

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



Alon Leviev



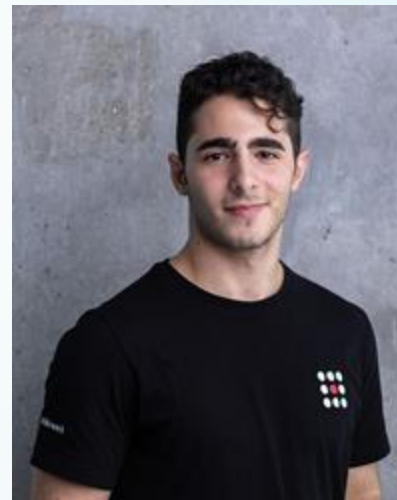
Security Researcher at SafeBreach

21 years old

Self-taught

OS internals, reverse engineering and vulnerability research

Former BJJ world and european champion



Agenda

Process Injection Background

Research Motivation & Questions

Detection Approach

Research Goals

User-mode Thread Pool Deep Dive

Introducing PoolParty

Process Injection Implications

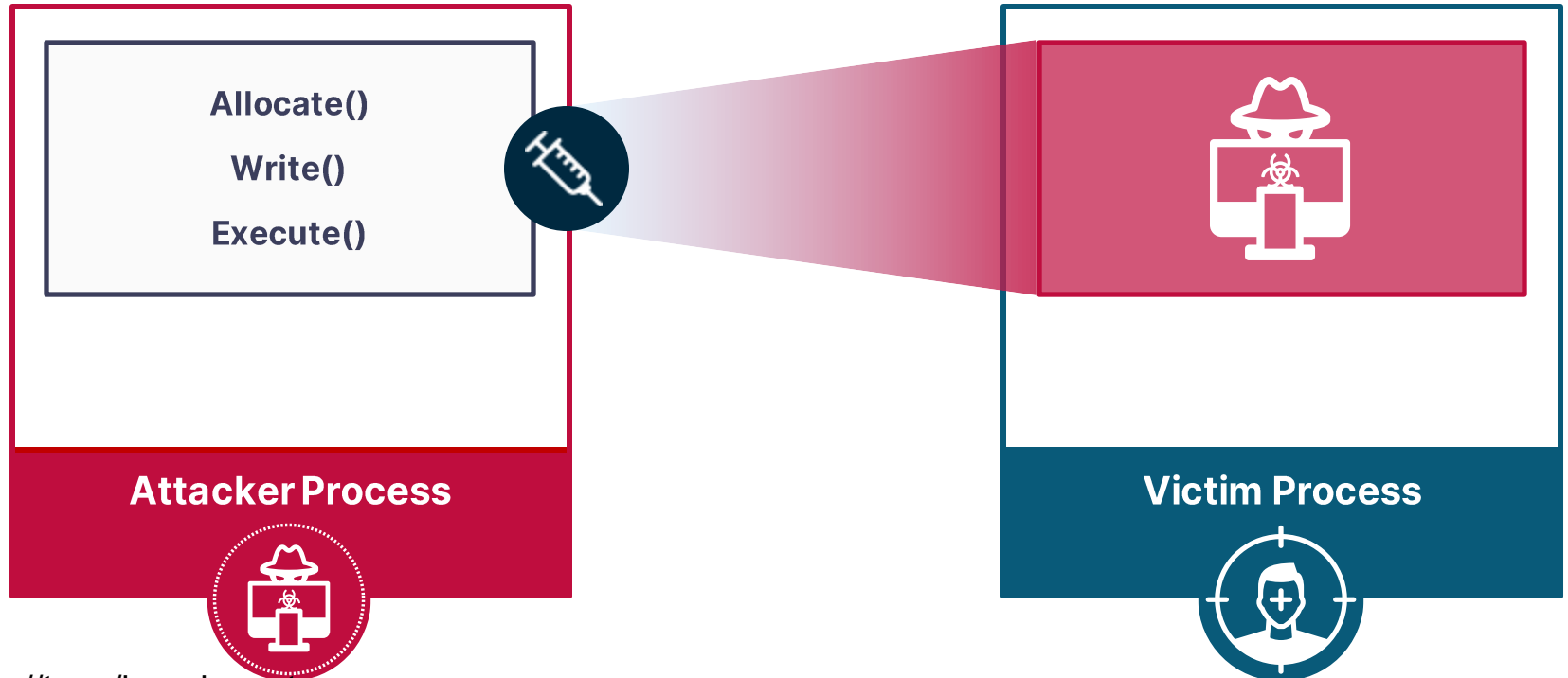
Takeaways



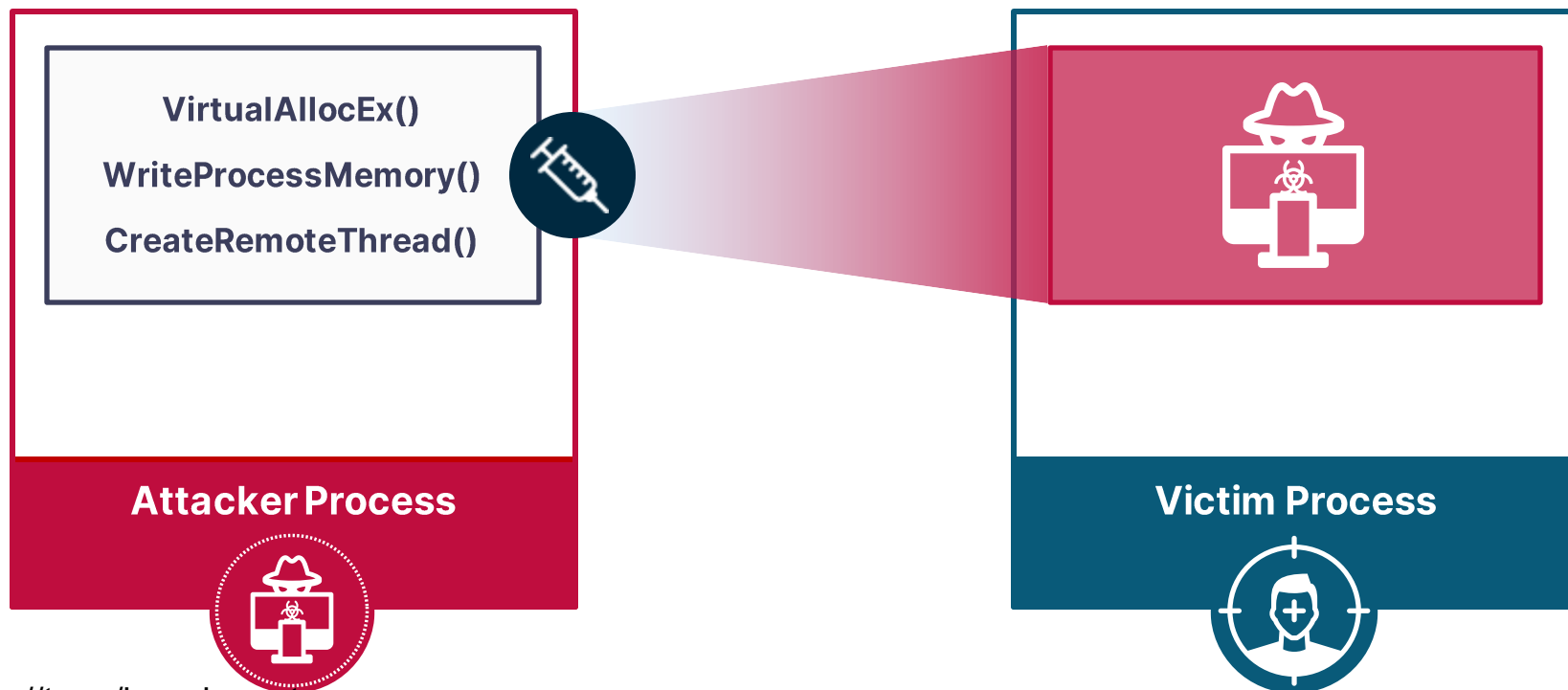
Process Injection Background



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?

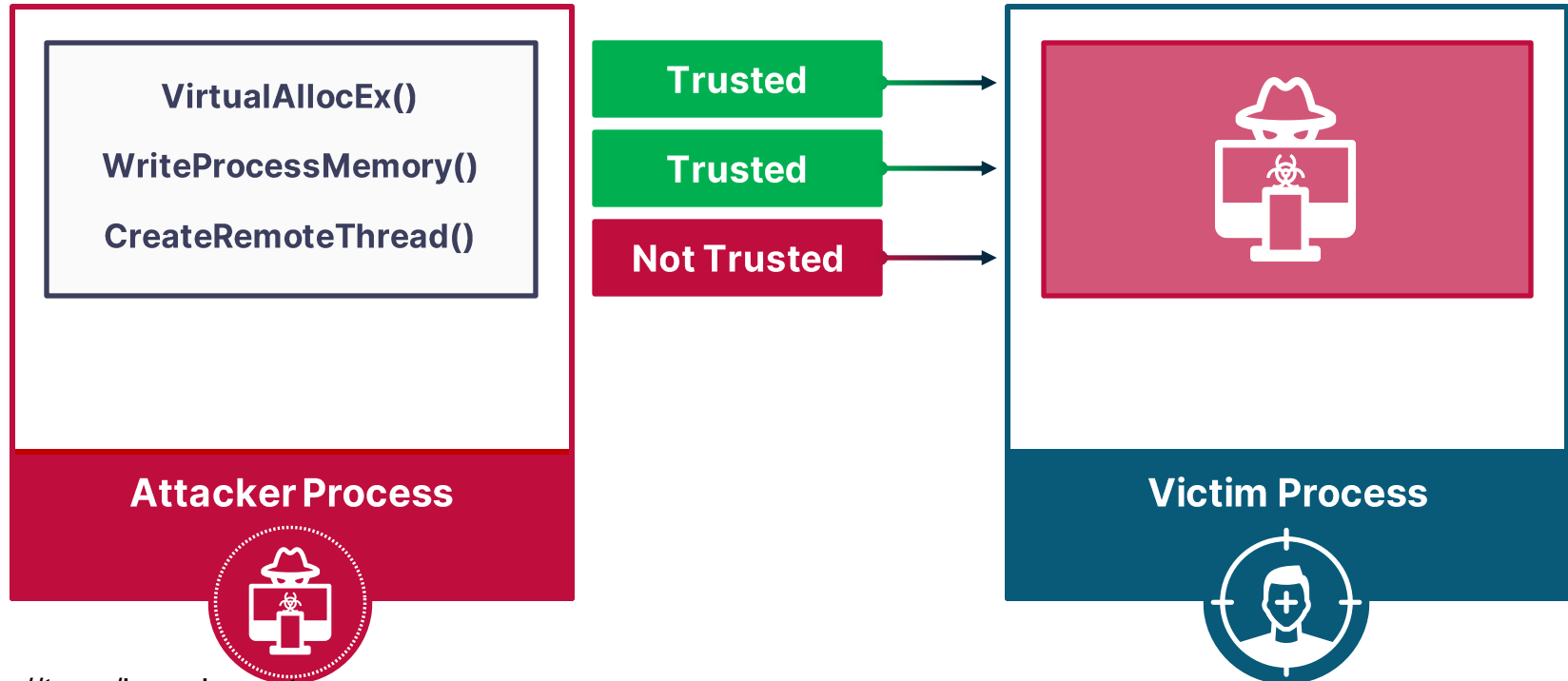
Is the current detection approach generic enough?



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 primitive is disguised as a legitimate action?



What Is a Thread Pool?

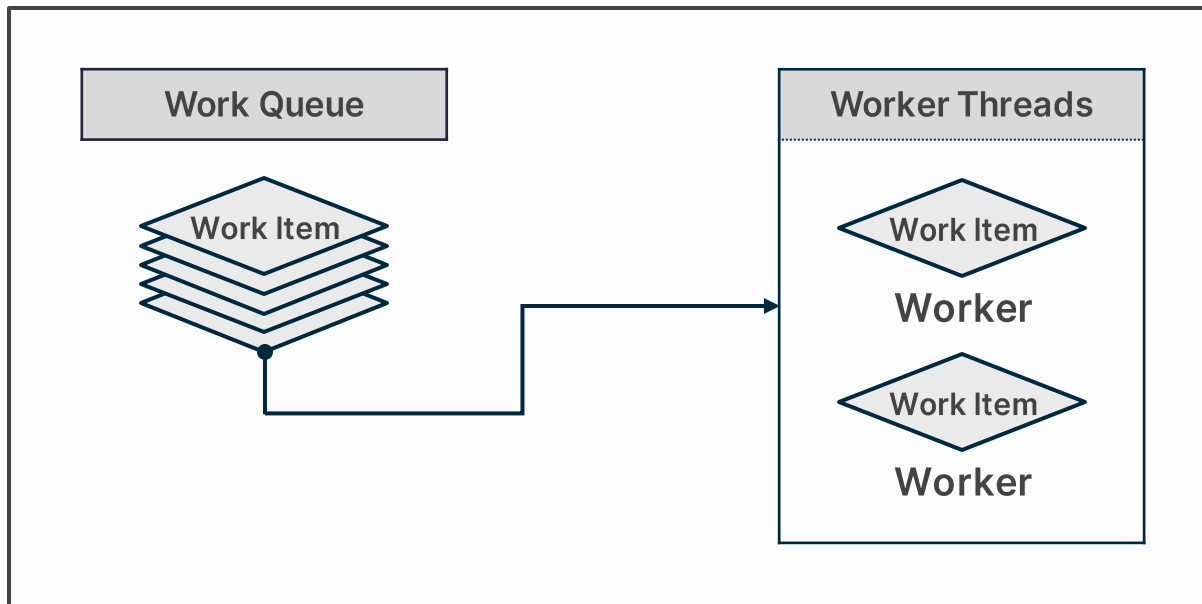
I wish these boxes could be sent in parallel



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

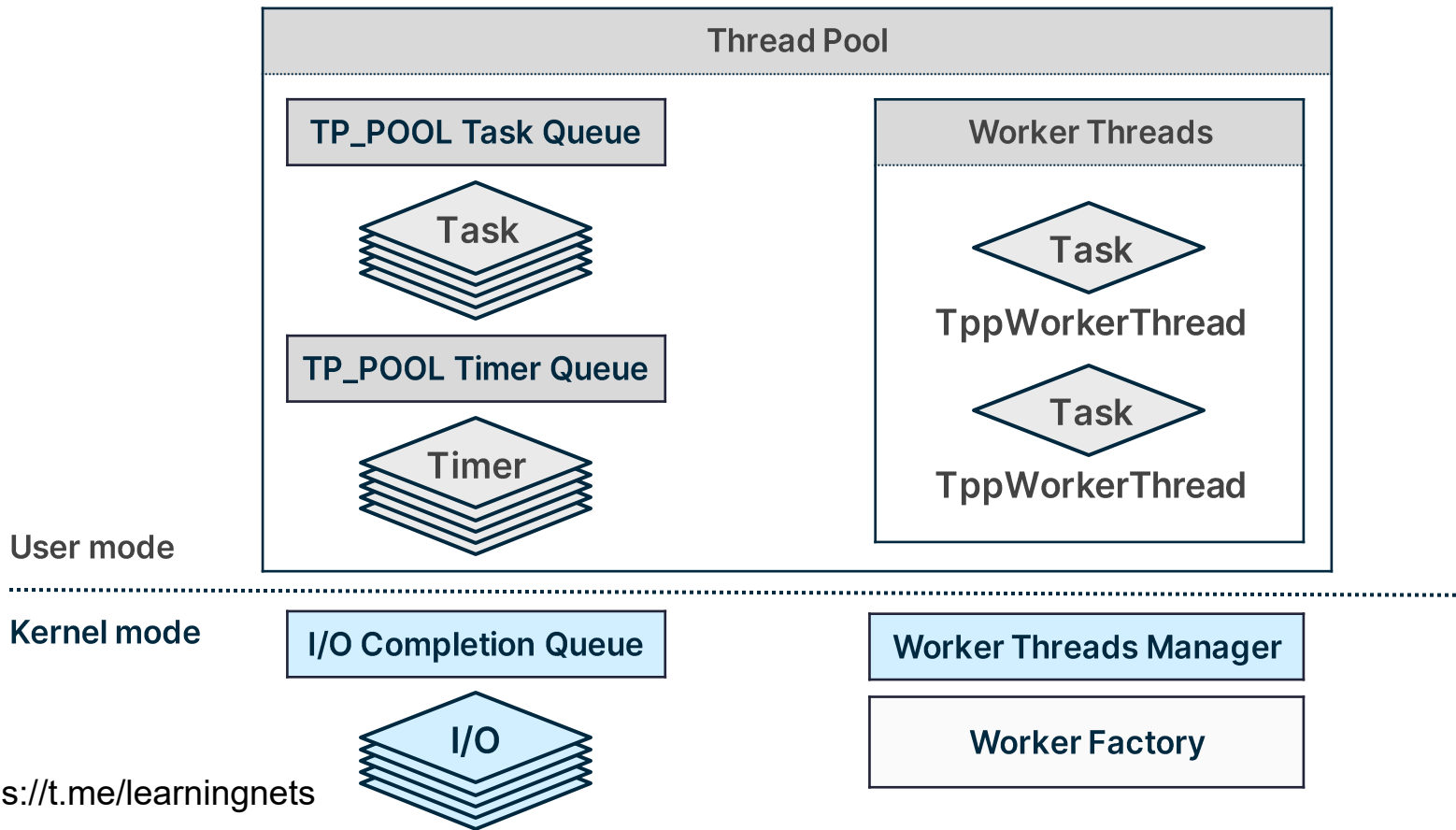
Multiple work item types are supported



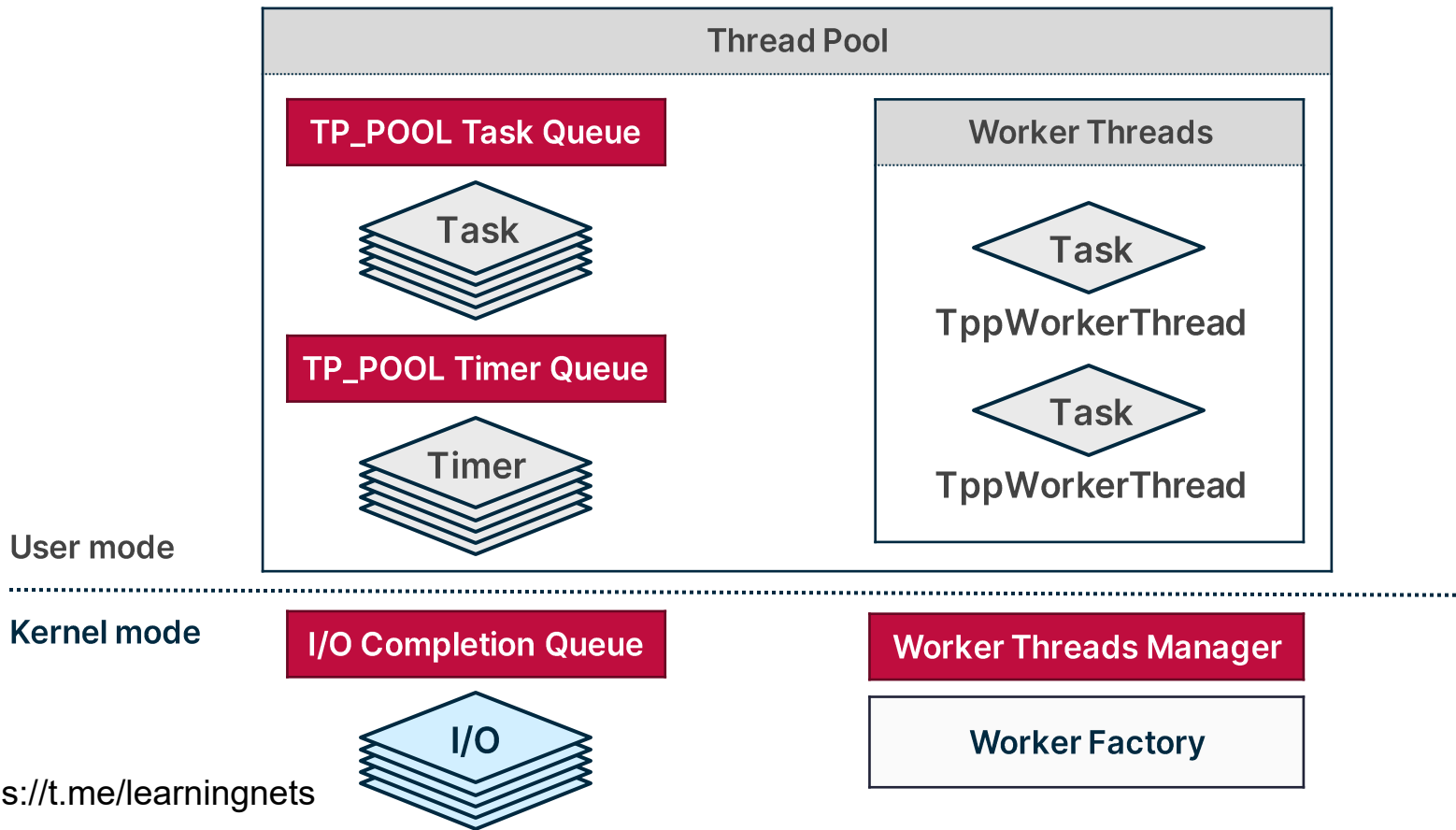
User-Mode Thread Pool Deep Dive



User-Mode Thread Pool Architecture



Defining Attack Surface



PoolParty State

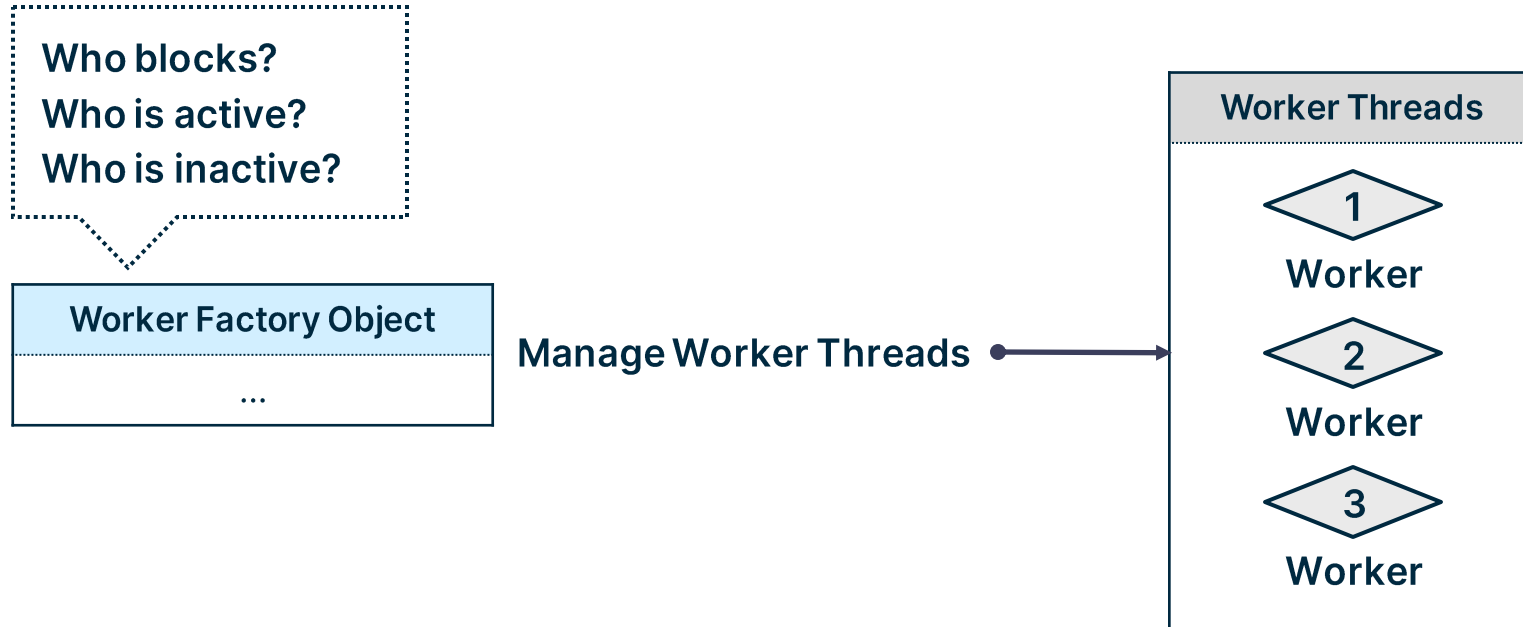
No friends in the pool



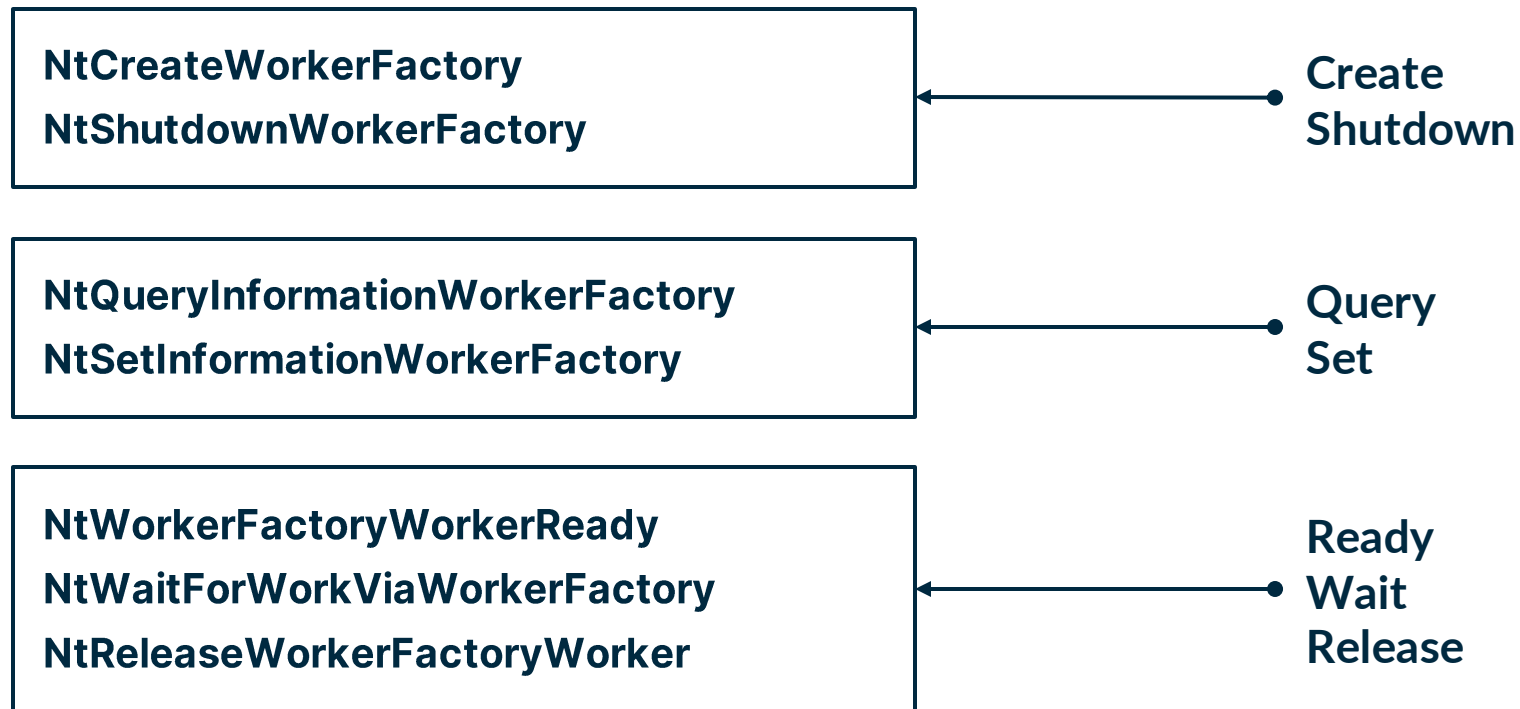
Attacking Worker Factories



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

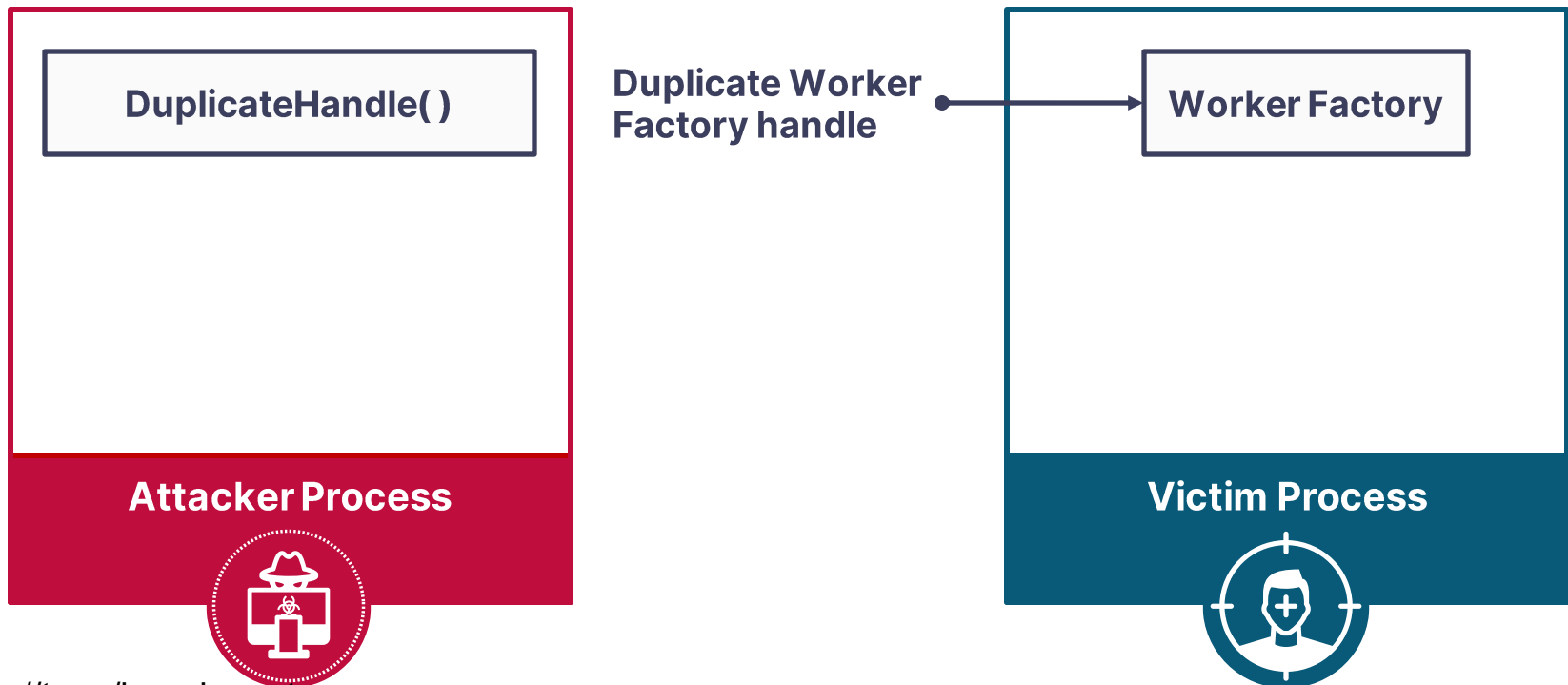
```
NTSTATUS NTAPI NtCreateWorkerFactory(..., HANDLE WorkerProcessHandle, ...)
{
    [snip]

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

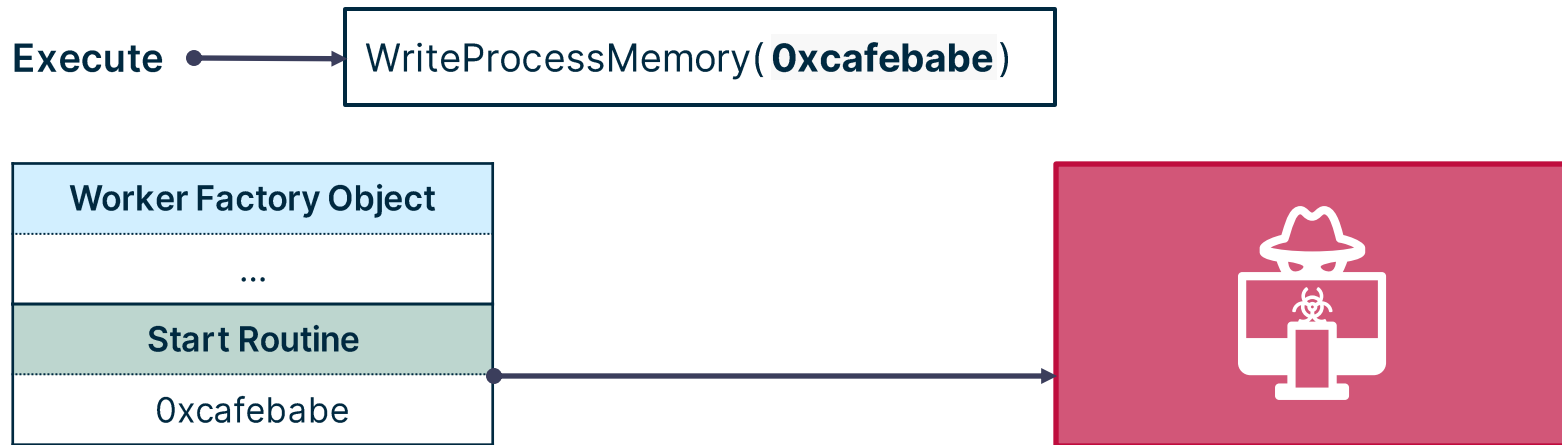
    if ( KeGetCurrentThread()->ApcState.Process != pWorkerProcessObject)
    {
        return STATUS_INVALID_PARAMETER;
    }

    [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,  
);
```

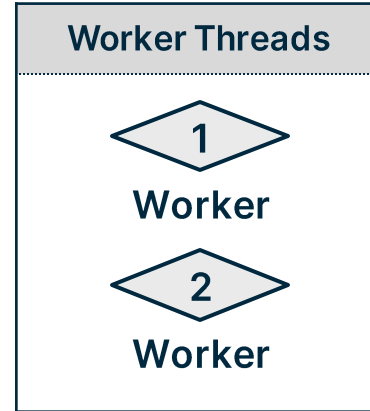
Attacking Worker Factories

```
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;
```

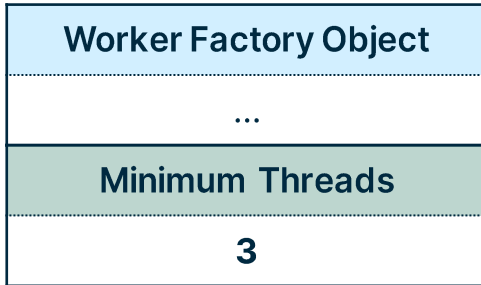
Attacking Worker Factories

Execute  `NtSetInformationWorkerFactory(Running Threads Num + 1)`

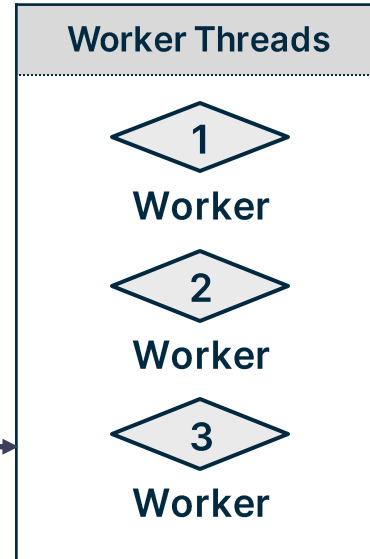
Worker Factory Object
...
Minimum Threads
2



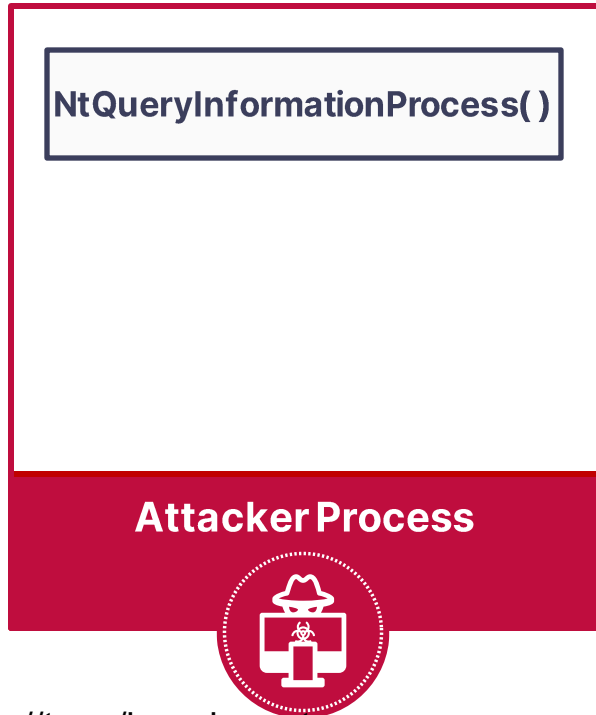
Attacking Worker Factories



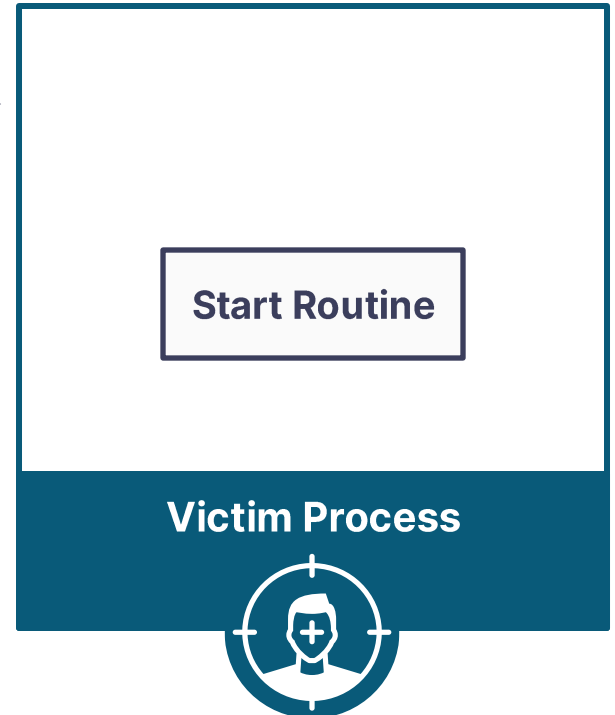
Create new worker thread →



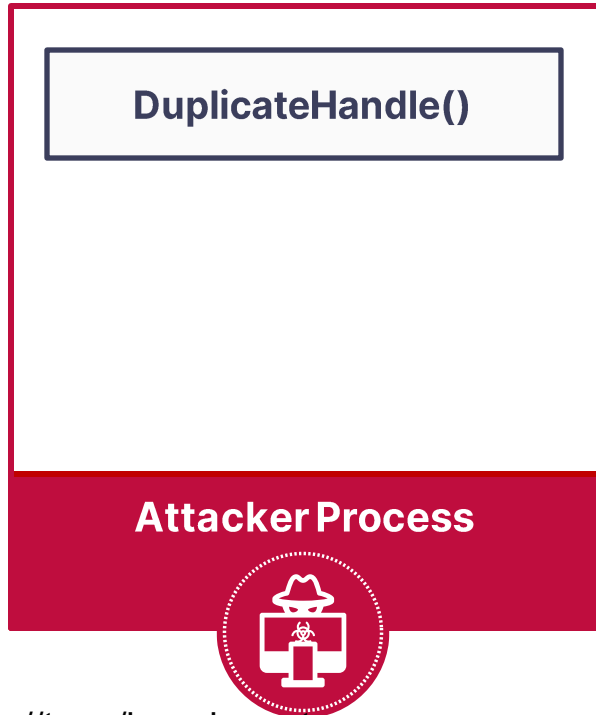
Attacking Worker Factories



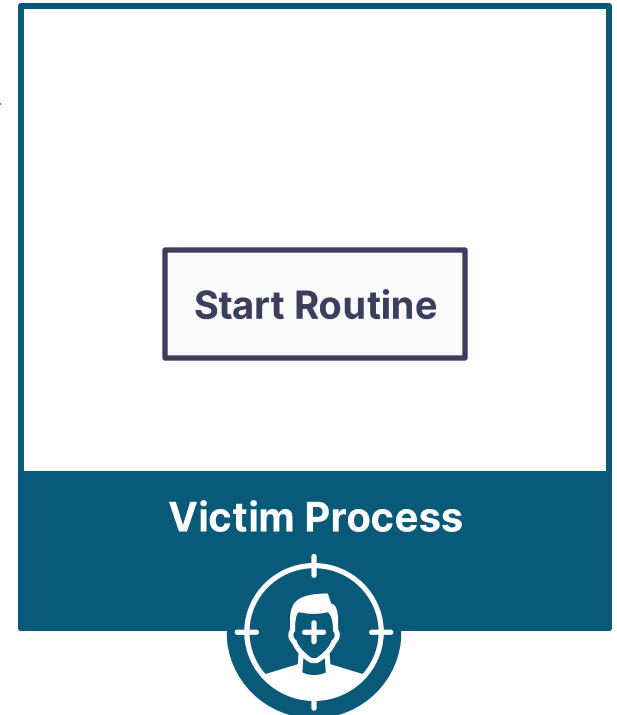
Get handle table →



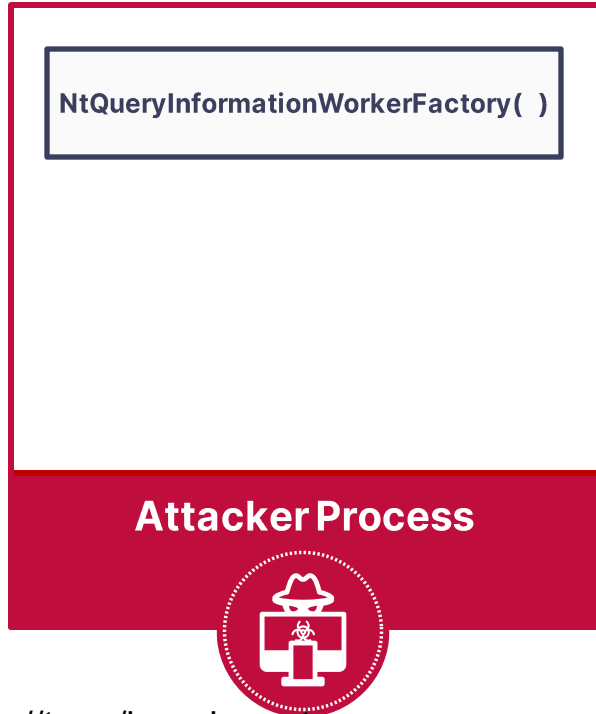
Attacking Worker Factories



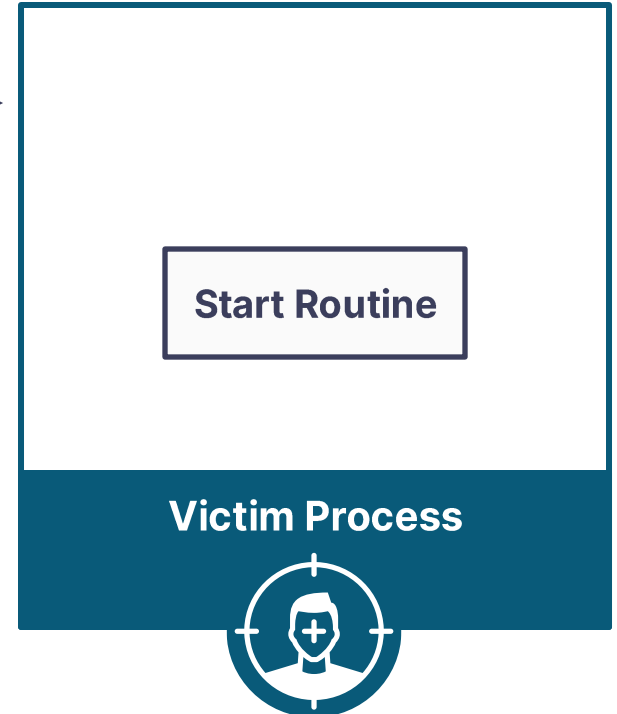
Duplicate
Worker Factory
handle →



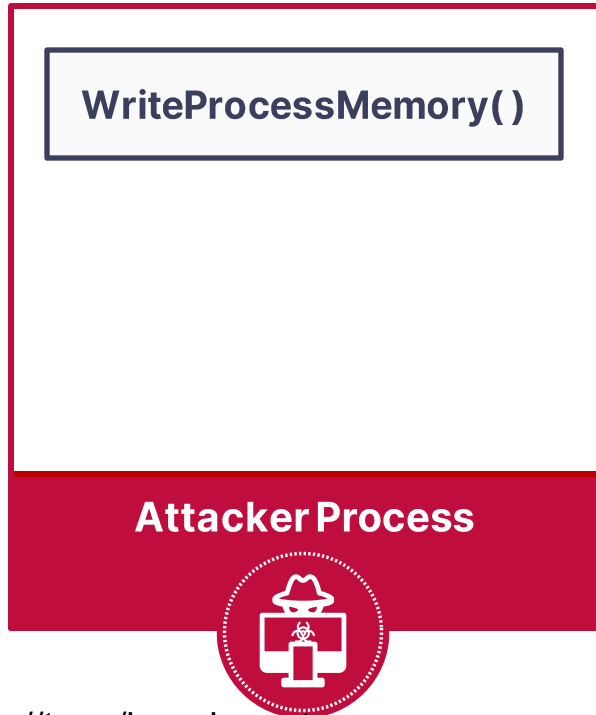
Attacking Worker Factories



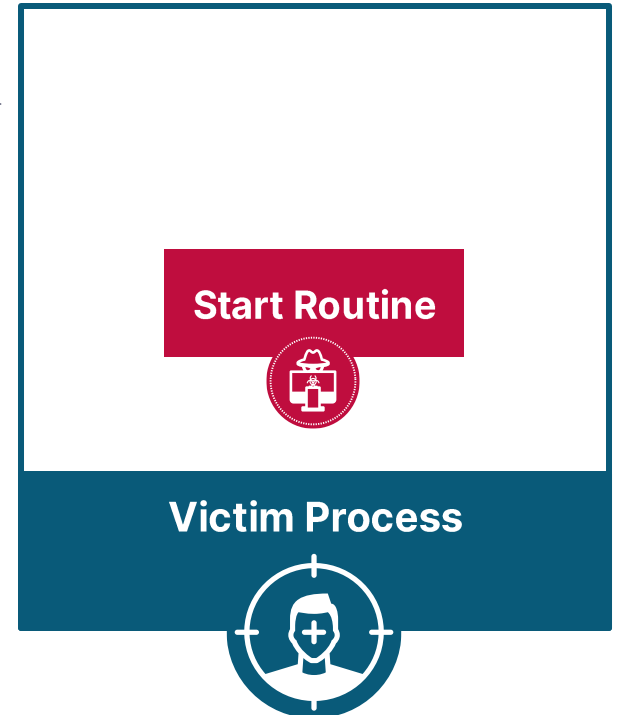
Get Worker
Factory info



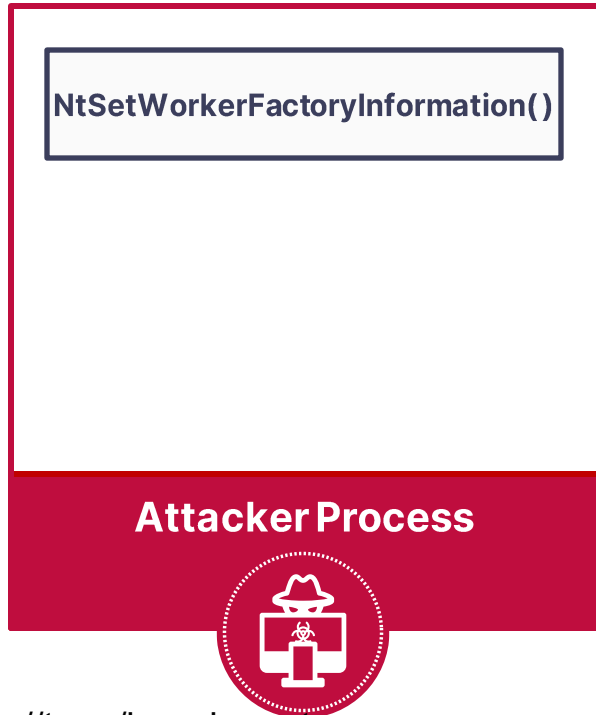
Attacking Worker Factories



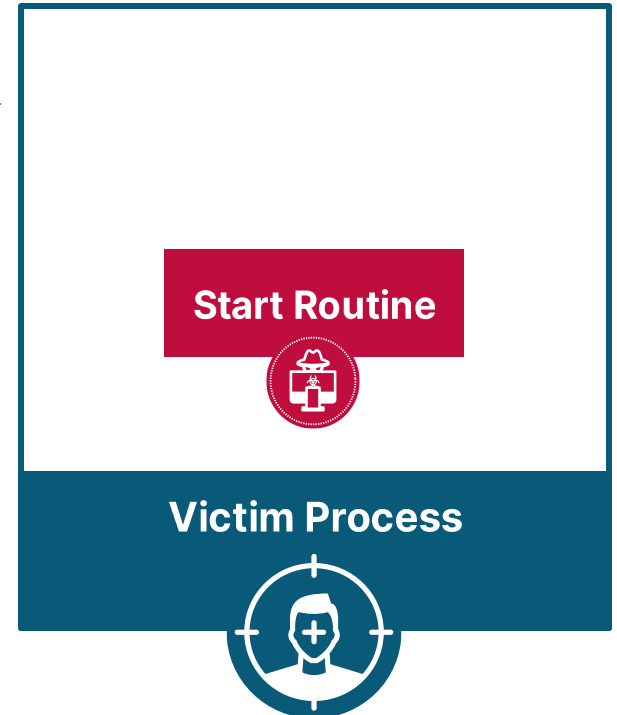
Write shellcode to start routine →



Attacking Worker Factories



Increase worker
factory minimum threads →



PoolParty State

First friend in the pool



Attacking Thread Pools



Why Thread Pool?

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

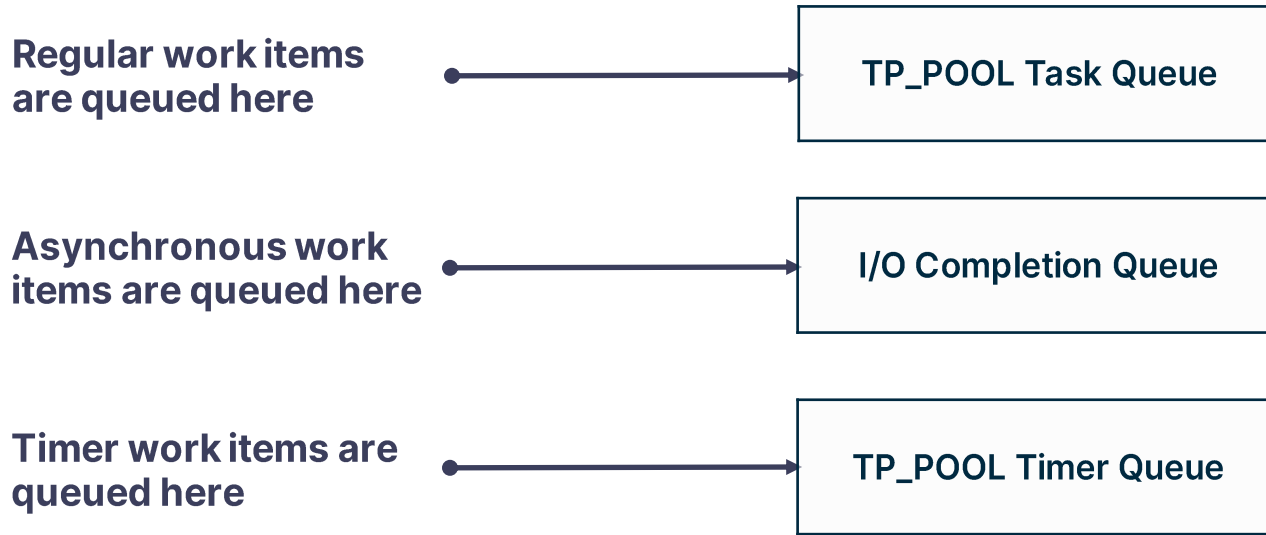
TP_JOB

TP_ALPC

Timer Work Items

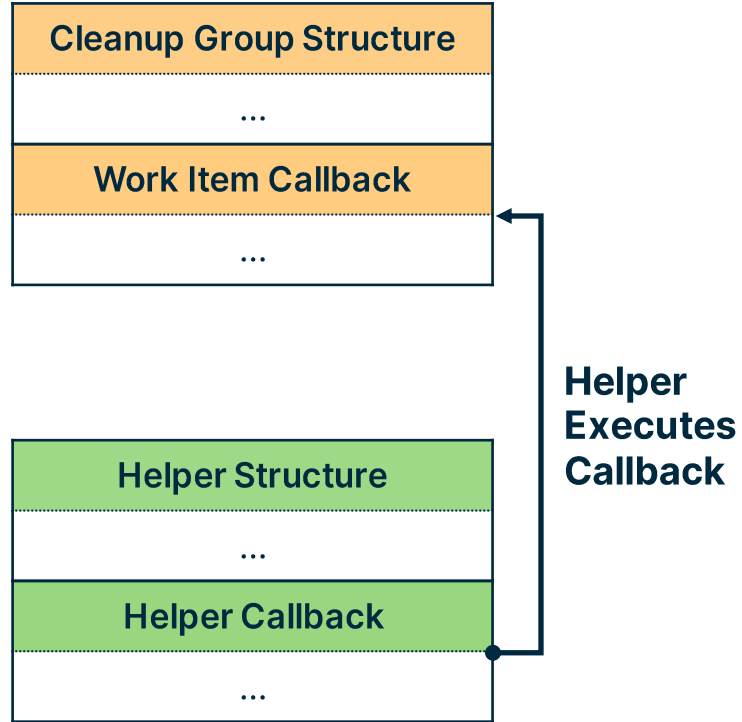
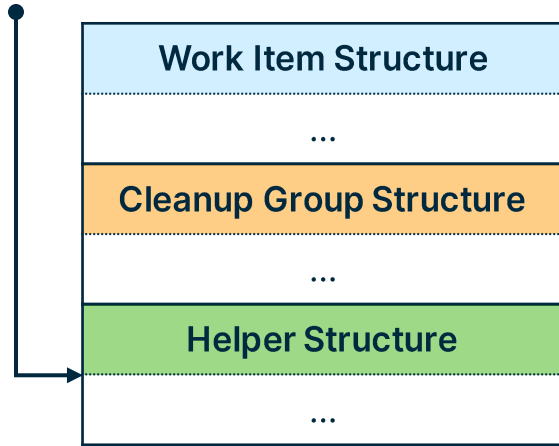
TP_TIMER

Attacking Thread Pools - Queue Types



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

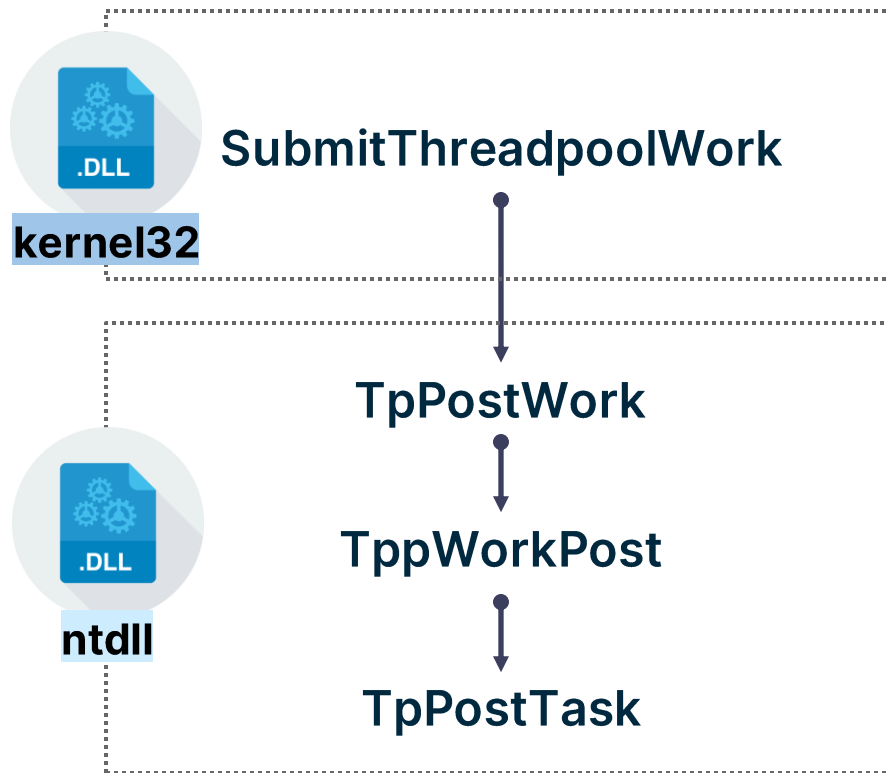
Attacking Thread Pools - TP_WORK

```
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



Attacking Thread Pools - TP_WORK

Ntdll::TpPostTask

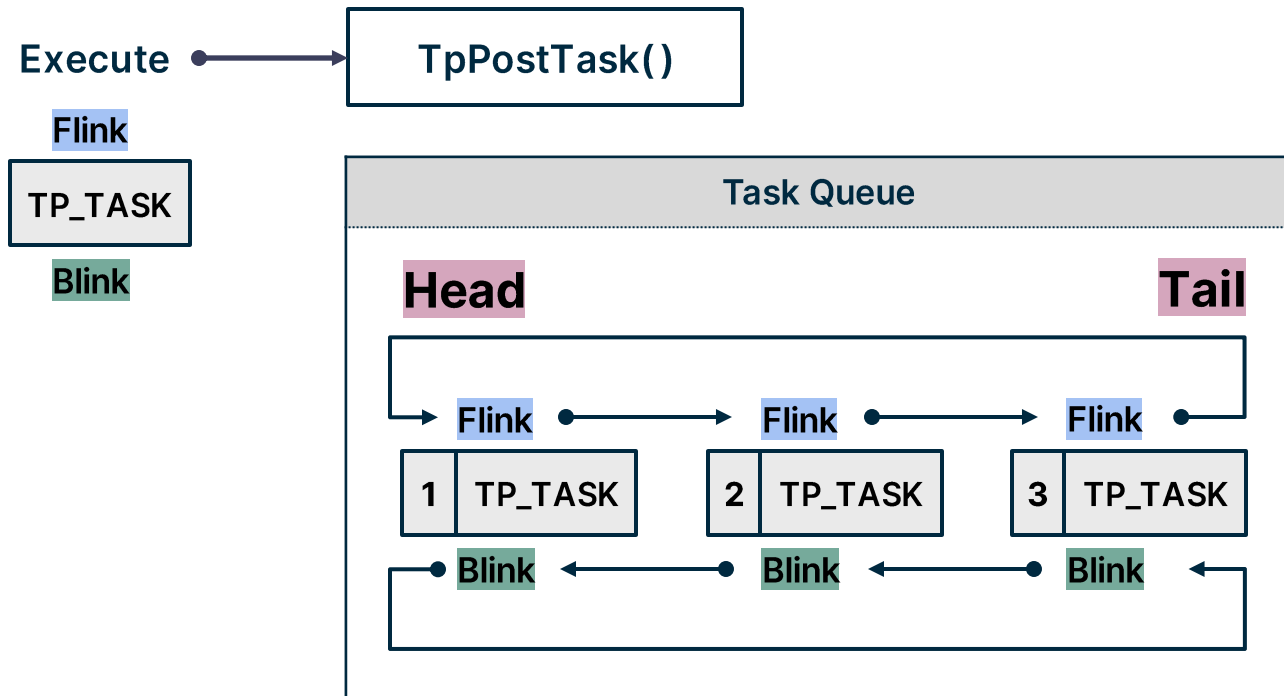
```
NTSTATUS NTAPI TpPostTask(TP_TASK* TpTask, TP_POOL* TpPool, int CallbackPriority, ...)
{
    [snip]

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

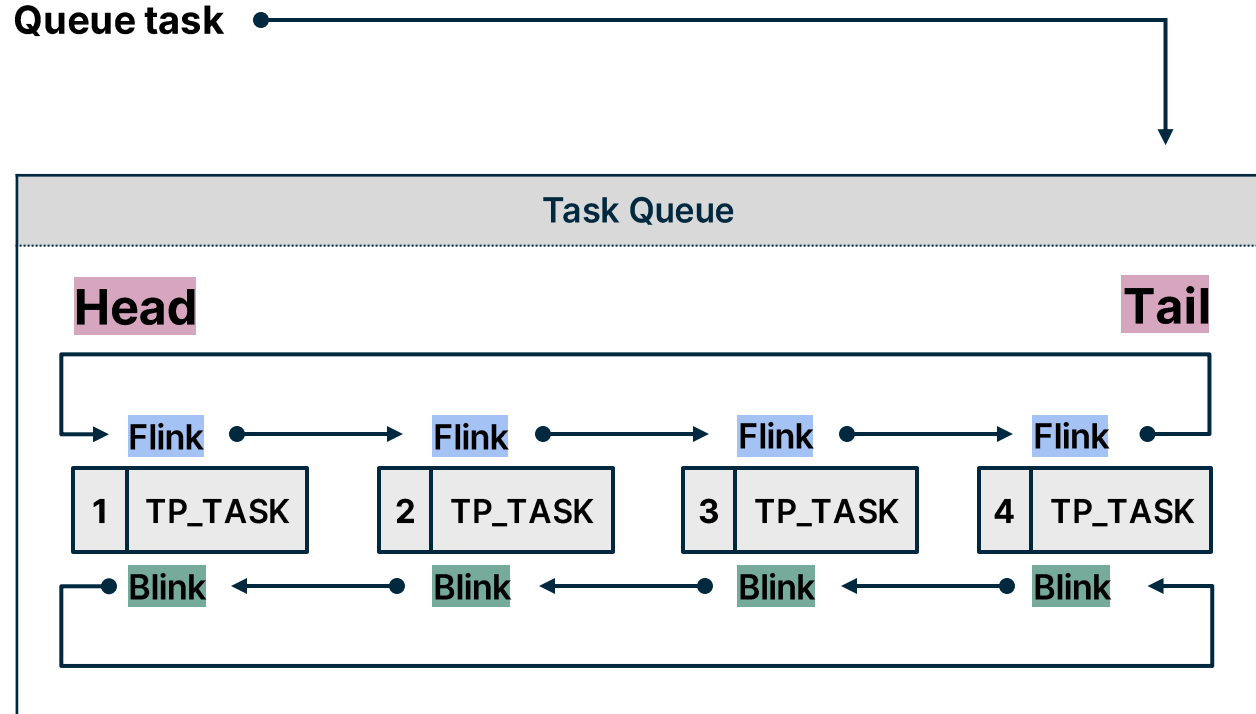
    InsertTailList(&TaskQueue->Queue, &TpTask->ListEntry);

    [snip]
}
```

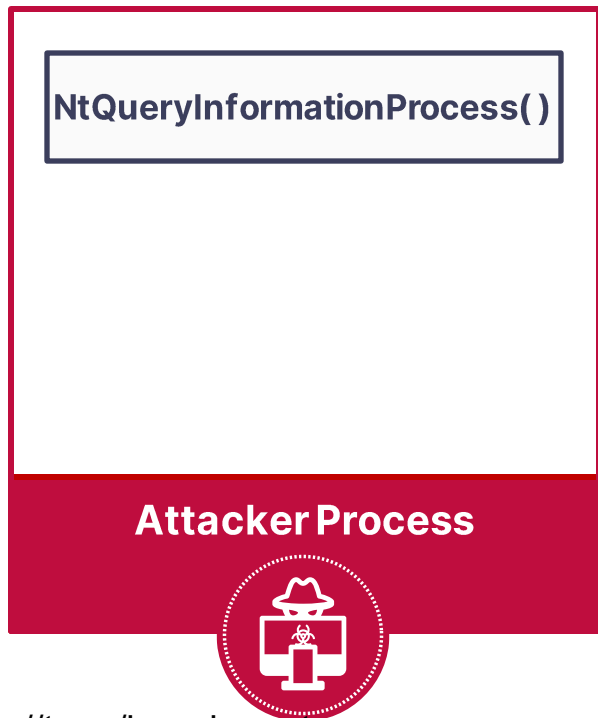
Attacking Thread Pools - TP_WORK



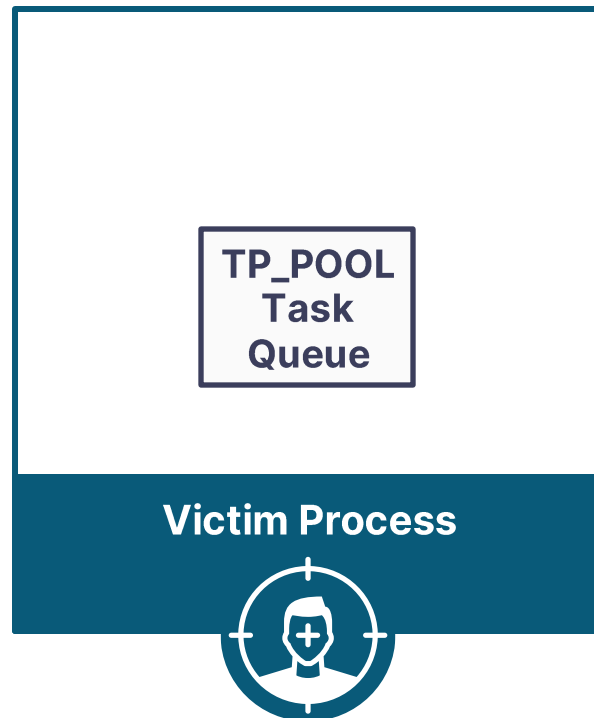
Attacking Thread Pools - TP_WORK



Attacking Thread Pools – TP_WORK



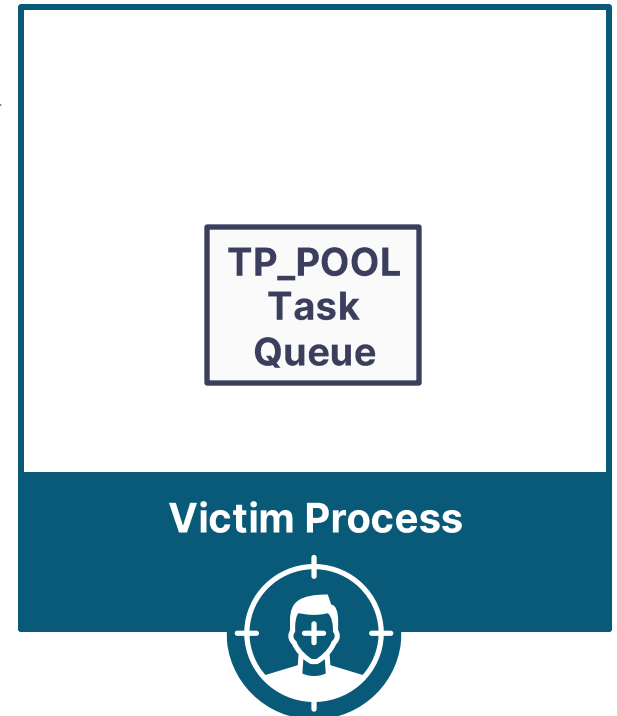
Get handle table →



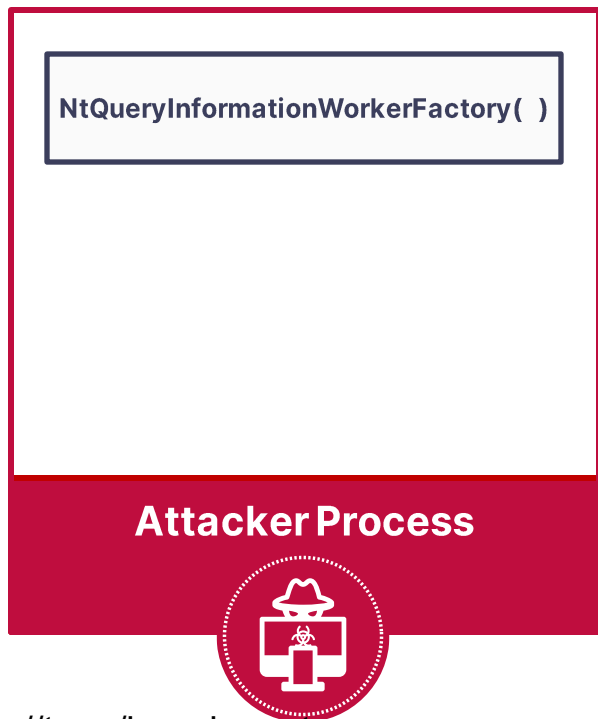
Attacking Thread Pools – TP_WORK



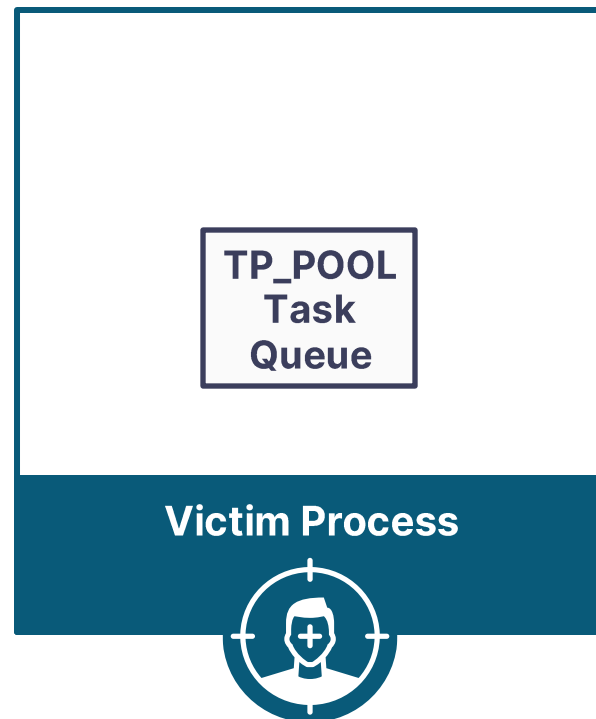
Duplicate
Worker Factory
handle →



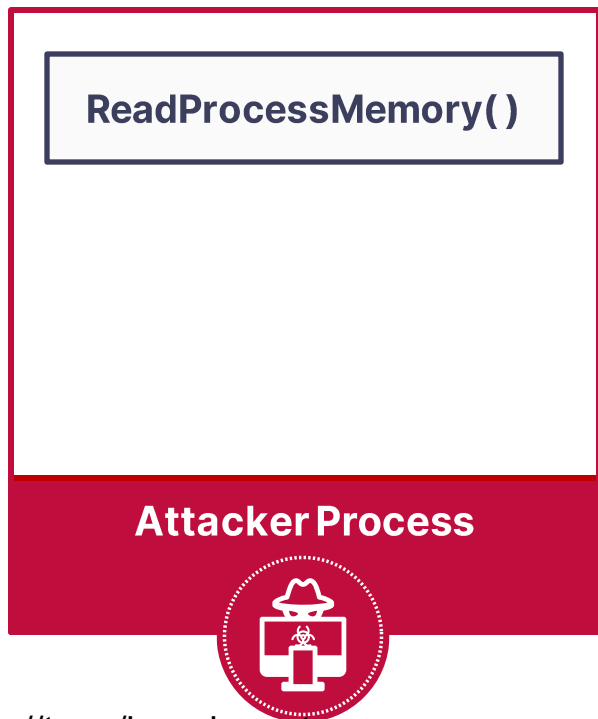
Attacking Thread Pools – TP_WORK



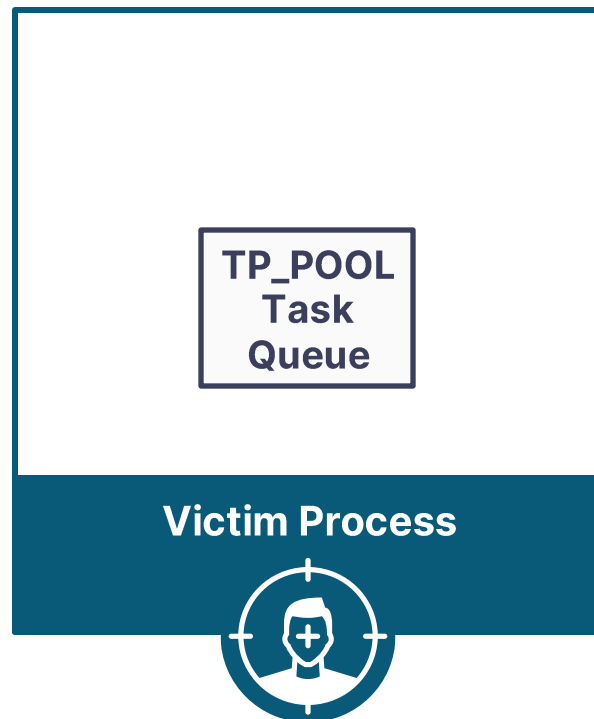
Get Worker
Factory info



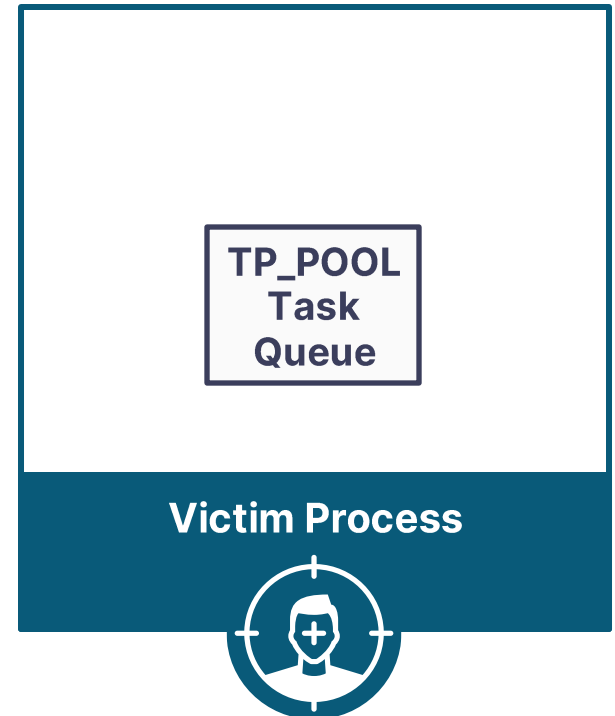
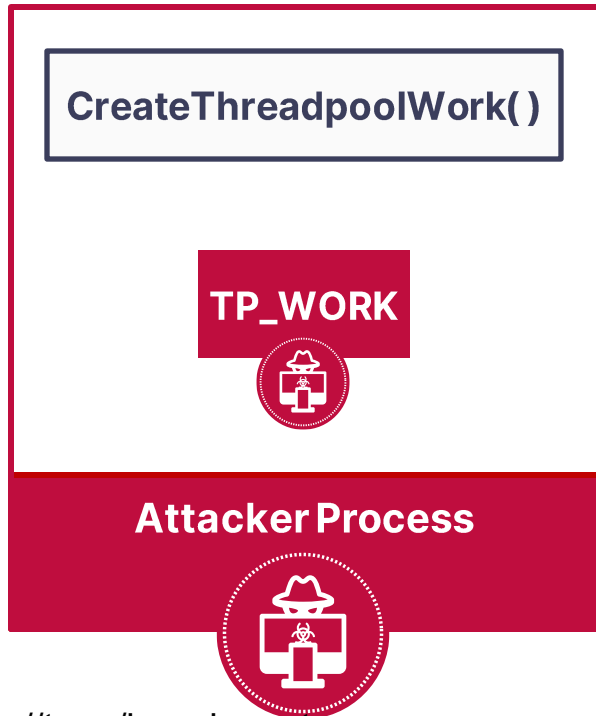
Attacking Thread Pools – TP_WORK



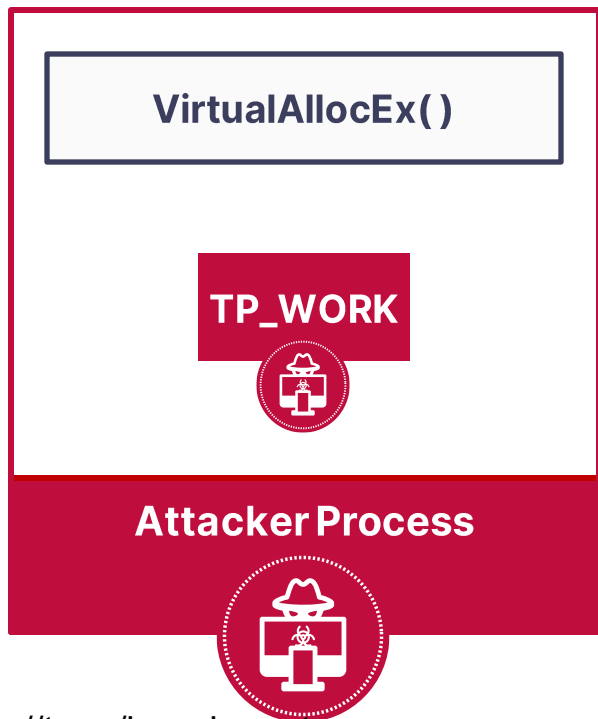
Read TP_POOL →



Attacking Thread Pools – TP_WORK



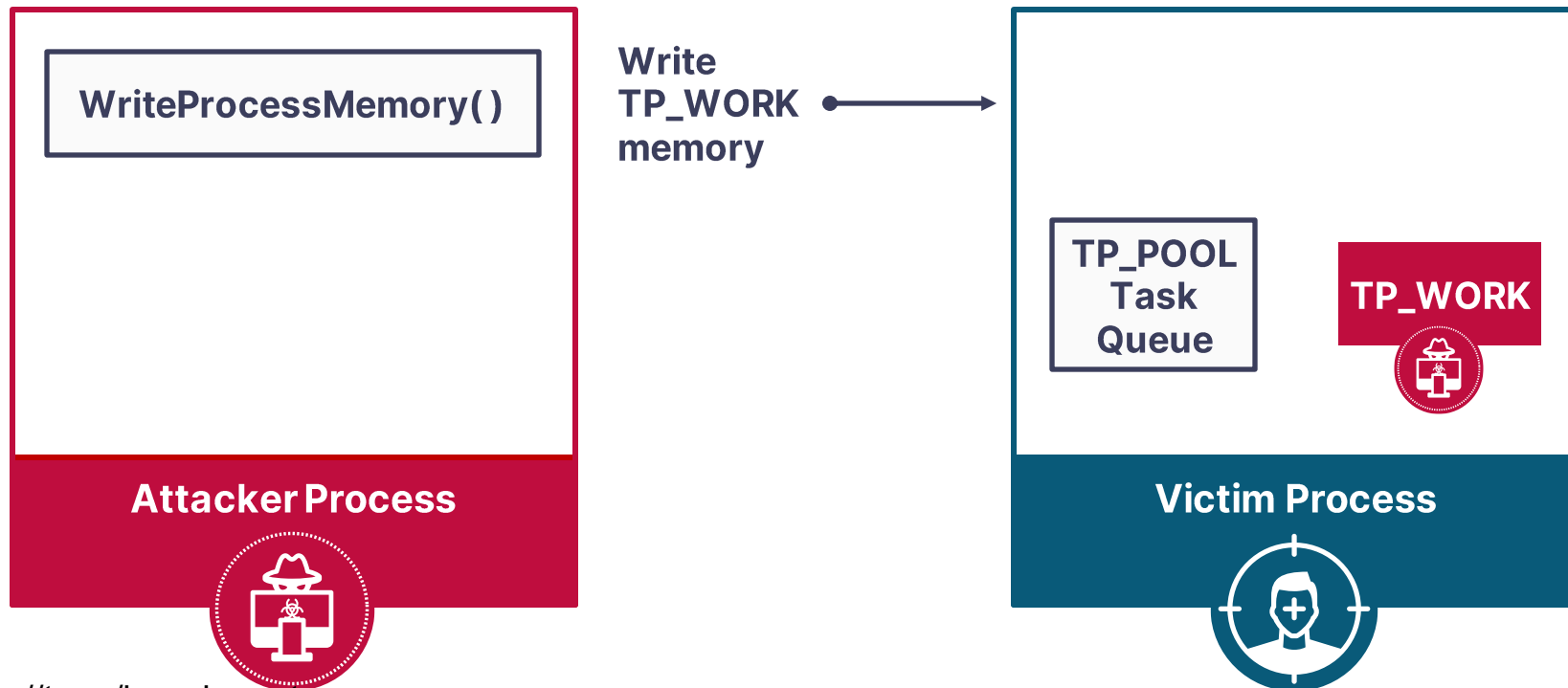
Attacking Thread Pools – TP_WORK



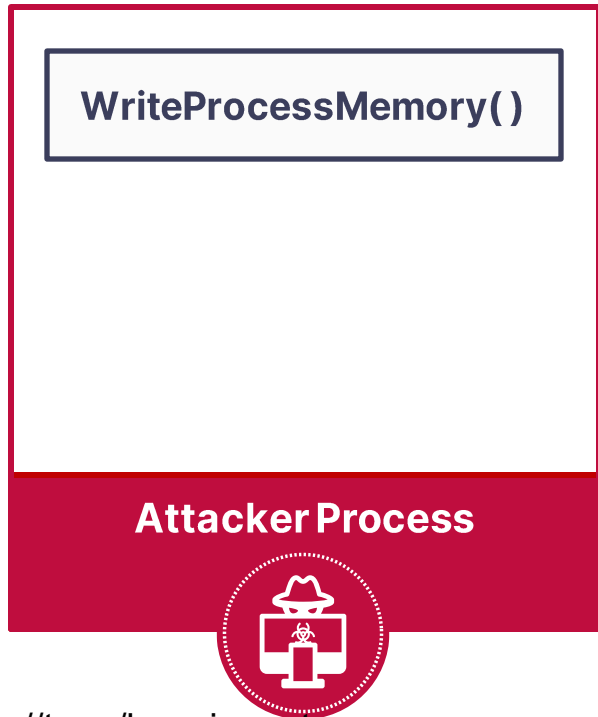
Allocate
TP_WORK
memory



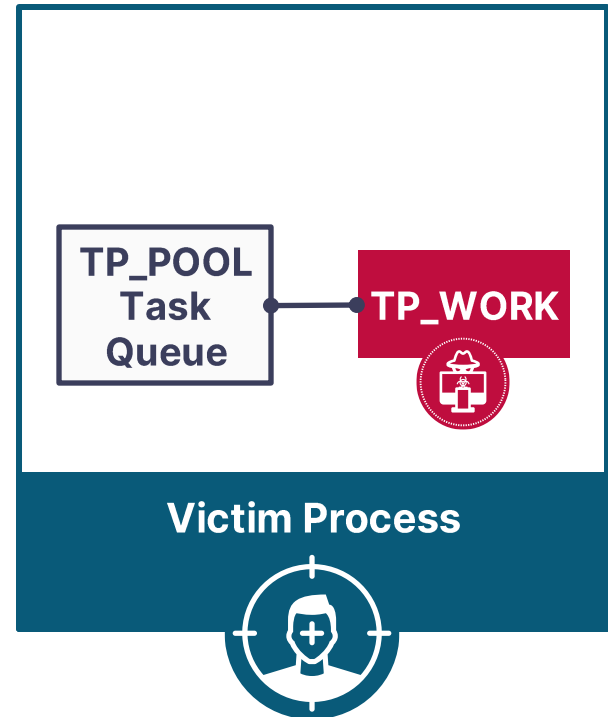
Attacking Thread Pools – TP_WORK



Attacking Thread Pools – TP_WORK



Insert
TP_WORK to
TP_POOL
task queue

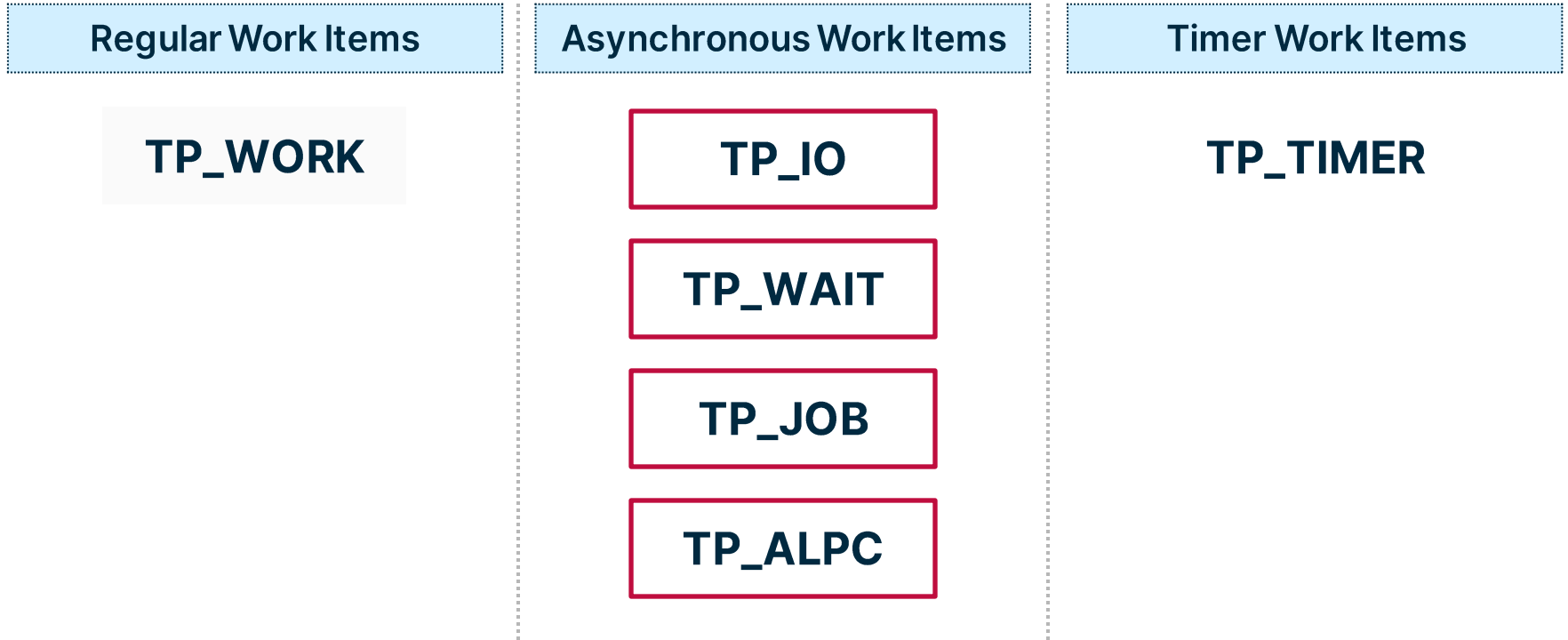


PoolParty State

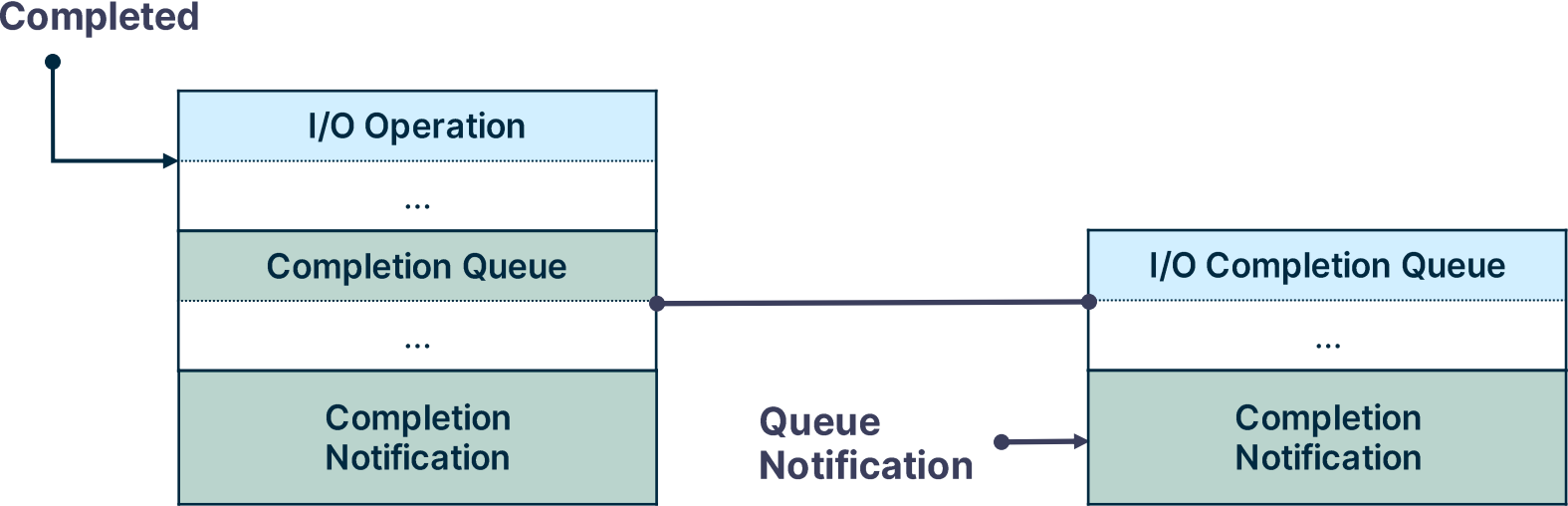
Second friend in the pool



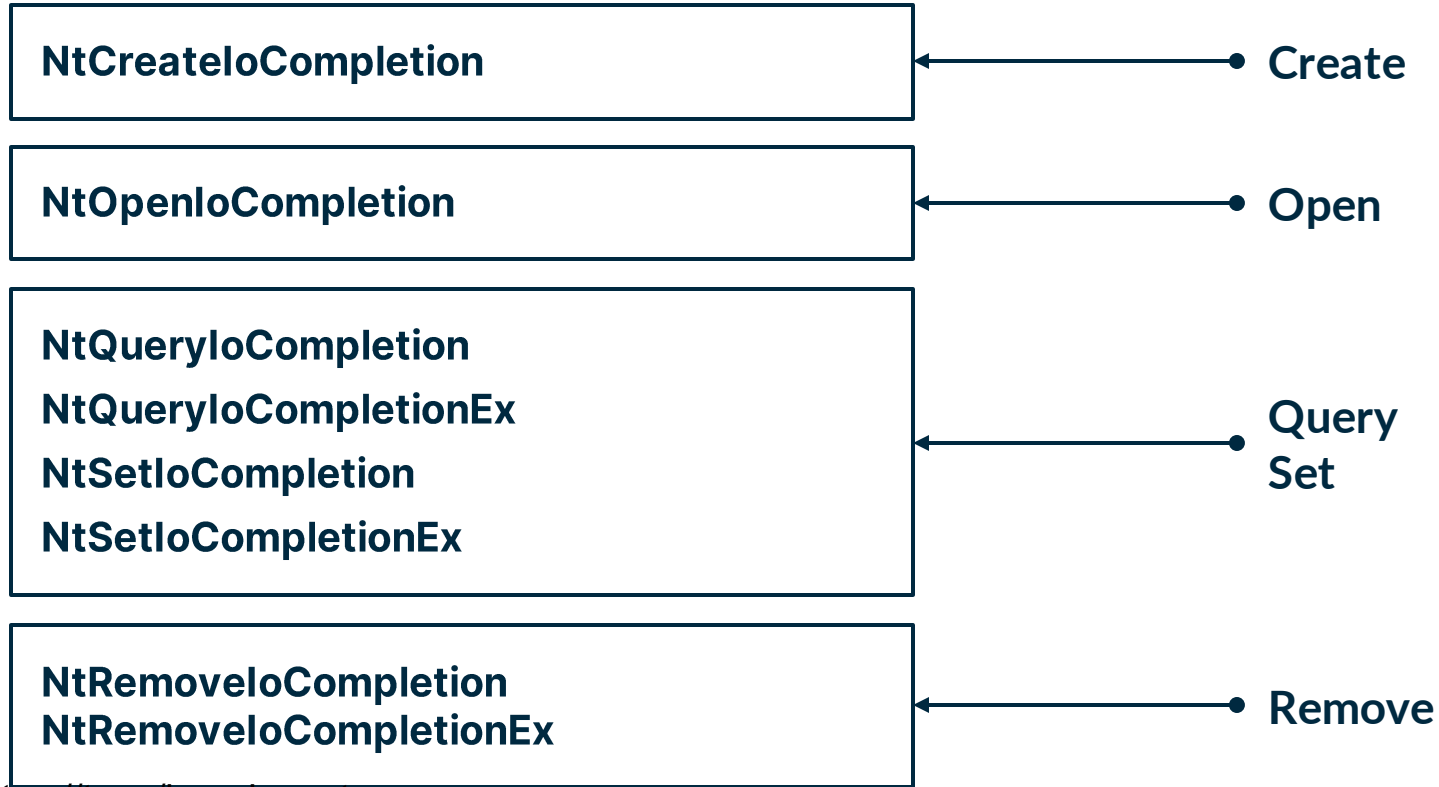
Attacking Thread Pools



I/O Completion Ports Introduction



I/O Completion Queues System Calls



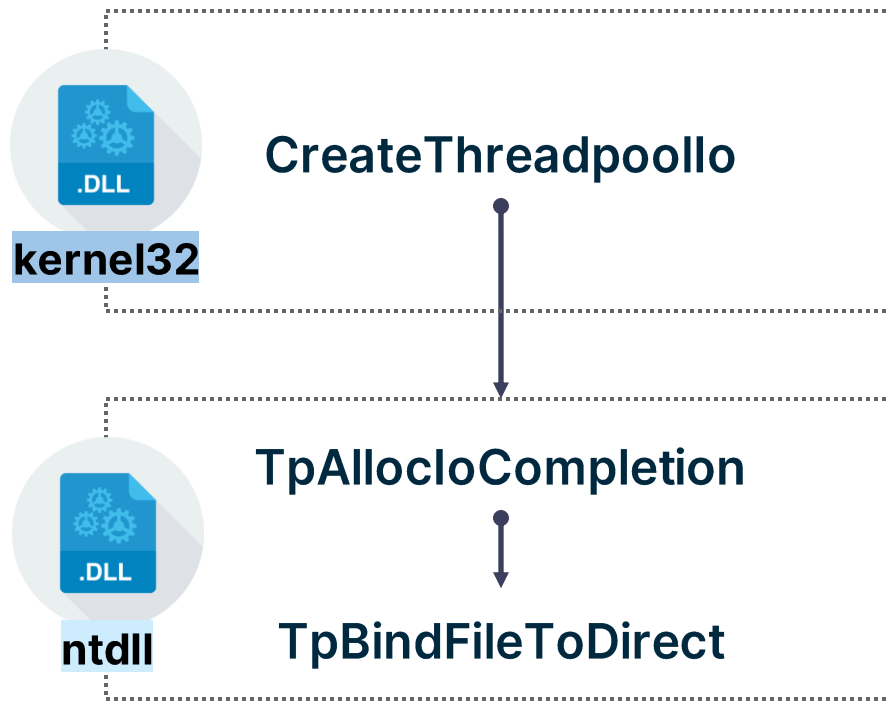
Attacking Thread Pools - TP_IO

```
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



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),
        FileCompletionInfo);

    [snip]
}
```

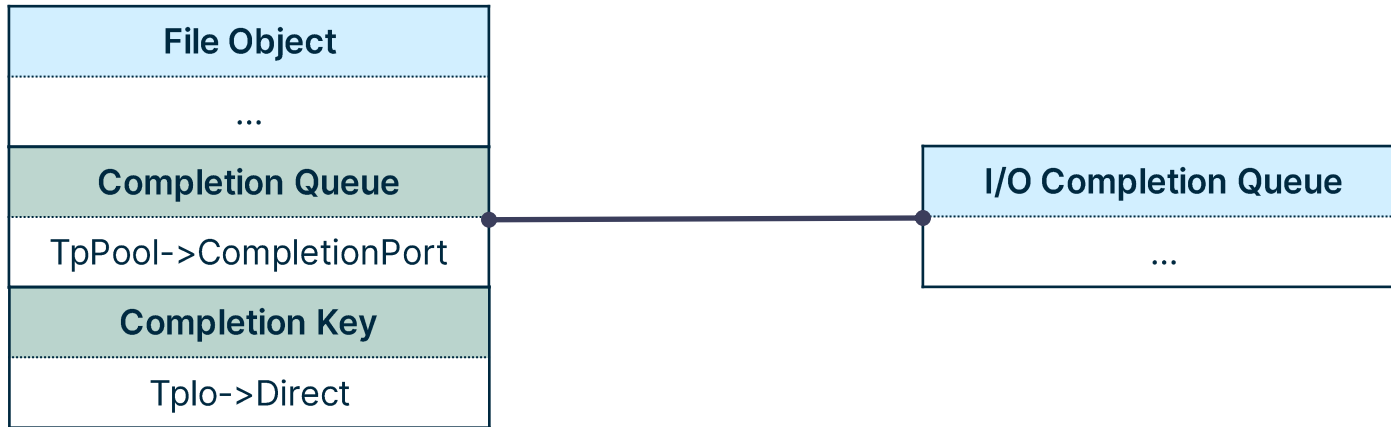
Attacking Thread Pools - TP_IO



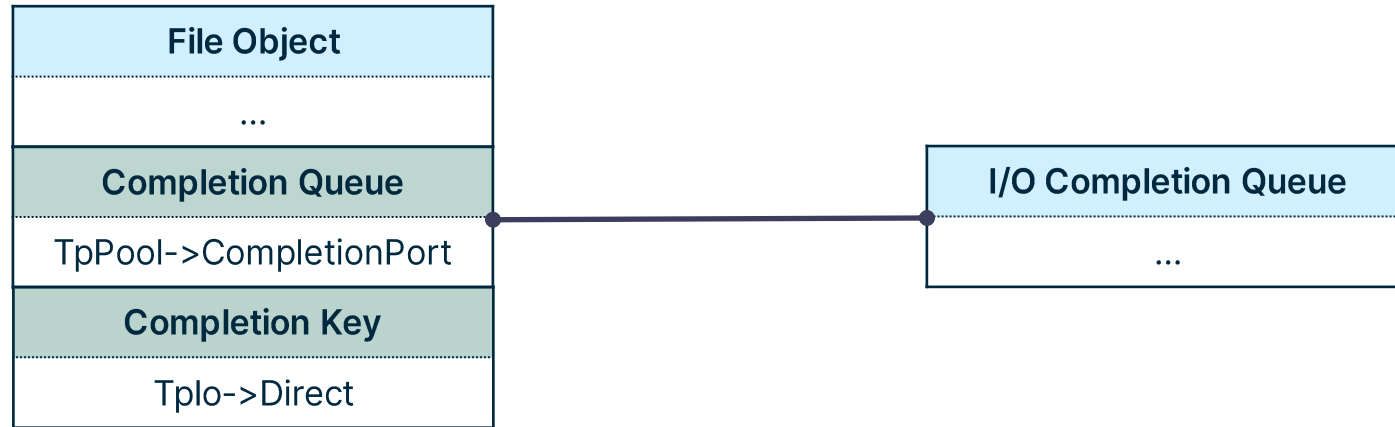
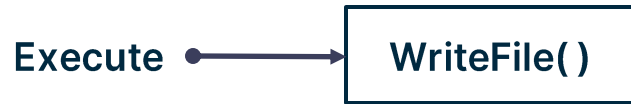
File Object
...
Completion Queue
NULL
Completion Key
NULL

I/O Completion Queue
...

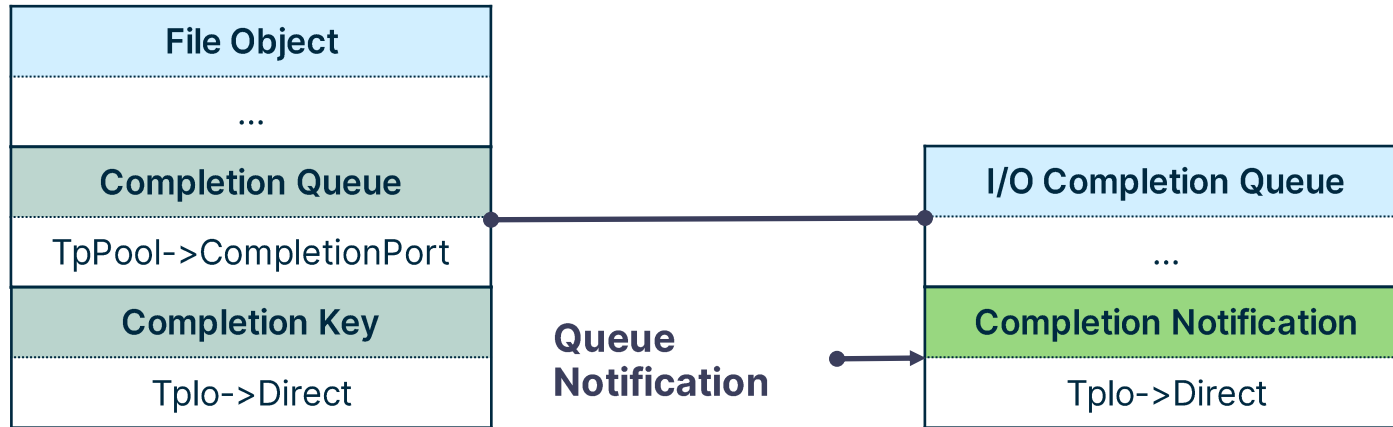
Attacking Thread Pools - TP_IO



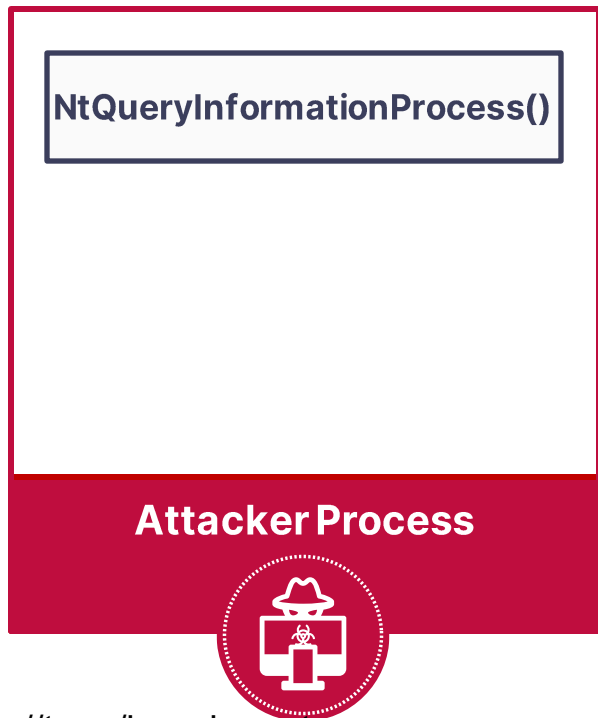
Attacking Thread Pools - TP_IO



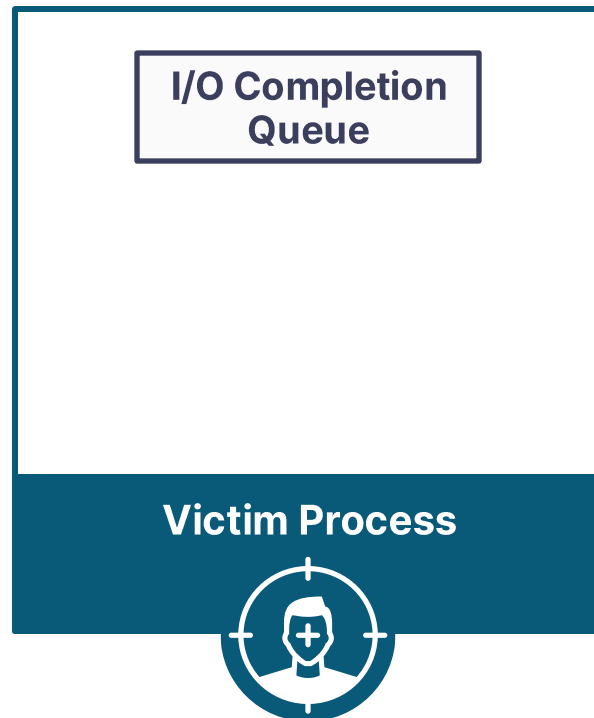
Attacking Thread Pools - TP_IO



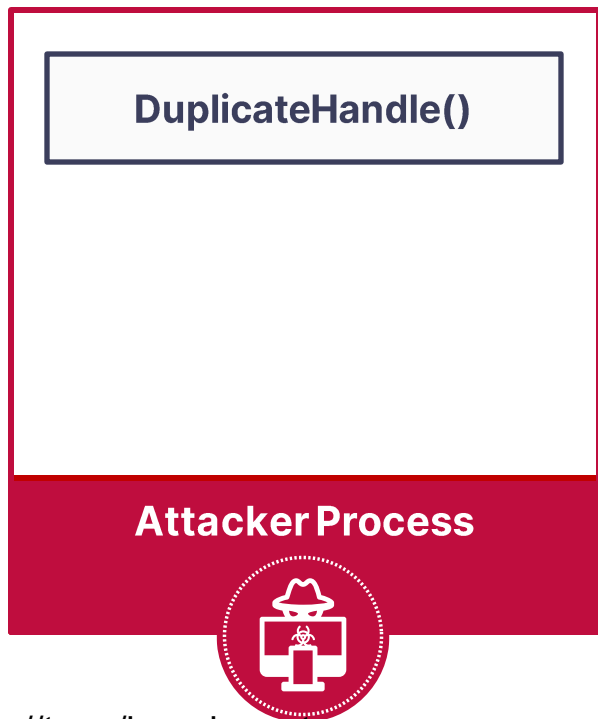
Attacking Thread Pools - TP_IO



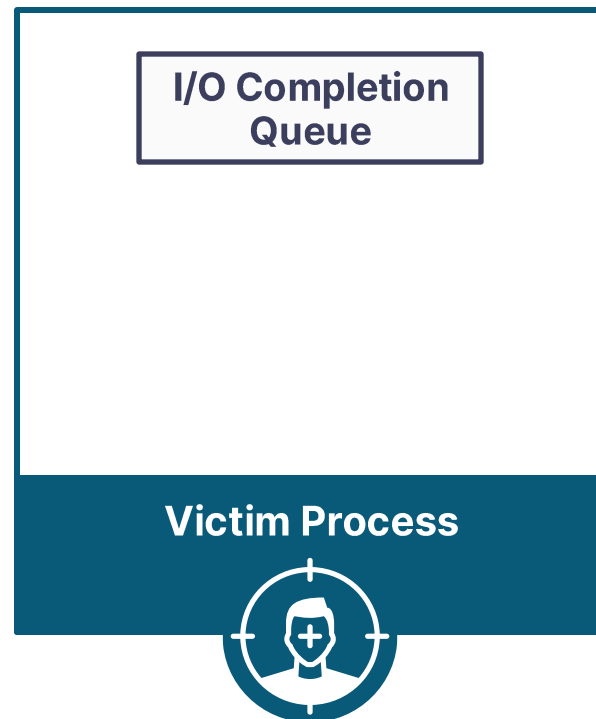
Get
handle
table



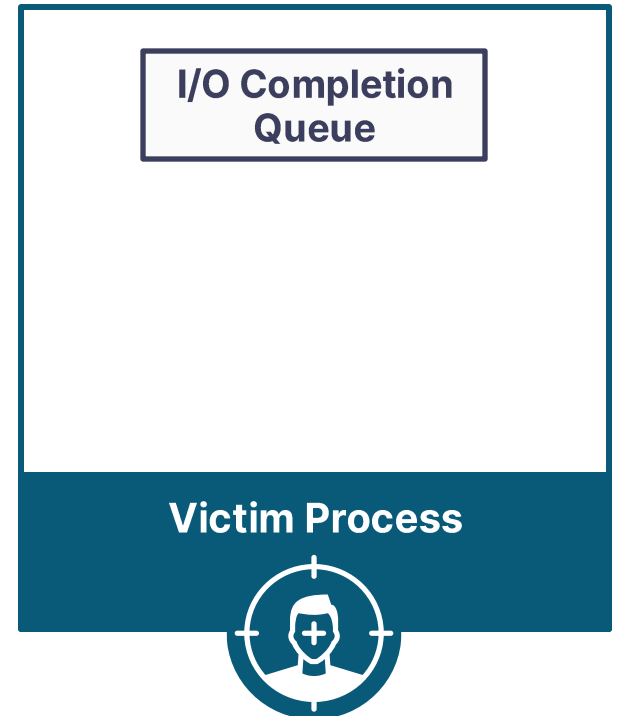
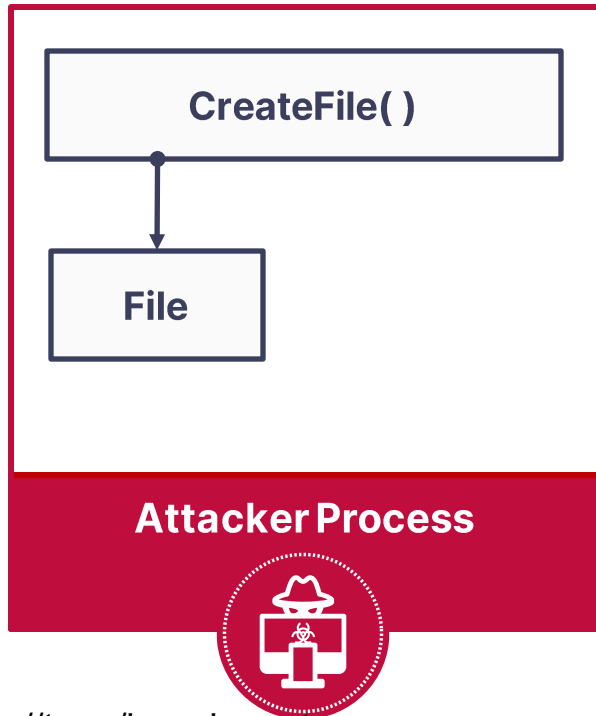
Attacking Thread Pools - TP_IO



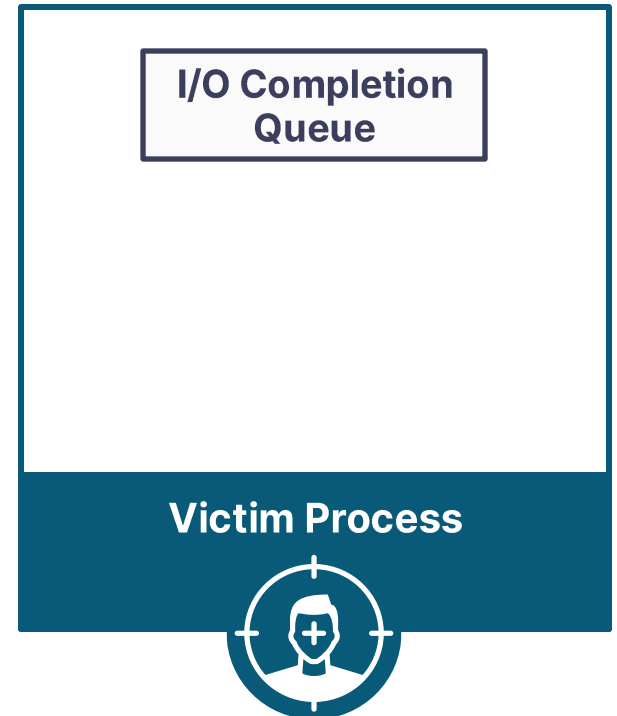
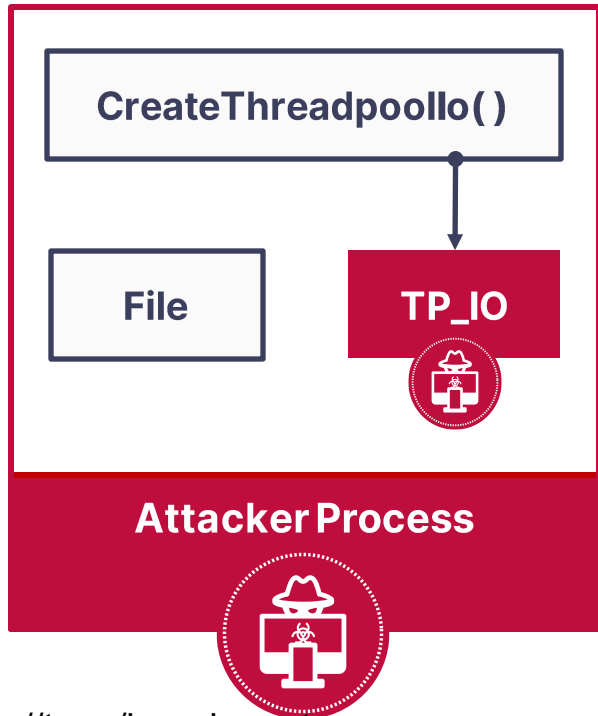
Duplicate I/O
Completion
queue handle



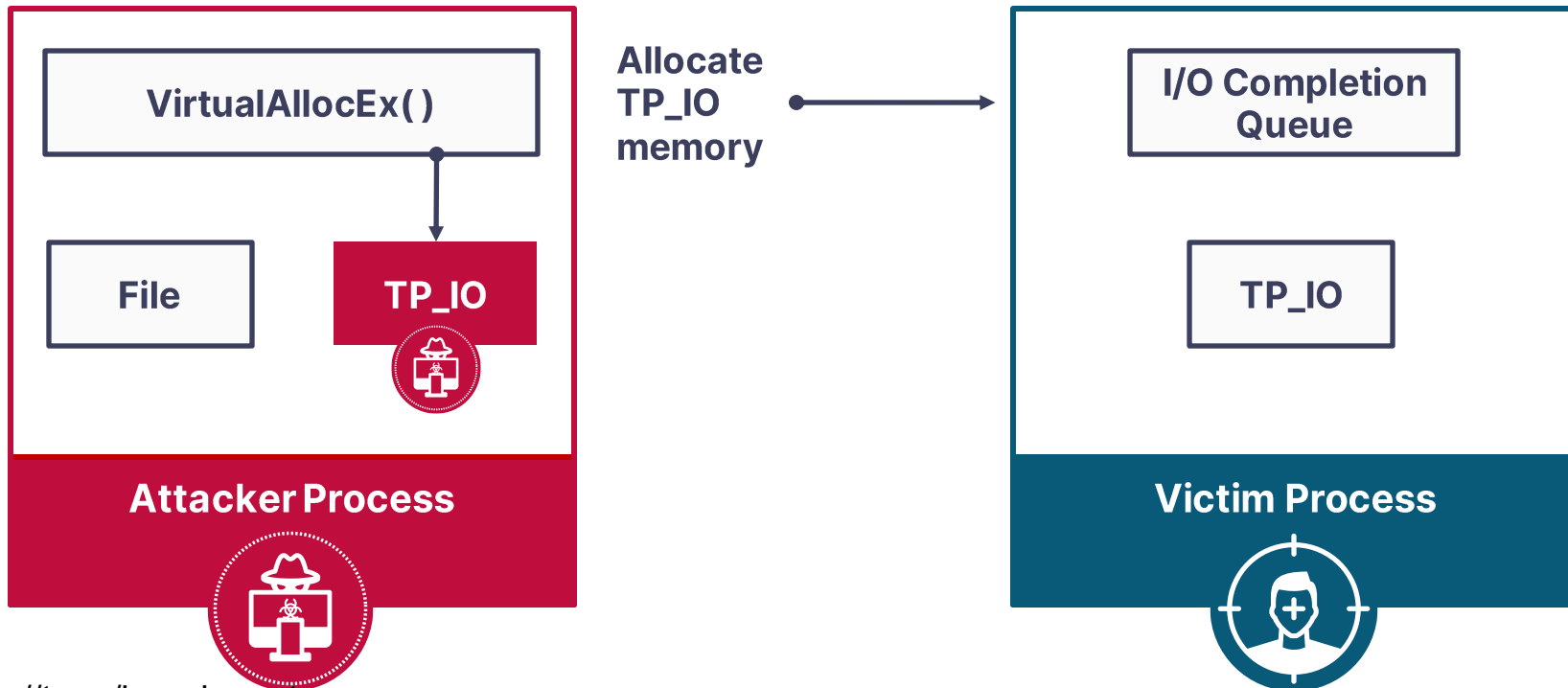
Attacking Thread Pools - TP_IO



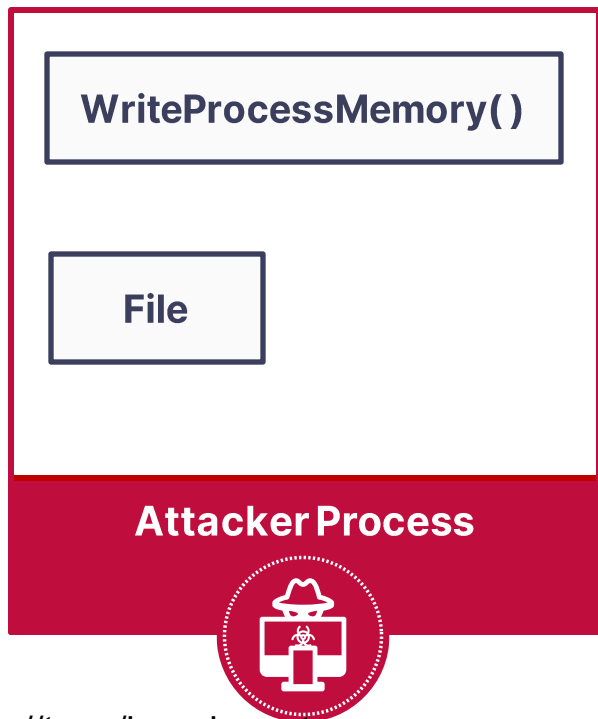
Attacking Thread Pools - TP_IO



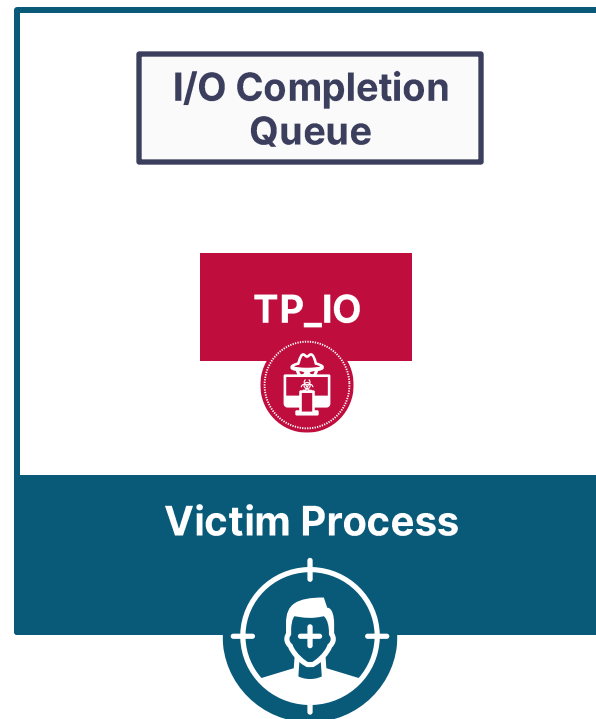
Attacking Thread Pools - TP_IO



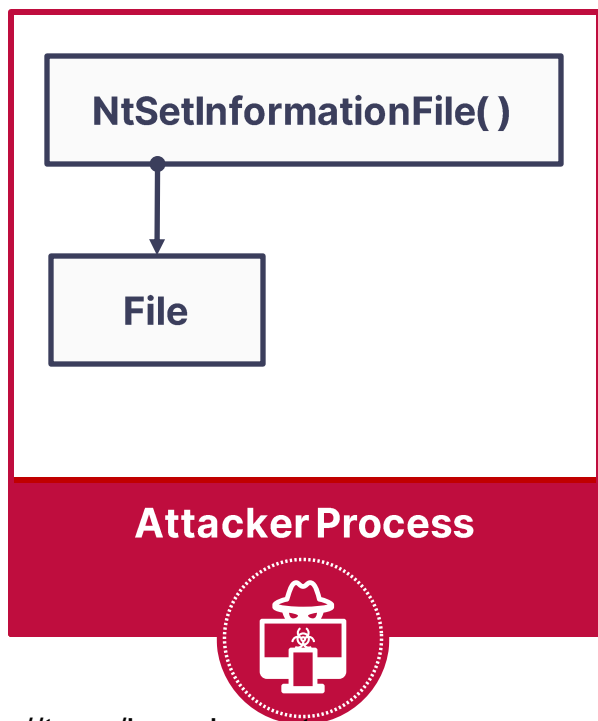
Attacking Thread Pools - TP_IO



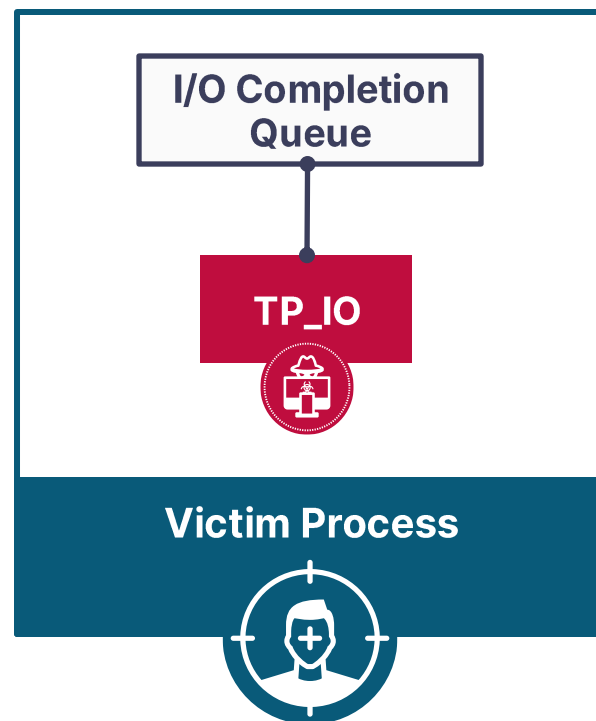
Write TP_IO memory



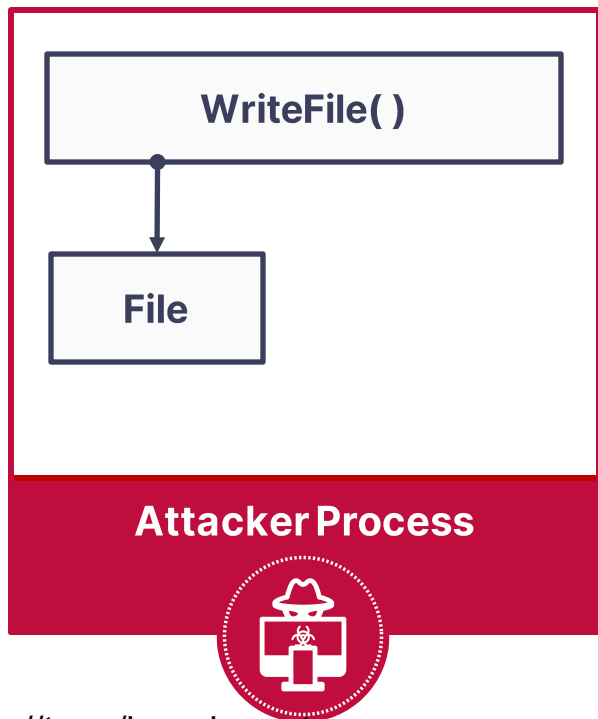
Attacking Thread Pools - TP_IO



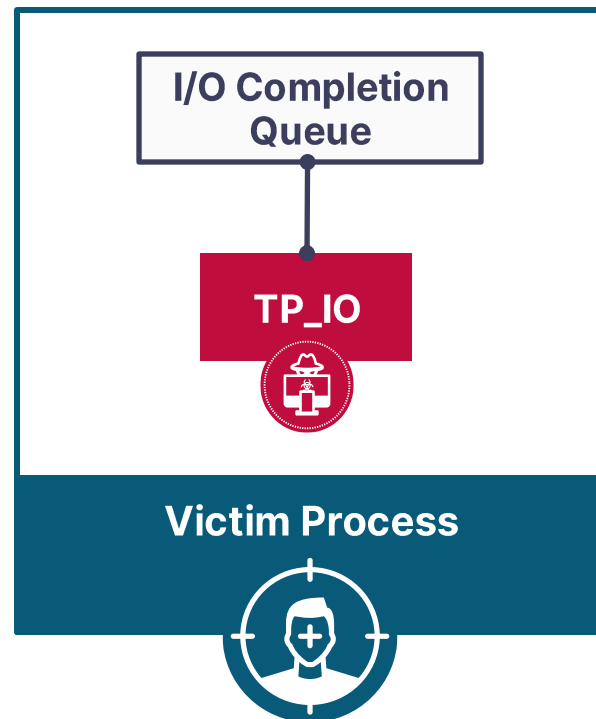
Associate TP_IO with target I/O completion queue



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

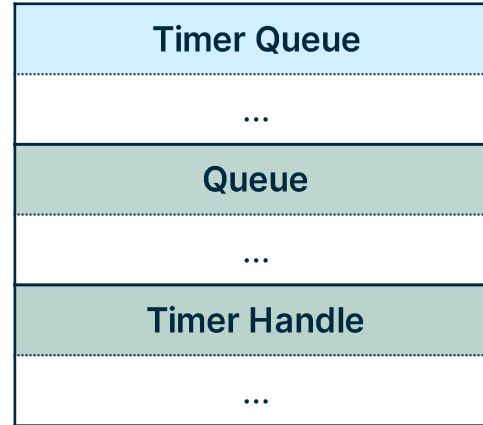
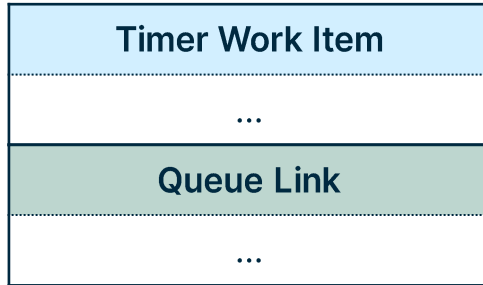
No timer handle is supplied



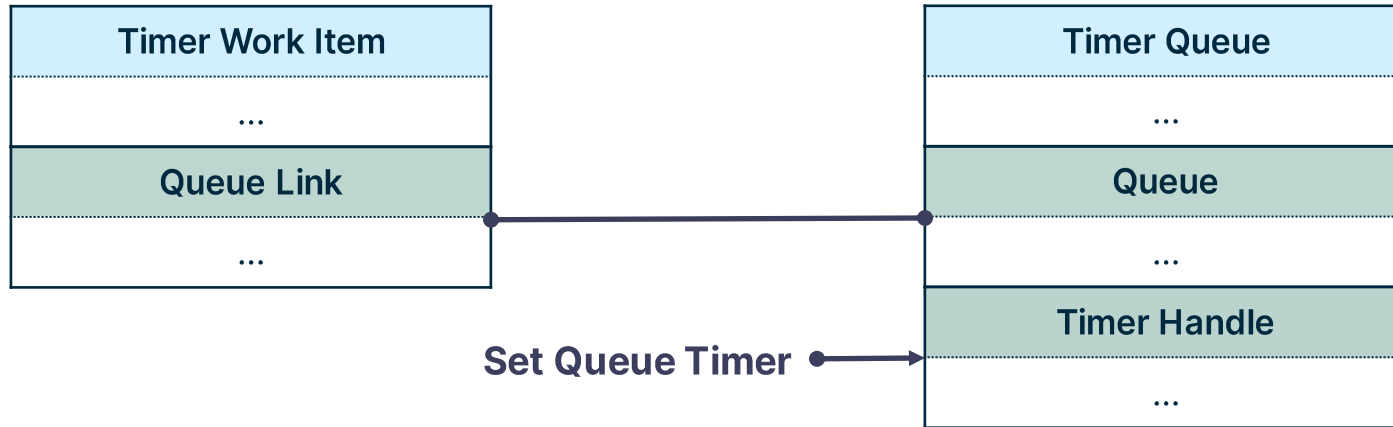
```
PTP_TIMER NTAPI CreateThreadpoolTimer(  
    _In_      PTP_TIMER_CALLBACK TimerCallback,  
    _In_Opt  PVOID TimerContext,  
    _In_Opt  PTP_CALLBACK_ENVIRON TpCallbackEnviron  
);
```

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

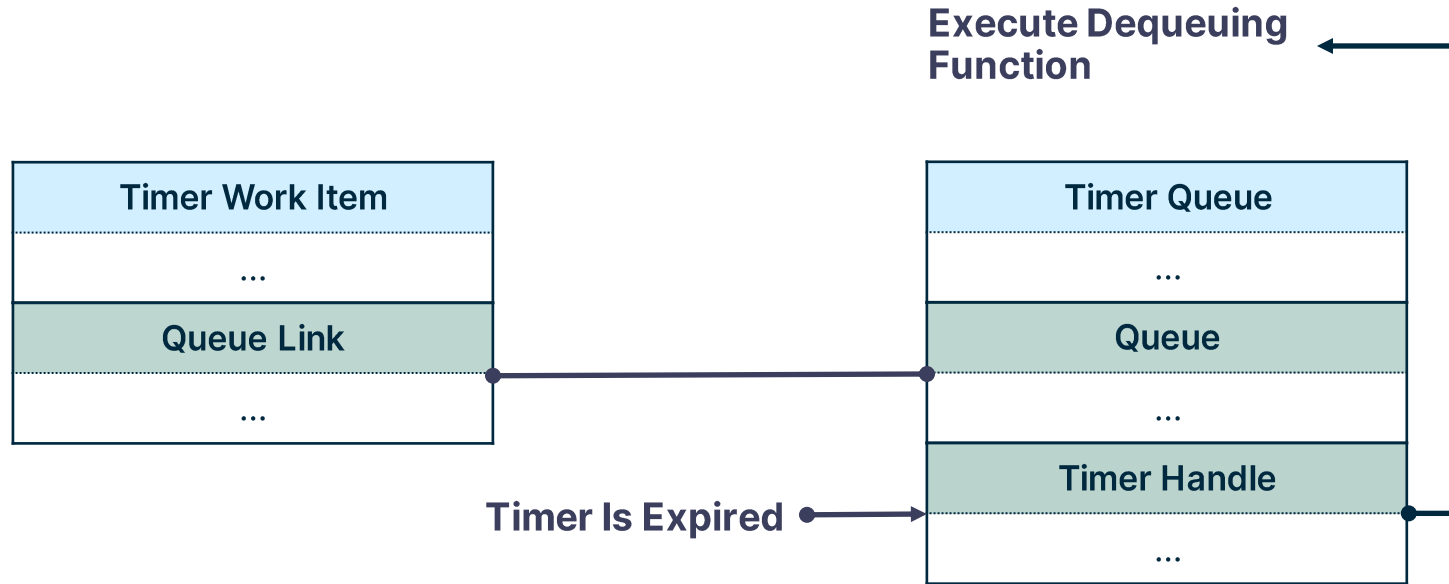
Attacking Thread Pools – TP_TIMER



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]
}
```

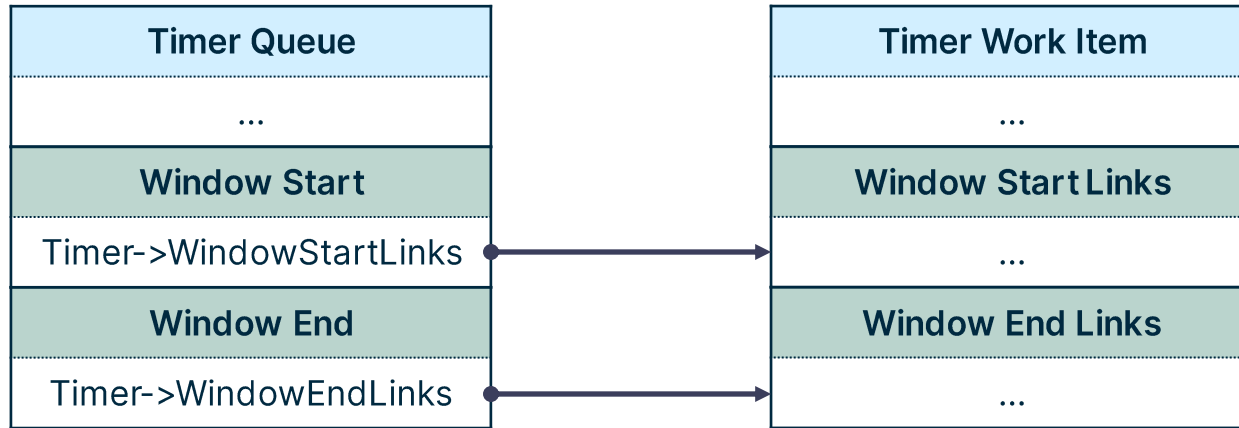
Attacking Thread Pools – TP_TIMER



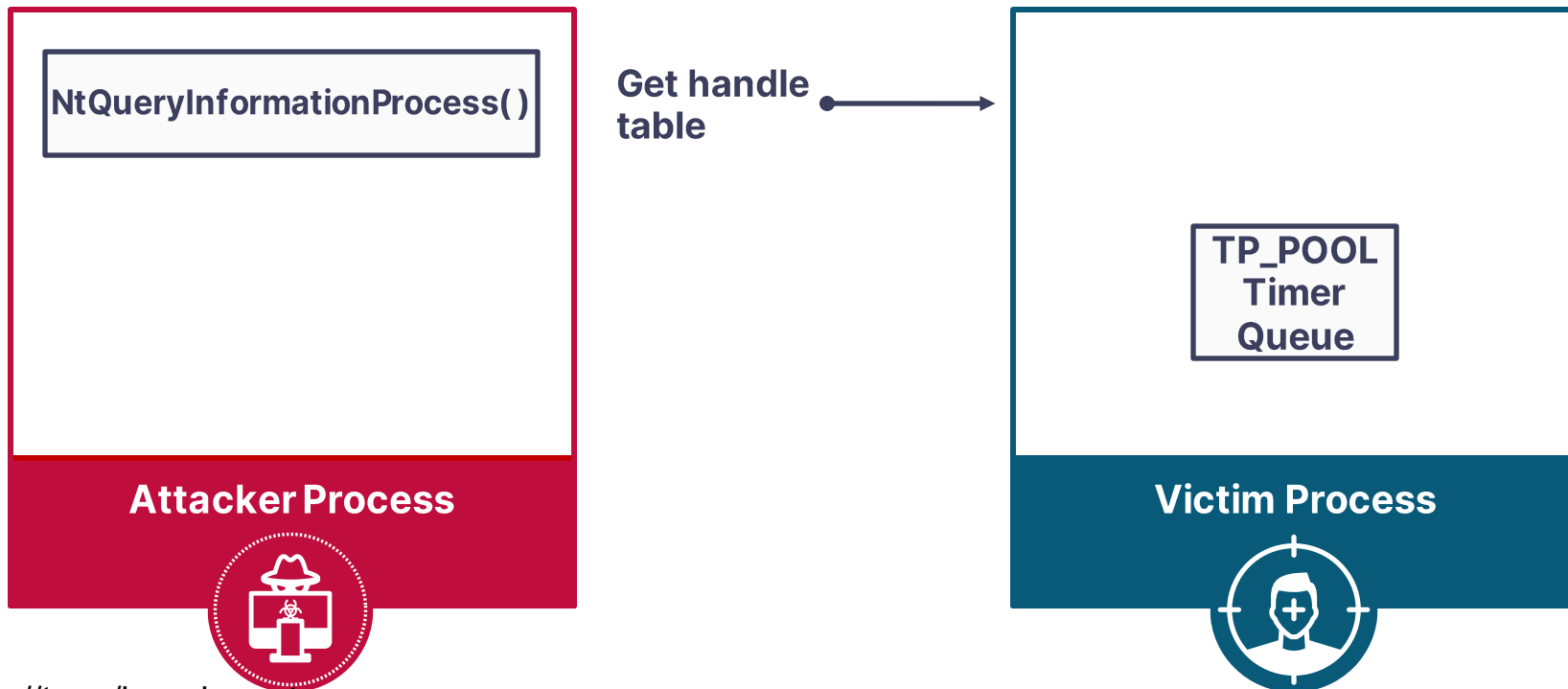
Timer Queue
...
Window Start
NULL
Window End
NULL

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

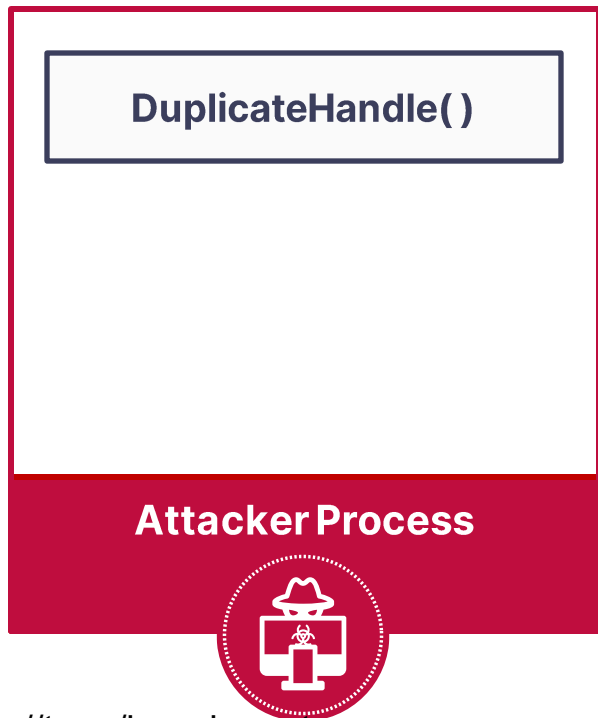
Attacking Thread Pools – TP_TIMER



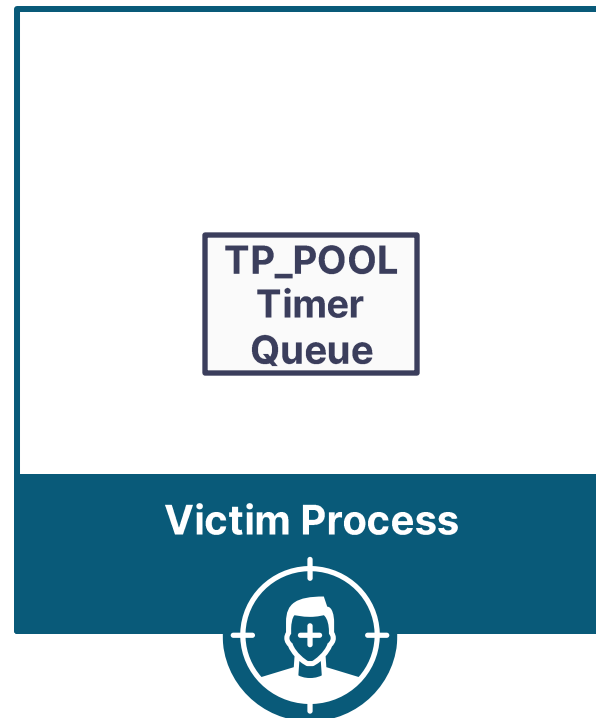
Attacking Thread Pools – TP_TIMER



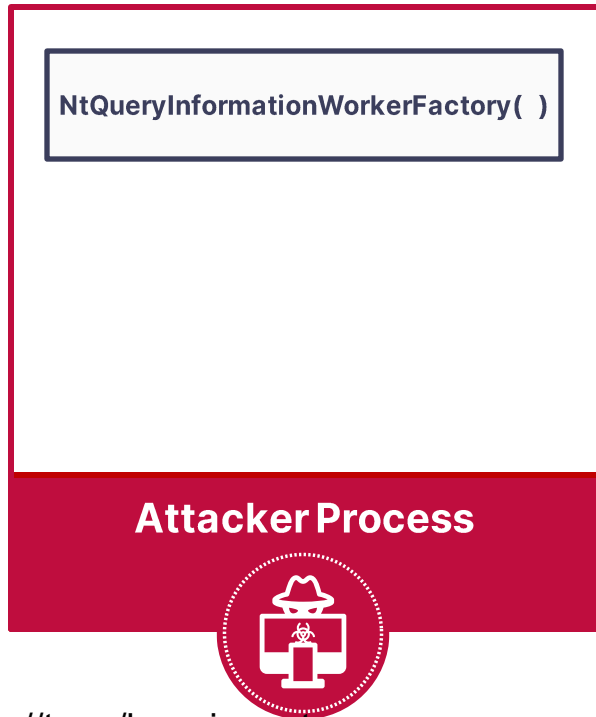
Attacking Thread Pools – TP_TIMER



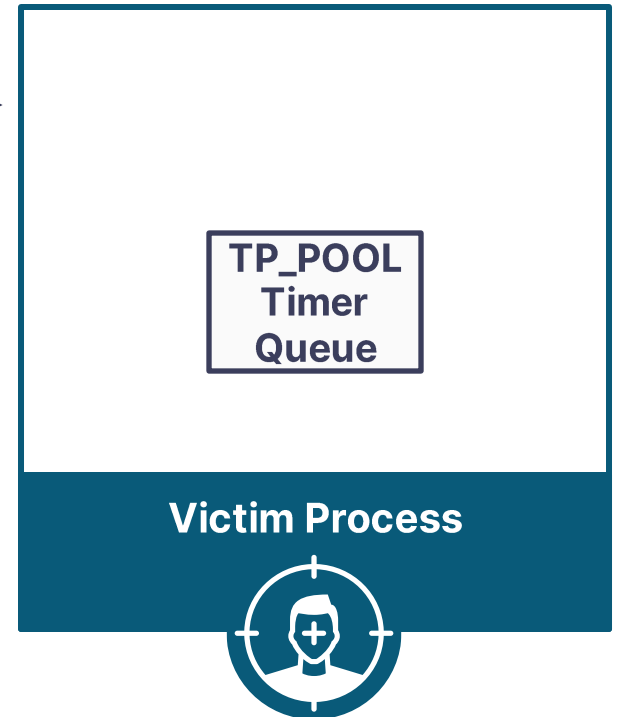
Duplicate
Worker Factory
handle →



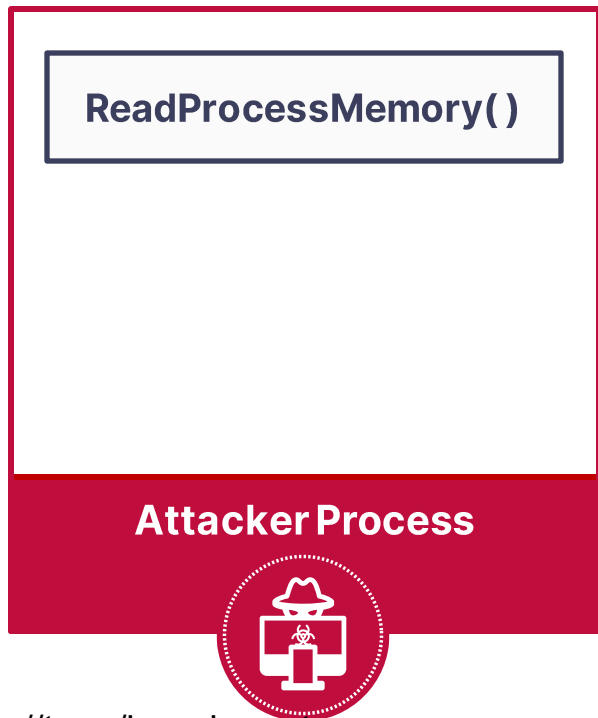
Attacking Thread Pools – TP_TIMER



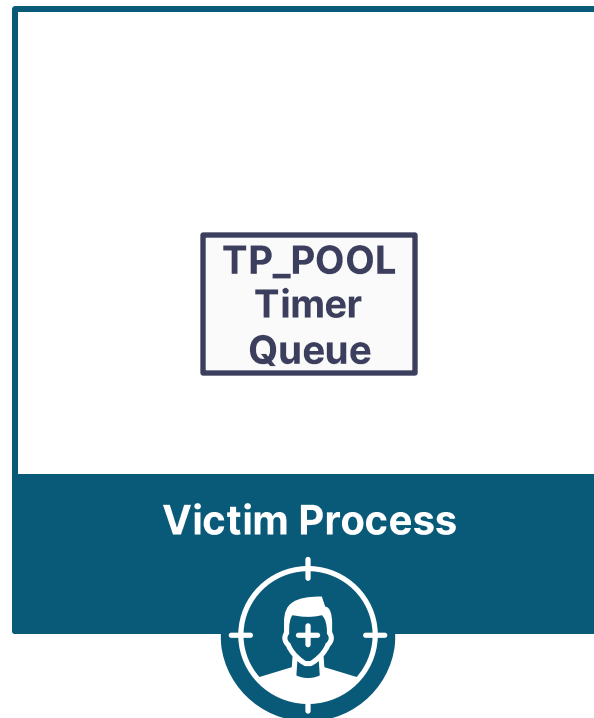
Get Worker
Factory
info



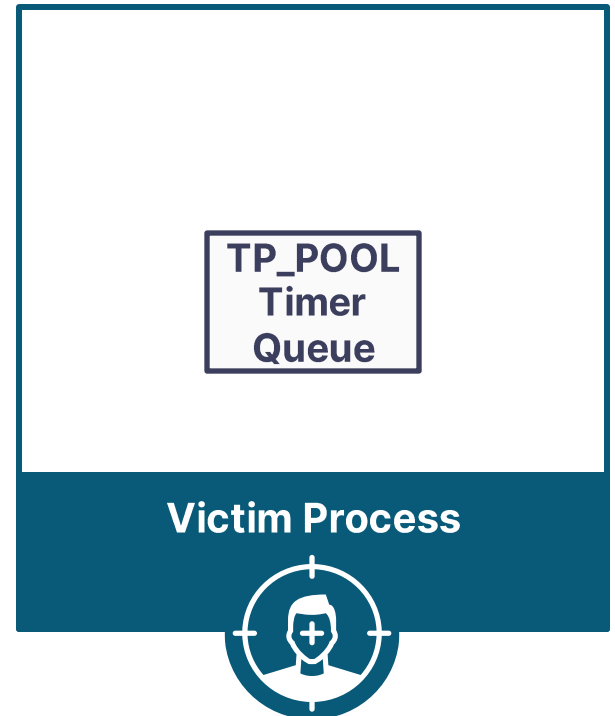
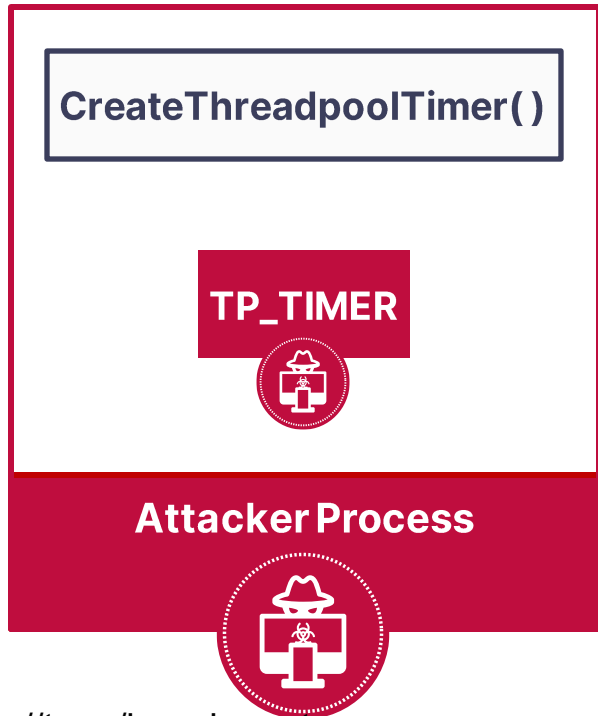
Attacking Thread Pools – TP_TIMER



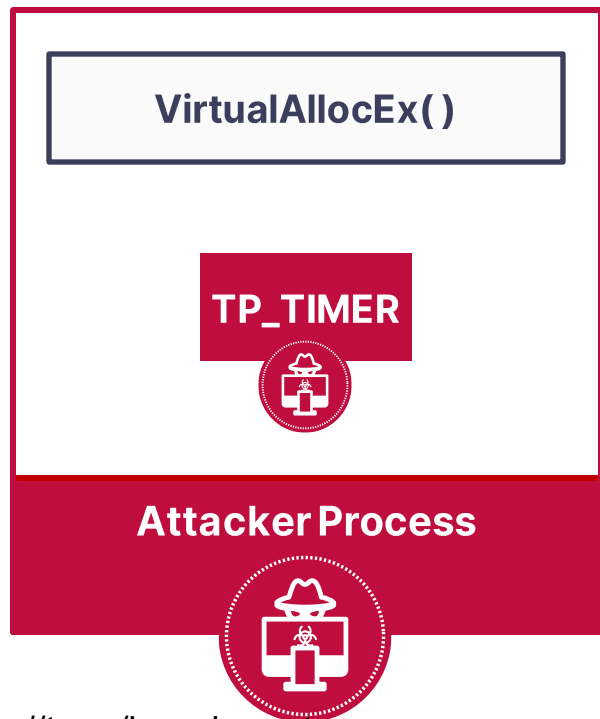
Read TP_POOL →



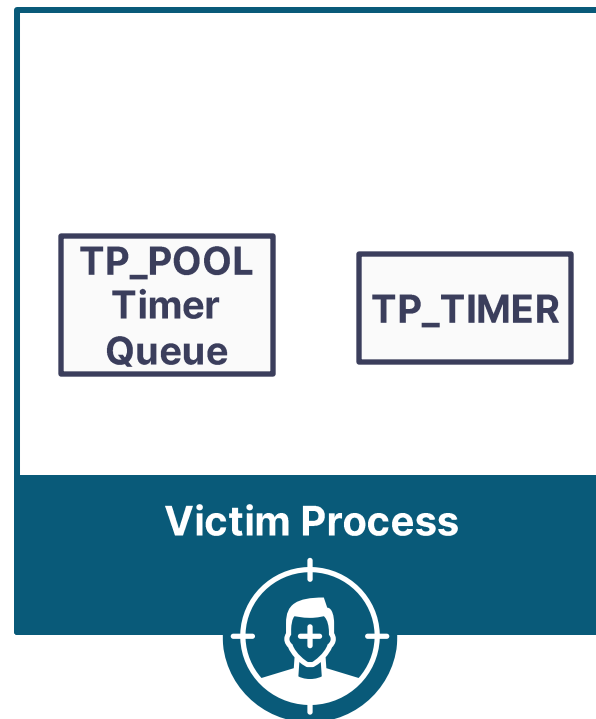
Attacking Thread Pools – TP_TIMER



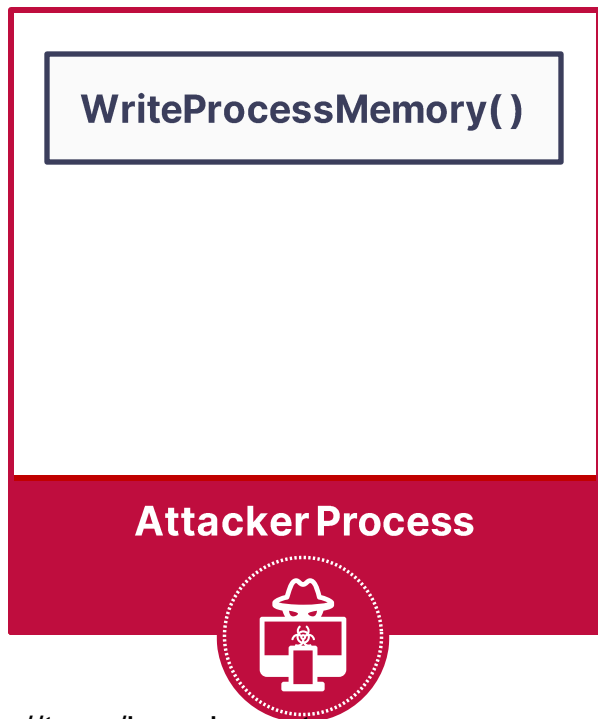
Attacking Thread Pools – TP_TIMER



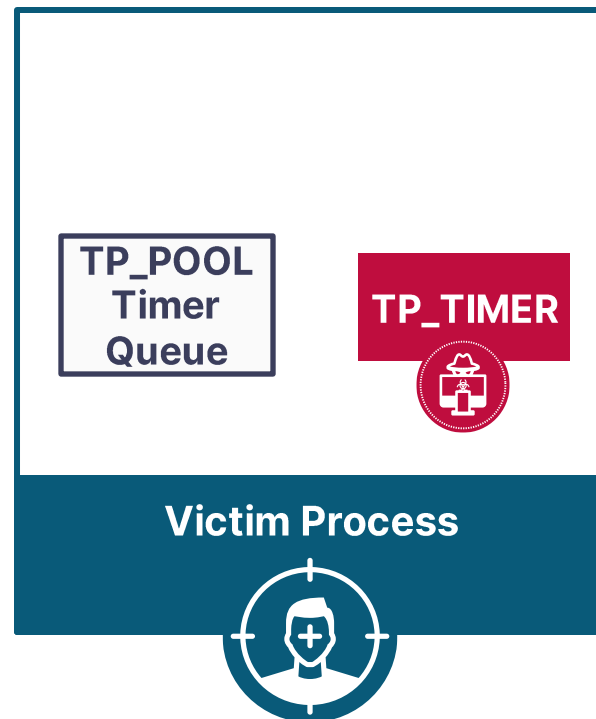
Allocate
TP_TIMER
memory



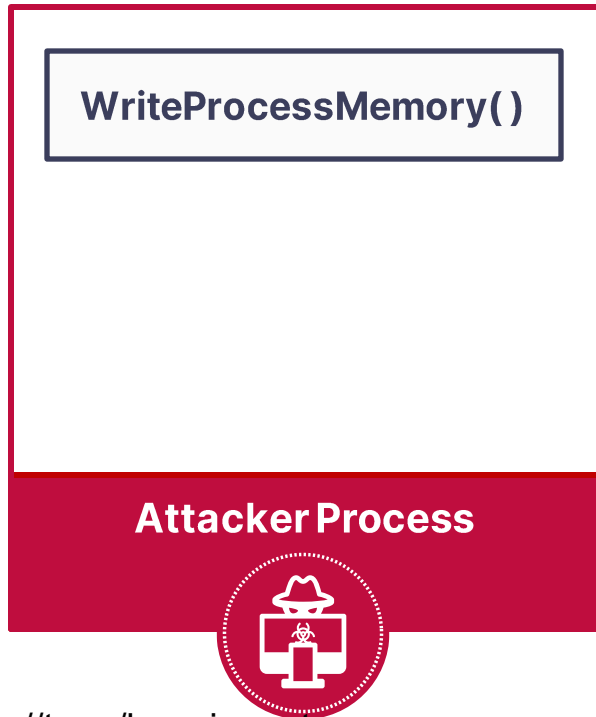
Attacking Thread Pools – TP_TIMER



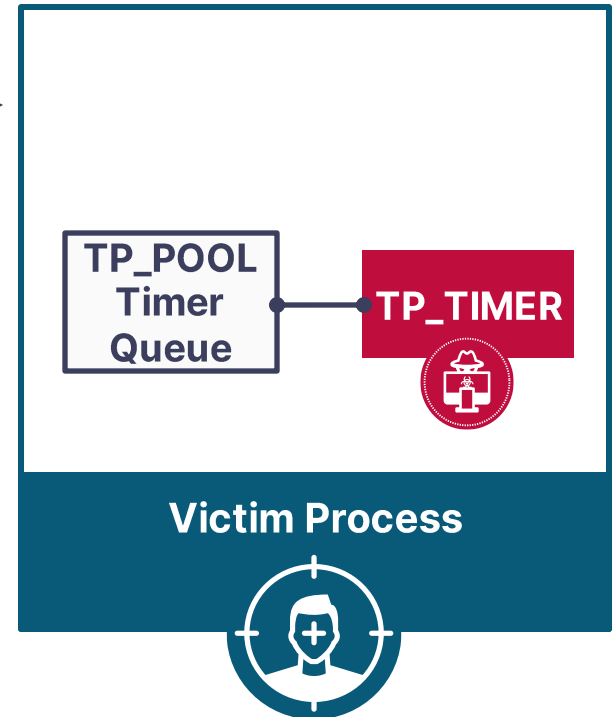
Write
TP_TIMER
memory →



Attacking Thread Pools – TP_TIMER



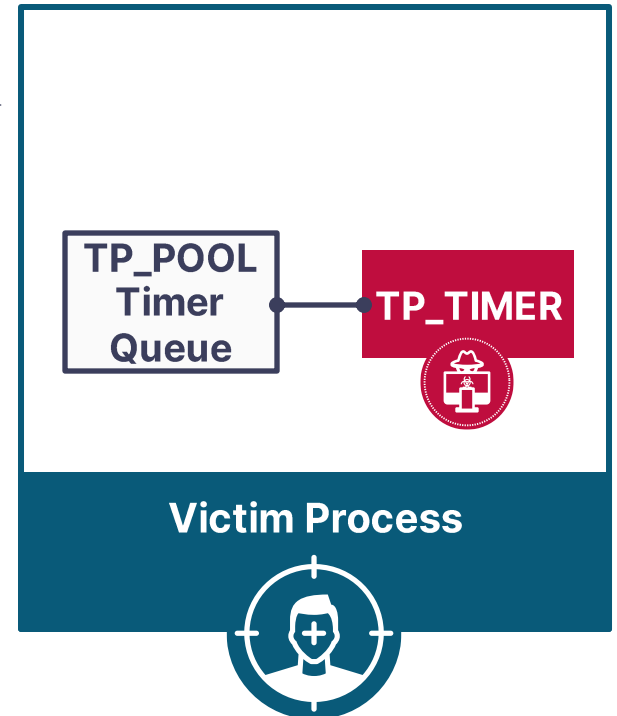
Insert
TP_TIMER to
TP_POOL
timer queue



Attacking Thread Pools – TP_TIMER



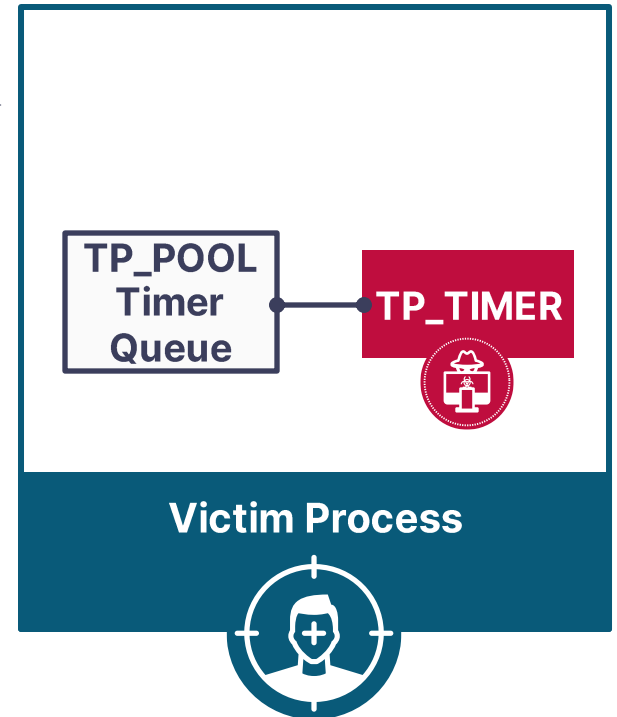
Duplicate
queue timer
handle



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)

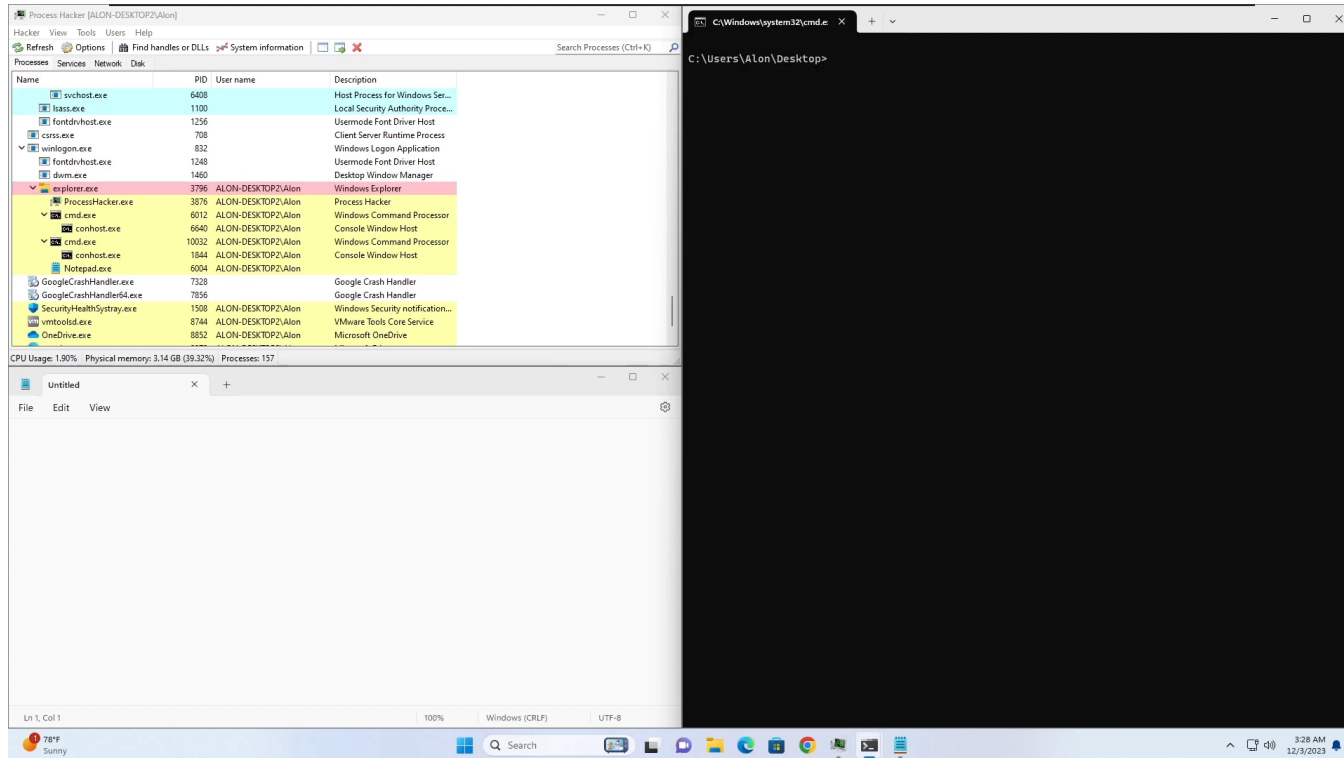
Introducing PoolParty - GitHub Repository



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

<https://t.me/learningnets>

Introducing PoolParty - Demo



<https://t.me/learningnets>



How it started



How it's going

Process Injection Implications



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 various files such as 'background_checks', 'bank_accounts', 'budget_spreadsheets.pptx', 'competitive_analysis.xlsx', 'confidential_memos', 'contracts', 'customer_data.pptx', 'employee_files', 'intellectual_property', and 'legal_documents'.
- Process Explorer:** Displays a list of running processes. The 'Private Bytes' column is highlighted, showing memory usage for various processes. Notable entries include 'svchost.exe' (20,384 K), 'VimPro-SE.exe' (8,520 K), and 'RuntimeBroker.exe' (5,968 K).
- Windows Defender:** A window titled 'Windows 11 MS Defender' is open, showing a dark interface.
- Command Prompt:** A terminal window is open, displaying the path 'C:\Users\A10h\Desktop>'.

The taskbar at the bottom shows the system tray with the time '4:10 AM' and date '8/4/2023'. The system status bar indicates 'CPU Usage 2.28%', 'Commit Charge 40.37%', 'Processes 138', and 'Physical Usage 59.48%'.

<https://t.me/learningnets>

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



Q & A

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



[@_0xDeku](https://twitter.com/_0xDeku)



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



alon.leviev@safebreach.com



<https://t.me/learningnets>