Haow do I sandbox?!?!

Cuckoo Sandbox Internals

Jurriaan Bremer @skier_t

Student (University of Amsterdam), Freelance Security Researcher

June 22, 2013

Introduction - Cuckoo Sandbox Team

Figure : Mark Schloesser, Claudio Guarnieri, Me, Alessandro Tanasi

Introduction - What this talk is NOT about!

Figure : Dragon Sandbox!

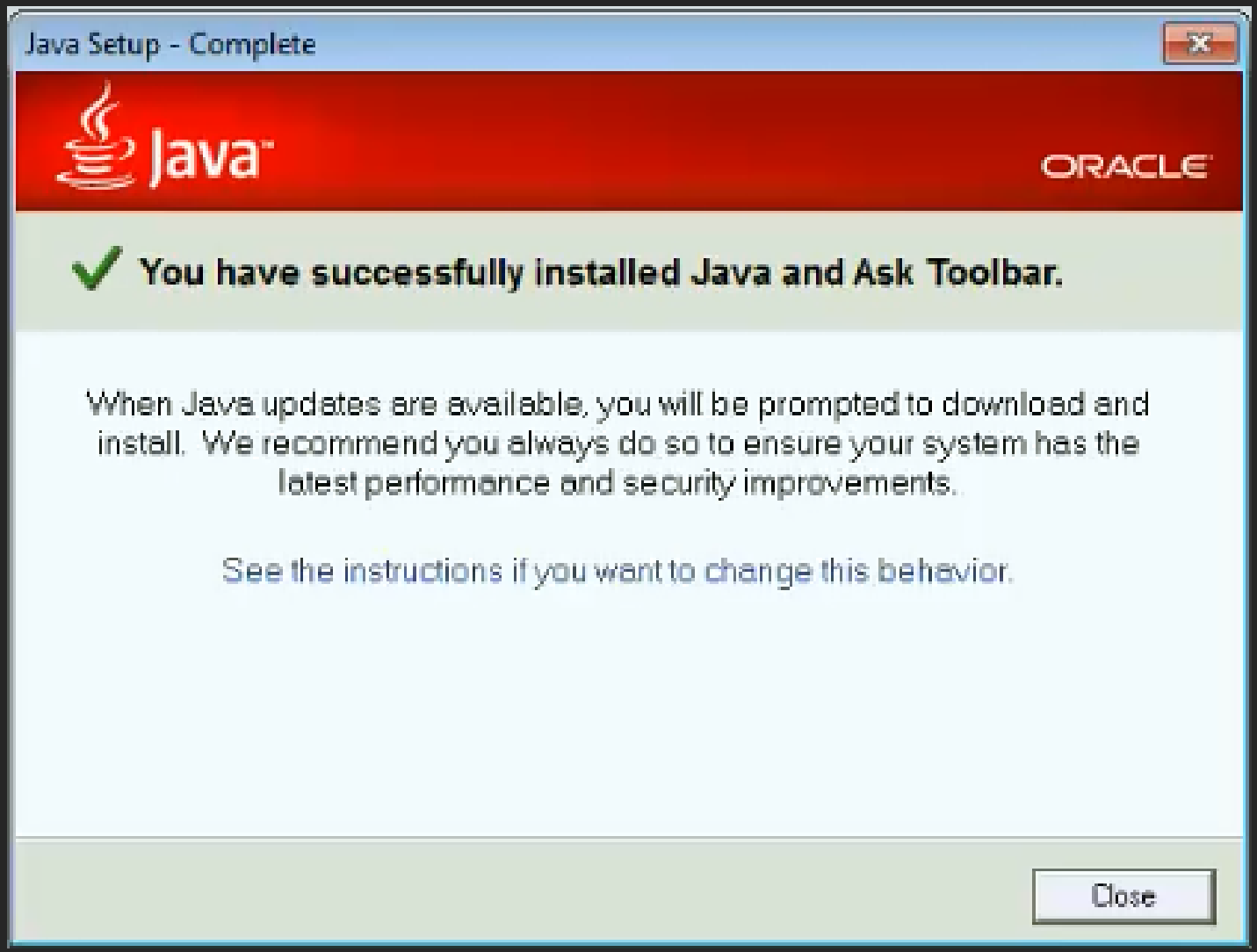
Introduction - What this talk is about!

- How we built Cuckoo
- How to evade Cuckoo
  - Left as an exercise for the attendee
  - Who would do such terrible thing though?

Introduction - Todays problems in Malware

- . . . Insert long list of problems . . .
- In the end, we prefer to blame..

Introduction - Todays problems in Malware



Introduction - Todays problems in Malware Analysis

- Static Analysis takes a lot of time
  - Obfuscation
  - Packers
- Dynamic Analysis also takes a lot of time
  - Multi-threaded malware
  - Anti-debugger, anti-virtual machine, etc.

Introduction - Sandboxing in General (1)

- Enter Sandboxes
  - Automated Malware Analysis - handles all repetitive work
    - Process thousands of samples in a reasonable time
  - Generic methods for bypassing anti's
- For the Client
  - User friendly - anyone can use it
  - Setup once, use it for eternity
    - For this step, see the manual :p

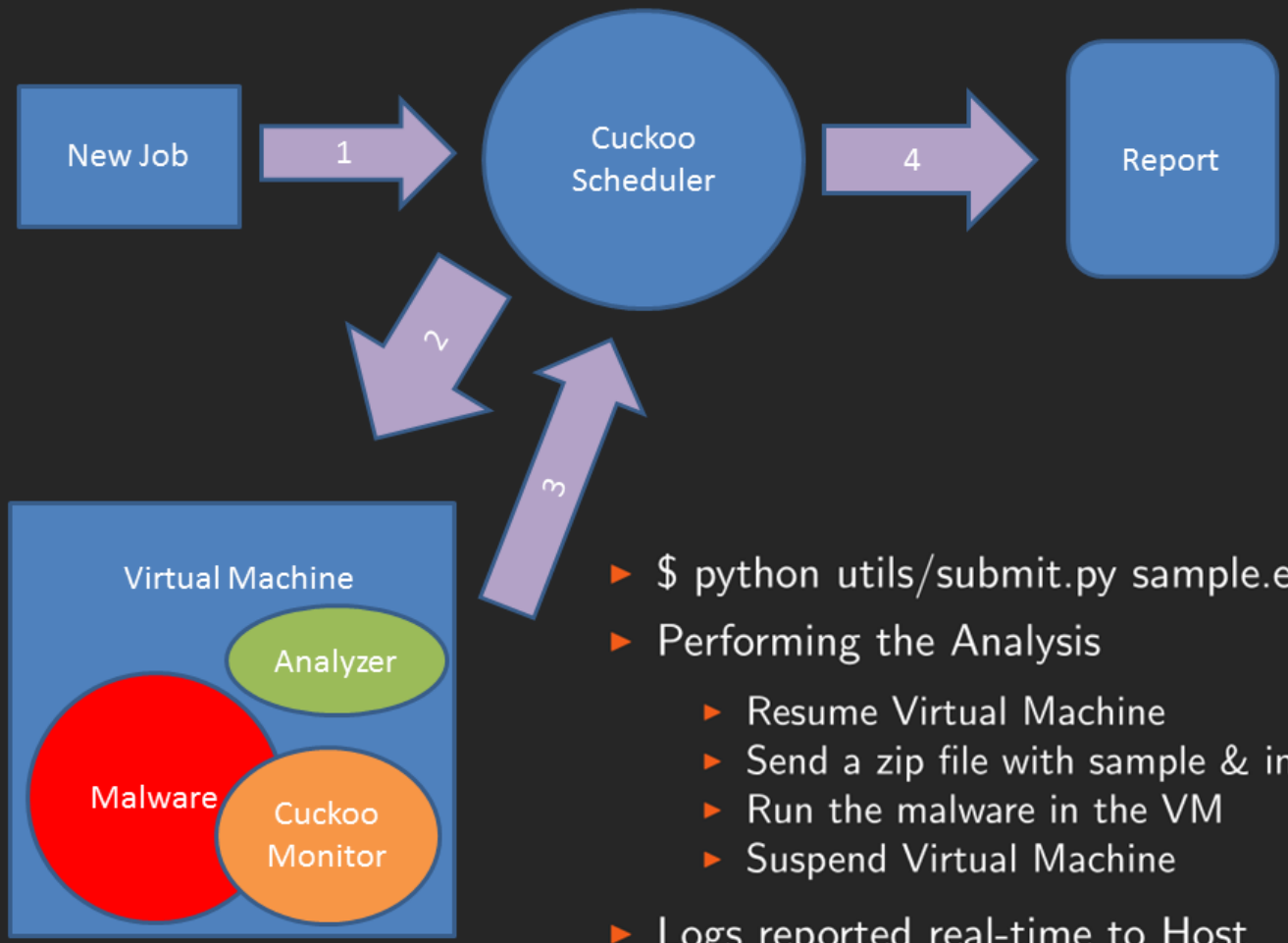
Introduction - Sandboxing in General (2)

- Existing Solutions
  - Closed Source
    - Not 100% customizable
  - Very expensive
- Enter Cuckoo Sandbox
  - Entirely Open Source
  - Free to use

Introduction - Disadvantages of Sandboxing

- Environment could be detected
  - Anti-sandbox
  - Randomize environment
    - Can only randomize so many things
- Various limitations depending on the implementation
  - We try our best to bypass these
    - E.g., Hook Detection by Malware
- Reports still have to be read by somebody

Cuckoo Sandbox Architecture

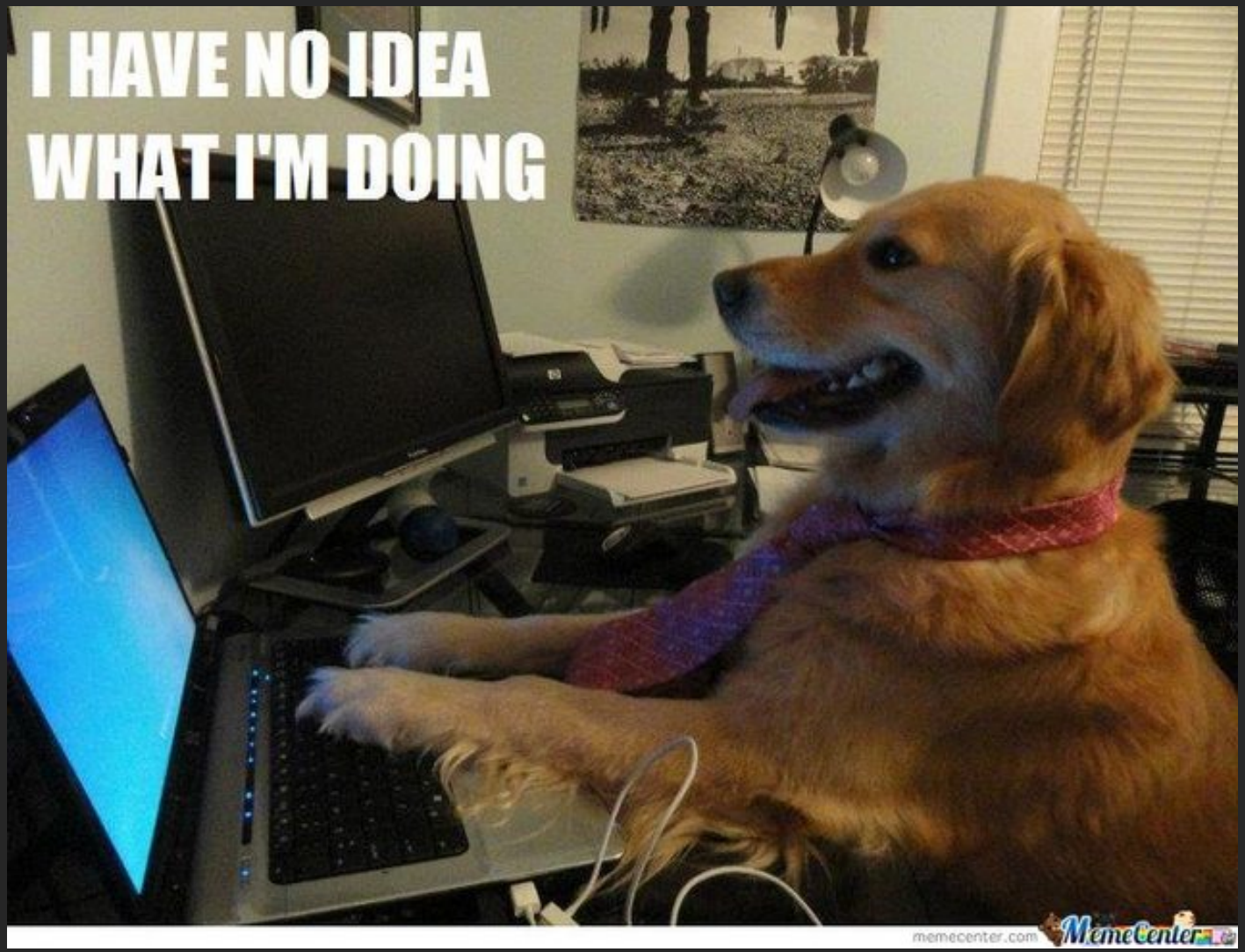


- $ python utils/submit.py sample.exe
- Performing the Analysis
  - Resume Virtual Machine
  - Send a zip file with sample & info
  - Run the malware in the VM
  - Suspend Virtual Machine
- Logs reported real-time to Host
- Generate a Report

Demonstration of analyzing a PDF exploit

- Demo showing the entire analysis process
- Quick look through the report

Cuckoo Sandbox Internals



Inside the Virtual Machine - Agent

- Listening Agent
  - Accepts a connection
  - Host connects
  - Host sends zip file
- Agent unpacks zip file
  - Python code
    - Easily upgrade Cuckoo to a new version!
  - Configuration files
  - The sample
- Agent runs the Analyzer
  - Which has been sent through the zip

Inside the Virtual Machine - Analyzer

- Analyzer
  - Initializes Cuckoo stuff
    - Open IPC Channel (Named Pipe)
    - Some handwaving etc
  - Dumps Configuration for the first Process
    - Name of the Named Pipe
    - IP and Port of the Result Server
    - (Will come back to that later)
  - Runs the specified Package

Inside the Virtual Machine - Packages

- Package starts an application with commandline parameters
  - Wrappers around `CreateProcess(CREATE_SUSPENDED)`
    - Packages for EXE, DLL, PDF, DOC, etc.
  - Inject Cuckoo Monitor DLL into the process
    - Using APC, `QueueUserAPC(...)`
  - Resume main thread of the process

Inside the Application - Cuckoo Monitor

- When resuming the main thread
  - Cuckoo Monitor is executed first
    - Due to the APC callback
  - Initializes internals & installs API Hooks
  - Notifies the Analyzer
    - Through Named Pipes
  - Real application is started

Outside the Virtual Machine - Result Server

- Cuckoo Monitor logs directly to the Host, over TCP/IP
  - IP and Port retrieved from the Configuration
- More stability than before, when we logged to a local file
  - VM Crashes resulted in no logs
  - Now real-time results

So, what now?

- We've covered the basics
- Useful for single-process stuff
- What's next?

More Advanced Malware (1)

- Some samples run new processes
  - RunPE, for Packers
  - Internet Exploder^WExplorer for URLs
- Some malware injects into other processes
  - Explorer.exe Injection to evade Firewalls
  - Banking Trojans

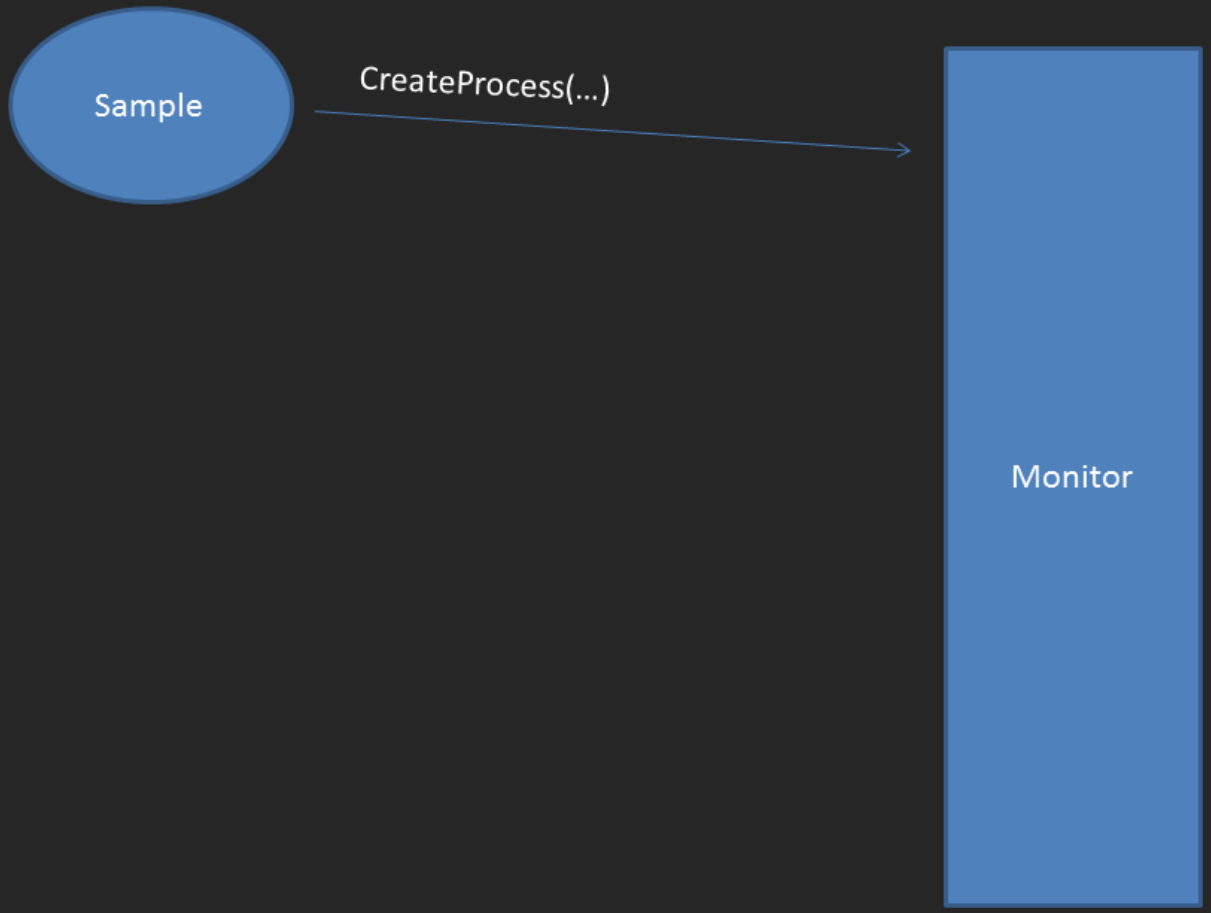
Child Process Injection

Before the new Process is executed, we want to inject Cuckoo Monitor.

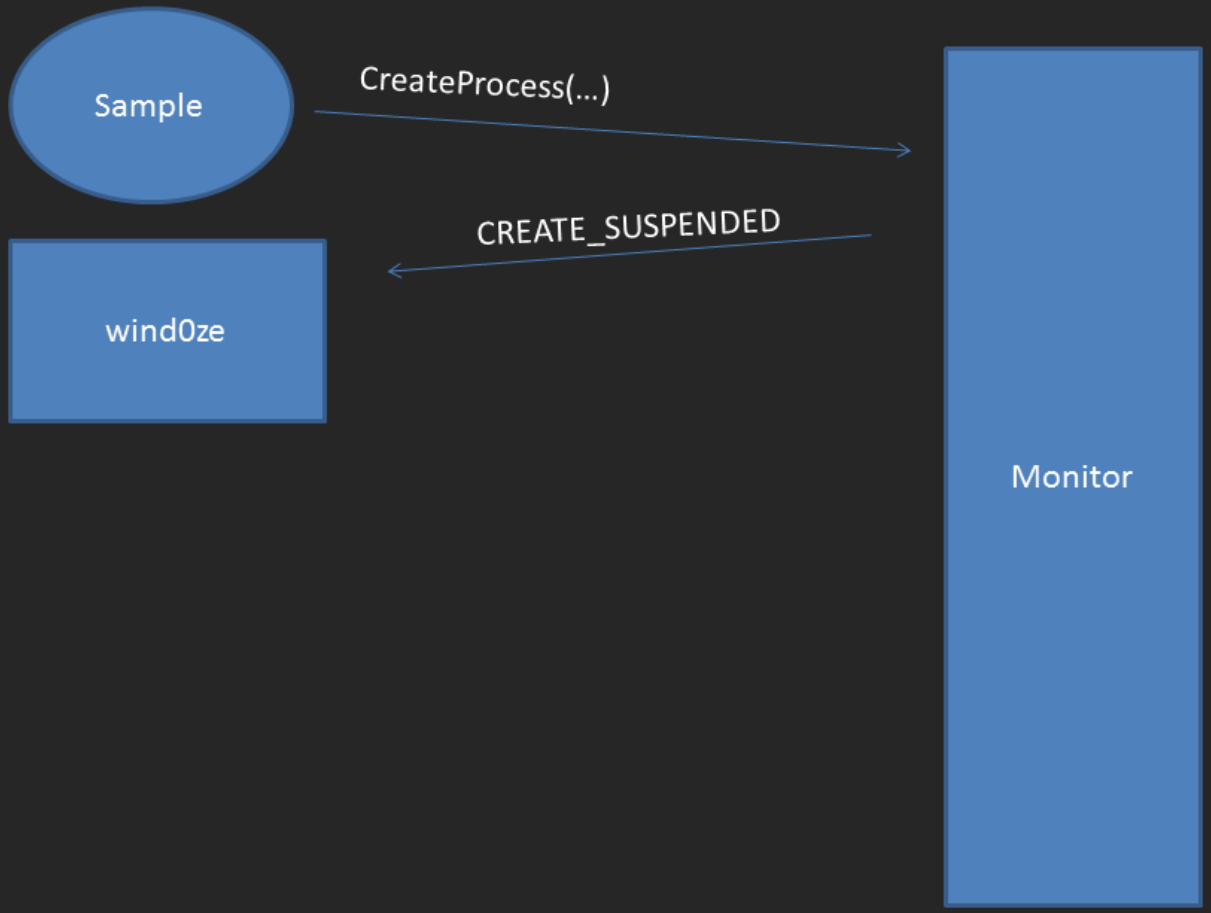
- Cuckoo Monitor notifies Analyzer
  - Asks to be injected into the target process
  - Analyzer dumps configuration file
  - Injection using APC

Child Injection

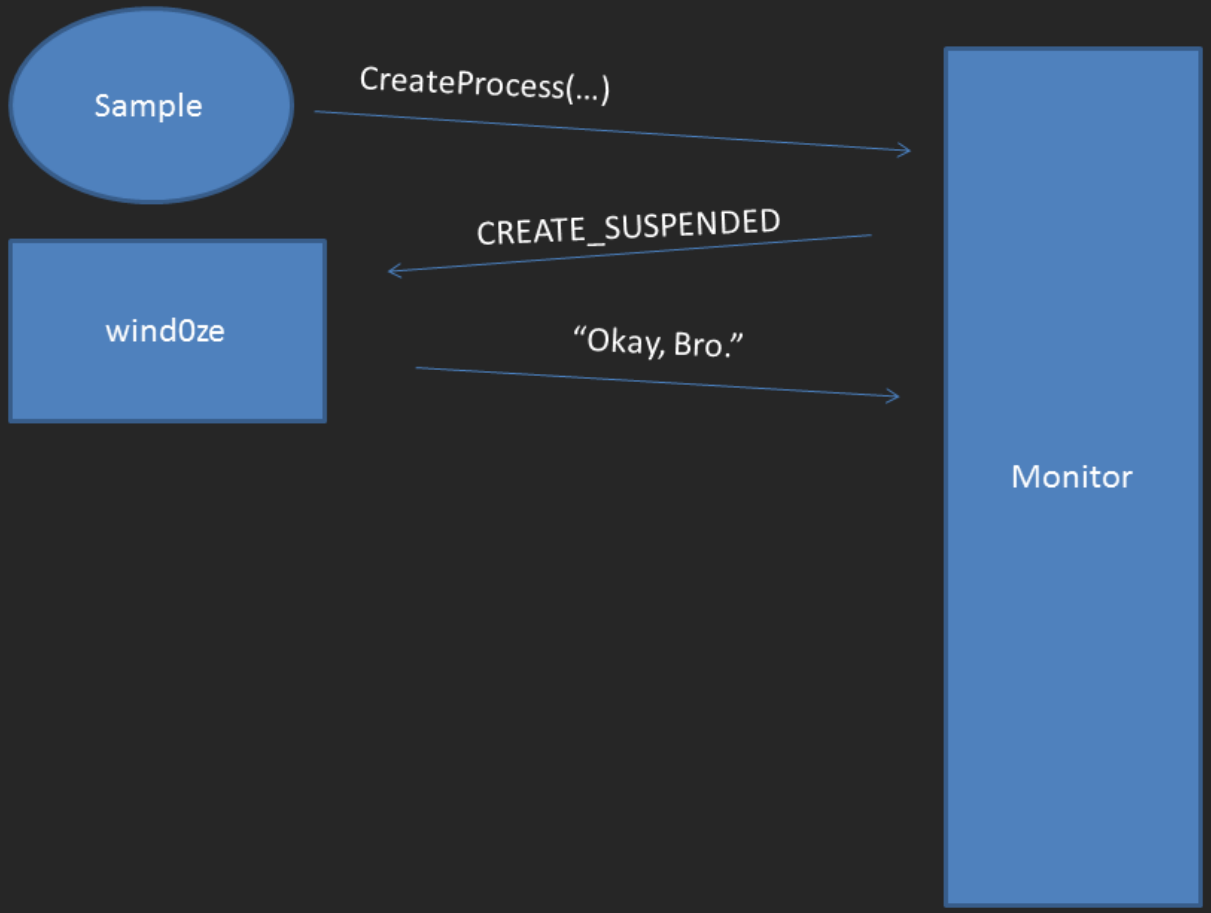
Child Injection



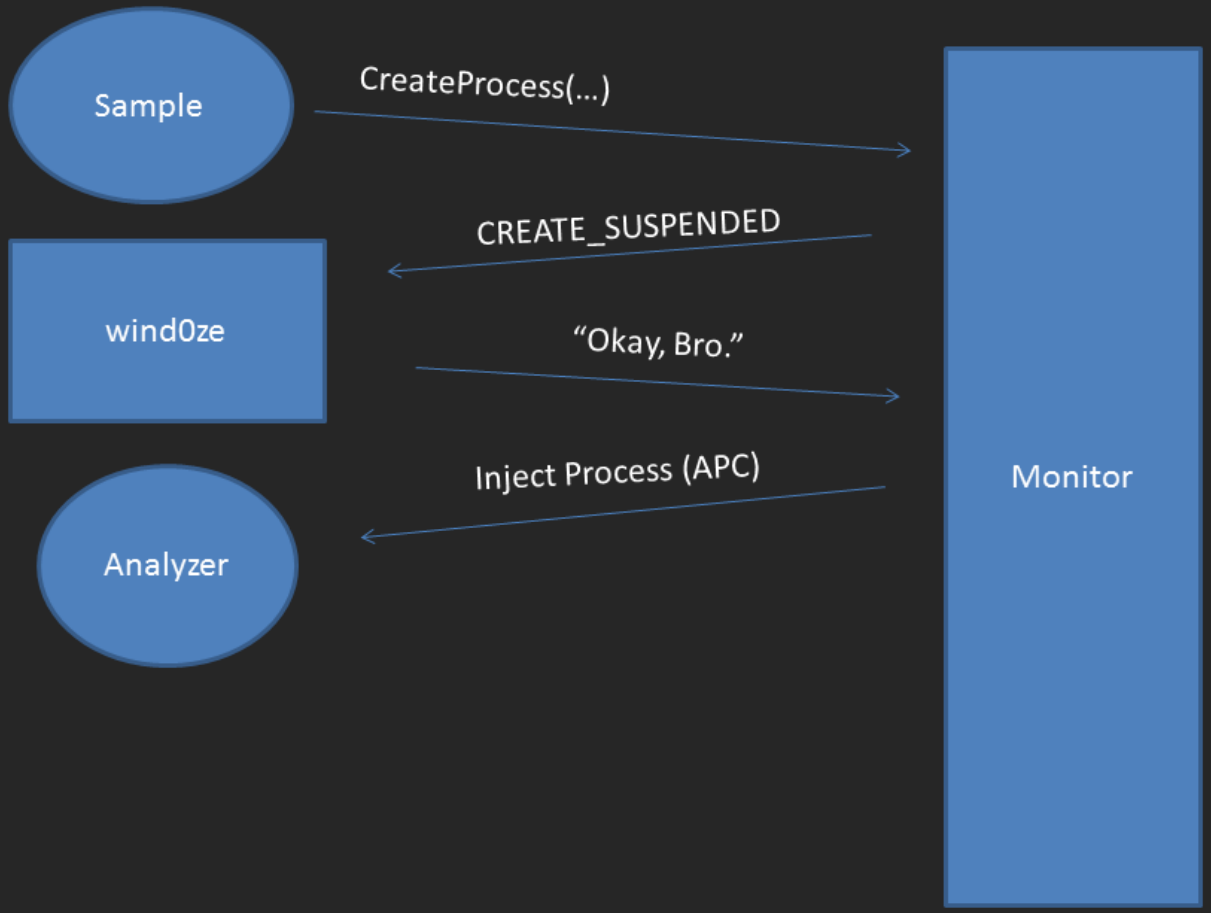
Child Injection



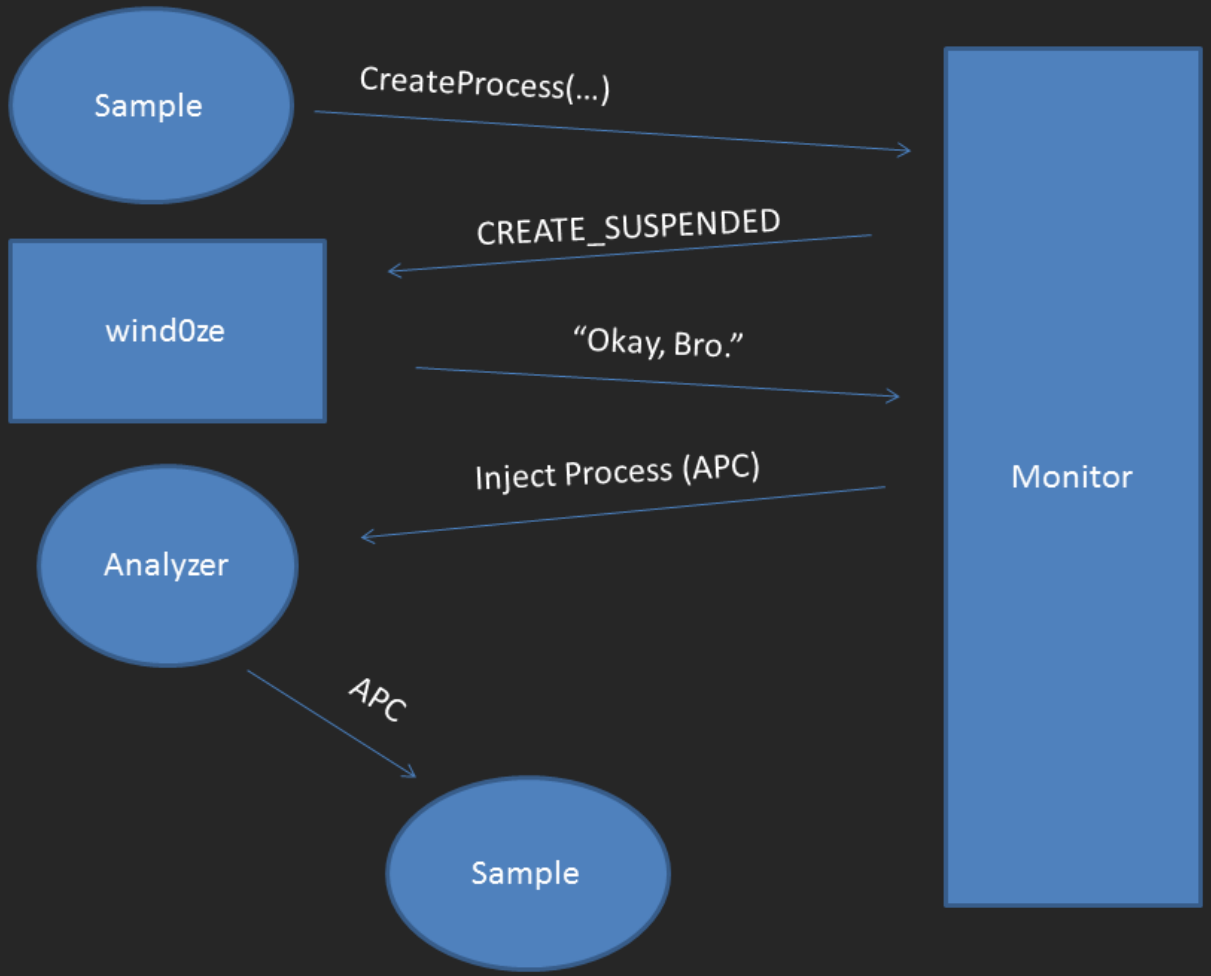
Child Injection



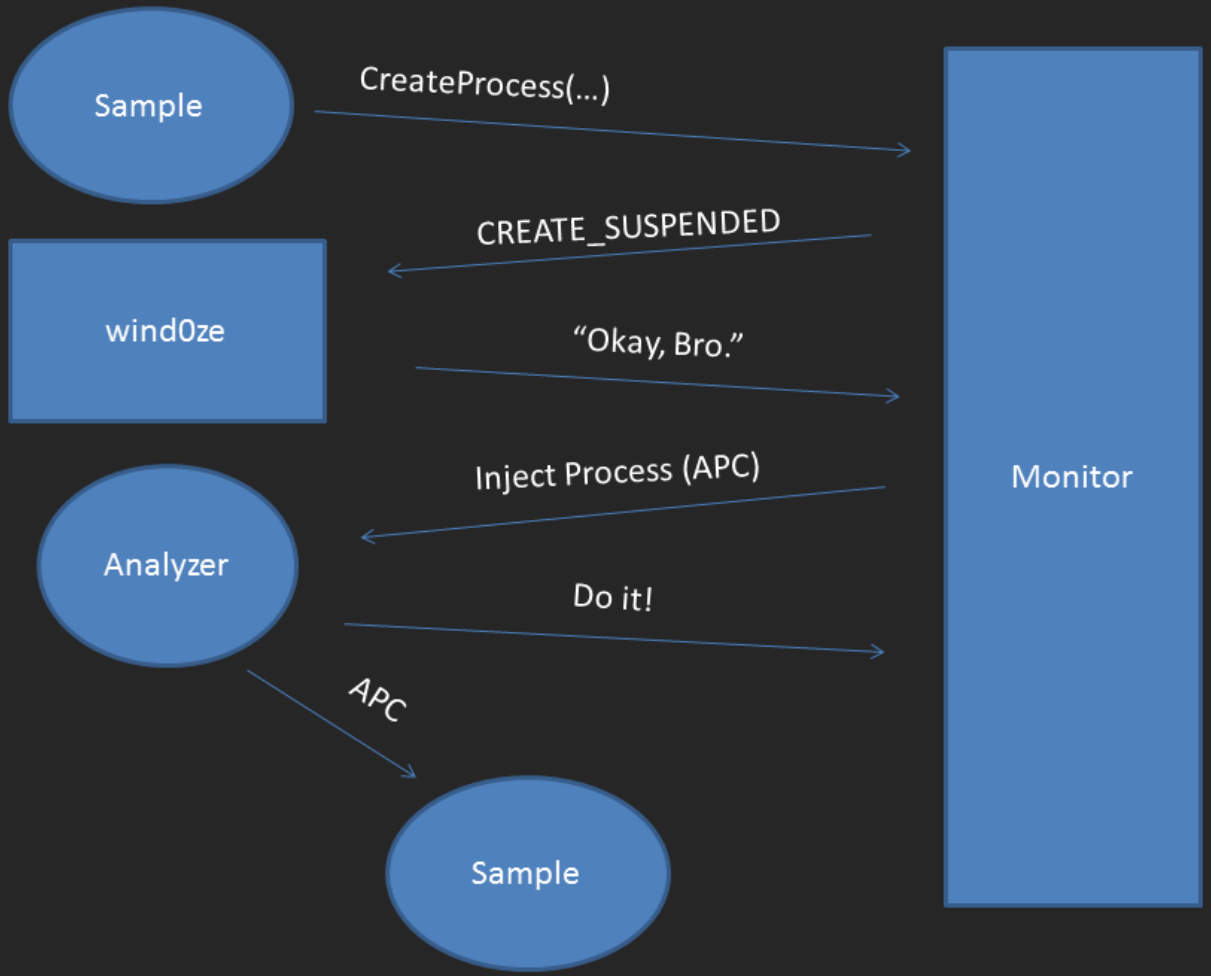
Child Injection



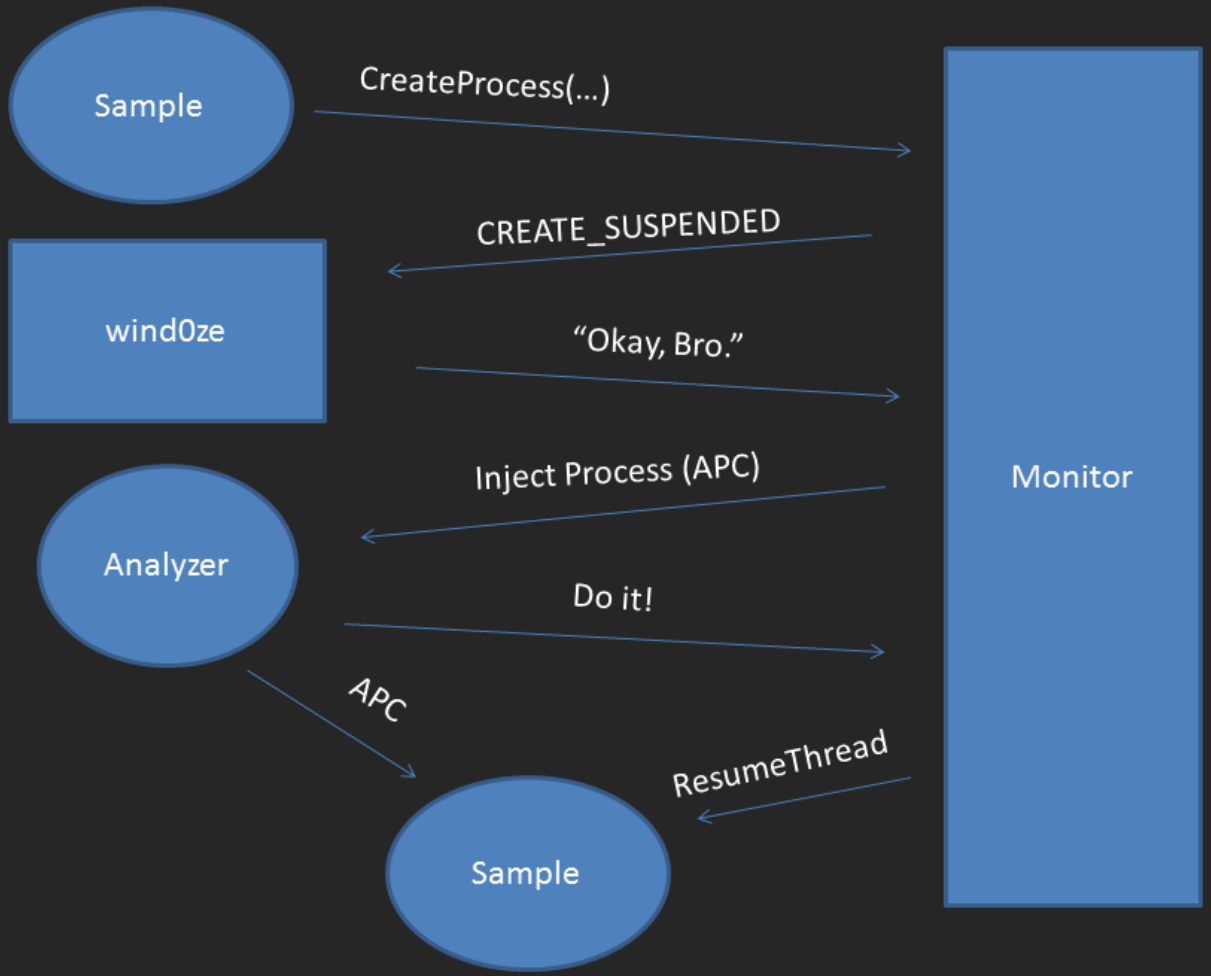
Child Injection



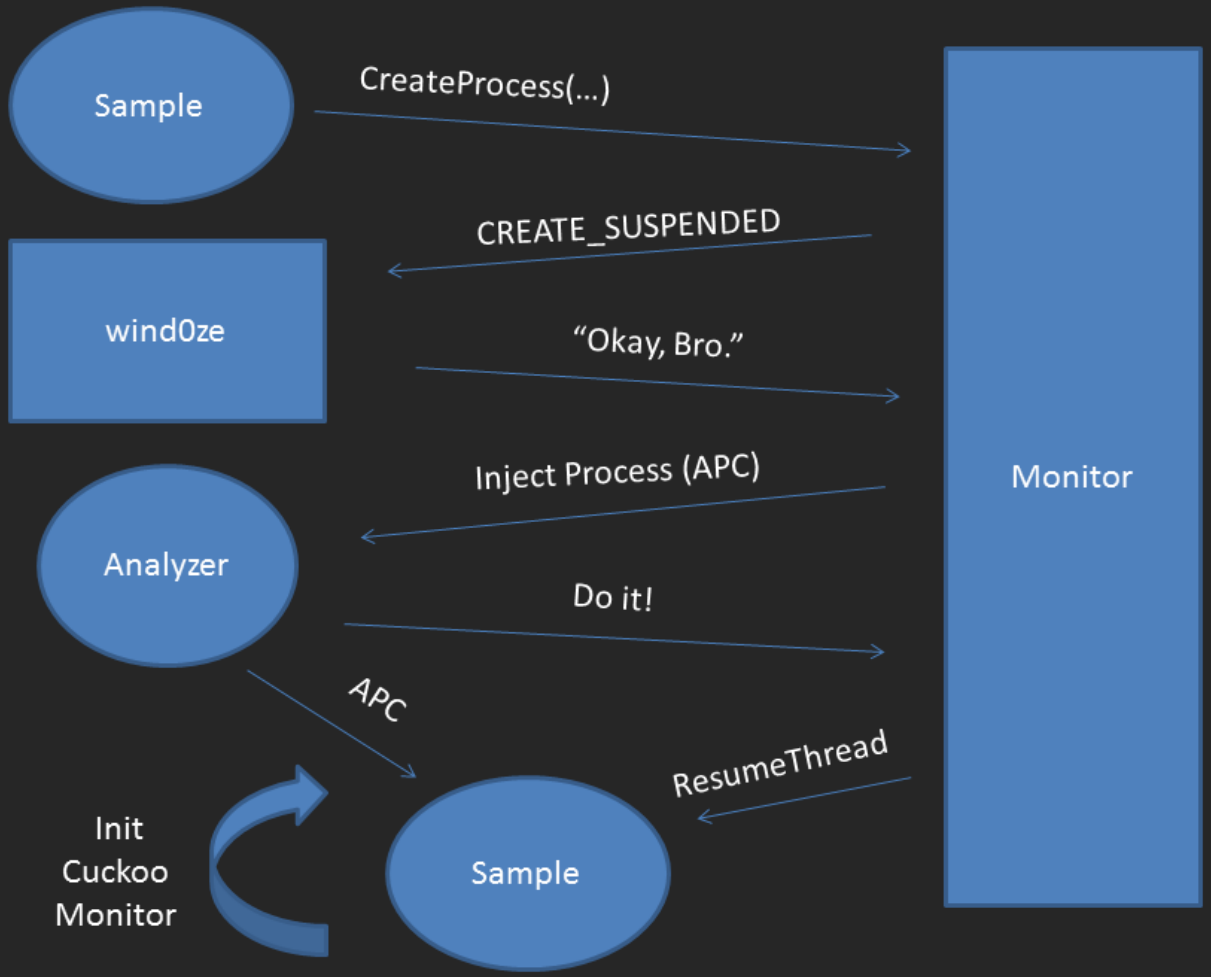
Child Injection



Child Injection



Child Injection



Process Injection

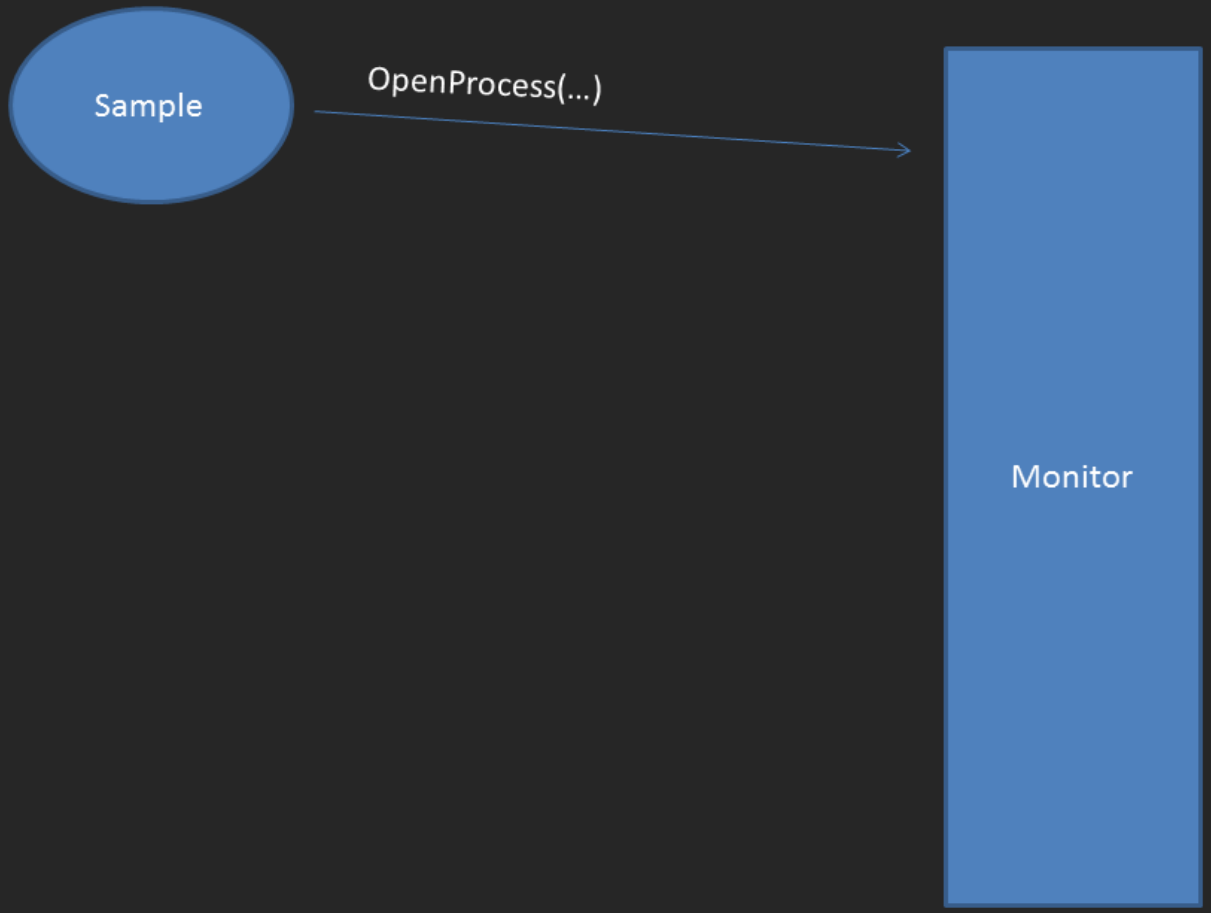
Before a sample injects and executes code into another process, we also want to inject Cuckoo Monitor.

Process Injection is similar to Child Injection, except for a few steps.

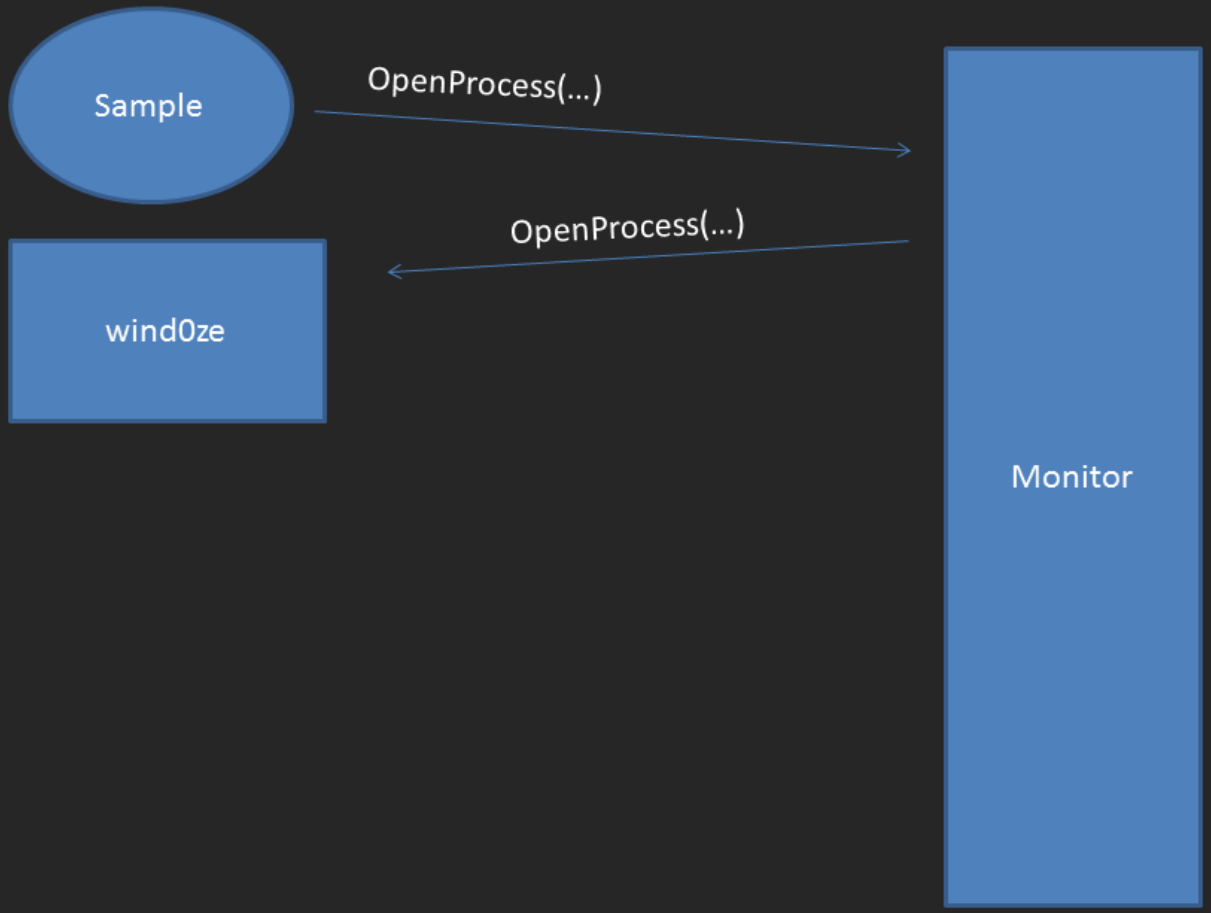
- No APC, but `CreateRemoteThread(...)`
  - Can't guarantee APC finishes in time
- Entirely inject Cuckoo Monitor before resuming execution
  - For Child Processes

Process Injection

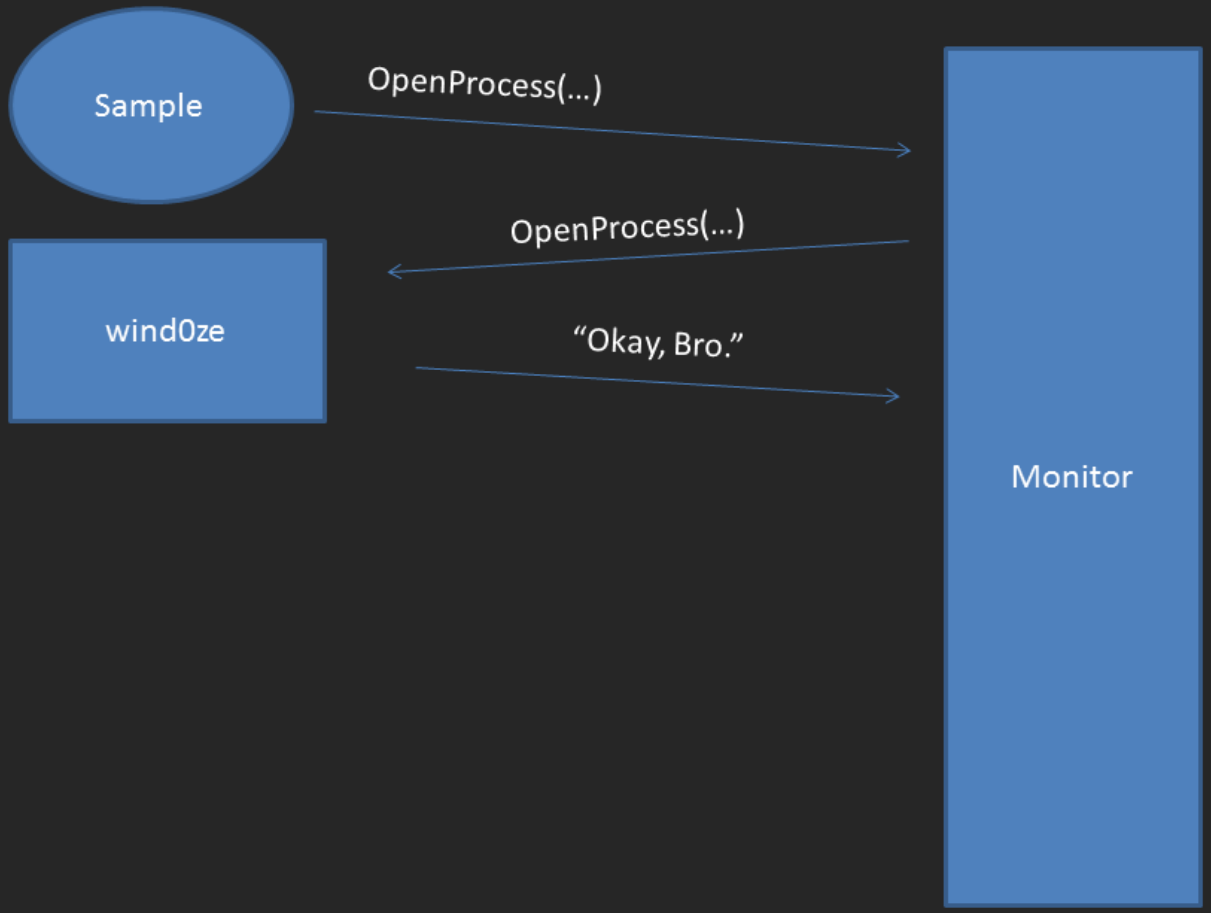
Process Injection



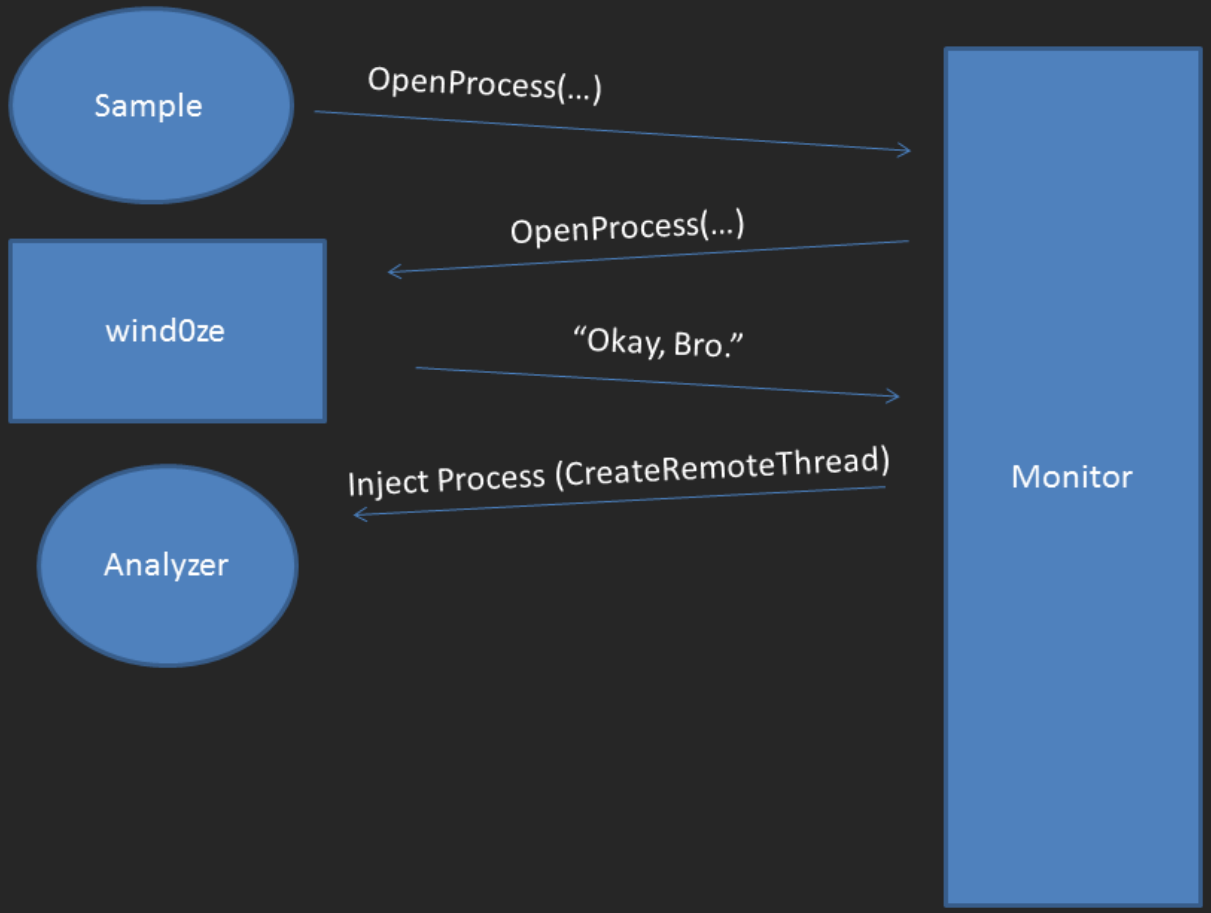
Process Injection



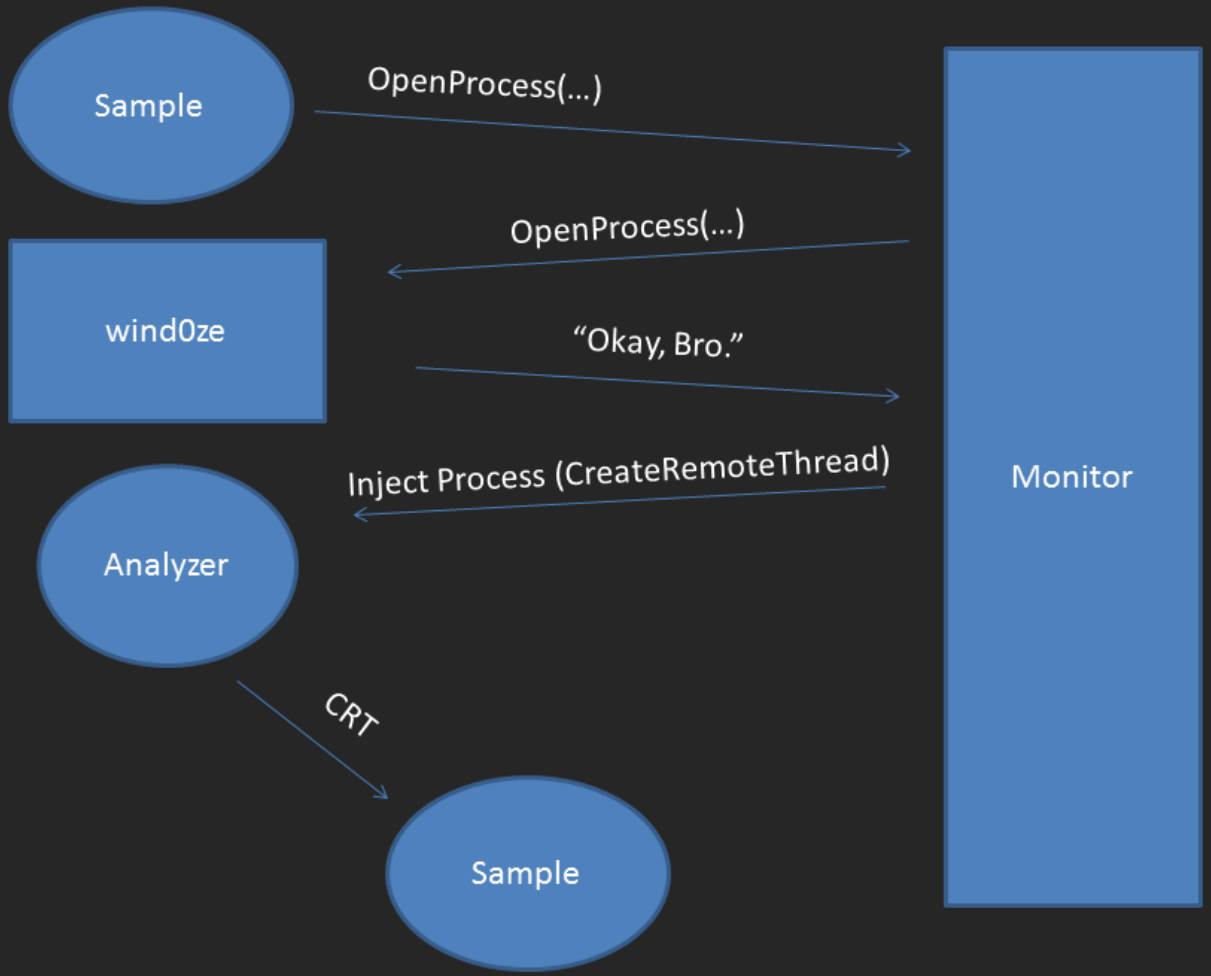
Process Injection



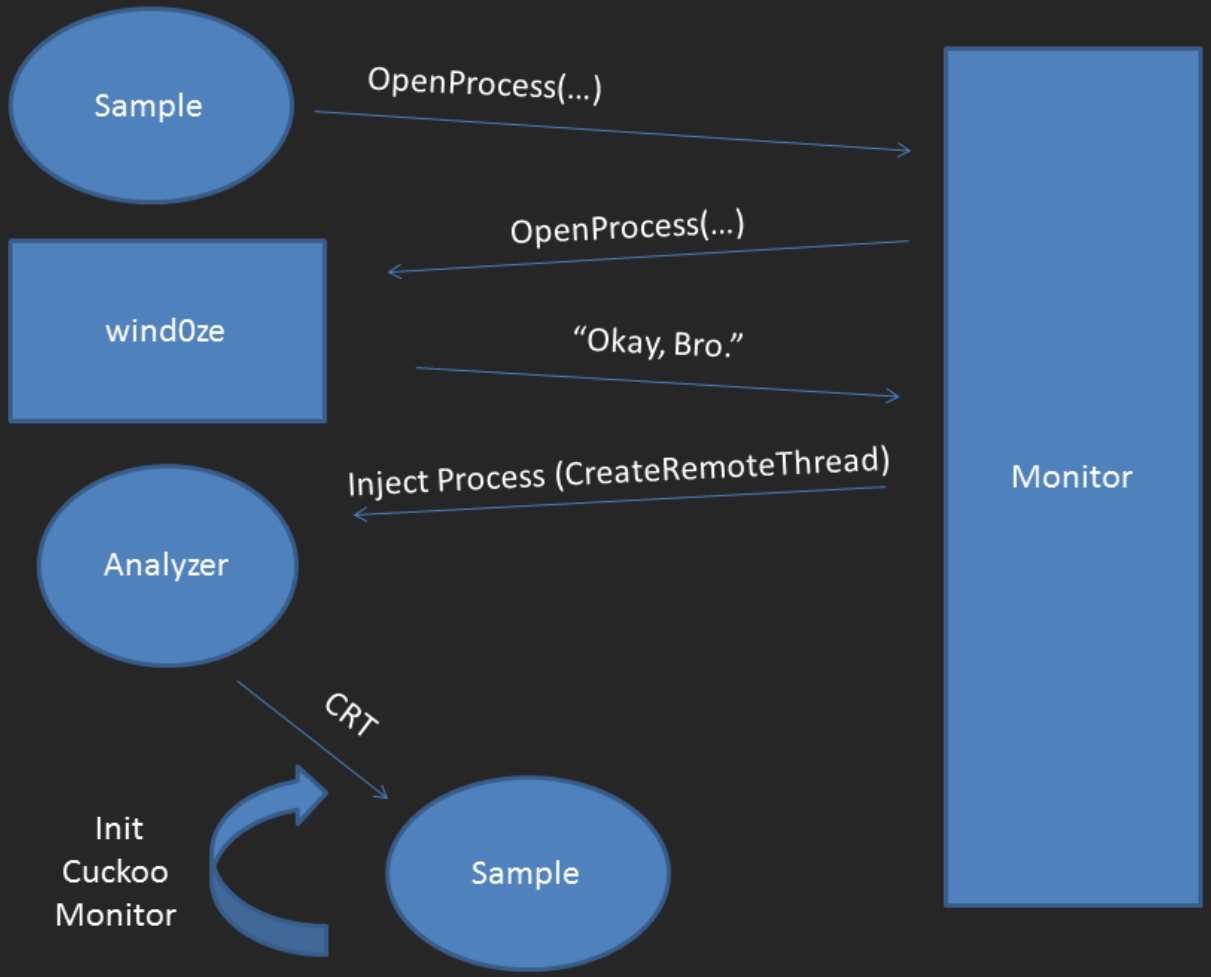
Process Injection



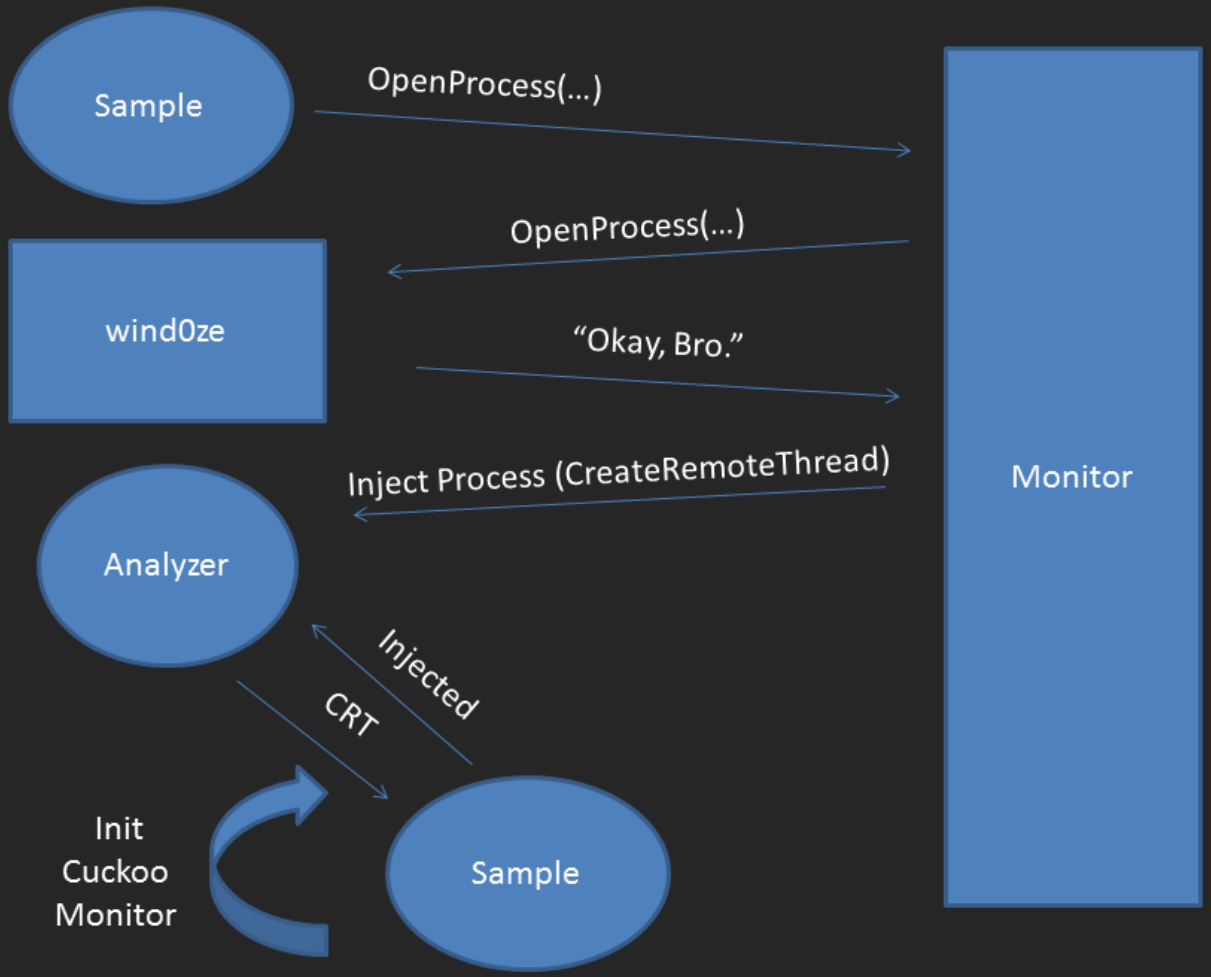
Process Injection



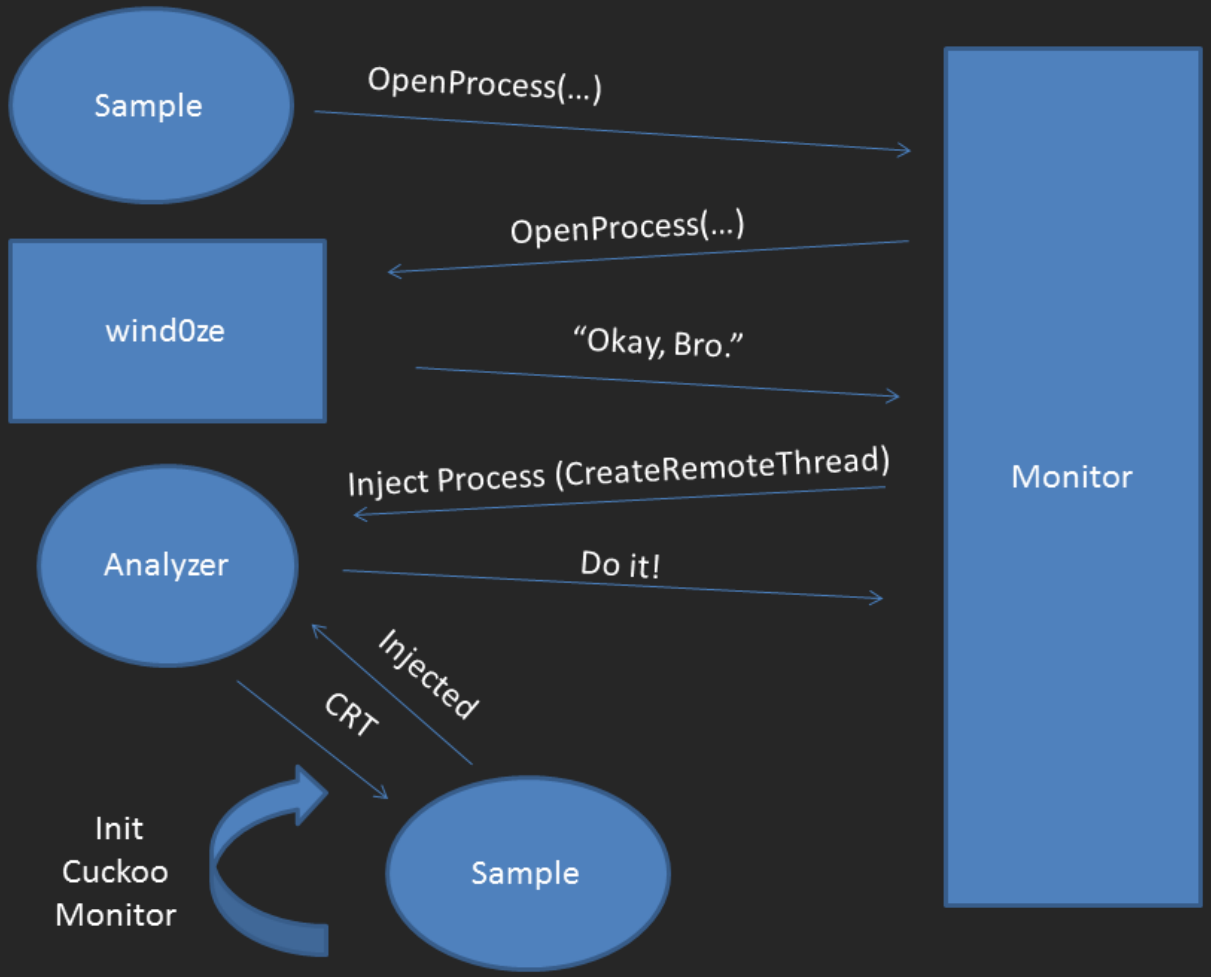
Process Injection



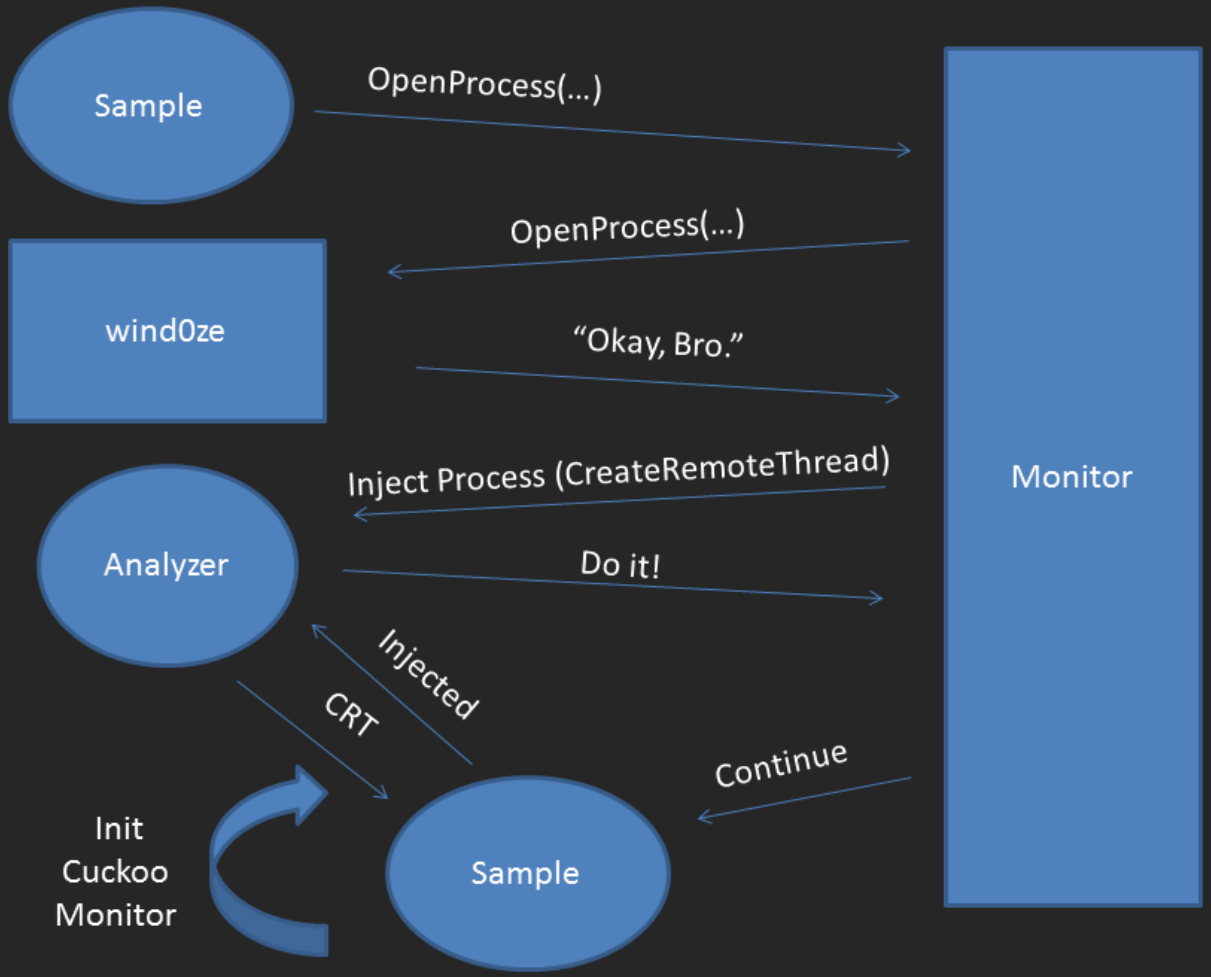
Process Injection



Process Injection



Process Injection



That said..

Figure : What the malware thinks it's doing.

That said..

Figure : What Cuckoo Sandbox thinks it's doing.

That said..

Figure : What really happens.

API Hooking - Overview

- Core functionality of Cuckoo Monitor
- Cuckoo Monitor logs about 170 APIs
  - We're adding APIs where needed
- Hooks lowest level APIs without loosing context
  - Not `CreateProcessA`
  - Not `CreateProcessW`
  - Not `CreateProcessInternalA`
  - But `CreateProcessInternalW`
- However, we also hook higher-level APIs
  - `ShellExecute` (supports protocol handlers, URLs, ..)
  - `system` (can pipe multiple processes)

API Hooking - Trampolines (1)

- Redirect execution using trampolines
  - Create a trampoline
  - Patch the function

http://jbremer.org/x86-api-hooking-demystified/

API Hooking - Trampolines (2)

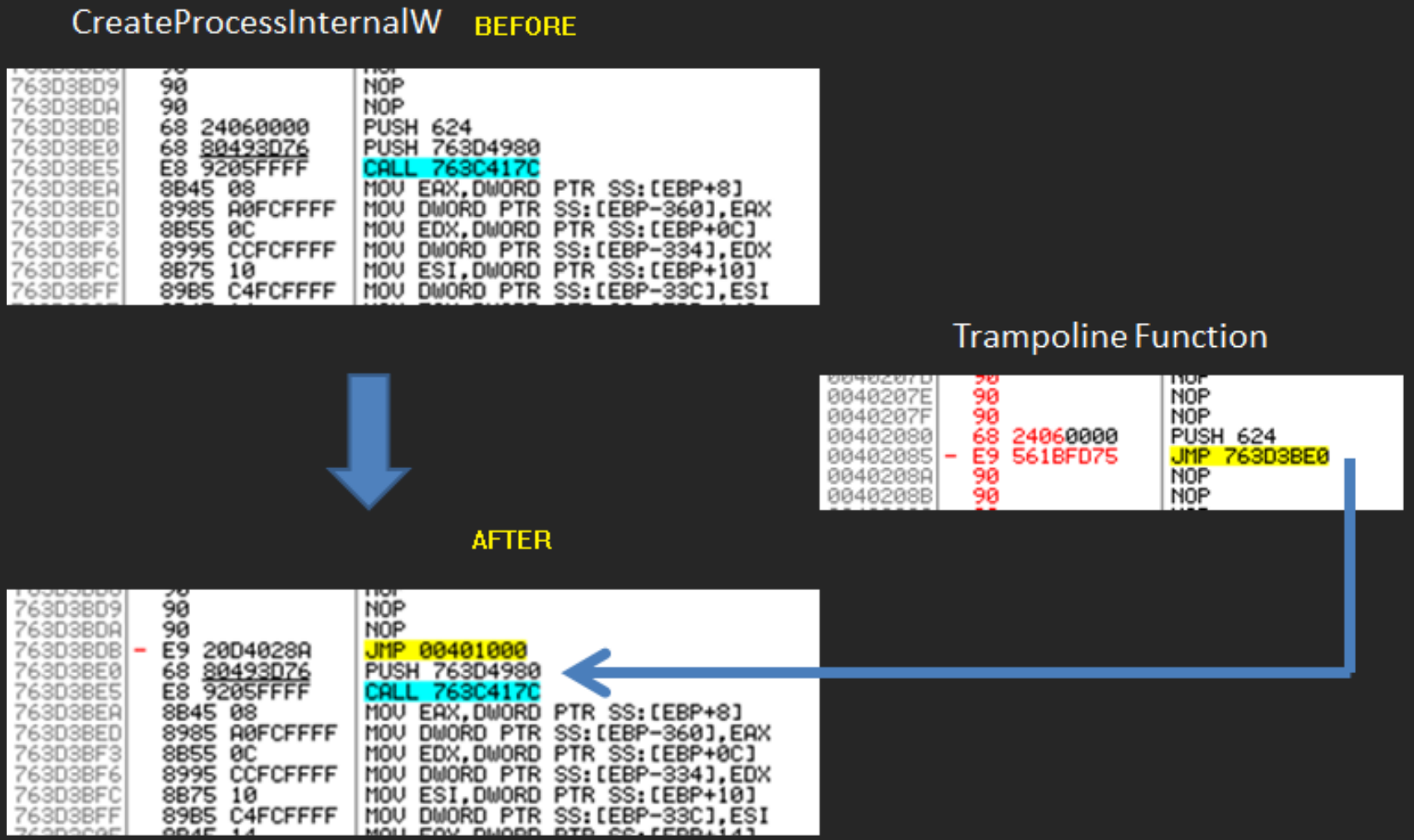


Figure : Trampolines are really basic.

API Hooking - Trampolines (3)

```
HOOKDEF(BOOL, WINAPI, CreateProcessInternalW,
    __in_opt    LPVOID lpUnknown1,
    __in_opt    LPWSTR lpApplicationName,
    __inout_opt LPWSTR lpCommandLine,
    __in_opt    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in_opt    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in        BOOL bInheritHandles,
    __in        DWORD dwCreationFlags,
    __in_opt    LPVOID lpEnvironment,
    __in_opt    LPWSTR lpCurrentDirectory,
    __in        LPSTARTUPINFO lpStartupInfo,
    __out       LPPROCESS_INFORMATION lpProcessInformation,
    __in_opt    LPVOID lpUnknown2
) {
    [...]

    BOOL ret = Old_CreateProcessInternalW(lpUnknown1, lpApplicationName,
        lpCommandLine, lpProcessAttributes, lpThreadAttributes,
        bInheritHandles, dwCreationFlags, lpEnvironment, lpCurrentDirectory,
        lpStartupInfo, lpProcessInformation, lpUnknown2);

    [...]
}
```

Figure : A day in the life of.. a hooked API.

API Hooking - Avoiding Hook Recursion (1)

```
HOOKDEF(BOOL, WINAPI, WriteFile,
    _In_          HANDLE hFile,
    _In_          LPCVOID lpBuffer,
    _In_          DWORD nNumberOfBytesToWrite,
    _Out_opt_     LPDWORD lpNumberOfBytesWritten,
    _Inout_opt_   LPOVERLAPPED lpOverlapped
) {

    [...]

    WriteFile(g_log_handle, "Hello Hook", 10, &bytes, NULL);

    [...]
}
```

Figure : Hello Hook?

API Hooking - Avoiding Hook Recursion (2)

- The first hooked API call is interesting, ignore the others.
  - Sounds easy enough right?
- Around 170 hooks.
  - Can't add code to each hook.
    - We're not coding for our local University.
- Solution: Transparently in the hooking mechanism.

API Hooking - Avoiding Hook Recursion (3)

- We need a counter
  - Zero -> execute the hook handler
  - Not Zero -> ignore this API call
- Let's go back to `WriteFile()`
  - count = 0
    - Increase counter
    - Execute the Hook Handler
  - count = 1
    - Ignore the Hook Handler

API Hooking - Avoiding Hook Recursion (4)

- We need one counter per thread
  - Thread Local Storage it is
- Increase it before executing the hook handler
- Decrease it after returning from the hook handler
  - Oh, we have to run our code after the hook handler returns
  - So we have to patch the return address
  - Oh, we have to store the original return address temporarily
    - TLS to the rescue?
- More on this later.

API Hooking - Get Last Error (1)

- Thread-specific Error Value, equivalent to `errno`
- Let's assume `CreateProcessInternalW()` returns failure
  - However, logging the failure is successful
  - Great!
- Last Error is stored in TLS as well
- After calling the trampoline function, we copy the Last Error
  - (Right before execution goes back to the hook handler)

API Hooking - Get Last Error (2)

```
HOOKDEF(BOOL, WINAPI, CreateProcessInternalW,
    __in_opt    LPVOID lpUnknown1,
    __in_opt    LPWSTR lpApplicationName,
    __inout_opt LPWSTR lpCommandLine,
    __in_opt    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in_opt    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in        BOOL bInheritHandles,
    __in        DWORD dwCreationFlags,
    __in_opt    LPVOID lpEnvironment,
    __in_opt    LPWSTR lpCurrentDirectory,
    __in        LPSTARTUPINFO lpStartupInfo,
    __out       LPPROCESS_INFORMATION lpProcessInformation,
    __in_opt    LPVOID lpUnknown2
) {
    [...]

    BOOL ret = Old_CreateProcessInternalW(lpUnknown1, lpApplicationName,
        lpCommandLine, lpProcessAttributes, lpThreadAttributes,
        bInheritHandles, dwCreationFlags, lpEnvironment, lpCurrentDirectory,
        lpStartupInfo, lpProcessInformation, lpUnknown2);

    [...]
}
```

Figure : Example `CreateProcessInternalW` hook.

API Hooking - Get Last Error (3)

- We have to temporarily backup the Last Error
  - Until the function returns, where we restore it
- TLS anyone?

API Hooking - Special Hooks (1)

- What about our Advanced Persistent Hooks?
- Some hook handlers should always be executed
  - Special `CreateProcessInternalW()`
    - Somebody has to inject those `system()`'d processes
  - (The normal `CreateProcessInternalW()` only logs)

API Hooking - Special Hooks (2)

- Treated as another hook
  - Special hook hooks the target function first
    - Normal hook hooks the Special hooks' hook (oboy)
  - Special hooks keeps its own data (Last Error, count, ...)

API Hooking - Result

Please enter Brainfart mode now.

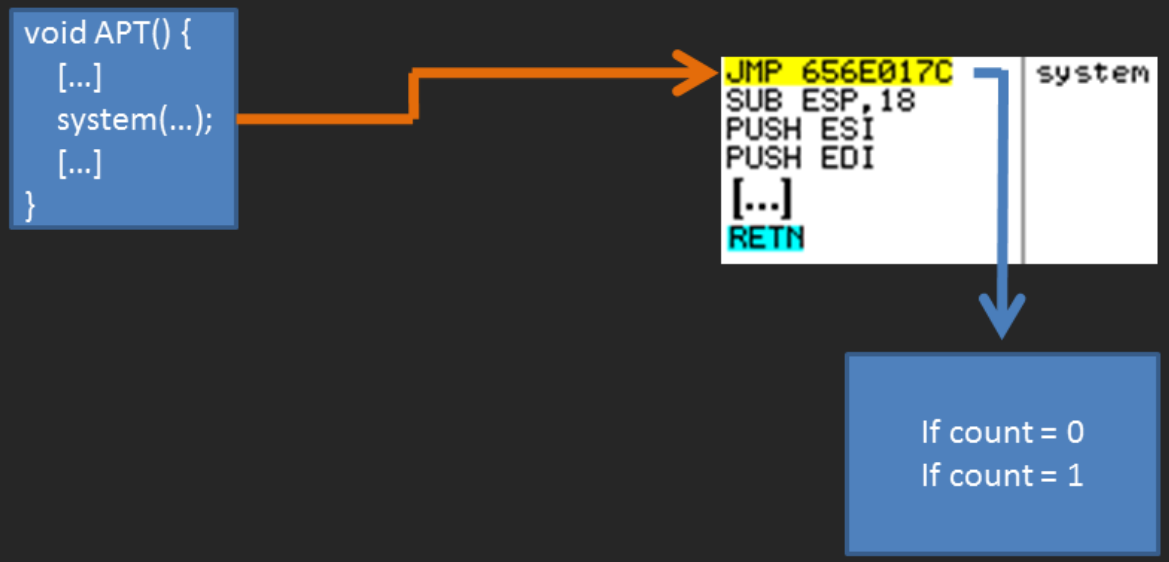
The following represents a `system()` hook as if it were the only hook.

API Hooking - Result

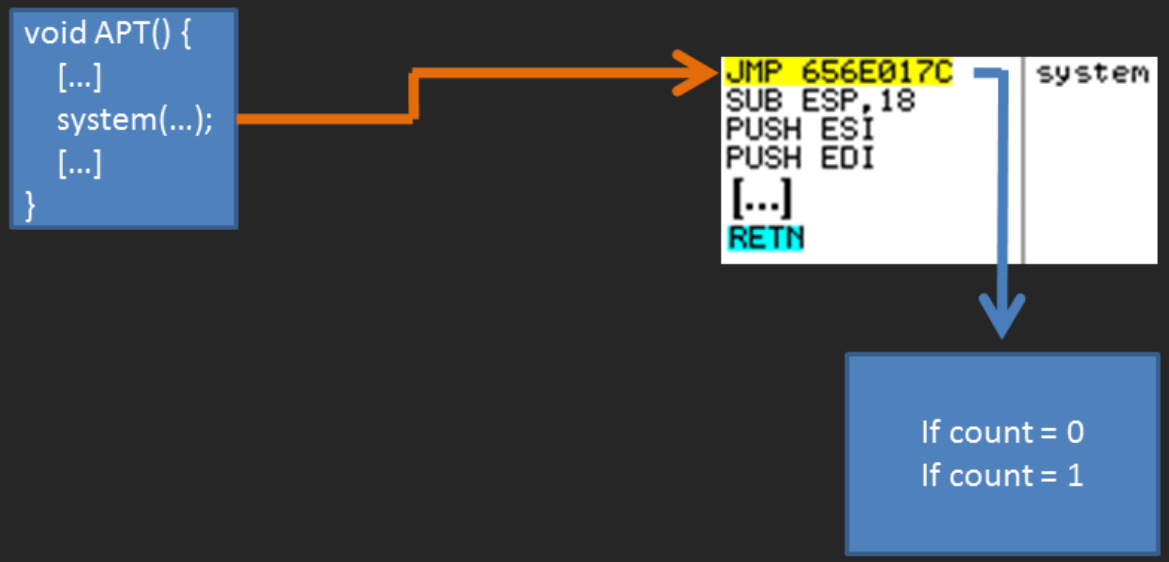
```
void APT() {
    [...]
    system(...);
    [...]
}
```

API Hooking - Result

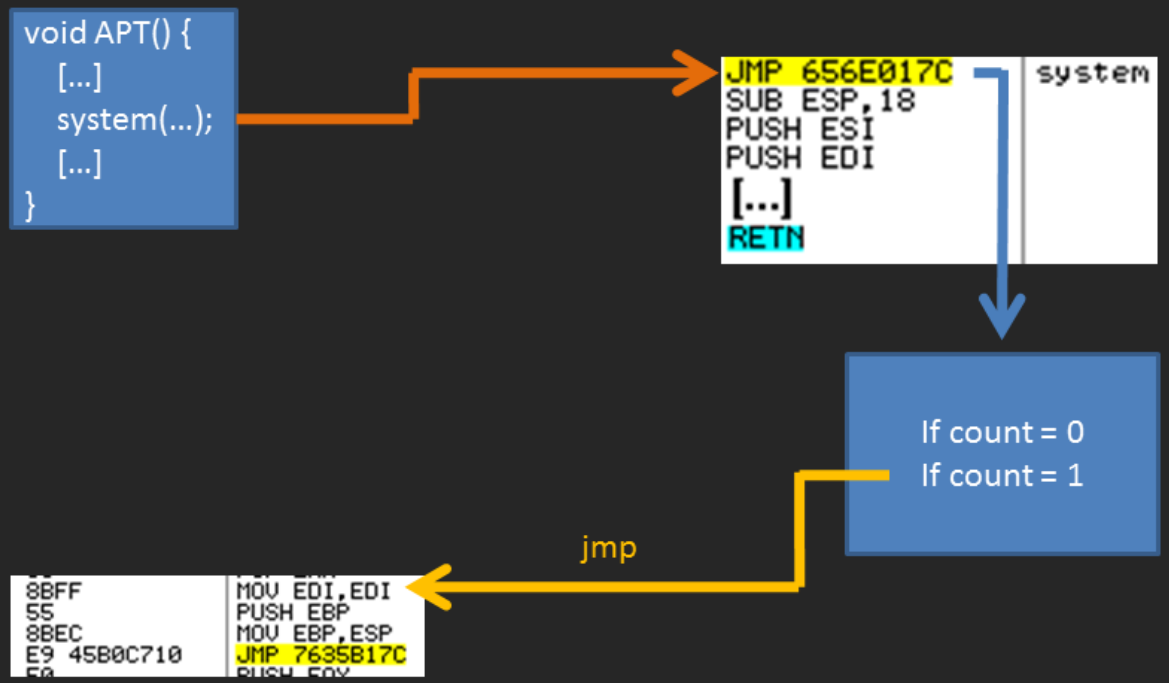
API Hooking - Result



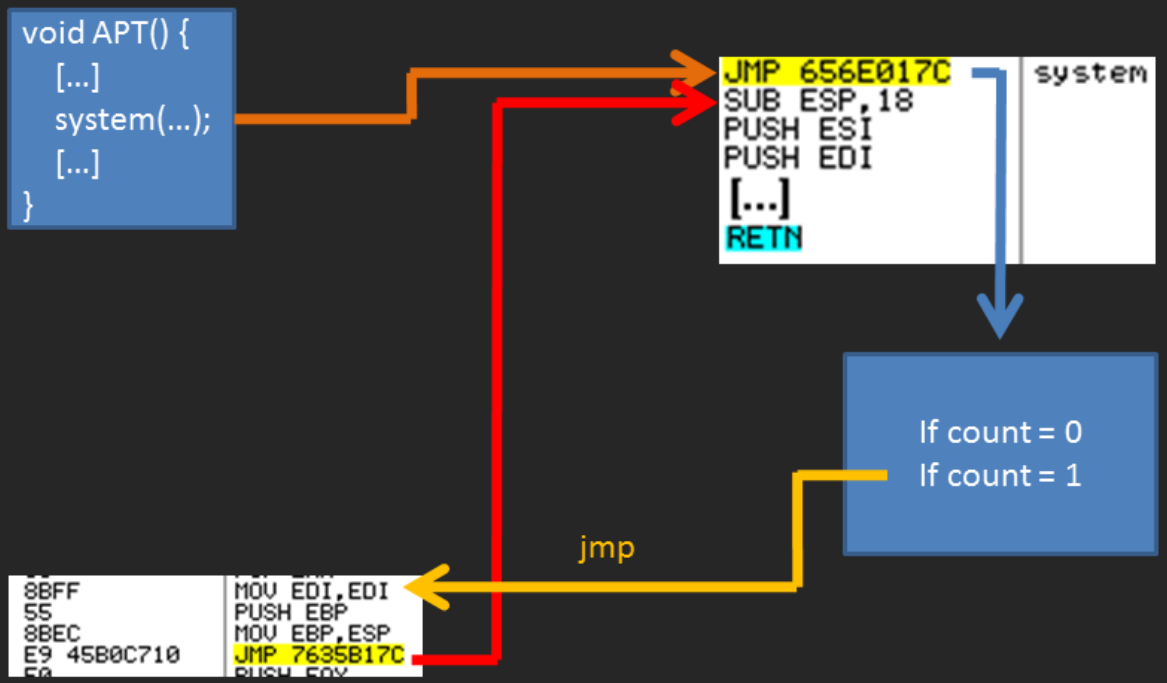
API Hooking - Result



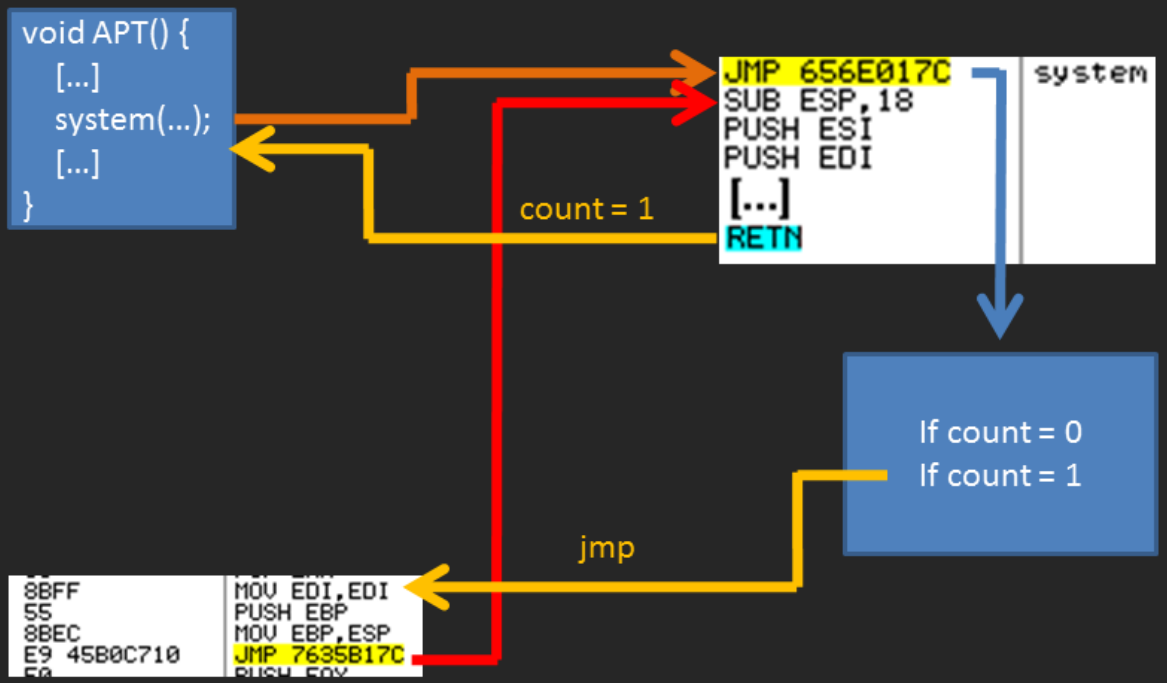
API Hooking - Result



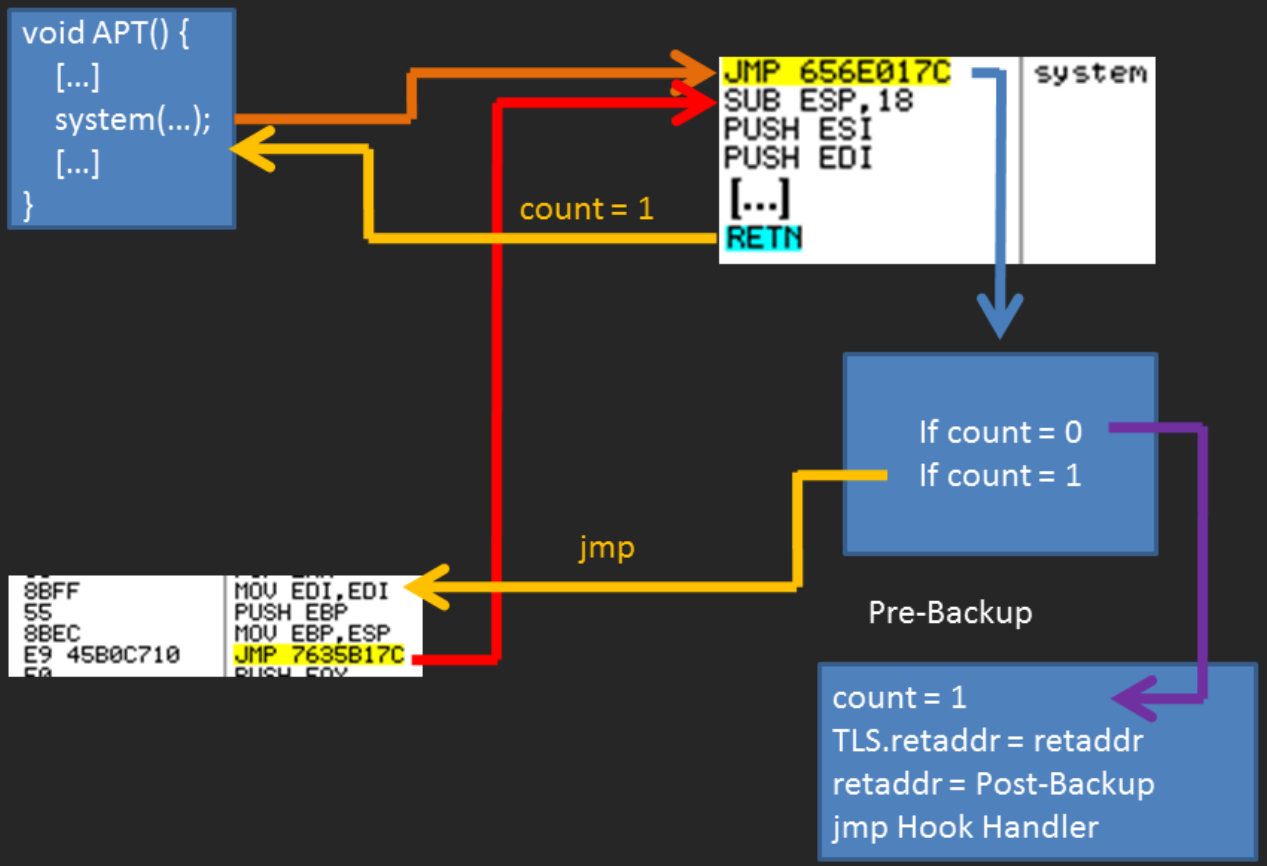
API Hooking - Result



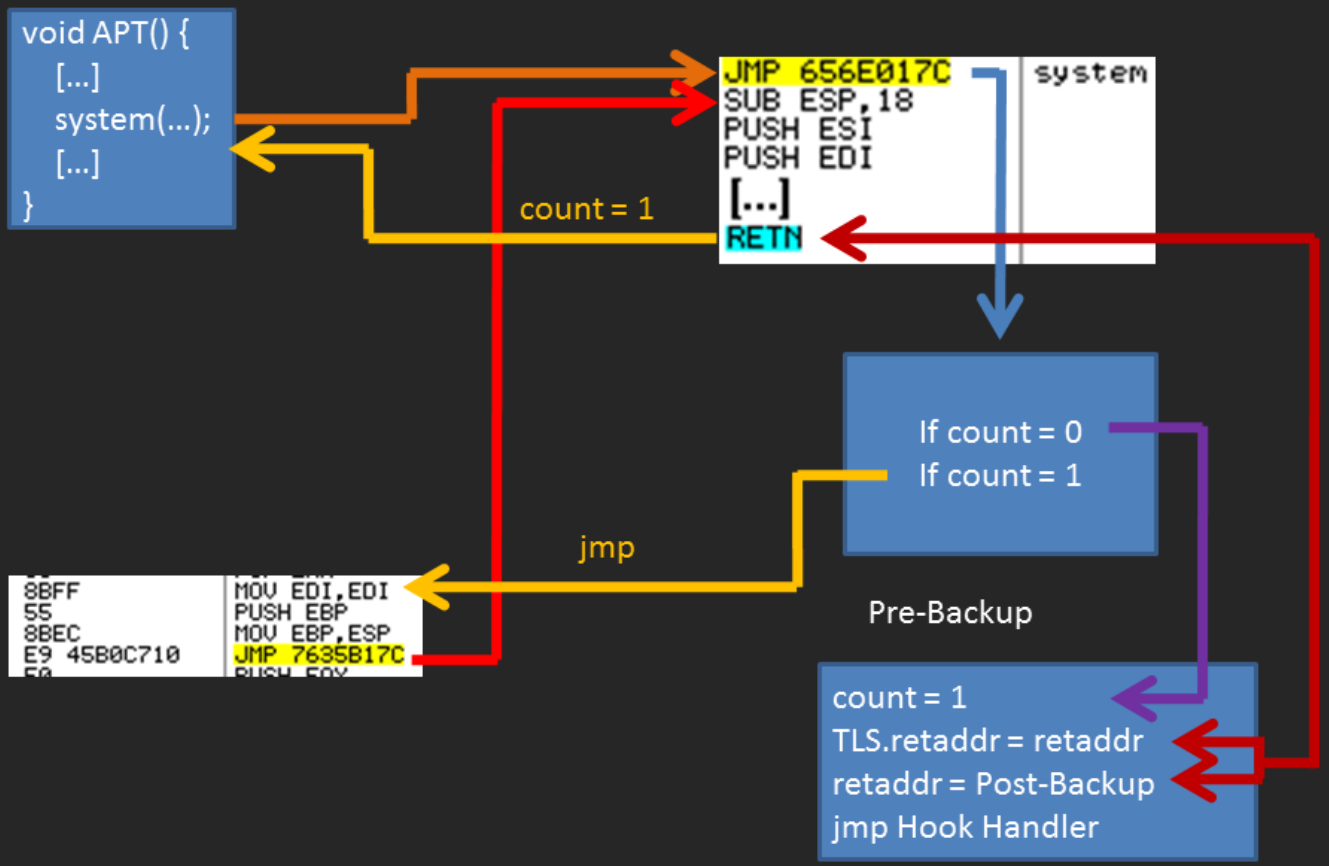
API Hooking - Result



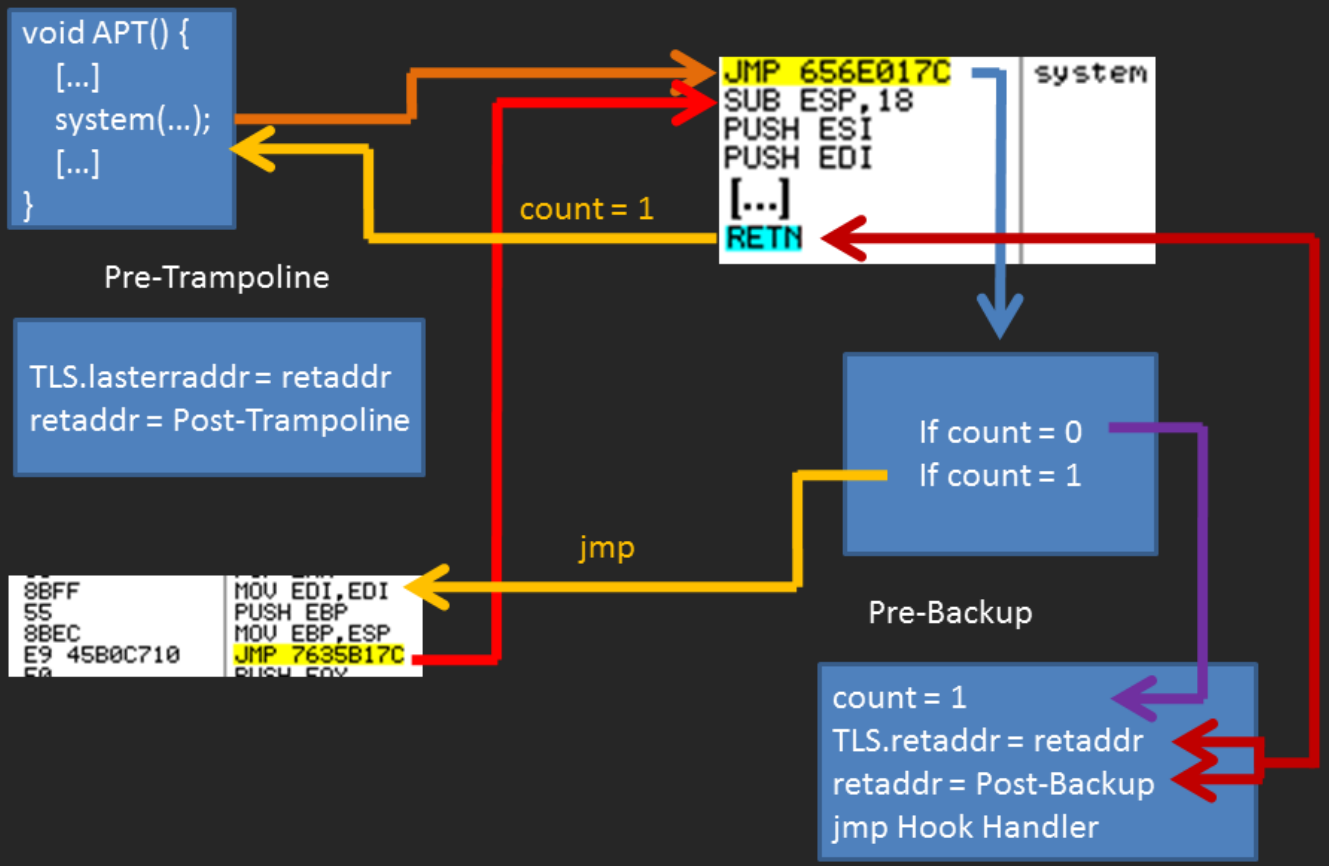
API Hooking - Result



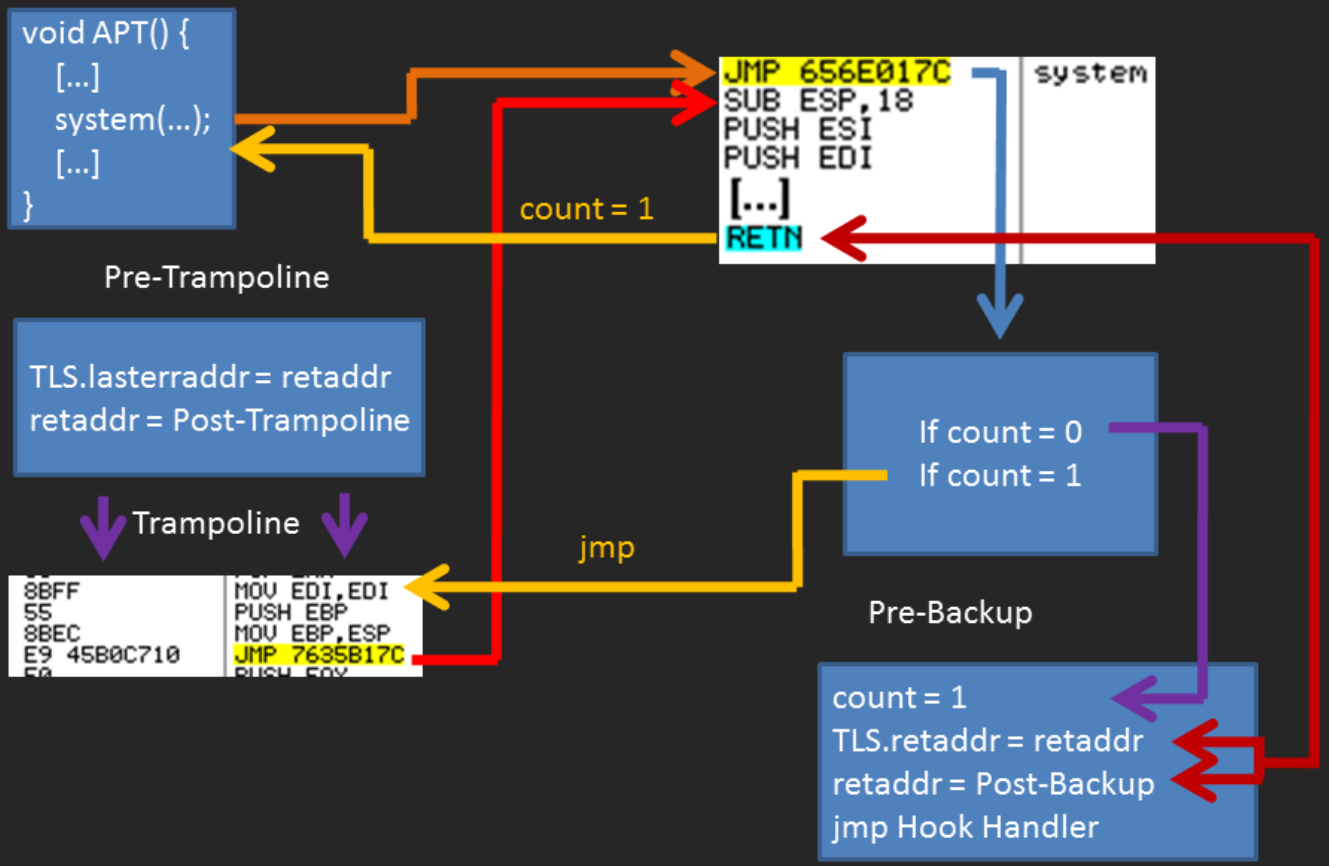
API Hooking - Result



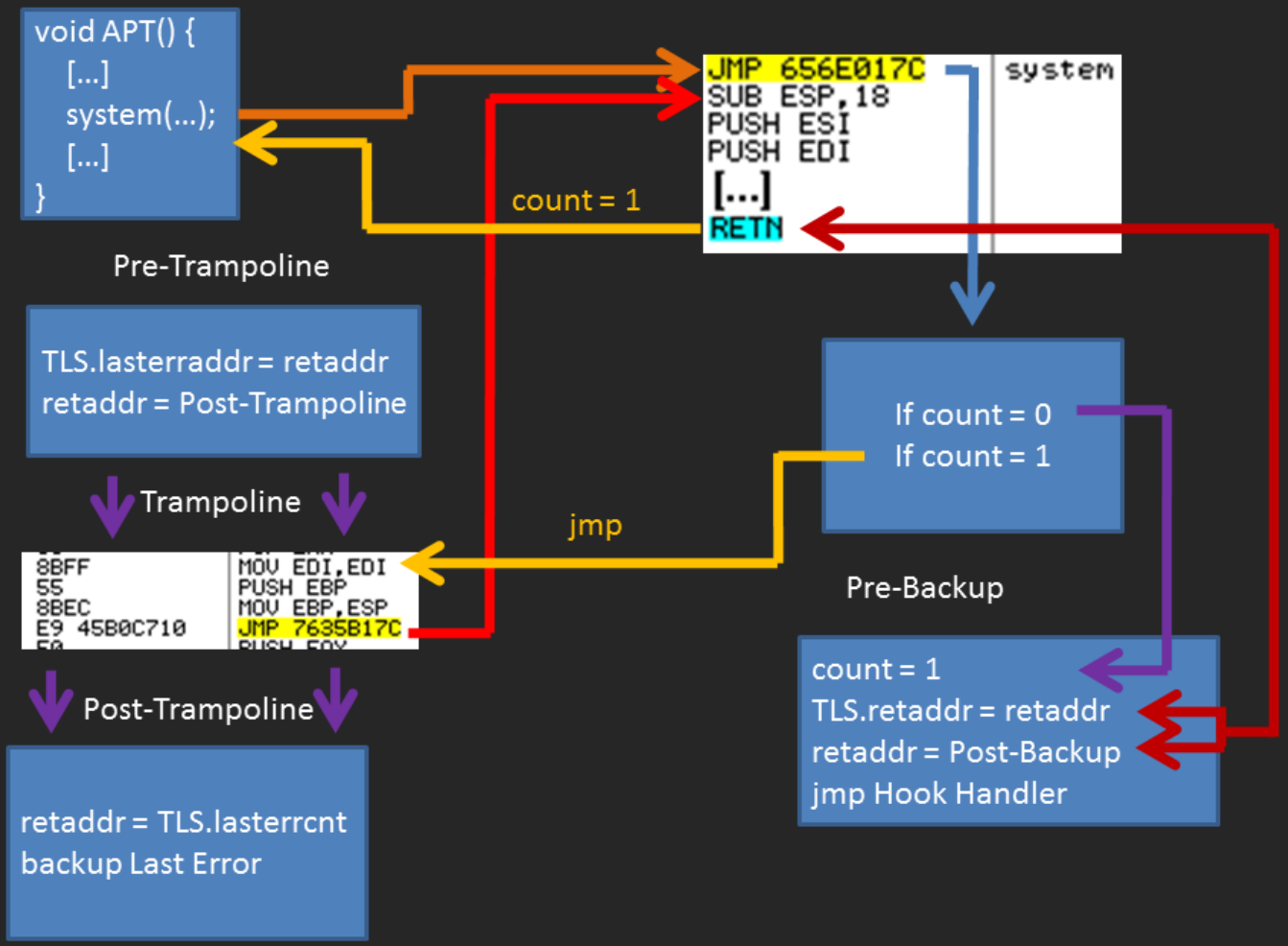
API Hooking - Result



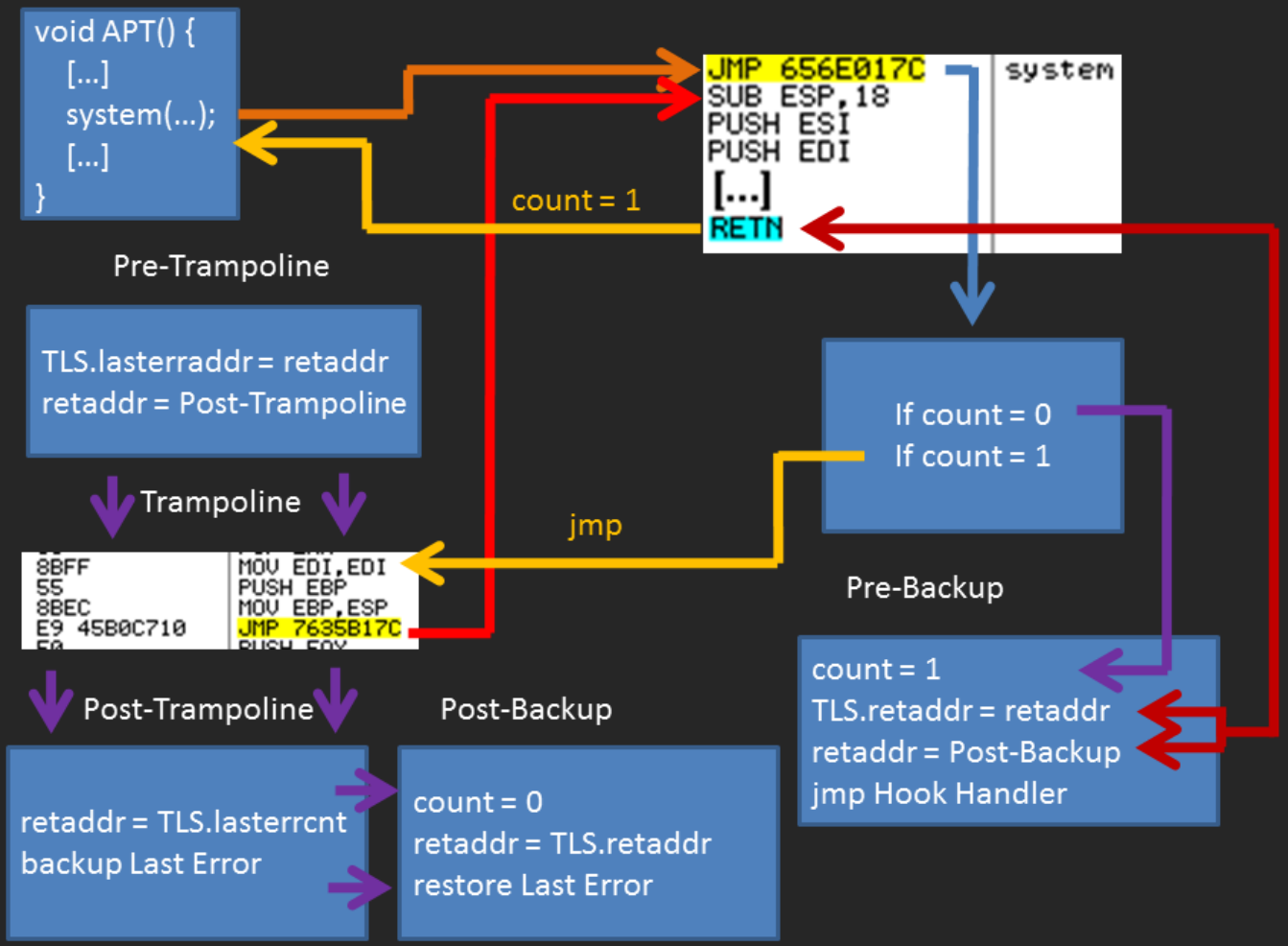
API Hooking - Result



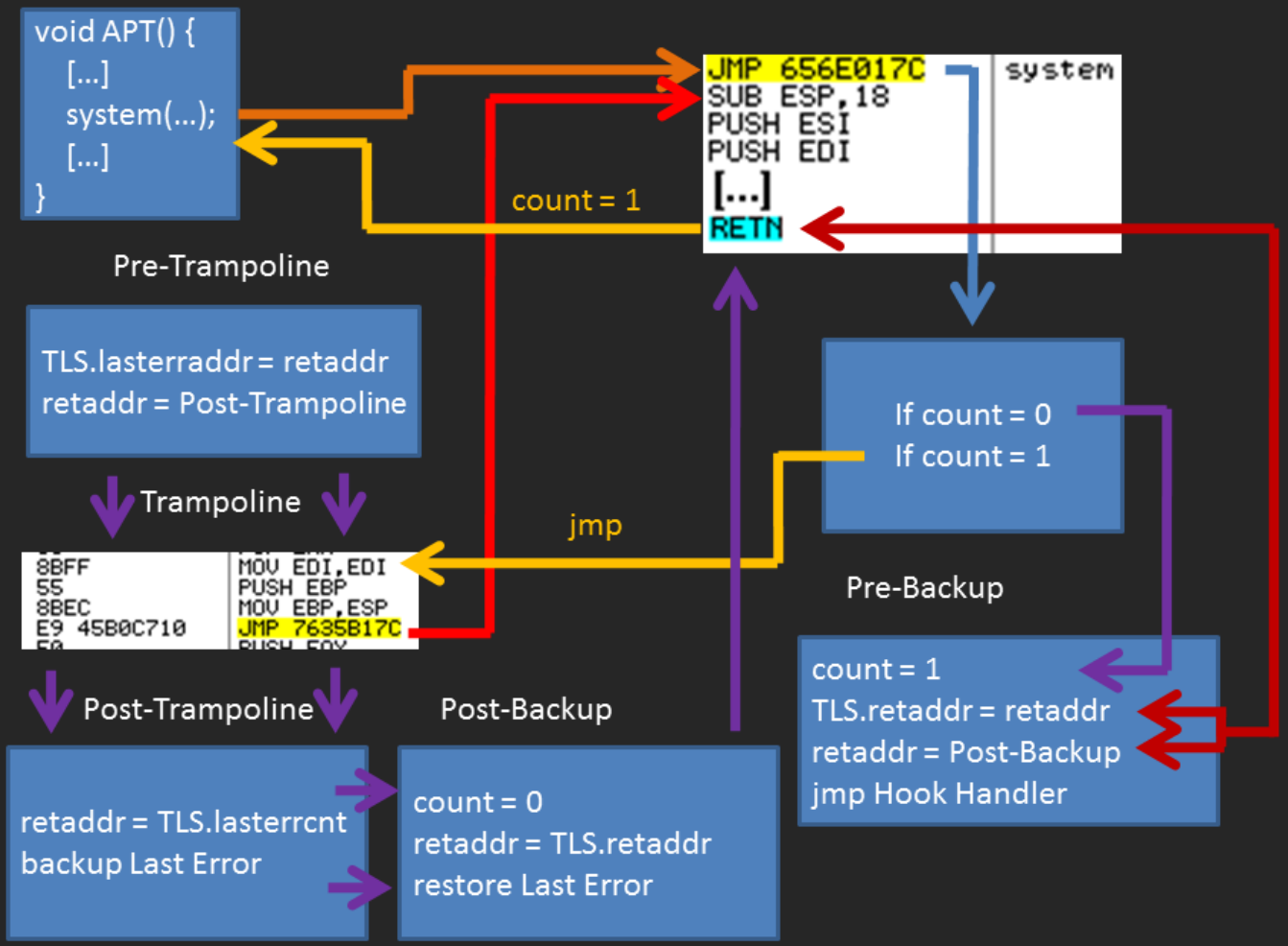
API Hooking - Result



API Hooking - Result



API Hooking - Result



Results

- What kind of logs are we interested in?
  - Process Management
  - Thread Management
  - Registry
  - File Input /Output
  - Sockets
  - ..
- Signatures & Reporting modules

Work in Progress - Return Address Checking Module (1)

- Sometimes APIs are not relevant
  - When injected into another process
- Check Return Address in the Stack Trace
  - Nothing interesting?
    - Don't log it
- As usual, sounds easier than it is
- Needs Taint Data
  - One process can write to another process

Work in Progress - Return Address Checking Module (2)

- Inter Process Communication required
  - `VirtualAllocEx`/`VirtualFreeEx`/.. go through the Analyzer
- `CreateRemoteThread(&LoadLibraryA, "evil.dll")`
  - `&LoadLibraryA` is now interesting

Work in Progress - Return Address Checking Module (3)

We were testing this code earlier, but got generic Cuckoo errors.

- Segfaults on `NtClose`/`VirtualFreeEx`
  - Unrelated to this module
  - However, necessary
- Once fixed, should work.

Work in Progress - StubDLL (1)

Some malware checks against hooks for common functions.

```
if(*(uint8_t *) addr == 0xe9) { ... }
```

- StubDLL doesn't hook a function
  - It generates a Shadow DLL in-memory
- Trampolines for every exported function
  - Restores context and jumps to original function
- Prologue is not altered

Work in Progress - StubDLL (2)

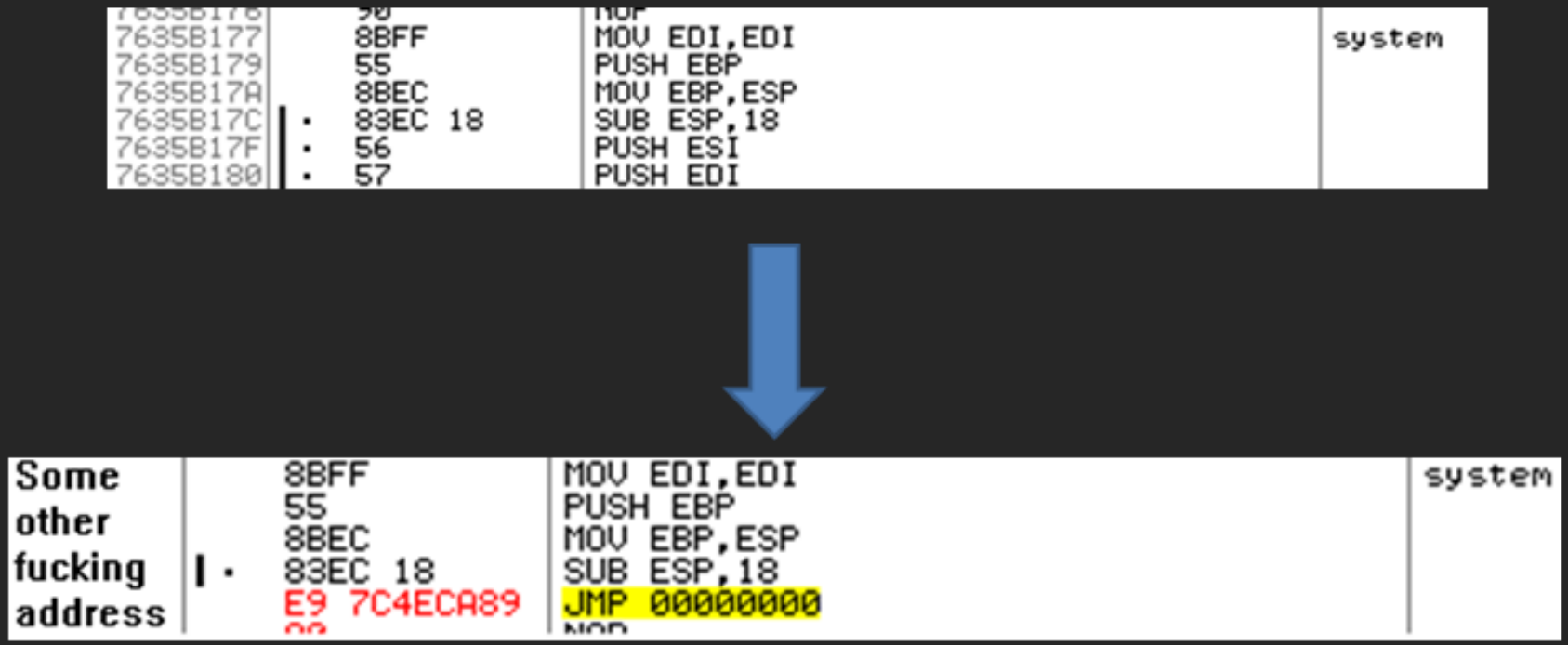


Figure : Simple old versus new `system`.

Questions?

.. :)