Please inject-me, a x64 code injection

By Alon Weinberg

August 2019

Alon Weinberg

I’m a Security researcher!
I’ve been working at Deep Instinct, Since 2017
And I was in the IDF Cyber Unit for 4.5 years

deepinstinct

Please inject-me

a x64 code injection

Intro

- Code injection and its importance
- Introducing Inject-Me

Technical background

- ReadProcessMemory
- X64 WinAPI calling convention

Inject-Me - Detailed flow

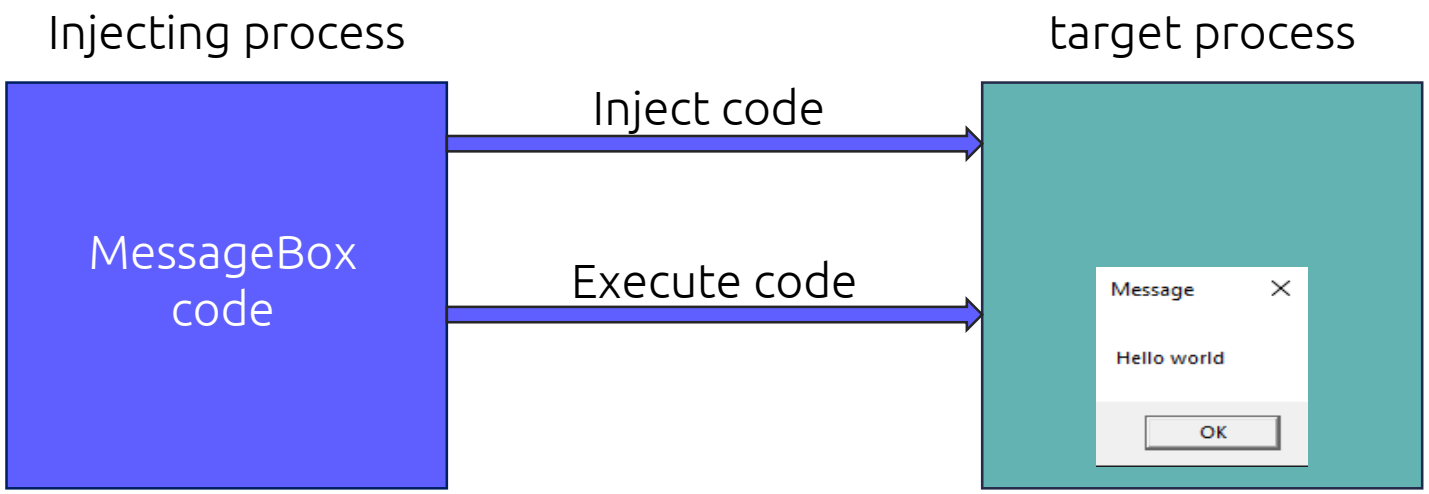
- Abusing ReadProcessMemory
- Copying data on the target process
- Finalizing the injection
  - Infinite running thread
  - Execution
- Demo

Intro

Please inject-me, a x64 code injection

What is code injection

Code injection is the general term of
introducing (or "injecting") code into a process
and executing it from the process context.



Why is code injection important?

Malicious use of code injection:

- Stealth - Hiding malware presence
- Evasion - Bypassing security solutions
- Stealing information from another process

Benign use of code injection

- Security solutions
- Adding functionality
- Monitoring, Analysis and Research

Introducing inject-me

- How it all started
- A new code injection for x64
- The idea behind Inject-Me
- “Injection-less” code injection

Technical background

Please inject-me, a x64 code injection

ReadProcessMemory function

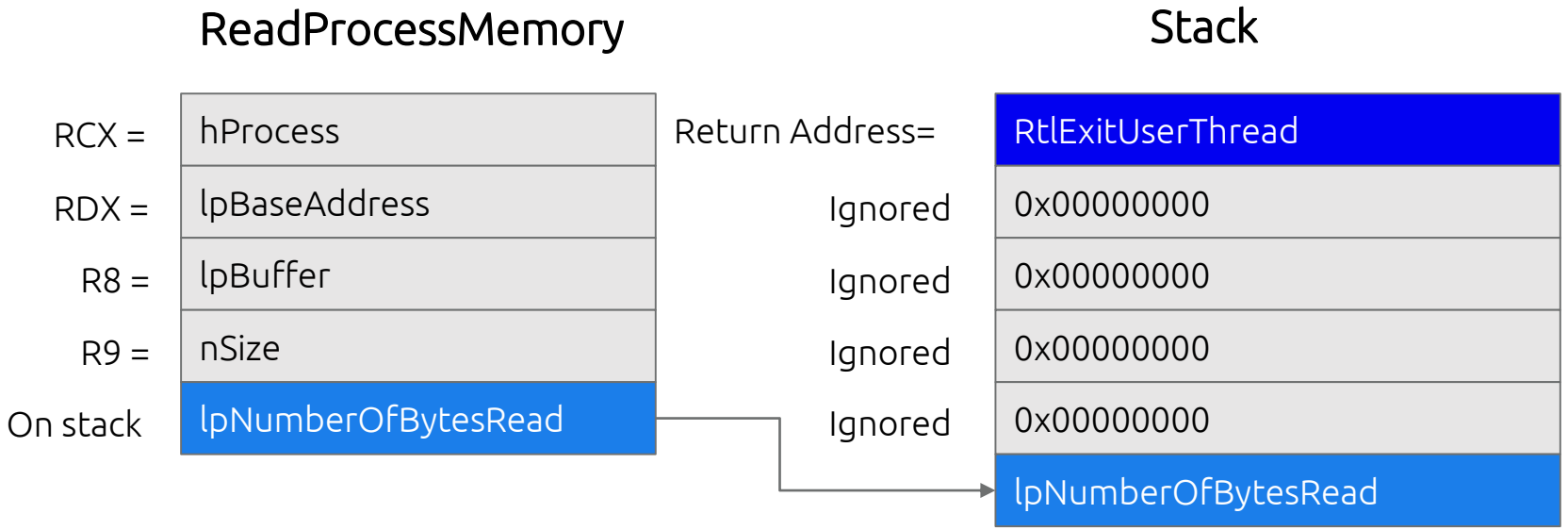
- Reads memory from a process

By running the function remotely in a target process, and controlling the parameters passed using **SetThreadContext** one can read\inject a shellcode into the target process.

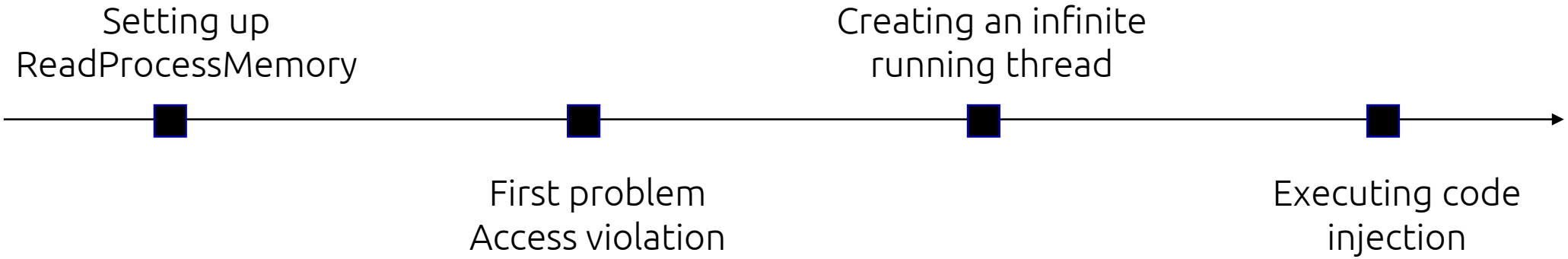
```
C++

BOOL ReadProcessMemory(
  HANDLE  hProcess,
  LPCVOID lpBaseAddress,
  LPVOID  lpBuffer,
  SIZE_T  nSize,
  SIZE_T  *lpNumberOfBytesRead
);
```

X64 WinAPI calling convention

- Integer arguments passed in registers **RCX, RDX, R8, and R9**
- Arguments after the fourth argument passed on the stack
- Function can be set with four or less arguments remotely using **SetThreadContext**



Details and flow of the Injection-less code injection



Setting up ReadProcessMemory for abuse

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection

- **ReadProcessMemory** gets 5 arguments
- Only 4 arguments can be passed through registers
- Fifth parameter can be **NULL**
- Creating a dummy stack - **VirtualAllocEx** allocates memory in a process and zeroes it
- Dummy stack will be used later as the stack when calling **ReadProcessMemory**

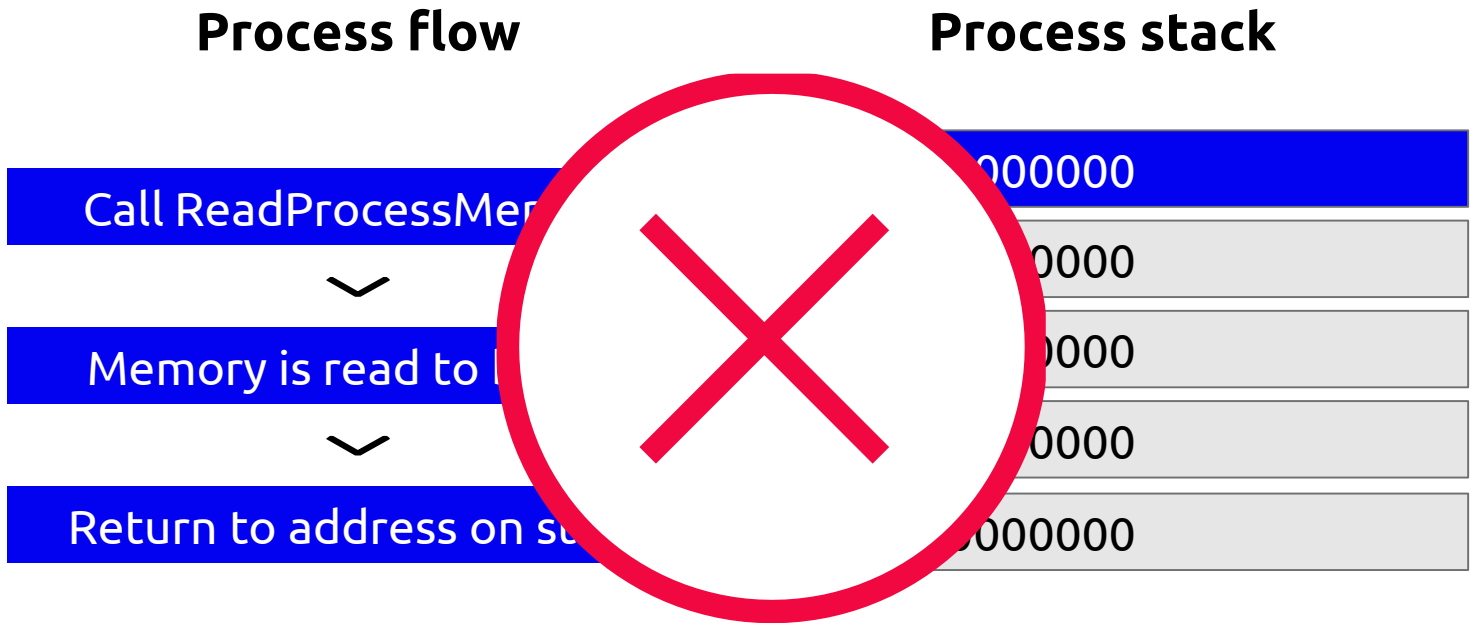
Setting up ReadProcessMemory for abuse

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection

- Using **DuplicateHandle** to duplicate injecting process handle to the target process
- Setting **hProcess** to Injecting process duplicated handle
- Allocating memory for the shellcode in the target process using **VirtualAllocEx**

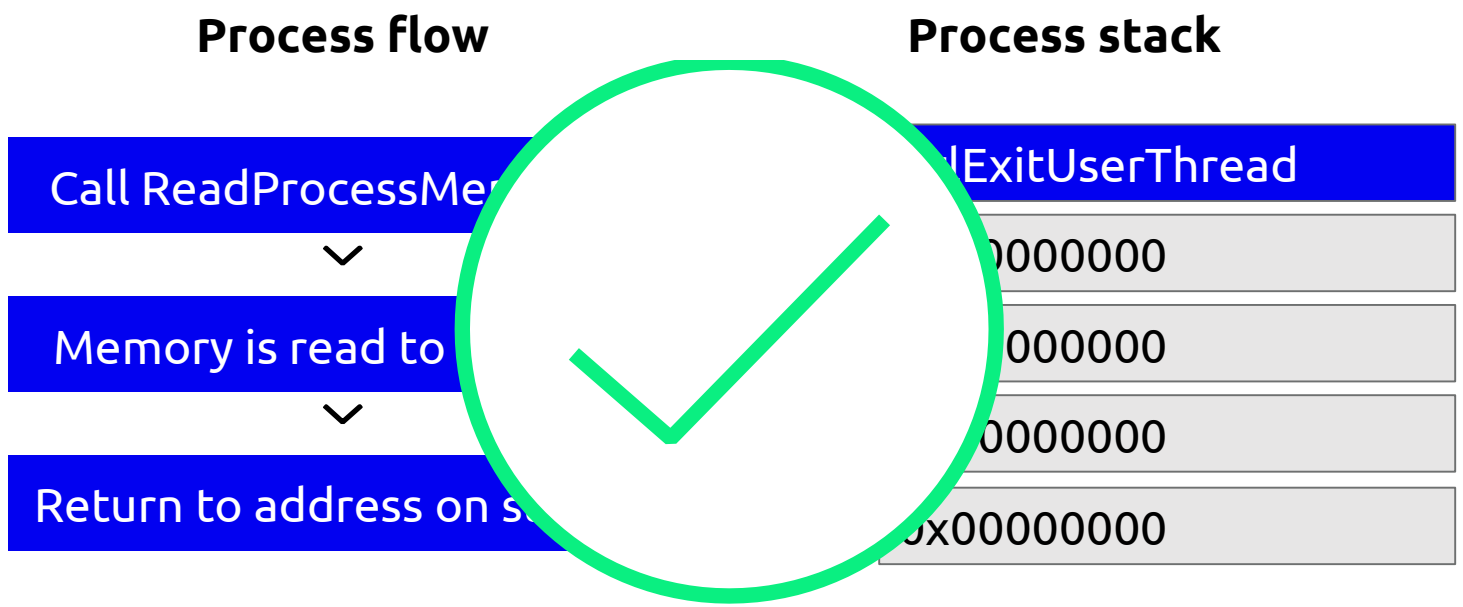
Access violation
return address is 0

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection



Access violation
return address is RtlExitUserThread

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection



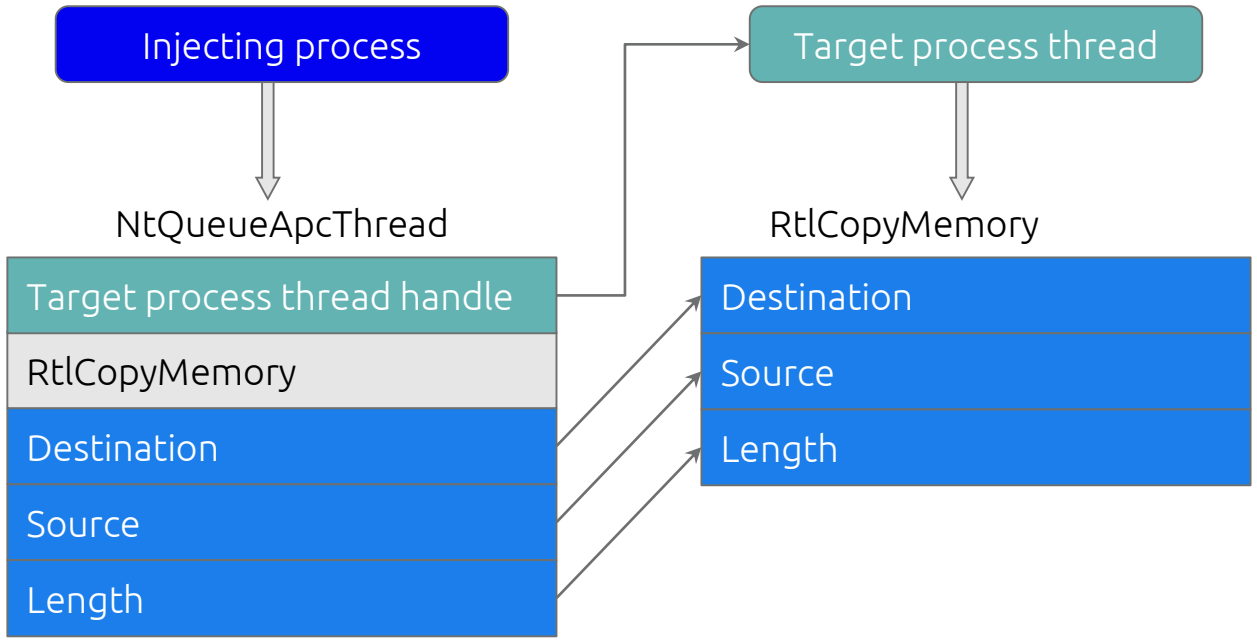
Copying RtlExitUserThread to the dummy stack

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection

- Kernel32.dll imports **RtlExitUserThread** from ntdll.dll
- **RtlExitUserThread** address should exist in kernel32.dll IAT (Import Address Table)
- kernel32.dll base address and IAT address are identical between processes
- Finding **RtlExitUserThread** in injecting process and copying it in the target process

How to copy data on the target process

- **NtQueueApcThread** calls a function in a process and passes 3 parameters to it
- **RtlCopyMemory** gets 3 parameters
- Copying data using **NtQueueApcThread** and **RtlCopyMemory**



Side note –

Recreating shellcode in a target process

- The method described earlier can be used to recreate a shellcode in the target process:
  - Finding each byte of the shellcode in the target process
  - Copying the shellcode byte by byte in the target process
- We’ve found a way to recreate shellcode in a target process!

Finalizing the code injection

Please inject-me, a x64 code injection

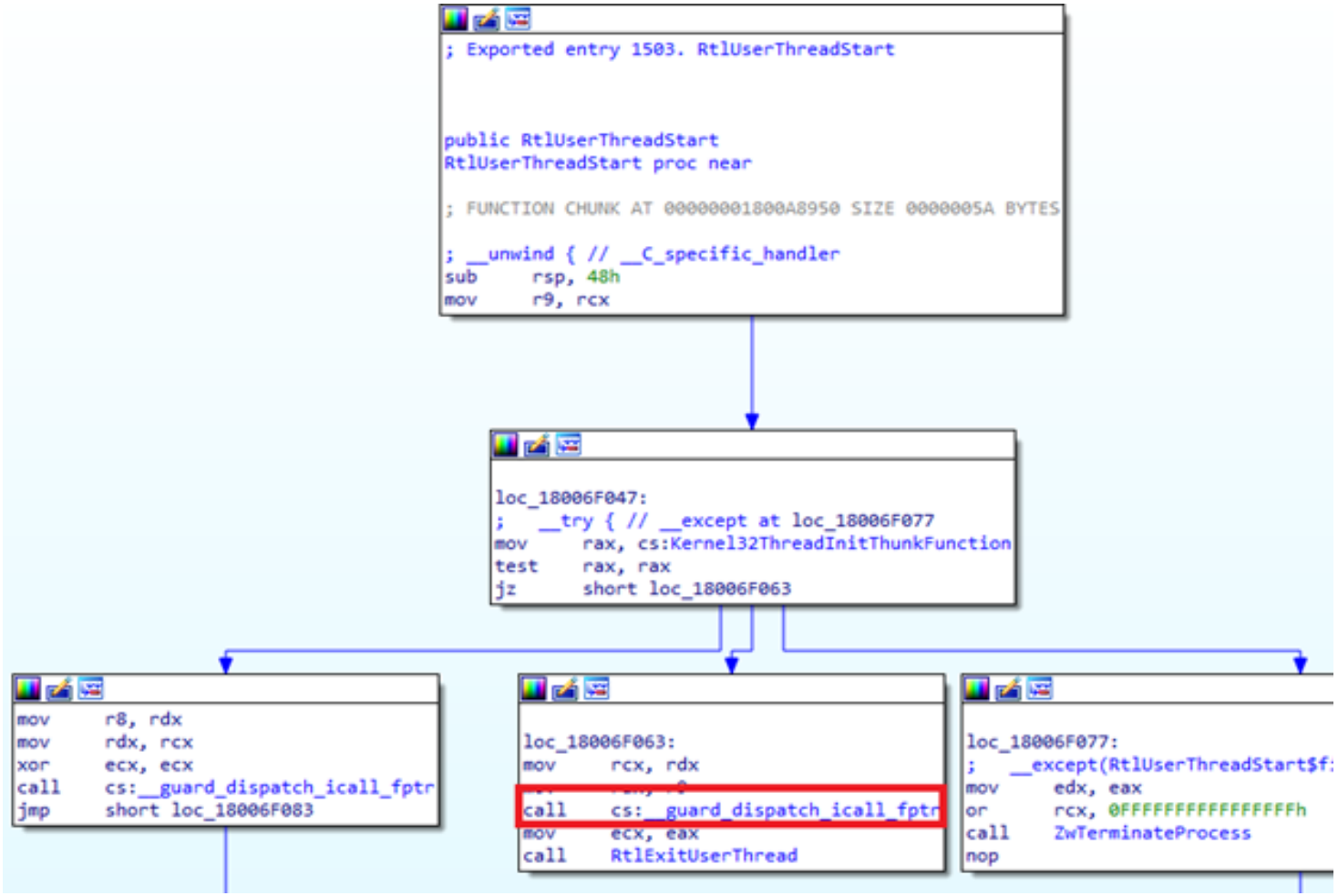
An Infinite running thread is needed
Describing a new problem

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection

- Set **RIP** register to **ReadProcessMemory**
  - Really?
- Setting the **RIP** register of a thread created suspended causes exception
  - Exception 0xC000000D, **STATUS_INVALID_PARAMETER**
  - The thread needs to initialize before it is manipulated
- A thread created in the target process will terminate before it can be manipulated
- Running an infinitely running thread will allow it to initialize

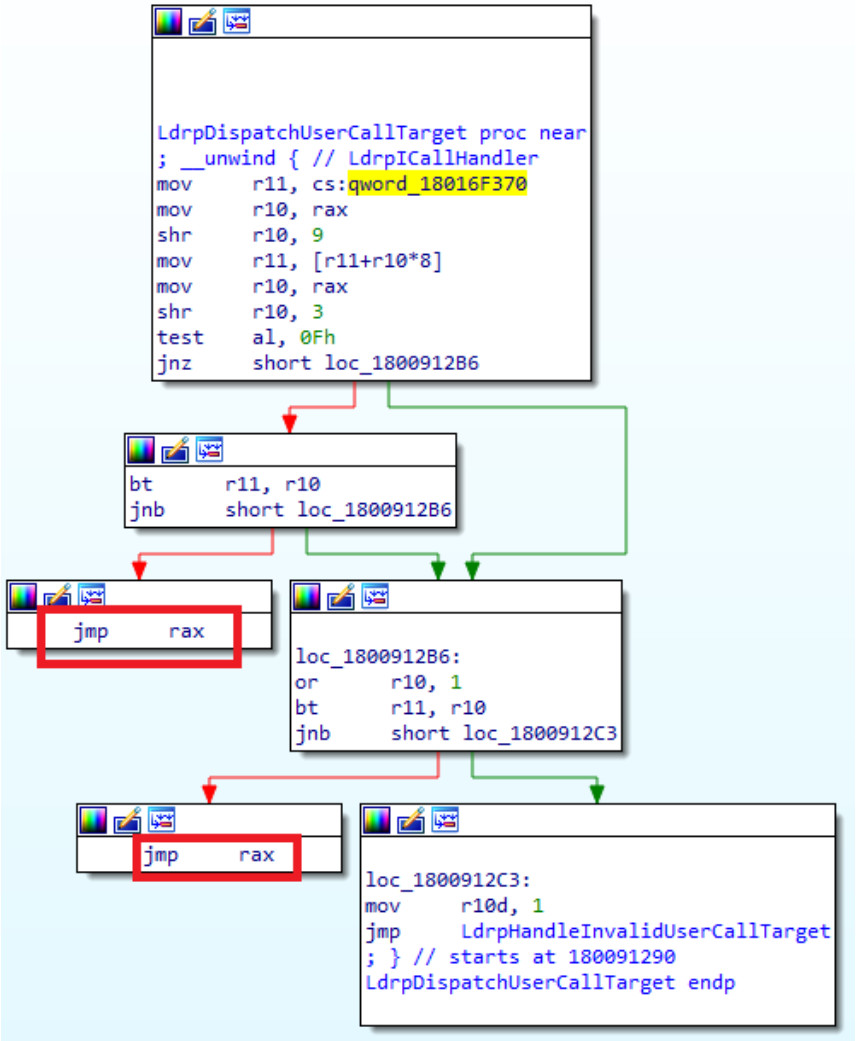
An Infinite running thread is needed

Looking at RtlUserThreadStart



An Infinite running thread is needed

Looking at RtlUserThreadStart



An Infinite running thread is needed

Running the infinitely running thread

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection

- Allocating RWX memory for **jmp RBX** opcode using **VirtualAllocEx**
- Looking for jump RBX opcode in our version of ntdll (opcode: 0xffe3)
- Copying jump RBX opcode in the target process using method described earlier
- Creating suspended thread using **CreateRemoteThread** function starting at **jmp RBX** opcode
- Setting **RBX** to point to **jmp RBX** opcode using **SetThreadContext**
- Resuming the thread

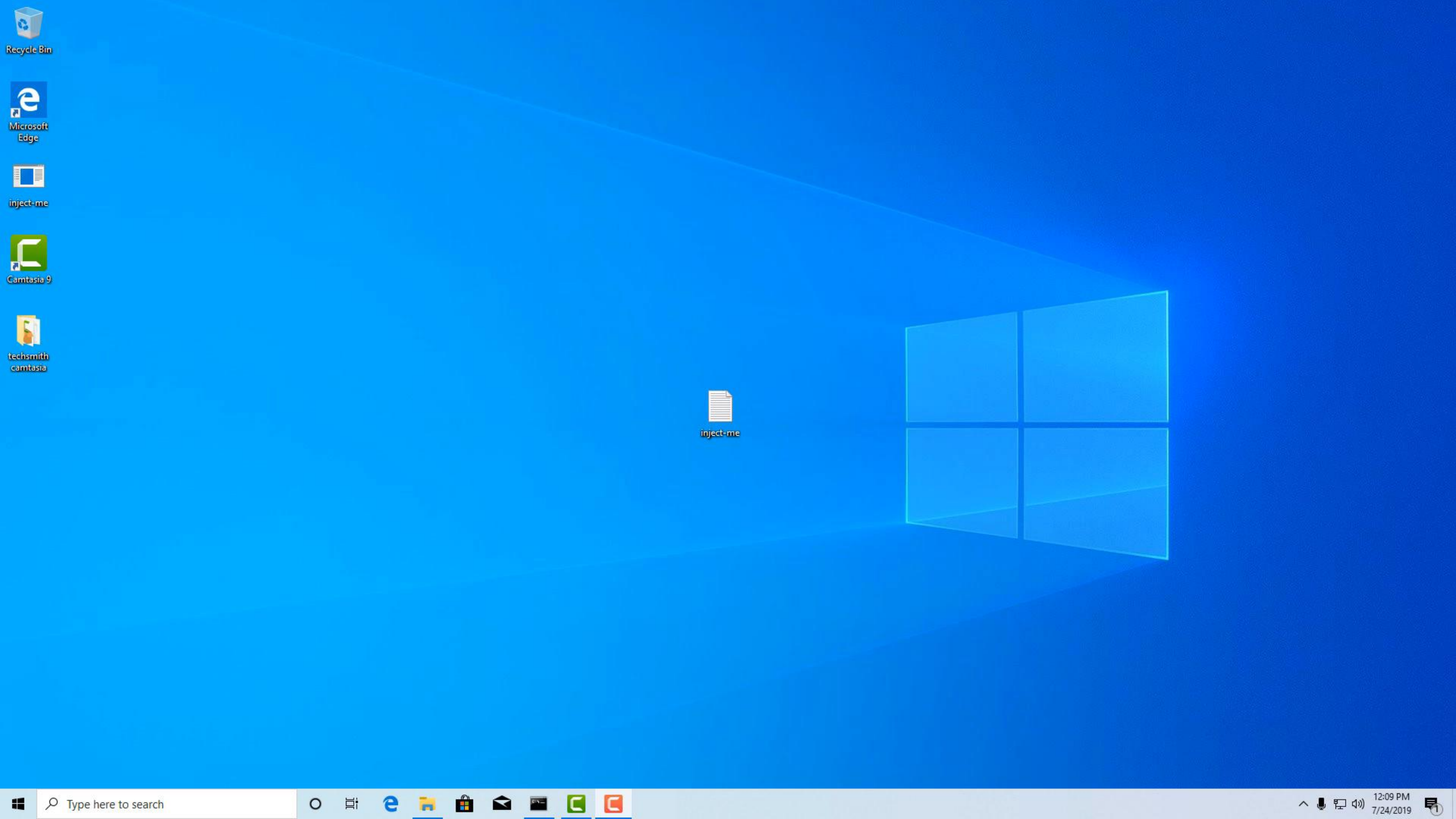
Executing the code injection

- Setting up ReadProcessMemory
- First problem Access violation
- Creating an infinite running thread
- Executing code injection

- Suspend the thread and check if **RIP** is at **jmp RBX** opcode address
- Setting the thread context using **SetThreadContext**
- Resuming the thread and waiting for the injection to occur
  - Using **WaitForSingleObject** to wait until the thread is done
- Executing the shellcode!

Demo

Please inject-me, a x64 code injection


Recycle Bin
Microsoft Edge
inject-me
Camtasia 9
techsmith camtasia
inject-me
Type here to search
12:09 PM
7/24/2019

Thank you!

For the full research paper visit this link | http://bit.ly/MeX64