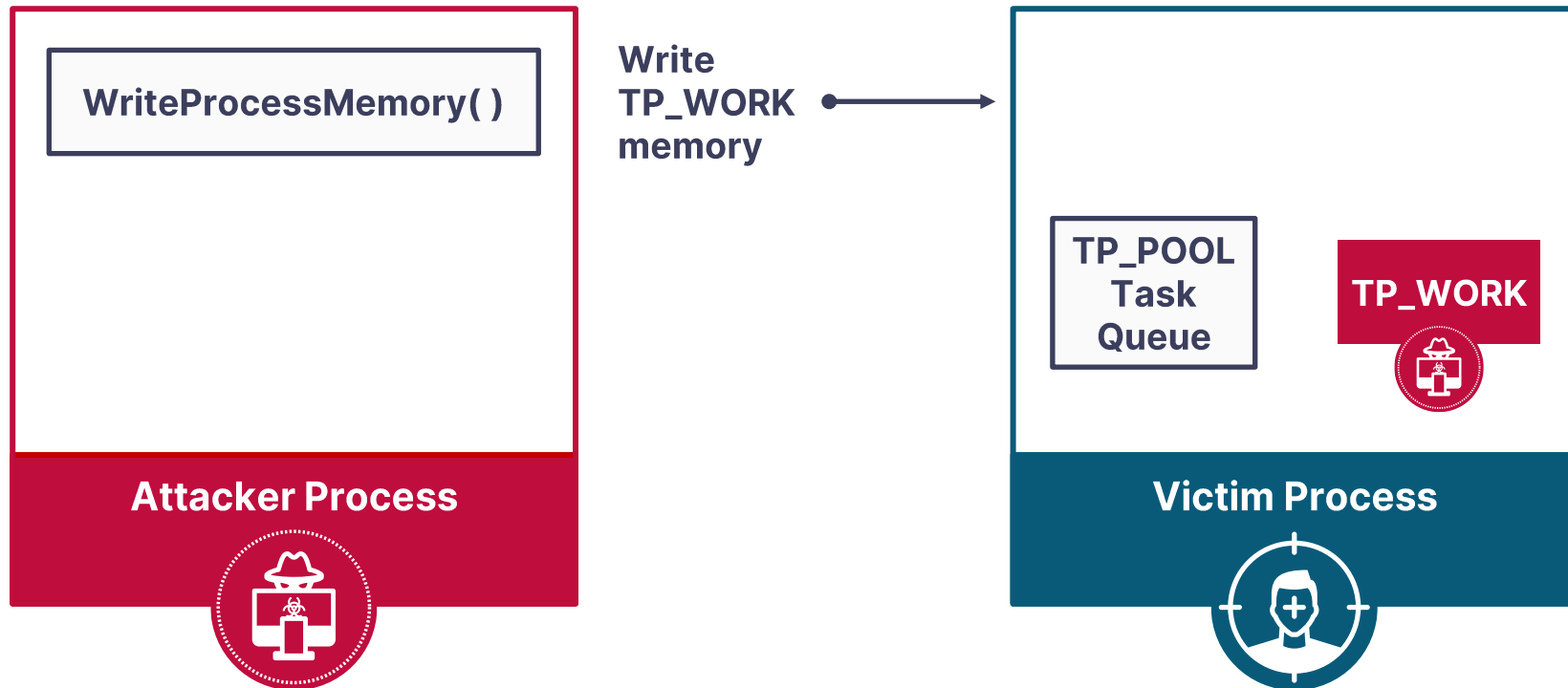
# Attacking Thread Pools – TP\_WORK



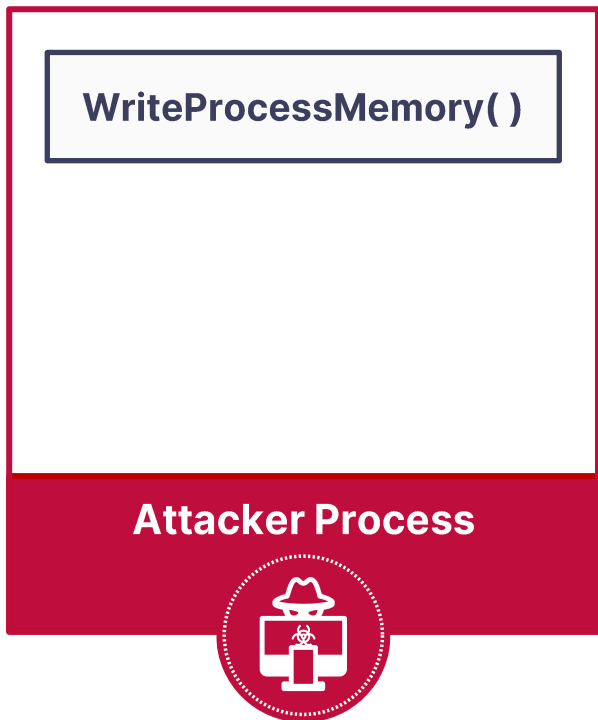
Allocate  
TP\_WORK  
memory



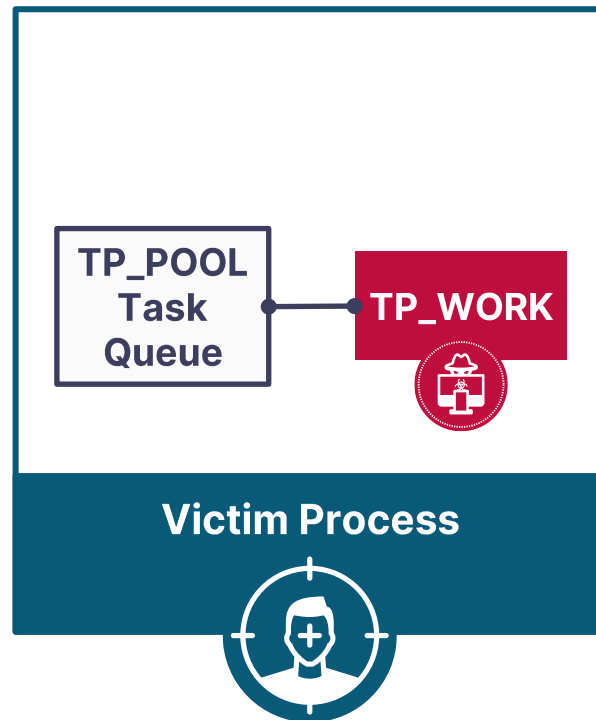
# Attacking Thread Pools – TP\_WORK



# Attacking Thread Pools – TP\_WORK



Insert  
TP\_WORK to  
TP\_POOL  
task queue



PoolParty State

Second friend in the pool





# Attacking Thread Pools

Regular Work Items

**TP\_WORK**

Asynchronous Work Items

**TP\_IO**

**TP\_WAIT**

**TP\_JOB**

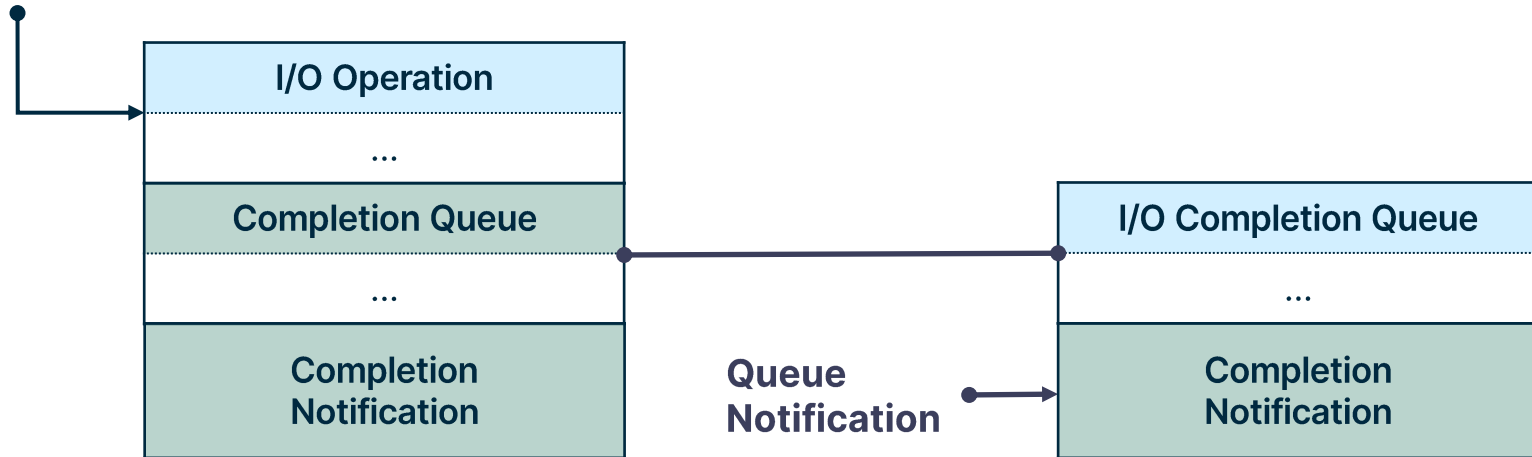
**TP\_ALPC**

Timer Work Items

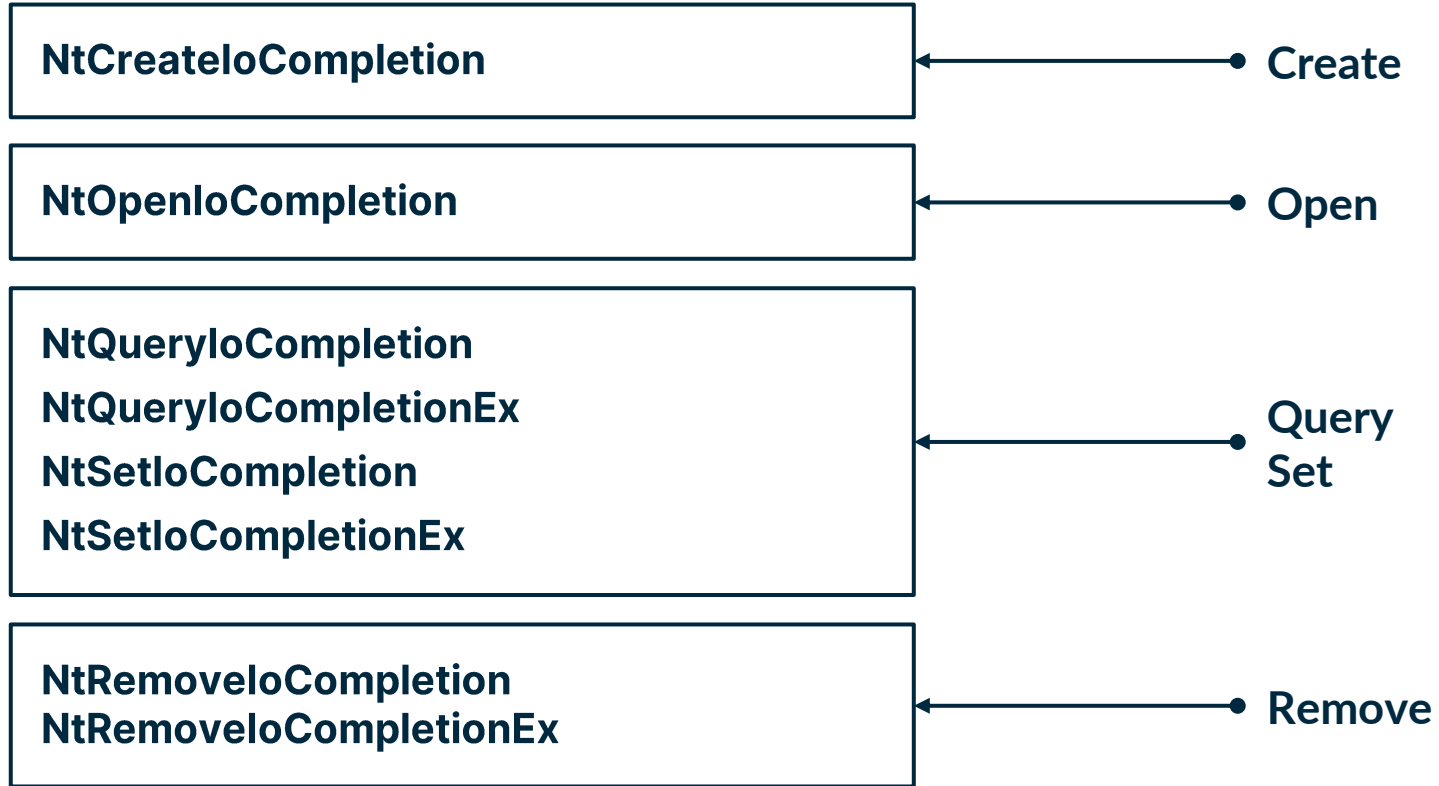
**TP\_TIMER**

# I/O Completion Queues Introduction

Completed



# I/O Completion Queues System Calls



# Attacking Thread Pools - TP\_IO

```
typedef struct _TP_IO
{
    _TPP_CLEANUP_GROUP_MEMBER CleanupGroupMember;
    TP_DIRECT Direct;
    HANDLE File;
    INT32 PendingIrpCount;
    INT32 __PADDING__[1];
} TP_WORK, * PTP_WORK;
```

Helper  
Structure



# Attacking Thread Pools - TP\_IO

## Ntdll:: TpBindFileToDirect

```
NTSTATUS NTAPI TpBindFileToDirect(HANDLE hFile, TP_DIRECT* TpDirect, TP_POOL* TpPool)
{
    [snip]

    FILE_COMPLETION_INFORMATION FileCompletionInfo{ 0 };
    FileCompletionInfo.Key = TpDirect;
    FileCompletionInfo.Port = TpPool->CompletionPort;

    NtSetInformationFile(
        hFile,
        &IoStatusBlock,
        &FileCompletionInfo,
        sizeof(FILE_COMPLETION_INFORMATION),
        FileCompletionInformation);

    [snip]
}
```

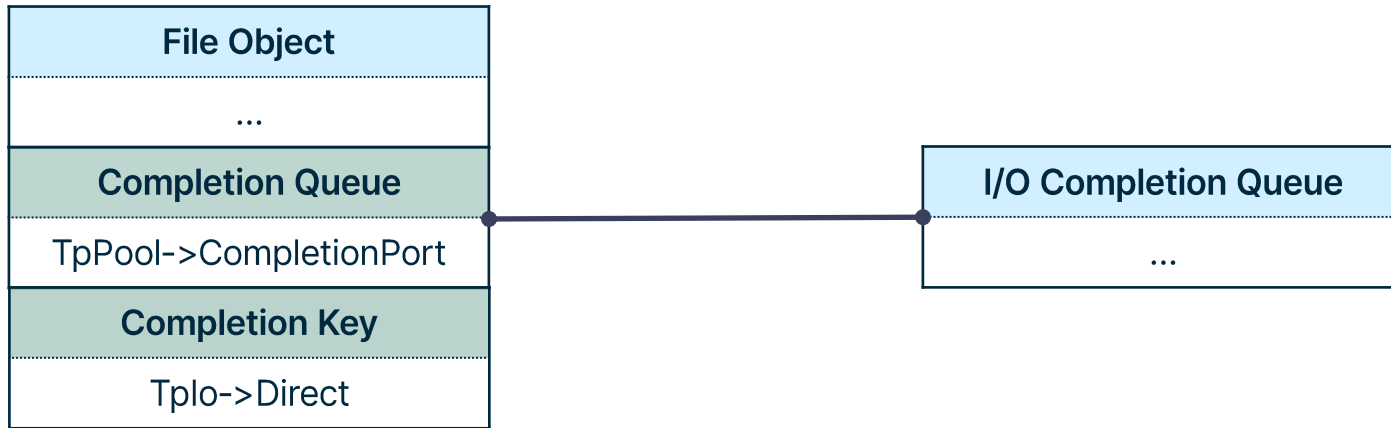
# Attacking Thread Pools - TP\_IO



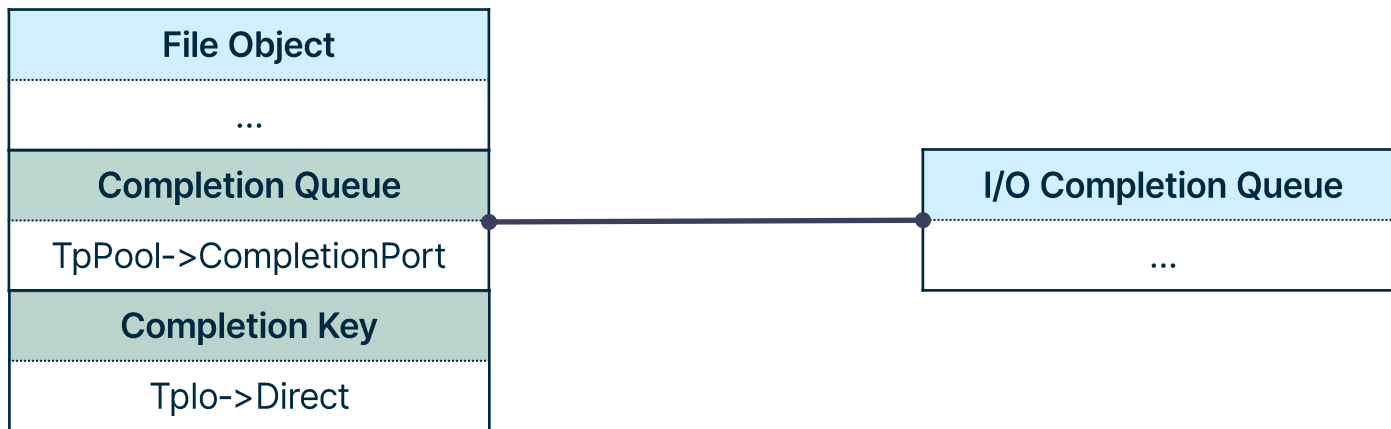
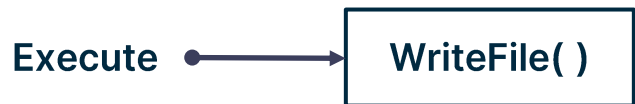
|                         |
|-------------------------|
| <b>File Object</b>      |
| ...                     |
| <b>Completion Queue</b> |
| NULL                    |
| <b>Completion Key</b>   |
| NULL                    |

|                             |
|-----------------------------|
| <b>I/O Completion Queue</b> |
| ...                         |

# Attacking Thread Pools - TP\_IO

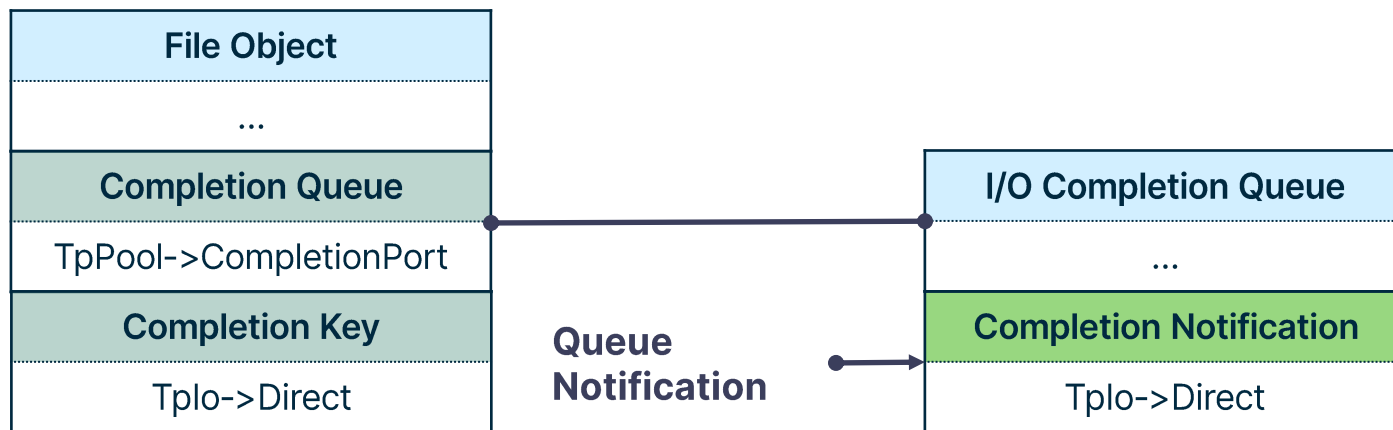


# Attacking Thread Pools - TP\_IO

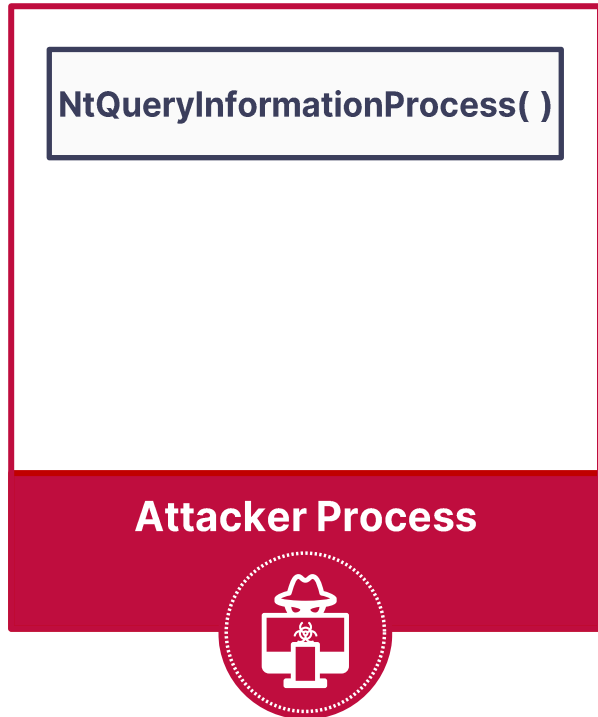




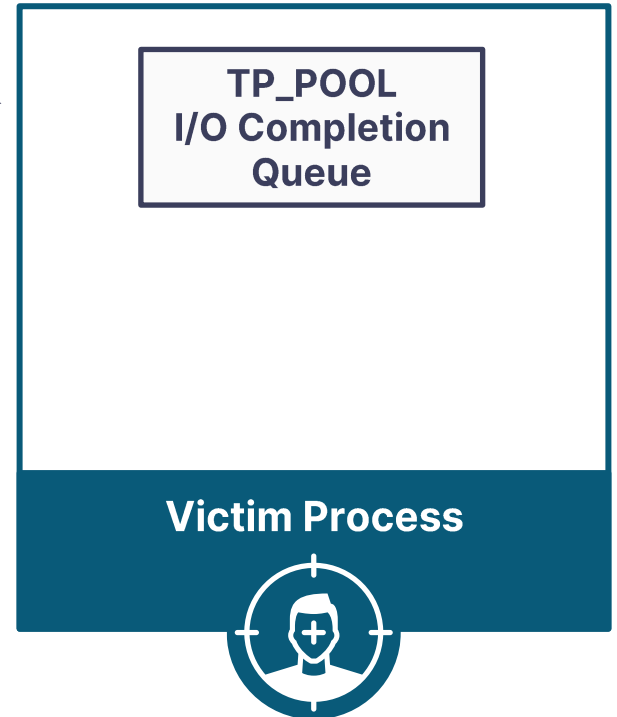
# Attacking Thread Pools - TP\_IO



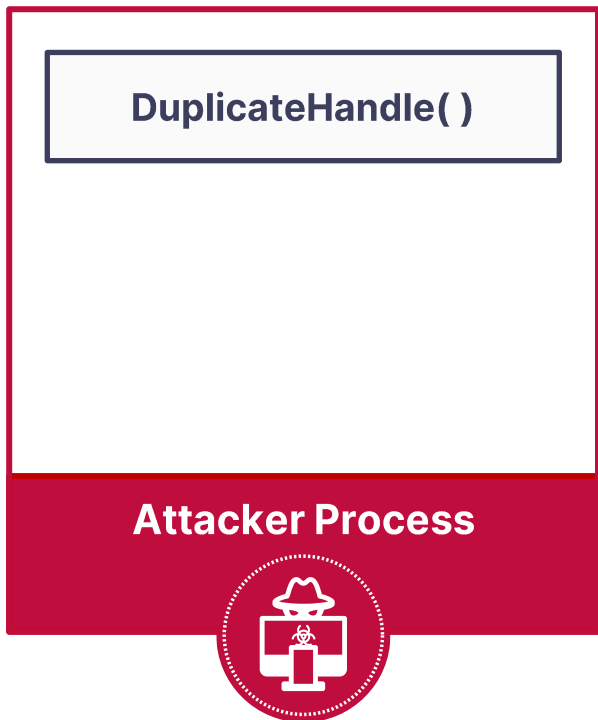
# Attacking Thread Pools - TP\_IO



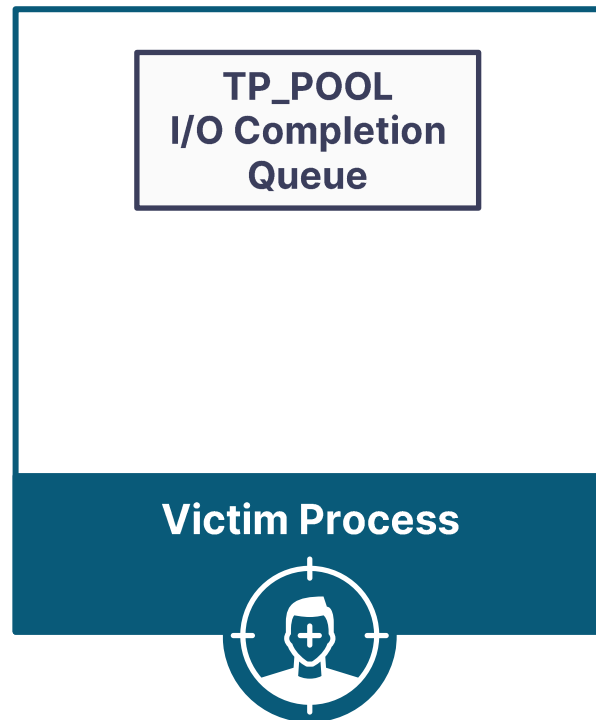
Get handle table



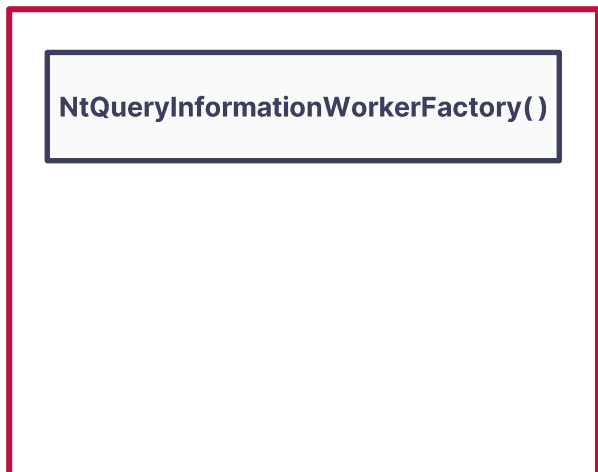
# Attacking Thread Pools - TP\_IO



Duplicate  
Worker Factory  
handle →



# Attacking Thread Pools - TP\_IO



**Attacker Process**



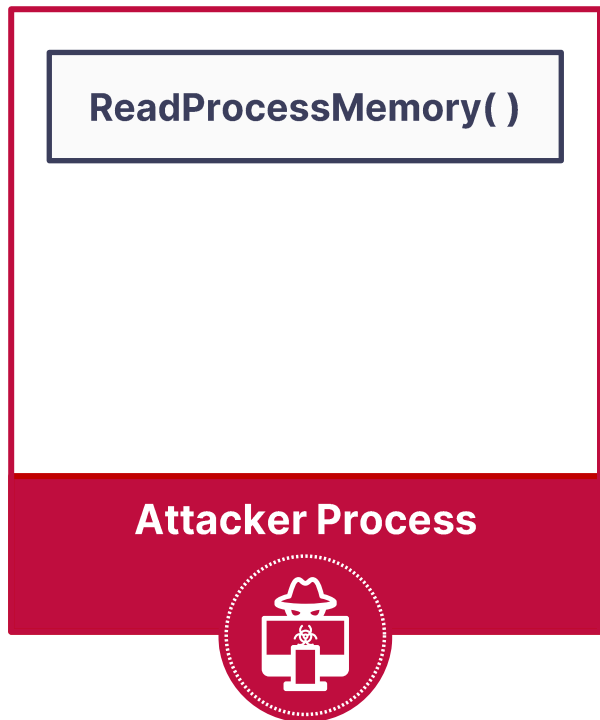
Get Worker  
Factory  
info



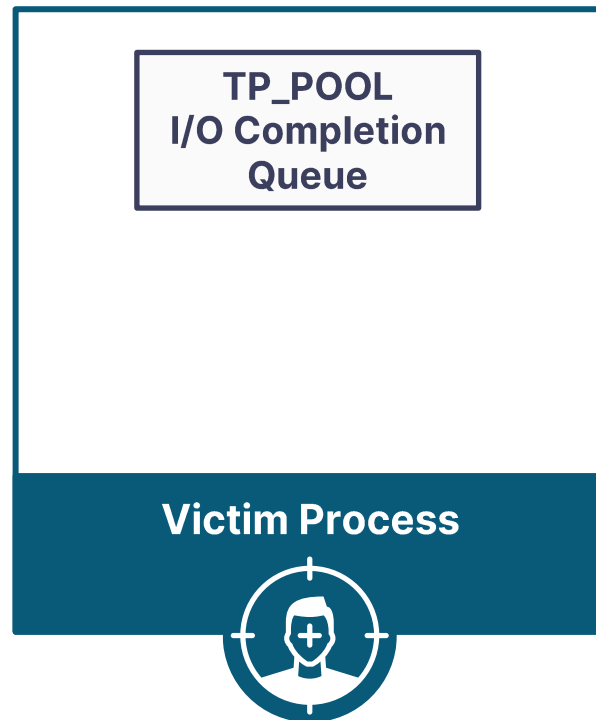
**Victim Process**



# Attacking Thread Pools - TP\_IO



Read TP\_POOL →



# Attacking Thread Pools - TP\_IO



**Attacker Process**



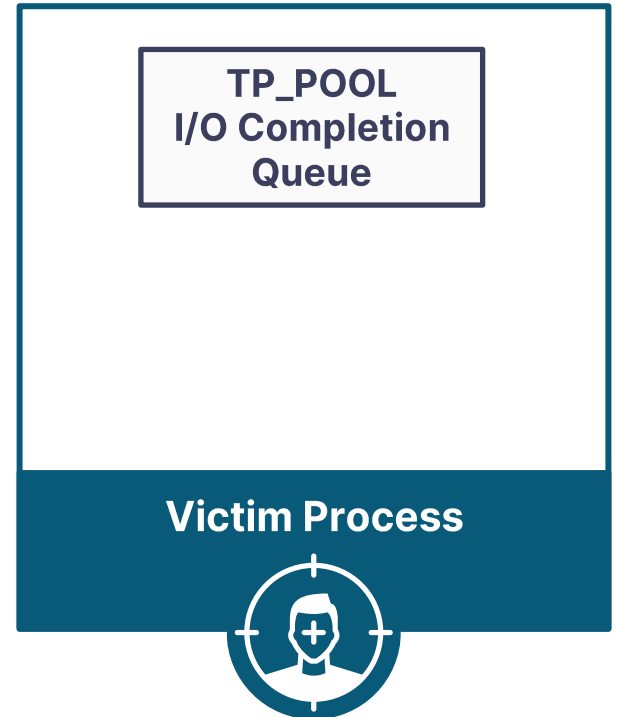
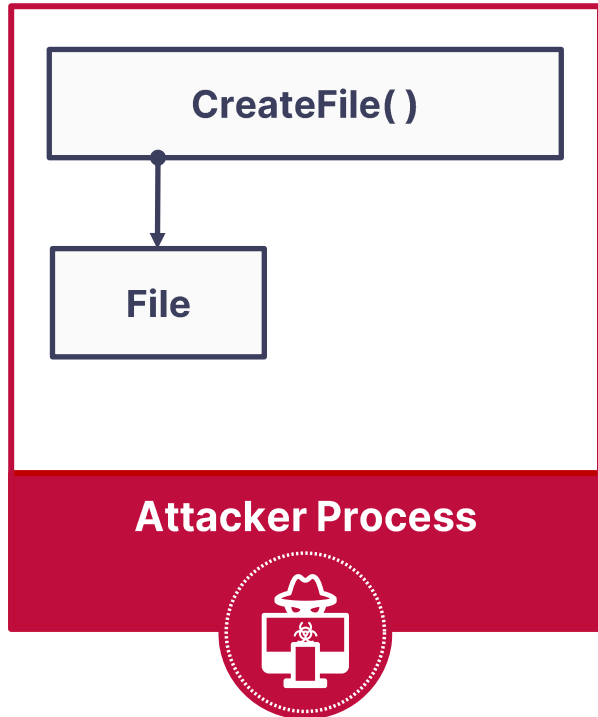
Duplicate I/O  
Completion  
queue handle



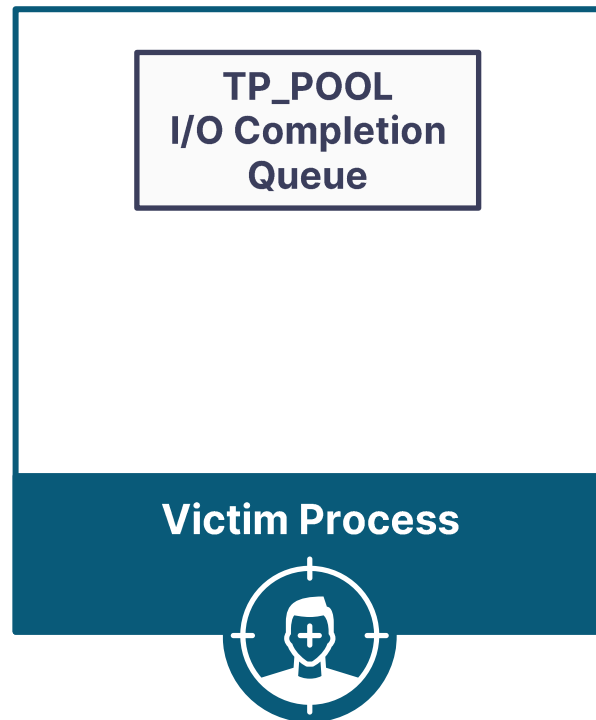
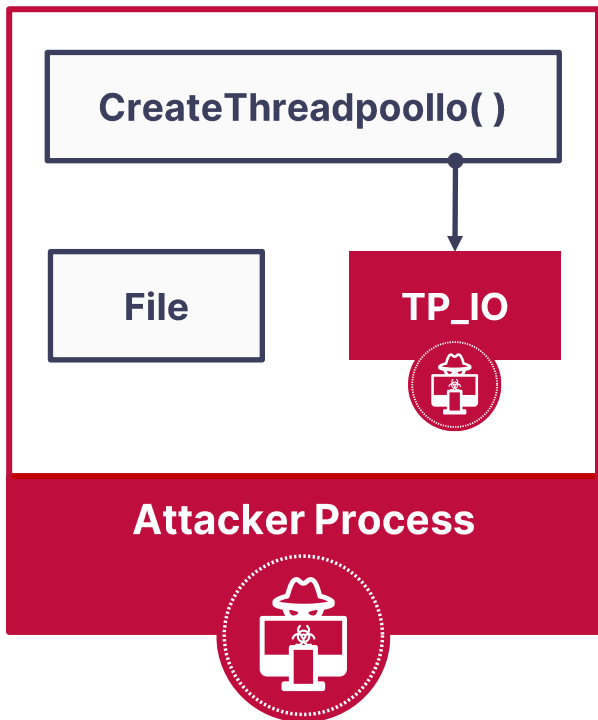
**Victim Process**



# Attacking Thread Pools - TP\_IO

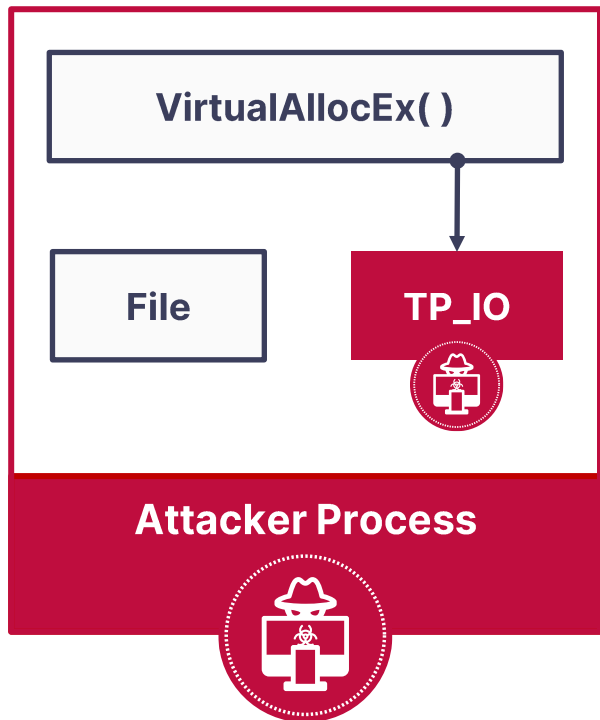


# Attacking Thread Pools - TP\_IO

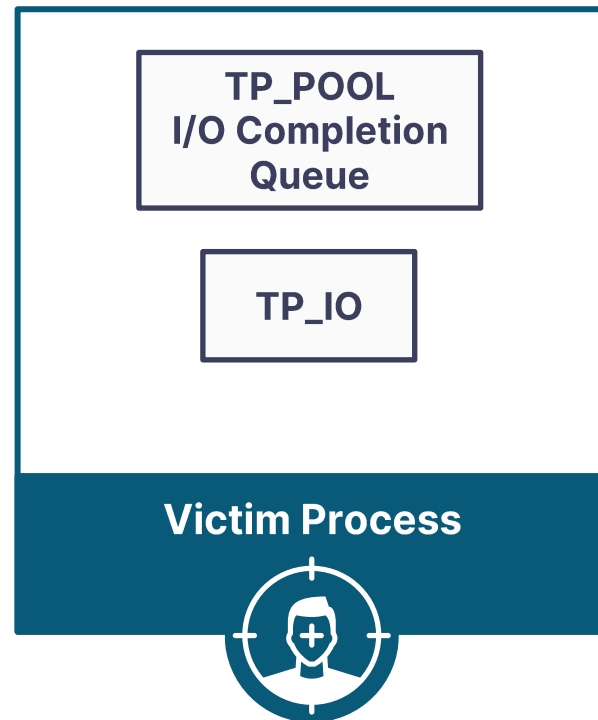




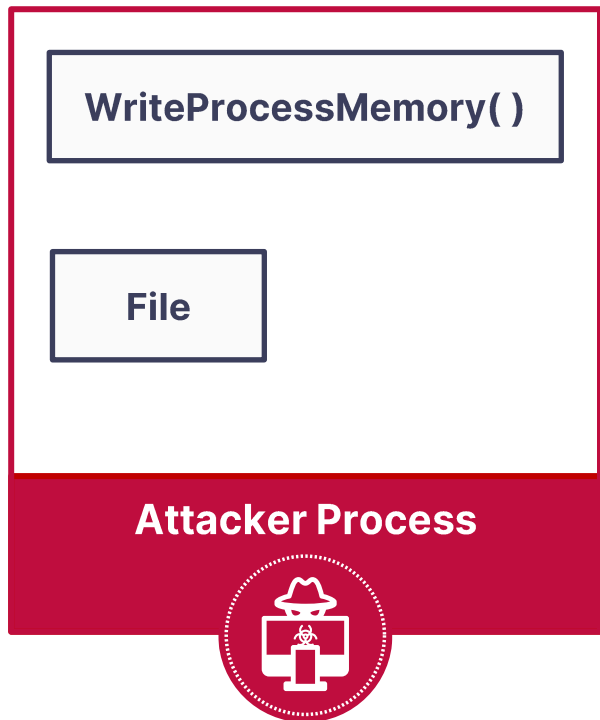
# Attacking Thread Pools - TP\_IO



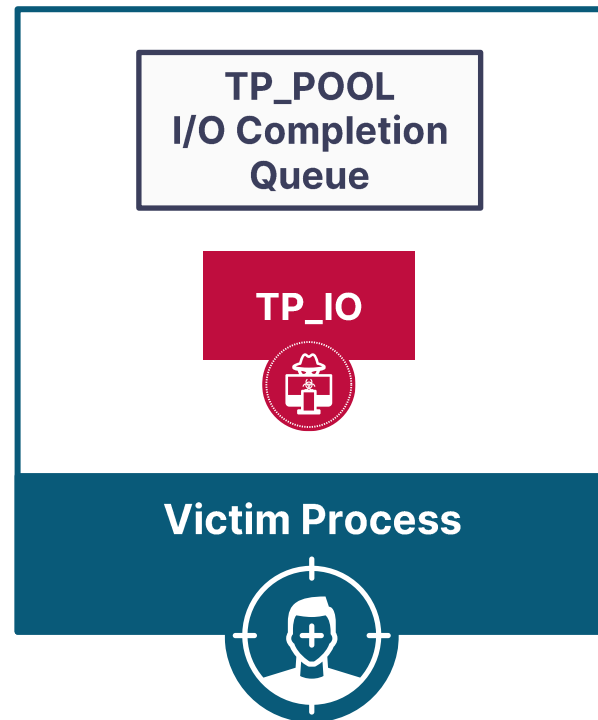
Allocate  
TP\_IO  
memory



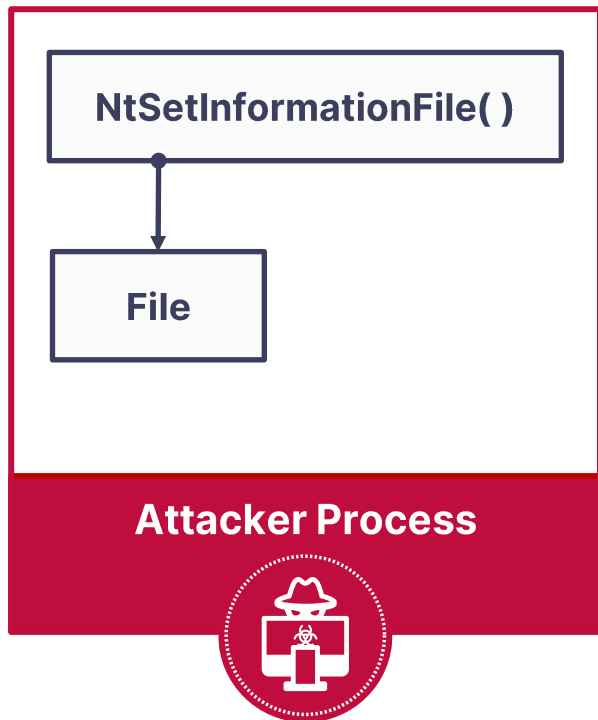
# Attacking Thread Pools - TP\_IO



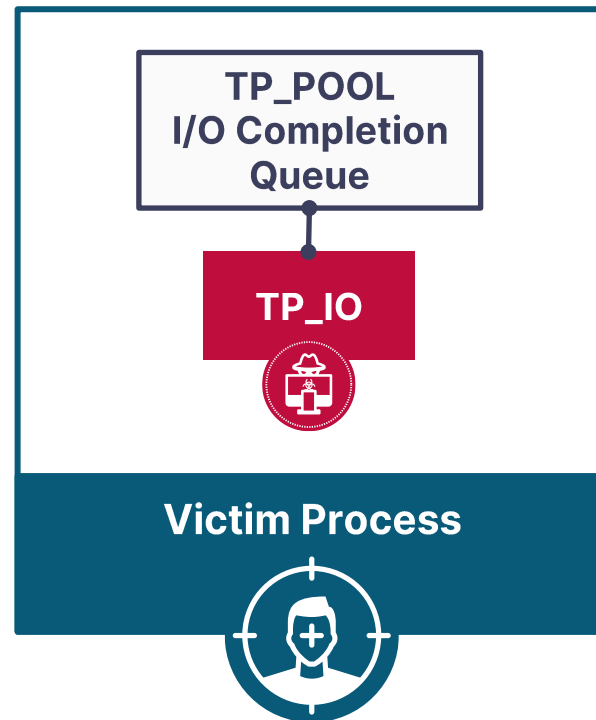
Write TP\_IO  
memory



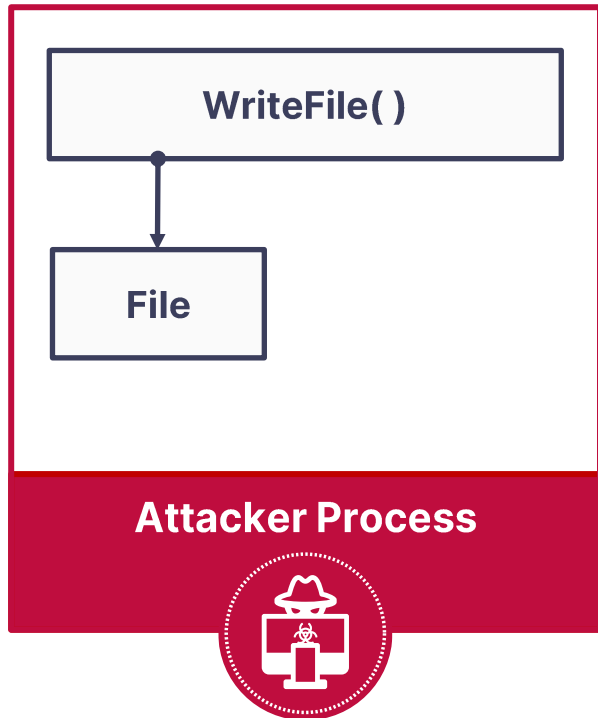
# Attacking Thread Pools - TP\_IO



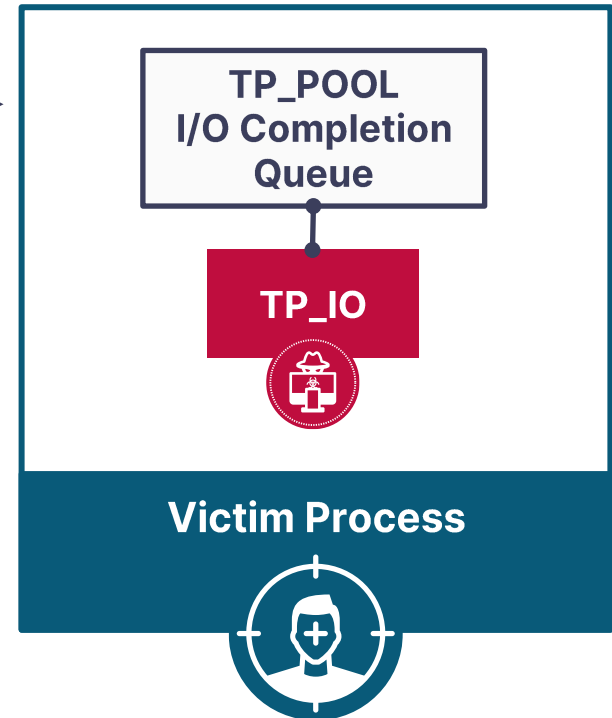
Associate TP\_IO with target I/O completion queue



# Attacking Thread Pools - TP\_IO



Queue  
notification  
to I/O  
completion  
queue



# Attacking Thread Pools - IO, ALPC, JOB, ...

---

Any TP\_DIRECT notification queued to I/O completion queue gets executed

---

Notifications can be queued by object operation completion

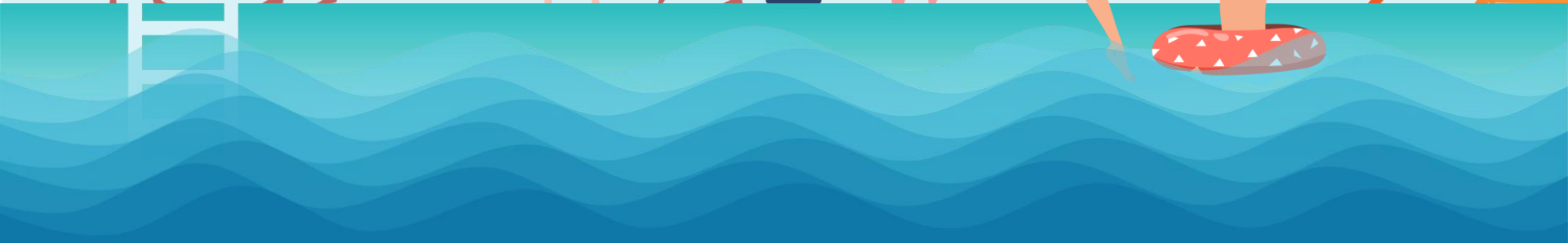
- File objects (TP\_IO)
  - ALPC port objects (TP\_ALPC)
  - Job objects (TP\_JOB)
  - Waitable objects – (TP\_WAIT)
- 

Notifications can be queued directly by NtSetIoCompletion system call

---

# PoolParty State

## Five new friends in the pool



# Attacking Thread Pools

Regular Work Items

**TP\_WORK**

Asynchronous Work Items

**TP\_IO**

**TP\_WAIT**

**TP\_JOB**

**TP\_ALPC**

Timer Work Items

**TP\_TIMER**

# Attacking Thread Pools - TP\_TIMER

**No timer handle is supplied**

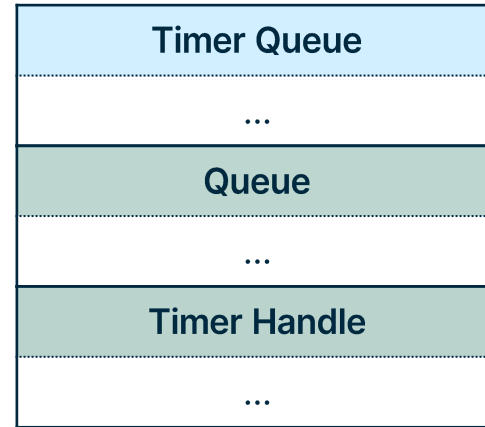
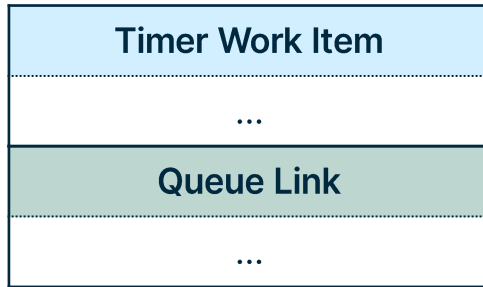


```
PTP_TIMER NTAPI CreateThreadpoolTimer(  
    _In_      PTP_TIMER_CALLBACK TimerCallback,  
    _In_Opt  PVOID TimerContext,  
    _In_Opt  PTP_CALLBACK_ENVIRON TpCallbackEnviron  
);
```

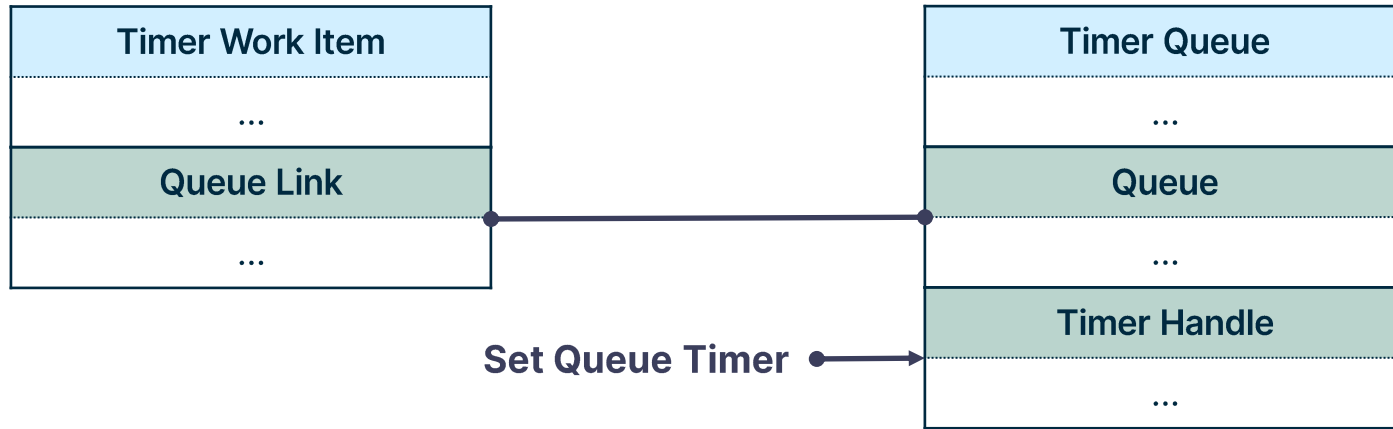
```
void NTAPI SetThreadpoolTimer(  
    _In_      PTP_TIMER_CALLBACK TimerCallback,  
    _In_Opt  PFILETIME DueTime,  
    _In_      DWORD Period,  
    _In_      DWORD WindowLength  
);
```



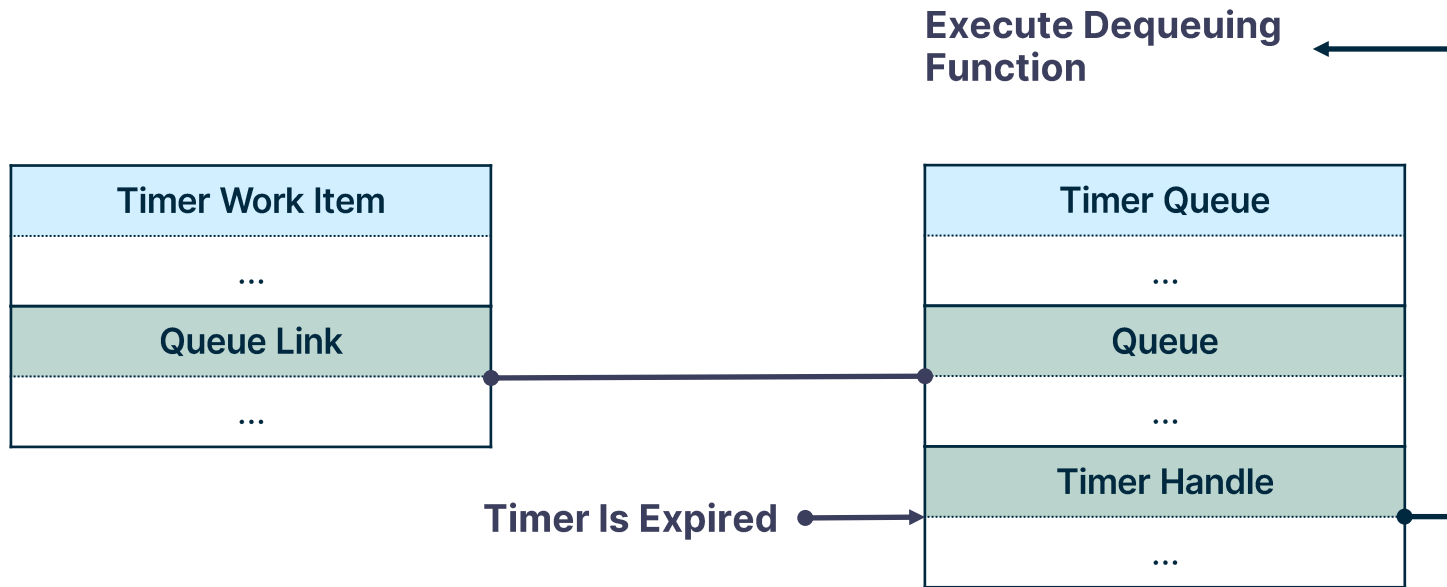
# Attacking Thread Pools – TP\_TIMER



# Attacking Thread Pools – TP\_TIMER



# Attacking Thread Pools – TP\_TIMER



# Attacking Thread Pools - TP\_TIMER

```
typedef struct _TP_TIMER
{
    [snip]
    TPP_PH_LINKS WindowEndLinks;
    TPP_PH_LINKS WindowStartLinks;
    [snip]
} TP_TIMER, * PTP_TIMER;
```

# Attacking Thread Pools - TP\_TIMER

## Ntdll:: TppEnqueueTimer

```
NTSTATUS NTAPI TppEnqueueTimer(TPP_TIMER_QUEUE* TimerQueue, TP_TIMER* TpTimer)
{
    [snip]
    TppPHInsert(&TimerQueue->WindowStart, &TpTimer->WindowStartLinks);
    TppPHInsert(&TimerQueue->WindowEnd, &TpTimer->WindowEndLinks);
    [snip]
}
```

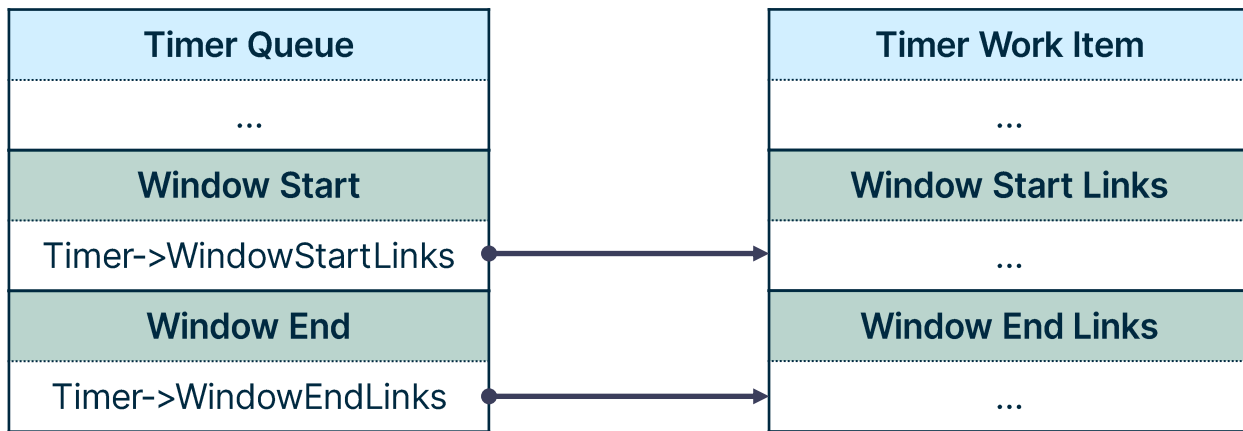
# Attacking Thread Pools – TP\_TIMER



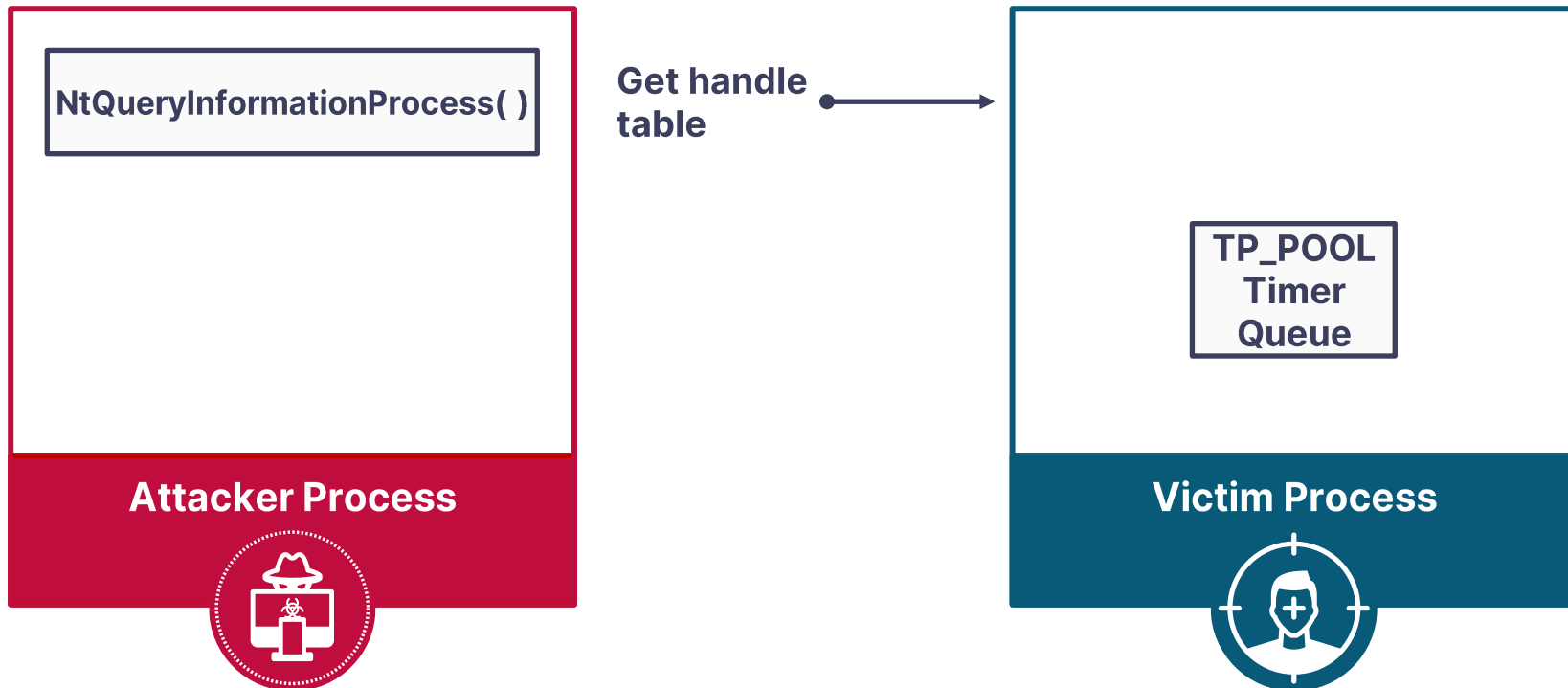
| Timer Queue  |
|--------------|
| ...          |
| Window Start |
| NULL         |
| Window End   |
| NULL         |

| Timer Work Item    |
|--------------------|
| ...                |
| Window Start Links |
| ...                |
| Window End Links   |
| ...                |

# Attacking Thread Pools – TP\_TIMER



# Attacking Thread Pools – TP\_TIMER

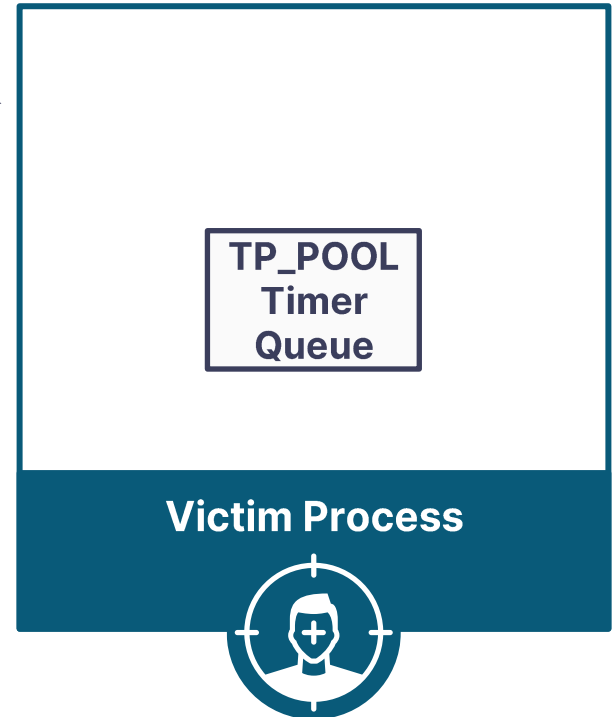




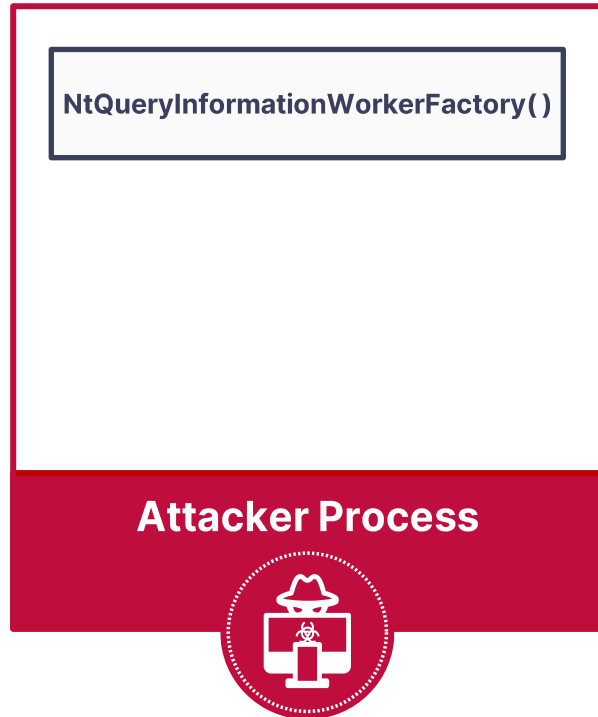
# Attacking Thread Pools – TP\_TIMER



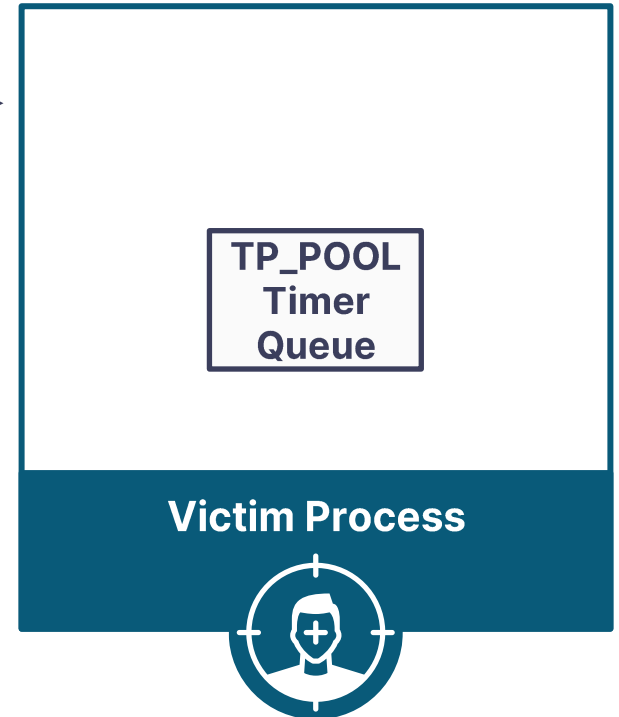
Duplicate Worker Factory handle →



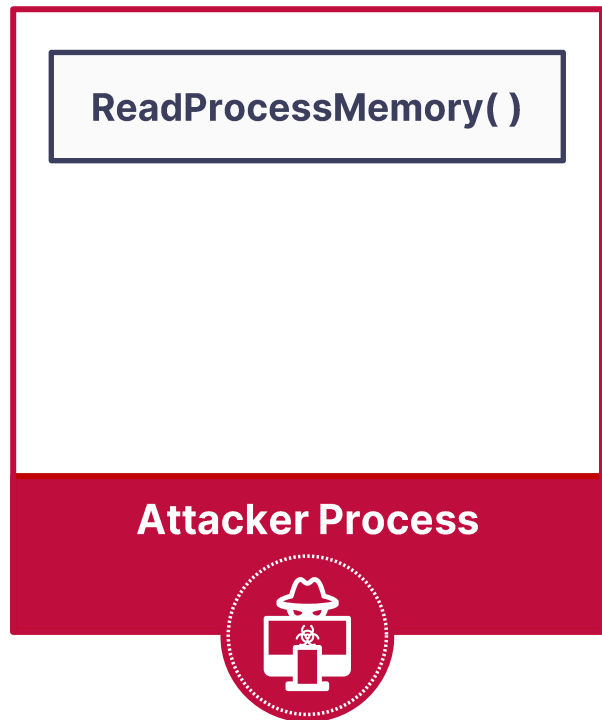
# Attacking Thread Pools – TP\_TIMER



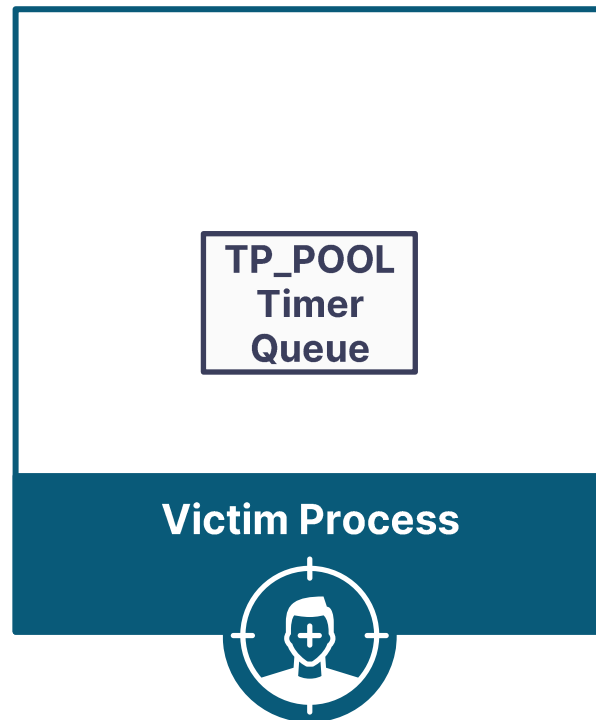
Get Worker  
Factory  
info



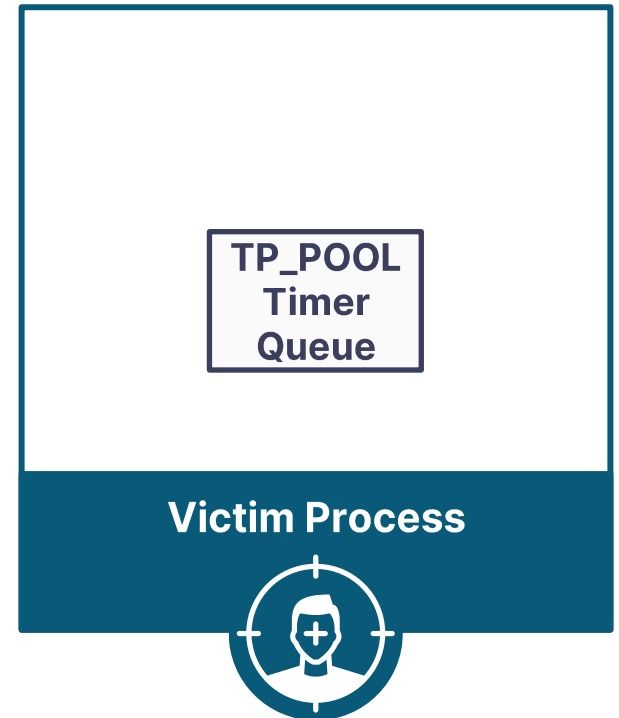
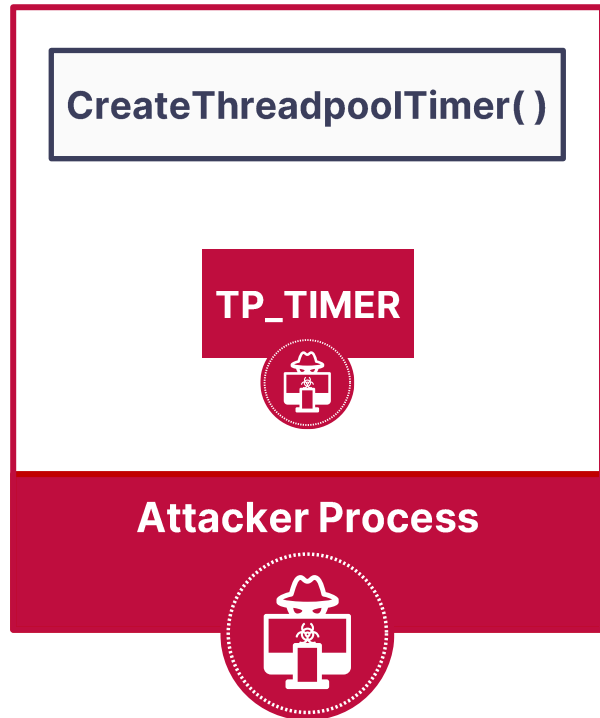
# Attacking Thread Pools – TP\_TIMER



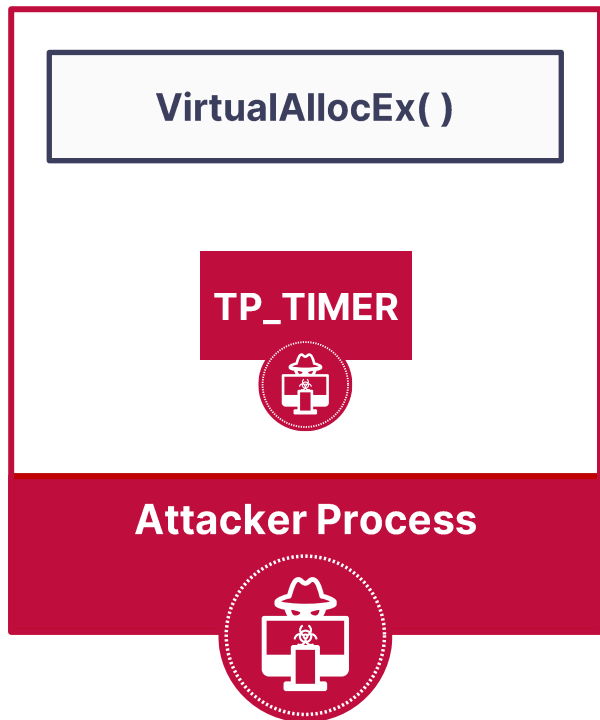
Read TP\_POOL →



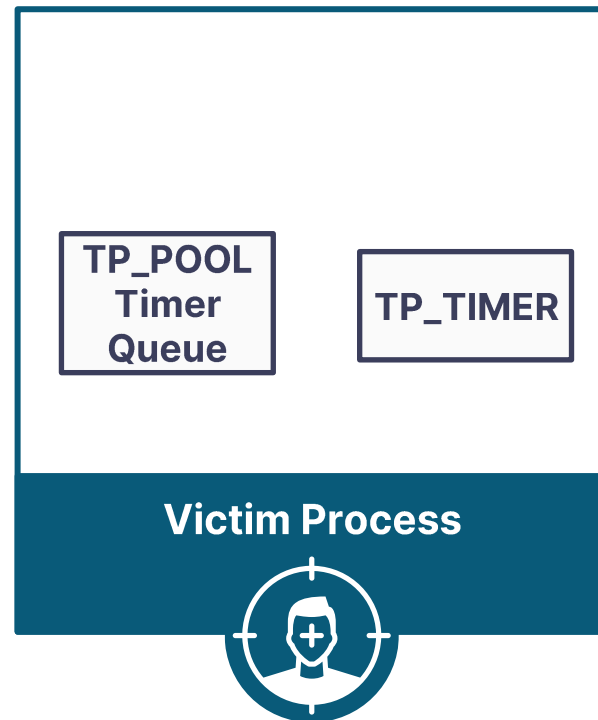
# Attacking Thread Pools – TP\_TIMER



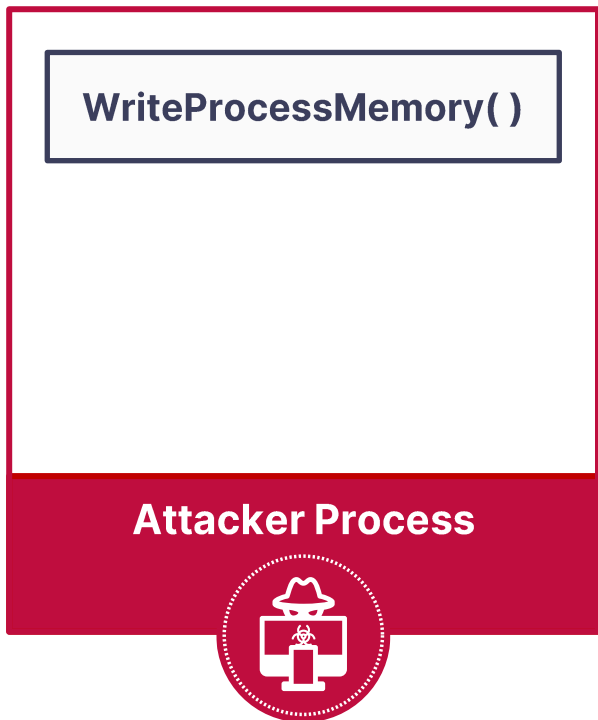
# Attacking Thread Pools – TP\_TIMER



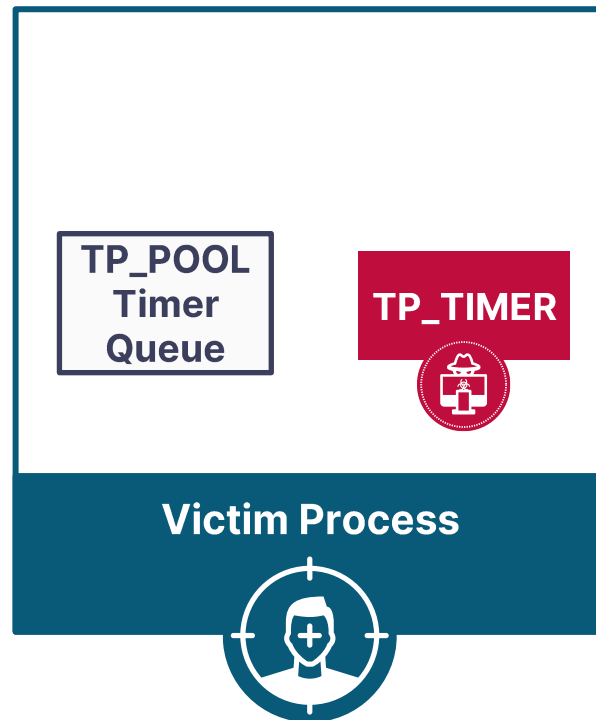
Allocate  
TP\_TIMER  
memory



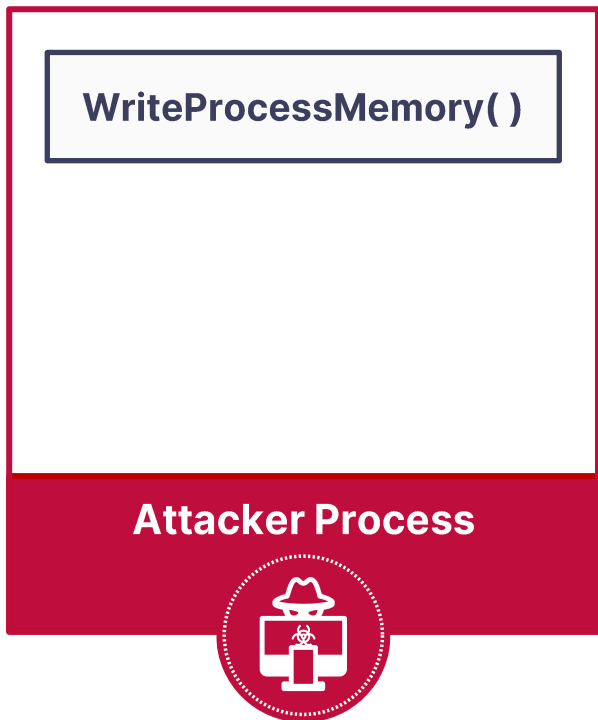
# Attacking Thread Pools – TP\_TIMER



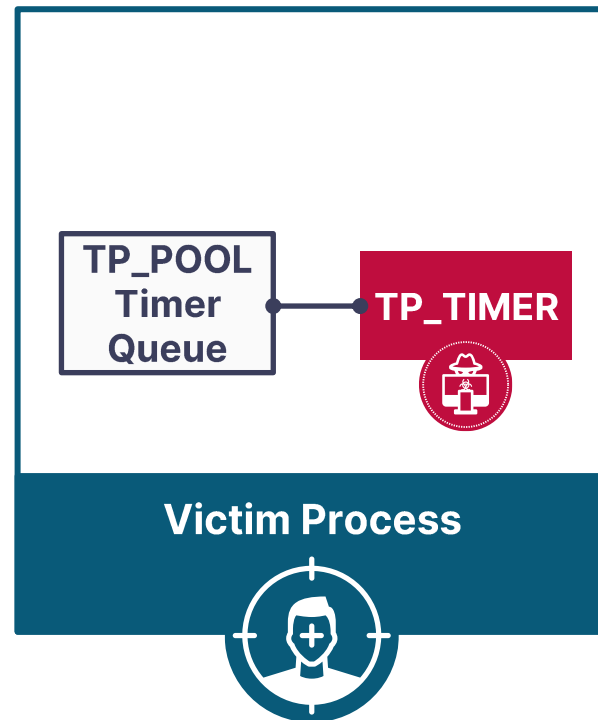
Write  
TP\_TIMER  
memory



# Attacking Thread Pools – TP\_TIMER



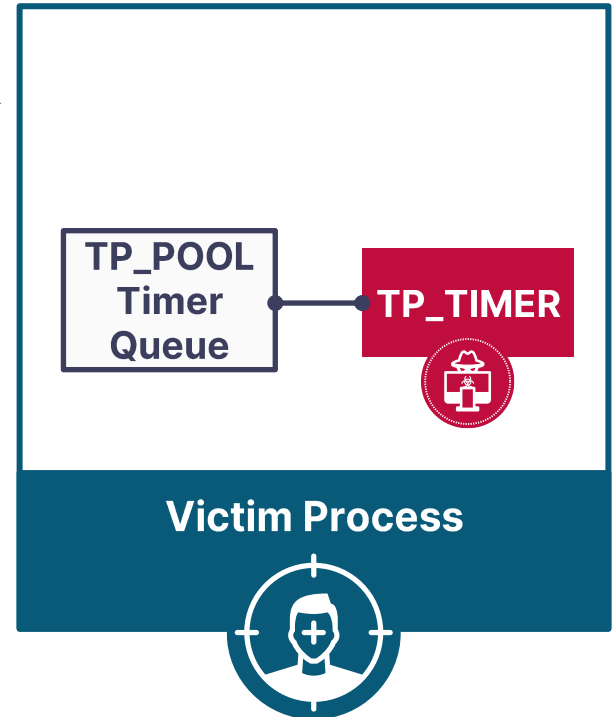
Insert  
TP\_TIMER to  
TP\_POOL  
timer queue



# Attacking Thread Pools – TP\_TIMER



Duplicate  
queue timer  
handle

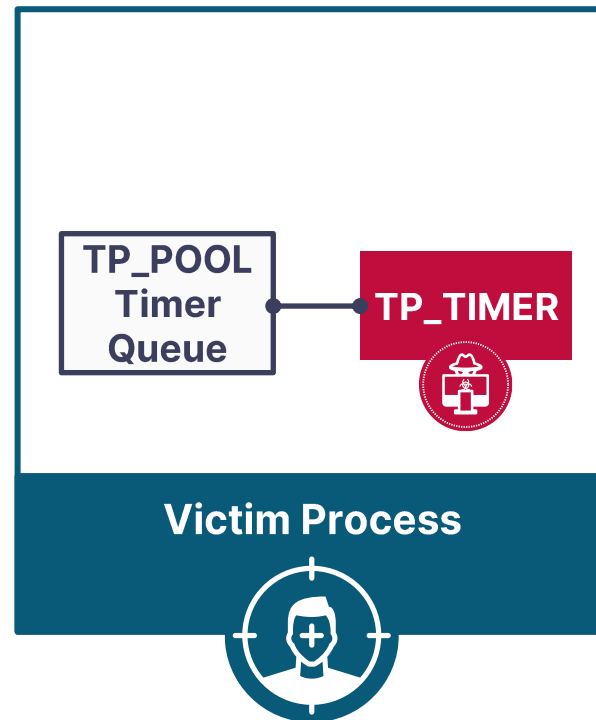




# Attacking Thread Pools – TP\_TIMER



Set queue  
timer to  
expire



## PoolParty State

One new friend in the pool



# Introducing PoolParty



# Introducing PoolParty – Supported Variants

- 1** Worker Factory Start Routine Overwrite
- 2** TP\_WORK Insertion
- 3** TP\_WAIT Insertion
- 4** TP\_IO Insertion
- 5** TP\_ALPC Insertion
- 6** TP\_JOB Insertion
- 7** TP\_DIRECT Insertion
- 8** TP\_TIMER Insertion

# Introducing PoolParty – Affected Products

Palo Alto Cortex



SentinelOne EDR



CrowdStrike Falcon



Microsoft Defender  
for Endpoint



Cybereason EDR



Figure 1: Magic Quadrant for Endpoint Protection Platforms



Source: Gartner (December 2022)

# Introducing PoolParty - Demo

The screenshot displays a Windows desktop environment. On the left, the Process Hacker application is open, showing a list of running processes. The 'explorer.exe' process is highlighted in pink, and several 'cmd.exe' processes are highlighted in yellow. Below the process list, system statistics are shown: CPU Usage: 1.80%, Physical memory: 3.14 GB (89.32%), and Processes: 157. An 'Untitled' text editor window is also open. On the right, a command prompt window is open, showing the current directory as 'C:\Users\Alon\Desktop>'. The Windows taskbar at the bottom shows the system tray with a temperature of 78°F, the date 12/3/2023, and the time 3:28 AM.

| Name                      | PID   | User name          | Description                       |
|---------------------------|-------|--------------------|-----------------------------------|
| svchost.exe               | 6408  |                    | Host Process for Windows Ser...   |
| ksass.exe                 | 1100  |                    | Local Security Authority Proce... |
| fontdrvhost.exe           | 1256  |                    | Usermode Font Driver Host         |
| csrss.exe                 | 708   |                    | Client Server Runtime Process     |
| winlogon.exe              | 832   |                    | Windows Logon Application         |
| fontdrvhost.exe           | 1248  |                    | Usermode Font Driver Host         |
| dmv.exe                   | 1460  |                    | Desktop Window Manager            |
| explorer.exe              | 3796  | ALON-DESKTOP2\Alon | Windows Explorer                  |
| ProcessHacker.exe         | 3876  | ALON-DESKTOP2\Alon | Process Hacker                    |
| cmd.exe                   | 6012  | ALON-DESKTOP2\Alon | Windows Command Processor         |
| conhost.exe               | 6640  | ALON-DESKTOP2\Alon | Console Window Host               |
| cmd.exe                   | 10032 | ALON-DESKTOP2\Alon | Windows Command Processor         |
| conhost.exe               | 1844  | ALON-DESKTOP2\Alon | Console Window Host               |
| Notepad.exe               | 6004  | ALON-DESKTOP2\Alon |                                   |
| GoogleCrashHandler.exe    | 7328  |                    | Google Crash Handler              |
| GoogleCrashHandler64.exe  | 7856  |                    | Google Crash Handler              |
| SecurityHealthSystray.exe | 1508  | ALON-DESKTOP2\Alon | Windows Security notification...  |
| vmtoolsd.exe              | 8744  | ALON-DESKTOP2\Alon | VMware Tools Core Service         |
| OneDrive.exe              | 8852  | ALON-DESKTOP2\Alon | Microsoft OneDrive                |



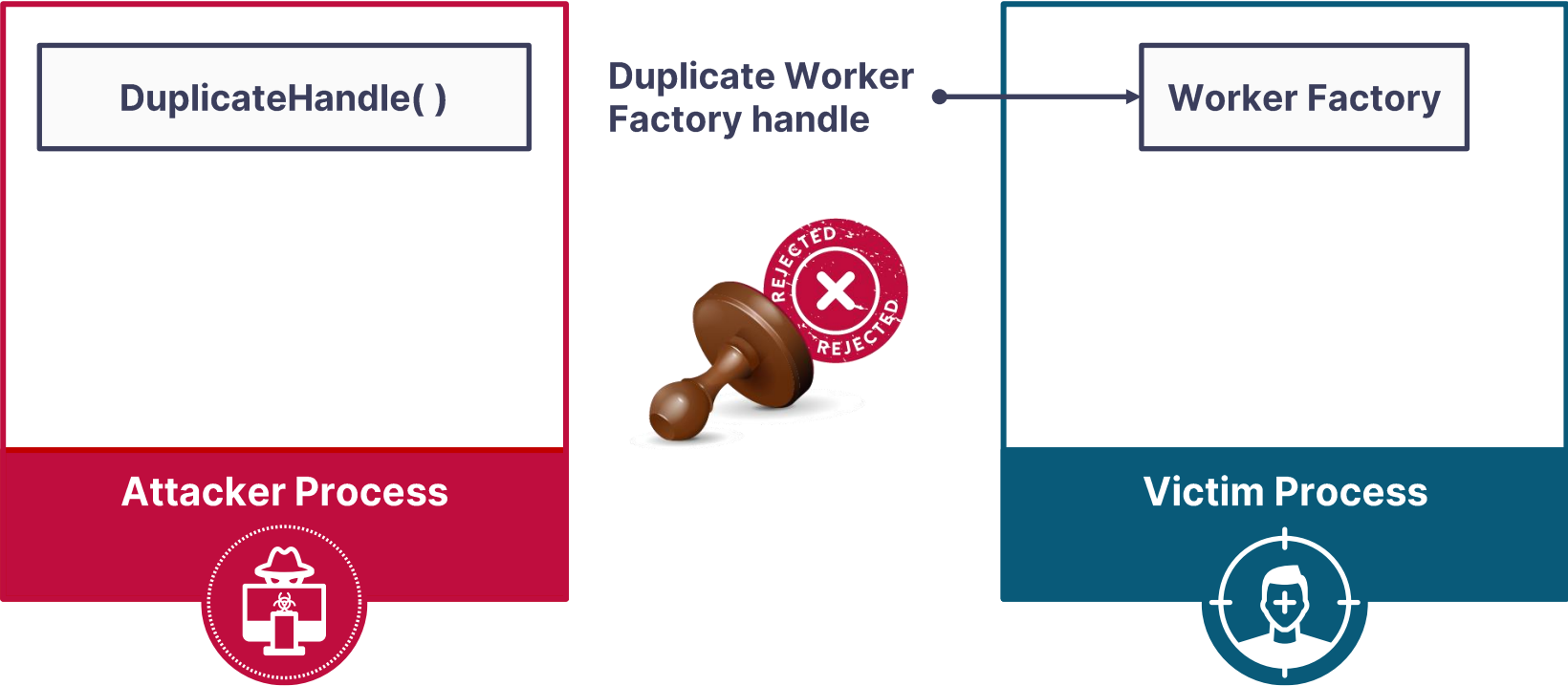
How it started



How it's going



# Initial Detection Against PoolParty



# Why The Worker Factory Is Needed?

---

Variant 1 – To query the start routine

---

Variant 2 – To get the task queue

---

Variants 3-7 – To get the I/O completion queue handle value

---

Variant 8 – To get the timer queue

---

# Avoiding Worker Factory Duplication For Variants 3-7

---

All Windows processes have a thread pool by default

---

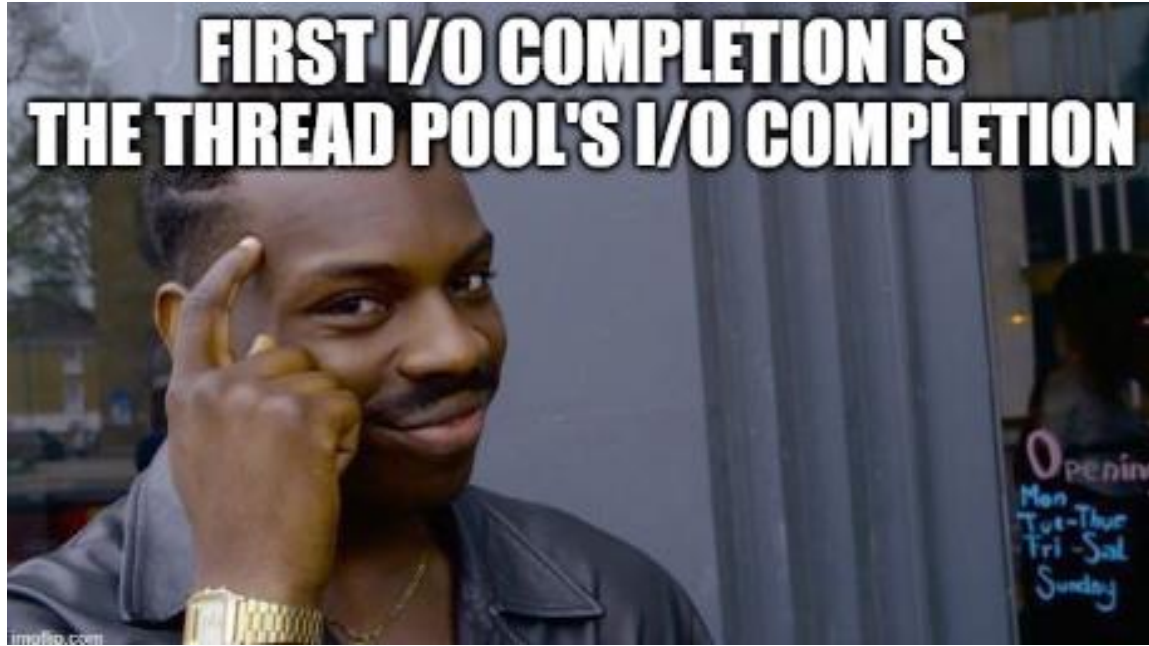
The thread pool is created before the main function is invoked

---

Handle values are sequential

---

# Avoiding Worker Factory Duplication For Variants 3-7



# Process Injection Implications



# Process Injection Implications – Evasive Credential Dumping

The image displays a Windows desktop environment with two primary windows open. The background window is the Windows Task Manager, showing a list of running processes. The foreground window is a File Explorer window titled 'Dumps', which is currently empty.

**Task Manager Process List:**

| Process             | CPU      | Private Bytes | Working Set | PID     | Description                   | Company Name          | Protection |
|---------------------|----------|---------------|-------------|---------|-------------------------------|-----------------------|------------|
| Winlogon.exe        | 9.684 K  | 18,572 K      | 5092        |         |                               |                       |            |
| Winlogon.exe        | 27,500 K | 31,126 K      | 7056        |         |                               |                       |            |
| WMIADAP.exe         | <0.01    | 2,100 K       | 8,204 K     | 7472    |                               |                       |            |
| svchost.exe         | 2,536 K  | 11,396 K      | 944         |         |                               |                       |            |
| svchost.exe         | 1,624 K  | 7,356 K       | 244         |         |                               |                       |            |
| Wdgate.exe          | 2,664 K  | 13,864 K      | 7348        |         | Microsoft Corporation         |                       |            |
| vmtoolsd.exe        | +0.01    | 9,676 K       | 21,544 K    | 3648    | VMware Tools Core Service     | VMware, Inc.          |            |
| vmtoolsd.exe        | +0.01    | 28,112 K      | 47,624 K    | 4872    | VMware Tools Core Service     | VMware, Inc.          |            |
| vmtoolsd.exe        | 1,572 K  | 7,128 K       | 3628        | 3628    | VMware SVGA Helper Service    | VMware, Inc.          |            |
| vmtoolsd.exe        | 1,664 K  | 7,580 K       | 3700        |         |                               |                       |            |
| VMToolsdService.exe | 3,024 K  | 11,972 K      | 3956        | 3956    | VMware Guest Authentication   | VMware, Inc.          |            |
| lsass.exe           | <0.01    | 26,776 K      | 27,704 K    | 3308    |                               |                       |            |
| lsass.exe           | <0.01    | 26,772 K      | 27,704 K    | 3436    |                               |                       |            |
| lsass.exe           | <0.01    | 26,768 K      | 27,704 K    | 3484    |                               |                       |            |
| lsass.exe           | <0.01    | 26,772 K      | 27,708 K    | 8680    |                               |                       |            |
| lsass.exe           | <0.01    | 26,772 K      | 27,704 K    | 8964    |                               |                       |            |
| lsass.exe           | <0.01    | 26,766 K      | 27,704 K    | 9000    |                               |                       |            |
| lsass.exe           | <0.01    | 26,776 K      | 27,704 K    | 9172    |                               |                       |            |
| lsass.exe           | <0.01    | 26,768 K      | 27,708 K    | 3324    |                               |                       |            |
| lsass.exe           | 7,140 K  | 18,632 K      | 6280        | 6280    | Host Process for Windows T... | Microsoft Corporation |            |
| lsass.exe           | 1,828 K  | 8,208 K       | 8308        |         |                               |                       |            |
| System Idle Process | 97.73    | 60 K          | 0 K         | 0       |                               |                       |            |
| System              | 0.18     | 52 K          | 156 K       | 4       |                               |                       |            |
| svchost.exe         | 18,764 K | 77,872 K      | 168,160 K   | 168,160 | Host Process for Windows T... | Microsoft Corporation |            |

**File Explorer Dumps Folder:**

The File Explorer window shows the 'Dumps' folder, which is currently empty. The interface includes a search bar and navigation pane on the left.

**Taskbar:**

The Windows taskbar at the bottom shows the search bar, system tray icons, and the date/time: 10:47 AM, 11/21/2023.

# Process Injection Implications – Controlled Folder Access Bypass

The screenshot displays a Windows 11 desktop environment with several open windows:

- File Explorer:** Shows the 'Documents' folder containing files such as 'background\_checks', 'bank\_accounts', 'budget\_spreadsheets.pptb', 'competitive\_analysis.xlsx', 'confidential\_memos', 'contracts', 'customer\_data.pptb', 'employee\_files', 'intellectual\_property', and 'legal\_documents'.
- Process Explorer:** Lists running processes with columns for CPU, Private Bytes, Working Set, PID, Description, and Company Name. Key processes include:

| Process                  | CPU      | Private Bytes | Working Set | PID  | Description                   | Company Name          |
|--------------------------|----------|---------------|-------------|------|-------------------------------|-----------------------|
| System Idle Process      | 37.88    | 92 K          | 5 K         | 0    |                               |                       |
| System                   | <0.01    | 40 K          | 144 K       | 4    |                               |                       |
| System Idle Process      | <0.01    | 8 K           | 6 K         | 8    | Hardware Interrupts and DPCs  |                       |
| smss.exe                 | 1.088 K  | 1,164 K       | 392         |      |                               |                       |
| svchost.exe              | 1.172 K  | 33,304 K      | 1954        |      |                               |                       |
| csrss.exe                | 1.960 K  | 5,436 K       | 576         |      |                               |                       |
| class.exe                | <0.01    | 1,932 K       | 5,308 K     | 648  |                               |                       |
| wininit.exe              | 1.492 K  | 6,796 K       | 656         |      |                               |                       |
| services.exe             | 5.036 K  | 19,026 K      | 768         |      |                               |                       |
| svchost.exe              | 8.908 K  | 25,820 K      | 920         |      | Host Process for Windows S... | Microsoft Corporation |
| VimPro-SE.exe            | 8.529 K  | 20,272 K      | 5040        |      |                               |                       |
| FlattenBaker.exe         | 9.716 K  | 43,204 K      | 540         |      | FlattenBaker                  | Microsoft Corporation |
| FlattenBaker.exe         | 5.968 K  | 25,704 K      | 878         |      | FlattenBaker                  | Microsoft Corporation |
| ghost.exe                | 5.160 K  | 18,300 K      | 5104        |      | COM Surrogate                 | Microsoft Corporation |
| lsass.exe                | 15.920 K | 45,876 K      | 3728        |      | Local System                  | Microsoft Corporation |
| FlattenBaker.exe         | 7.840 K  | 28,856 K      | 7244        |      | FlattenBaker                  | Microsoft Corporation |
| VimPro-SE.exe            | 25.572 K | 32,876 K      | 7364        |      |                               |                       |
| smss.exe                 | 2.504 K  | 3,520 K       | 8124        |      | Windows Defender SmarSc...    | Microsoft Corporation |
| FlattenBaker.exe         | 3.496 K  | 17,608 K      | 1044        |      | FlattenBaker                  | Microsoft Corporation |
| ApplicationFrameHost.exe | 6.004 K  | 29,796 K      | 8607        |      | Application Frame Host        | Microsoft Corporation |
| MicrosoftHost.exe        | Sup...   | 20,520 K      | 47,356 K    | 3004 |                               | Microsoft Corporation |
| SearchHost.exe           | Sup...   | 144,800 K     | 227,300 K   | 8292 |                               | Microsoft Corporation |
| System Idle Process      | 26.524 K | 73,040 K      | 3000        |      | Windows Start Experience H... | Microsoft Corporation |
| Windows.exe              | 7.444 K  | 17,796 K      | 1000        |      |                               | Microsoft Corporation |
- Windows Defender:** Shows a dark screen, likely indicating a security alert or scan in progress.

# Takeaways





# Takeaways

---

We need a generic detection approach for process injections

---

The impact of process injections is larger than we thought

---

Enhance your focus on detecting anomalies rather than placing complete trust in processes based solely on their identity

---



# PoolParty GitHub + Q&A

<https://github.com/SafeBreach-Labs/PoolParty>



**@\_0xDeku**



**<https://il.linkedin.com/in/alonleviev>**



**[alon.leviev@safebreach.com](mailto:alon.leviev@safebreach.com)**

---

