Inside the SYSTEMBC Command-and-Control Server

By Dave Truman

Published: 2024-01-19 · Archived: 2026-04-29 02:11:33 UTC

Throughout Q2 and Q3 2023, Kroll has observed an increased use of the malicious “SYSTEMBC” tool to maintain access in a compromised network. SYSTEMBC was first observed in the wild in 2018 with its core functionality revolving around its ability to act as SOCKS5 proxy. This provides a useful capability for threat actors as a persistent access mechanism or for purposes of leaving behind a backdoor in case of discovery of their initial access method. The SOCKS protocol allows for a threat actor to access a victim network almost as if their workstation was directly connected. The tool has been leveraged by a number of threat actors across several campaigns as well as used alongside many malware families, including RHYSIDIA, BLACKBASTA, CUBA, GOOTLOADER, COBALTSTRIKE and EMOTET.

SYSTEMBC can be purchased on underground marketplaces and is supplied in an archive containing the implant, a command-and-control (C2) server, and a web administration portal written in PHP.

The Kroll Cyber Threat Intelligence (CTI) team conducted research into SYSTEMBC, focusing on its C2 server.

Watching the initial walkthrough of the SYSTEMBC C2 server from our threat intelligence expert, Dave Truman

SYSTEMBC Server Installation

SYSTEMBC comes as an installation package with several software components:

- SYSTEMBC implant in standalone preinstallation environment (PE) Windows executable (.EXE) file format
- SYSTEMBC implant as PE dynamic link library (DLL) file (32-bit and 64-bit versions)
- Server executable in Windows .EXE format
- Server executable in Linux executable and linkable format (ELF) format
- PHP file for rendering the C2 panel interface
- GeoIP2 data

Alongside the software components, the developers of the malware provide a text file containing both English and Russian language installation steps for the server.

```
================================================== RUSSIAN ==================================================

ВНИМАНИЕ! socks.exe стучит через 5 минут после запуска!

сервер имеет ограничение на число поддерживаемых socks5 соединений. не более 45151 соединений (порты 4000-49151)

1Gbit server / 1000 socks = 1 mbit per socks

windows:

        1. установить IIS + PHP 7. инструкция на английском https://docs.microsoft.com/en-us/iis/application-frameworks/scenario-build-a-php-website-on-
iis/configuring-step-1-install-iis-and-php

[TRUNCATED FOR DISPLAY]

================================================== ENGLISH ==================================================

ATTENTION! socks.exe come online after 5 minutes from start!

server have limit supporting connections. no more 45151 (ports 4000-49151)

1Gbit server / 1000 socks = 1 mbit per socks

windows:

        1. install IIS + PHP 7. instruction here https://docs.microsoft.com/en-us/iis/application-frameworks/scenario-build-a-php-website-on-iis/configuring-
step-1-install-iis-and-php
```

Figure 1 - Installation Instructions Provided in Both Russian and English

Installation instructions are provided for both Windows and Linux, containing detailed steps on packages to install and execution commands to run. While the Windows instructions are much more succinct, Linux instructions are more in-depth. They can be found below:

```
linux:

        1. install apache + php 7

                apt update
                apt install apache2 -y
                apt install php -y

        2. upload and run server.out

                chmod 777 server.out
                ./server.out &

if u are using CentOS need turn off firewall

# sudo systemctl stop iptables
# sudo systemctl disable iptables

/www need install to root folder of web server
http://yourserver/systembc/password.php show online sockses (pass adm443). for security u can rename folder 'sytembc'
```

Figure 2 – Linux Installation

Finally, the installation document details how to set up fast flux for use with SYSTEMBC, although it states that fast flux is not recommended to be used with the tool.

```
================================================== FASTFLUX ==================================================

not recommended use fastflux. because ff need domain. more better use ip instead domain (domain can block)

for fastflux need additional settings. example config for haproxy

listen socks5_bc
  bind *:4001 # (need to change with your port in builder)
  mode tcp
  option clitcpka
  option srvtcpka
  server target [redacted] send-proxy-v2

[redacted] replace to ip of your server

+ need execute 3 commands on server of fastflux (not direct server. on ff)

sysctl net.ipv4.tcp_keepalive_time=60
sysctl net.ipv4.tcp_keepalive_intvl=10
sysctl net.ipv4.tcp_keepalive_probes=3

else sockses will disconnect every 1 minute
```

Figure 3 – Fast Flux Warning and Instructions

Within the fast-flux instructions are details on setting kernel-level parameters for network “keep alive” behaviors as a mechanism to avoid the SOCKS connections from disconnecting.

SYSTEMBC Server Configuration

At the heart of the C2 side of the application is the “server” binary for Windows. This is called “server.exe”; for Linux, it is called “server.out”. Our research focused on the Linux server due to its common use.

The server opens two or more Transmission Control Protocol (TCP) ports:

- One port for inter-process communication (IPC) between itself and the PHP-based panel interface, usually port 4000 and bound to localhost.
- One port for C2 traffic. We have seen many different port numbers for C2 across the different servers that the Kroll CTI team found in the wild; however, 443 is likely the default.
- For each active implant, the server opens a port. These range in number from 4001 to 49151.

The configuration for the C2 and IPC ports are represented as plain text string within the binary itself. Both port number strings are prepended with a label acting as an identifying string “PORT0:” for IPC and “PORT1:” for C2. Each is also padded with null characters so the space allocated for each would total six characters, allowing for a five-printable and one null-terminating character representation of all valid TCP port numbers (1–65535).

```
00003510  78 69 74 00 45 78 63 65  70 74 69 6f 6e 20 61 64  |xit.Exception ad|
00003520  64 72 3a 20 50 4f 52 54  30 3a 34 30 30 30 00 00  |dr: PORT0:4000..|
00003530  50 4f 52 54 31 3a 34 34  33 00 00 00 0d 0a 0d 0a  |PORT1:443.......|
00003540  00 0d 0a 51 55 49 54 0a  00 00 00 00 00 00 00 00  |...QUIT.........|
00003550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00003560  00 00 00 00 03 00 01 00  e8 00 40 00 00 00 00 00  |..........@.....|
00003570  00 00 00 00 00 00 00 00  00 00 00 00 03 00 02 00  |................|
```

Figure 4 – Port Settings Within Binary with Visible Padding

The way the data is stored within the binary indicates that it is likely to be configurable, potentially with a tool that reads the binary and updates the values by searching for the labels and updating the strings. Testing modifying

the values with a hex editor allowed for configuration of the ports.

There are core three data files that the server uses:

Figure 5 – Table of Server Executable's Configuration Files

These files get created when the server needs them, so if no comments have been made for any implants/victims, there will not be a .comments file.

The use of the value 49151 for the maximum port number hints that the developer might be used to lower-level programming such as C or Assembly. While 49151 in its decimal notation doesn't appear relevant, in hex it is 0xBFFF, which is one lower than a round number (0xC000). This indicates an awareness of the memory allocation of integers.

Figure 6 – Source Filename Contained Within the Linux Server Binary

While it is generally reported that SYSTEMBC is written in C, there is reference to a source file within the Linux Server binary that could indicate that part of SYSTEMBC was coded directly in Assembly language.

SYSTEMBC Control Panel

The SYSTEMBC panel is written as a single monolithic PHP script. The code itself appears mainly written as a series of "if" statements, with little use of functions. At the beginning of the code, we can see the setup of the TCP connection to the server binary with the port number of 4000 hardcoded. This indicates that while this port is configurable in the binary, it is likely intended to be left at 4000. Rather than utilizing PHP's built-in ability to act as code nested within an HTML document, instead most of the HTML elements of the page are printed out with the echo function. While both are valid approaches, the choice might provide insights into the developer's previous development experience.

Figure 7 – Control Panel PHP code with Hardcoded Connection String to Server IPC Port

One of the few separated-out functions, which solves an arguable non-complex problem, is a function called "secondsToTime", which appears to be copied and modified from a stack overflow post with the original comments left in.

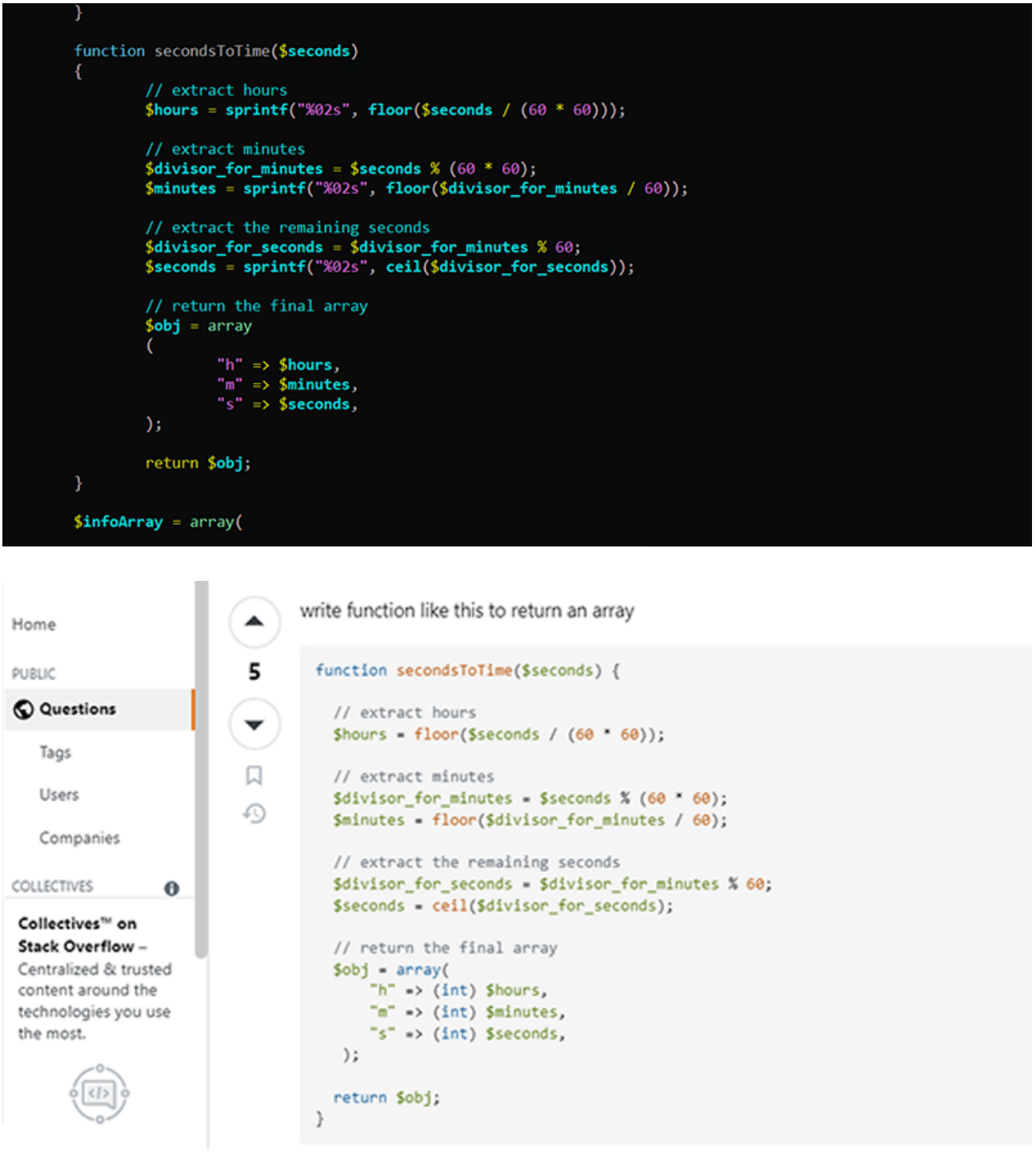

Figure 8 – Side-by-Side Comparison of Function Within PHP Panel and Stack Overflow Post

Along with the previously mentioned maximum port, there are indicators that might also point to the possibility of the developer having more experience (or interest) in lower-level programming, with the PHP being a means to an end. For example, the developer would seek a faster way to get a working web interface rather than one of their go-to technologies. Some of these indicators are:

- The minimalist and monolithic nature of the PHP code
- Printing the HTML elements rather than nesting of the PHP within an HTML document
- Choice to do most of the business logic within the server executable

- Binary configuration files
- Utilization of externally sourced functions for relatively easy-to-write PHP code

The control panel is the attacker's view of the machines they have infected. It is dominated by a table with a row for the key details of each machine, most important of which is the port on which the C2 server is listening for SOCKS traffic to tunnel into victim networks.

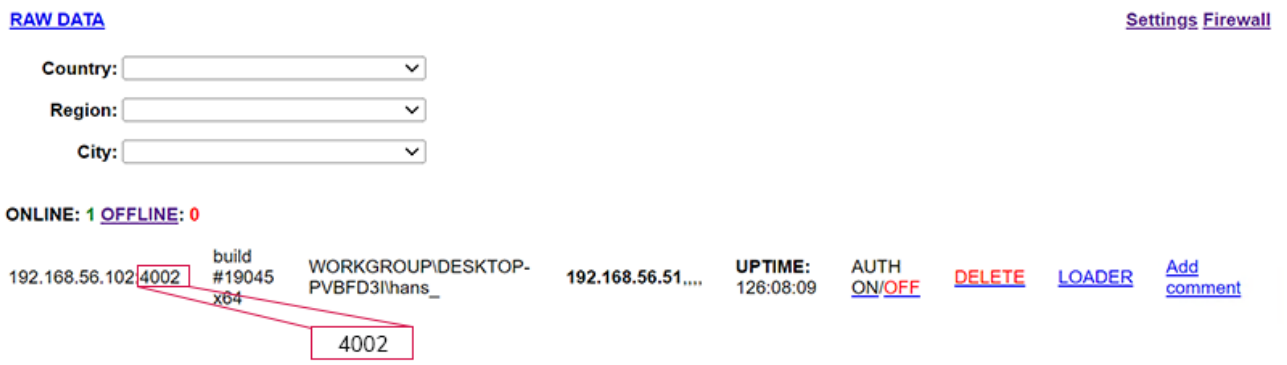

Figure 9 – SYSTEMBC C2 Panel Showing Active Implant with its SOCKS Access Port Highlighted

Core Functionalities of SYSTEMBC

SYSTEMBC has a small set of core functionalities; however, it is entirely based upon enabling other functionalities. The functionality of SYSTEMBC can be broken down into 3 components.

- SOCKS5 proxying onto a compromised host
- Loader functionality for running separate scripts and executables
- Module loading for on-the-fly expanding of SYSTEMBC implants core functionality

SOCKS5

Upon check-in, the implant establishes an outbound network connection from the victim machine to the C2 server; that connection is kept open while the implant is running. On the C2, a listening TCP socket is created and assigned a port number, which in turn is associated with the specific implant. This relationship is maintained via the user of the .settings file. The server implements the SOCKS5 protocol on the port associated with the implant and accepts any SOCKS5 connections for which it will then send the network data to be proxied along with the network connection established from the victim machine. Once received, the implant will forward traffic on this network to the intended destination as if the victim machine had made the request.

A threat actor can use this SOCKS connection as a beachhead within the target network forwarding on any tool with SOCKS support, including vulnerability scanners, password sprayers and remote connections. Often overlooked, it also provides the actor with a means for attacking other networks from the victim network. This can be useful in abusing trusted relationships between companies and for attacking from IP addresses that belong to legitimate entities with a low-risk profile, rather than from anonymized IPs via VPNs or Tor that might trigger security events on well-defended networks.

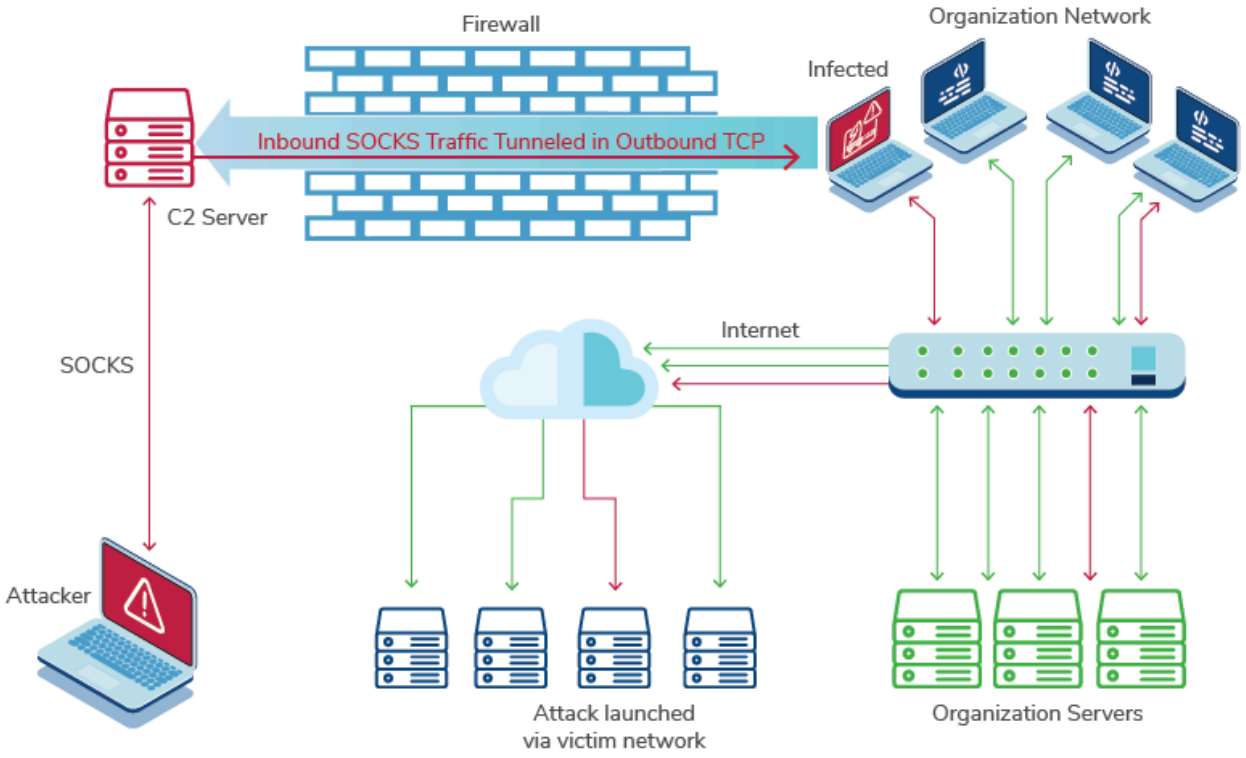

Figure 10 – SOCKS Communication Across a Victim Network

The server-side configuration of the SOCKS access allows for two optional settings for the SOCKS proxy. Both are geared toward securing the connection on behalf of the actor.

The first is the option to enable authentication individually for each connection. This is done by switching a toggle for the specific implant on the main page. When toggled, the page changes to show a username and password that can be used to authenticate via SOCK5 protocol as defined in RFC 1929 for that specific implant. The username and password are randomly generated and not set by the control panel user. Turning the authentication toggle off and on again will cause the username and password to rotate. The configuration for this is stored in the .settings file.

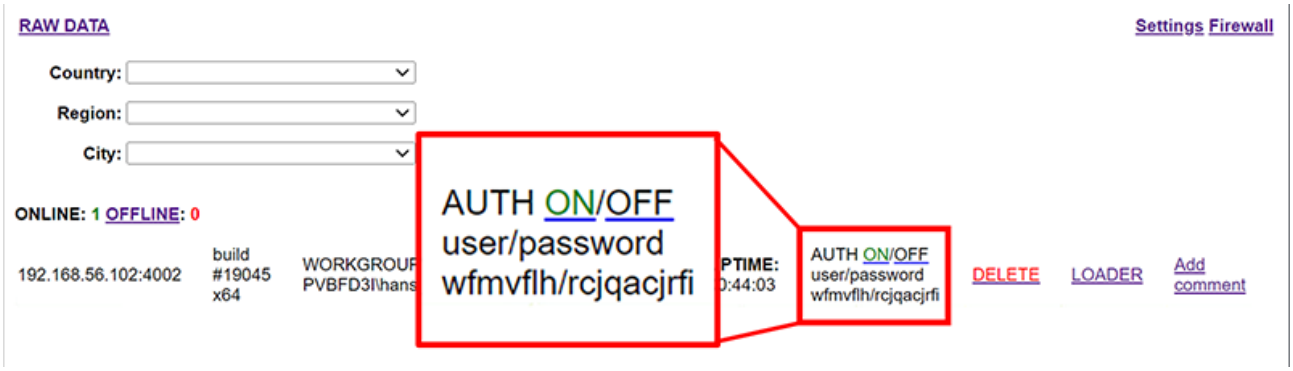

Figure 11 – Authentication Toggle and Auto-Generated SOCKS Credentials

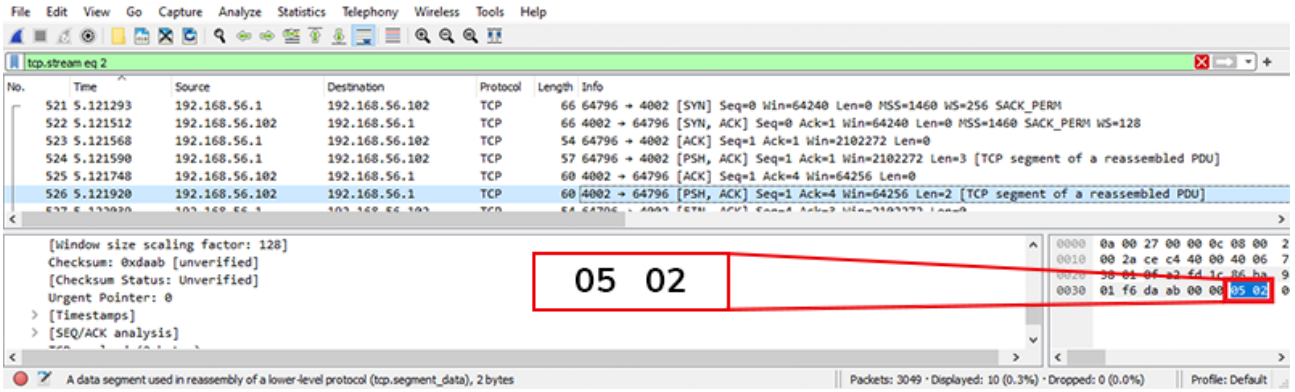

Figure 12 – Packet Capture Showing Server Requiring SOCKS5 (05) Credential Authentication (02) as per RFC 1929

The second configuration option that affects the behavior of the SOCKS proxy functionality is the configuration that is accessed via the link labeled “Firewall”.

Figure 13 – Link to Host-based Access Control Configuration

By entering one or more IP addresses in the text area, the server will accept SOCKS connections from that list of IP addresses. This is a global setting, and all implants will now only be SOCKSaccessible via these IP specific addresses.

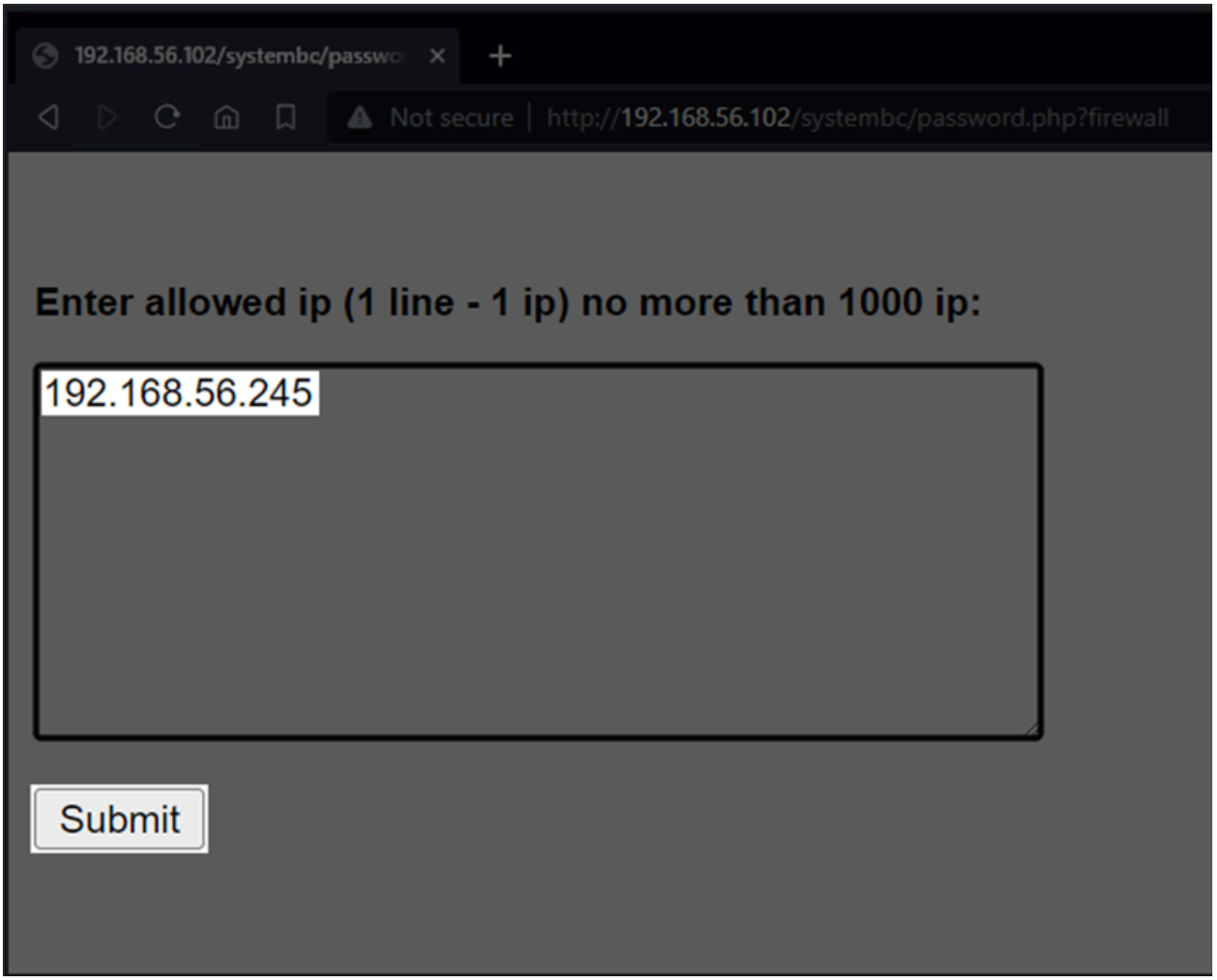

Figure 14 – Host-based Access Control Configuration

The access control mechanism is not a firewall as it does not block access to the port and the application behind it. Instead, the server process will immediately close the connection, instigating a graceful shutdown upon first use.

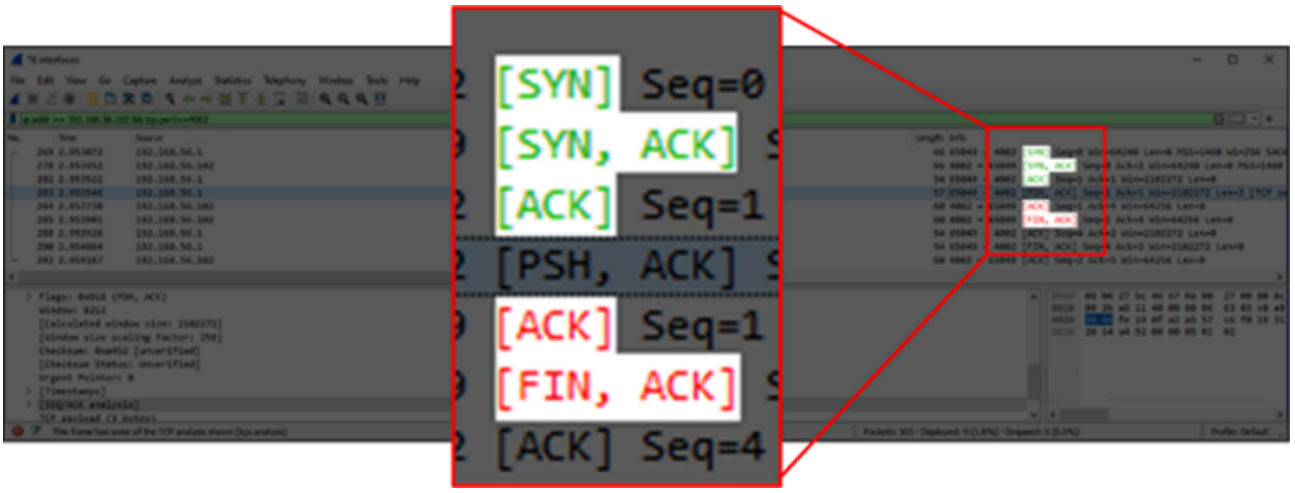

Figure 15 – Graceful Shutdown Initiated by Server on First Use After TCP Handshake

The TCP port will still show up as open via a port scan from an IP address not in the allowed list; however, indications of it handling SOCKS5 protocol are no longer present in the packet data of that connection.

Loader

In order to run a file on a victim machine, a direct URL needs to be submitted to the file in the “LOAD URL” textbox within the “LOADER” page. This will trigger the implant to execute a GET request for file download. The implant can download via both HTTP and HTTPS.

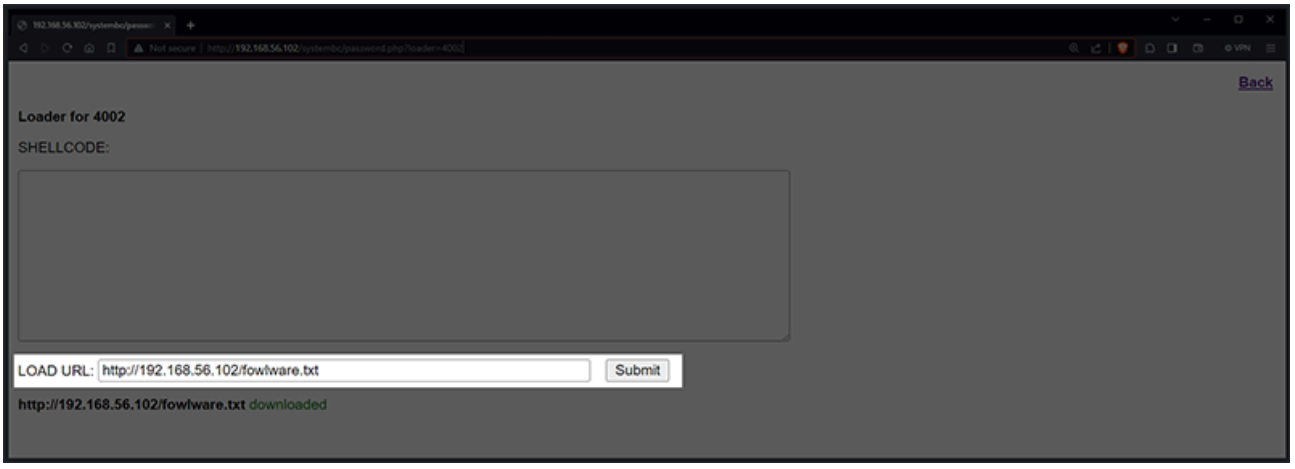

Figure 16 – C2 Panel LOADER Page with Example URL

The loader contains functionality for multiple file extensions. Viewing the code in a decompiler, we can see that the downloaded file will be treated as an .EXE by default and changed if one of the other extensions is used.

```
            /* Extension: EXE */
  local_714[0] = 0x657865;
  pcVar2 = FUN_00406924((char *)(puVar17 + 2));
            /* Extension: VBS */
  if (*(int *)(pcVar2 + 4 + (int)puVar17) == 0x7362762e) {
    local_714[0] = 0x736276;
  }
            /* Extension: BAT */
  if (*(int *)(pcVar2 + 4 + (int)puVar17) == 0x7461622e) {
    local_714[0] = 0x746162;
  }
            /* Extension: CMD */
  if (*(int *)(pcVar2 + 4 + (int)puVar17) == 0x646d632e) {
    local_714[0] = 0x646d63;
  }
            /* Extension: PS1 */
  if (*(int *)(pcVar2 + 4 + (int)puVar17) == 0x3173702e) {
    local_714[0] = 0x317370;
  }
```

Figure 17 – File Extensions Executable by SYSTEMBC

This means that any file being sent without one of the extensions listed will be treated as an .EXE file, allowing a filename to appear in network logs that might blend in more easily.

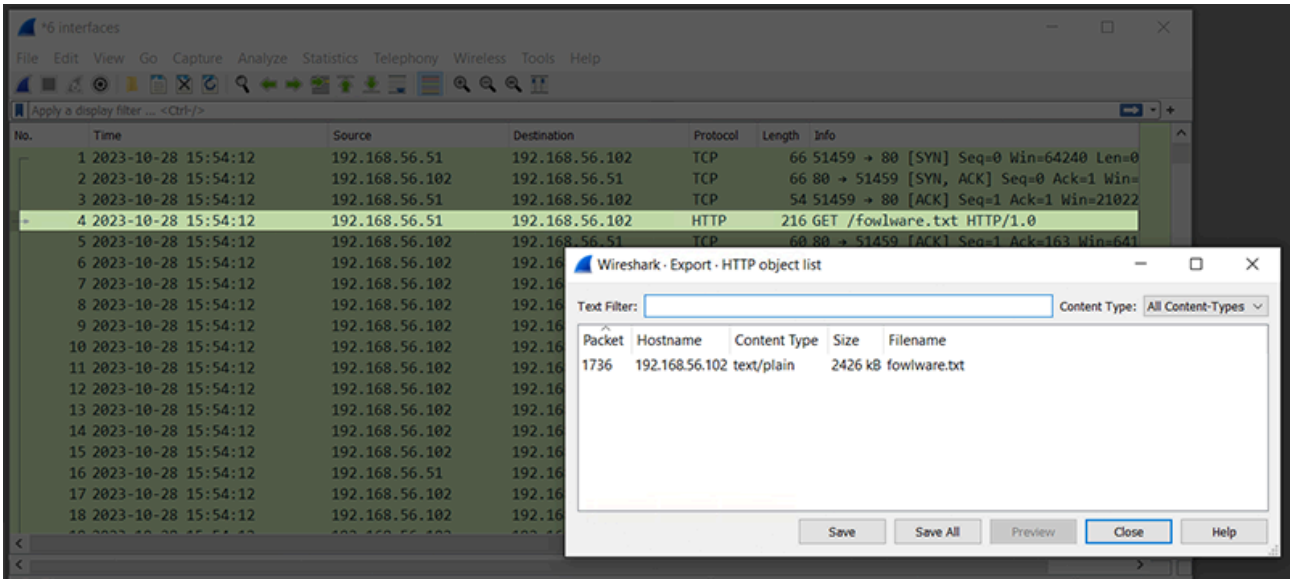

Figure 18 – Network Traffic Capture of Implant Downloading a Remote File

Once the file has been downloaded, the implant saves the file to the %TEMP% directory with a filename consisting of random lowercase characters and the file extension. The implant will set up a scheduled task to run the downloaded file. The task list is made up of 19 lowercase characters. Since this scheduled task is set up to only run once, it will no longer be visible after it is triggered.

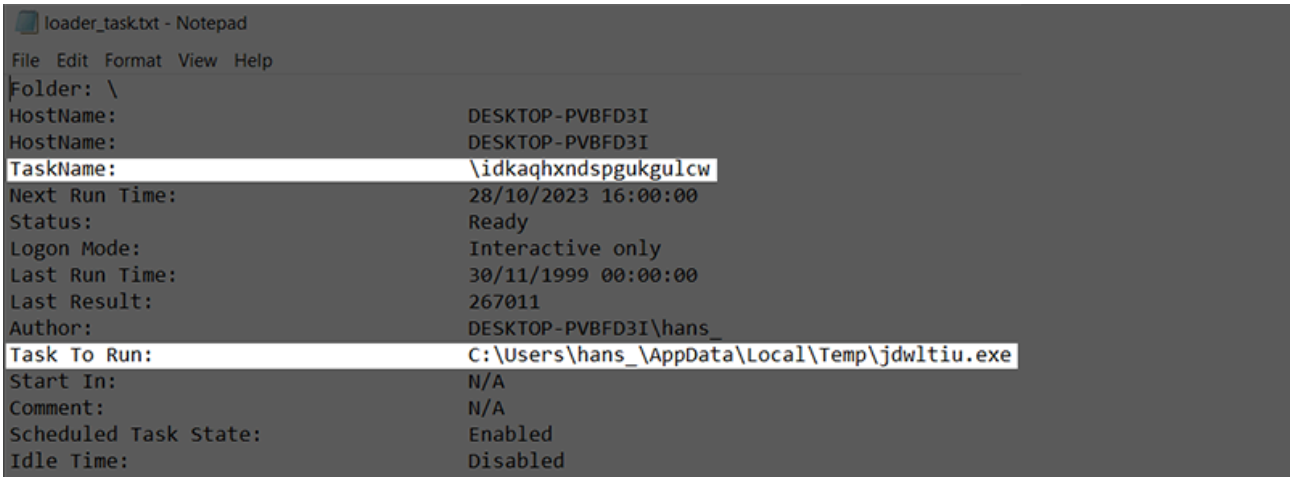

Figure 19 – Example Scheduled Task Created for Loading Other Malware on Compromised System

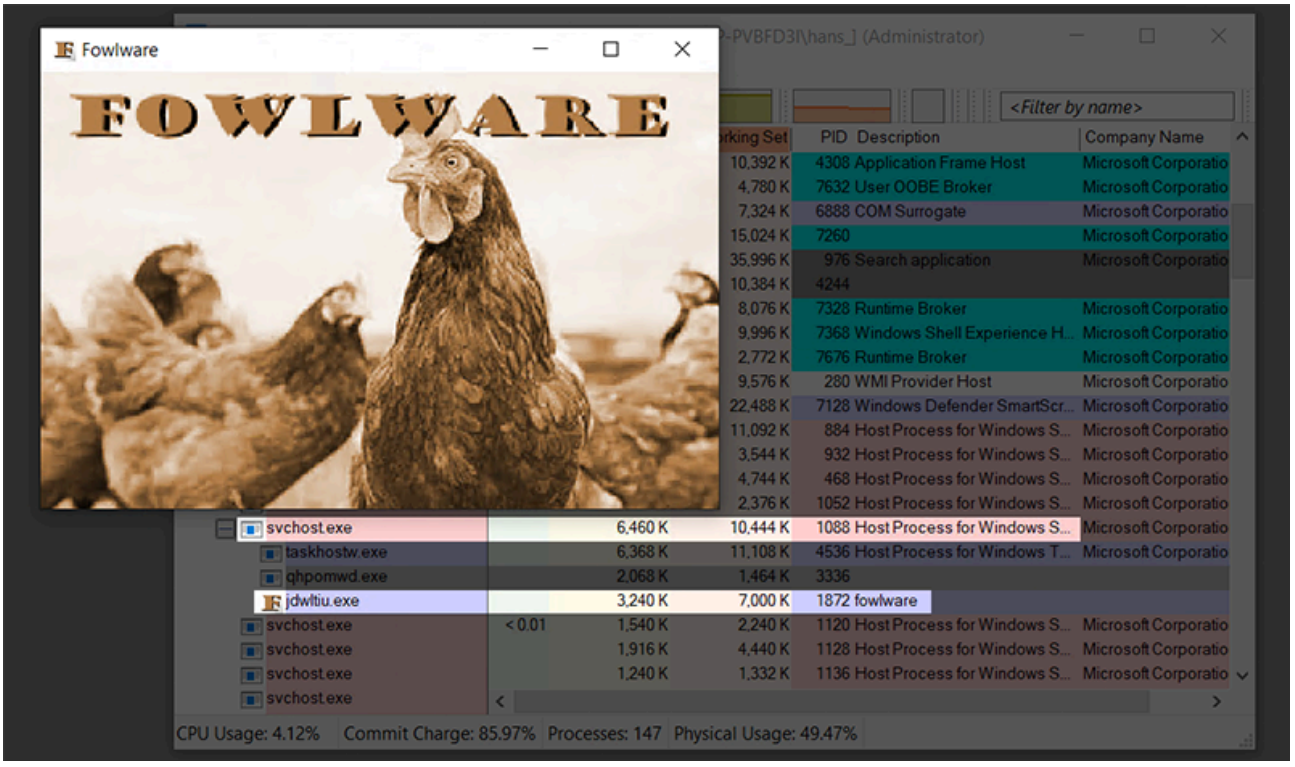

Figure 20 – Downloaded Executable Running with Parent Process svchost.exe

The launch of the downloaded file via a scheduled task cannot easily be traced back to the implant by antivirus solutions at runtime, as the parent process of that file will be svchost.exe and not the implant. This means that if the new file gets blocked or quarantined, the implant might be able to continue unnoticed.

Shellcode

The shellcode functionality, which is also accessible via the “LOADER” page, allows for a different approach in executing malicious code. Instead of downloading and executing a file, the shellcode in backslash, escaped, hexadecimal format can be sent to the implant directly via a text area on the page.

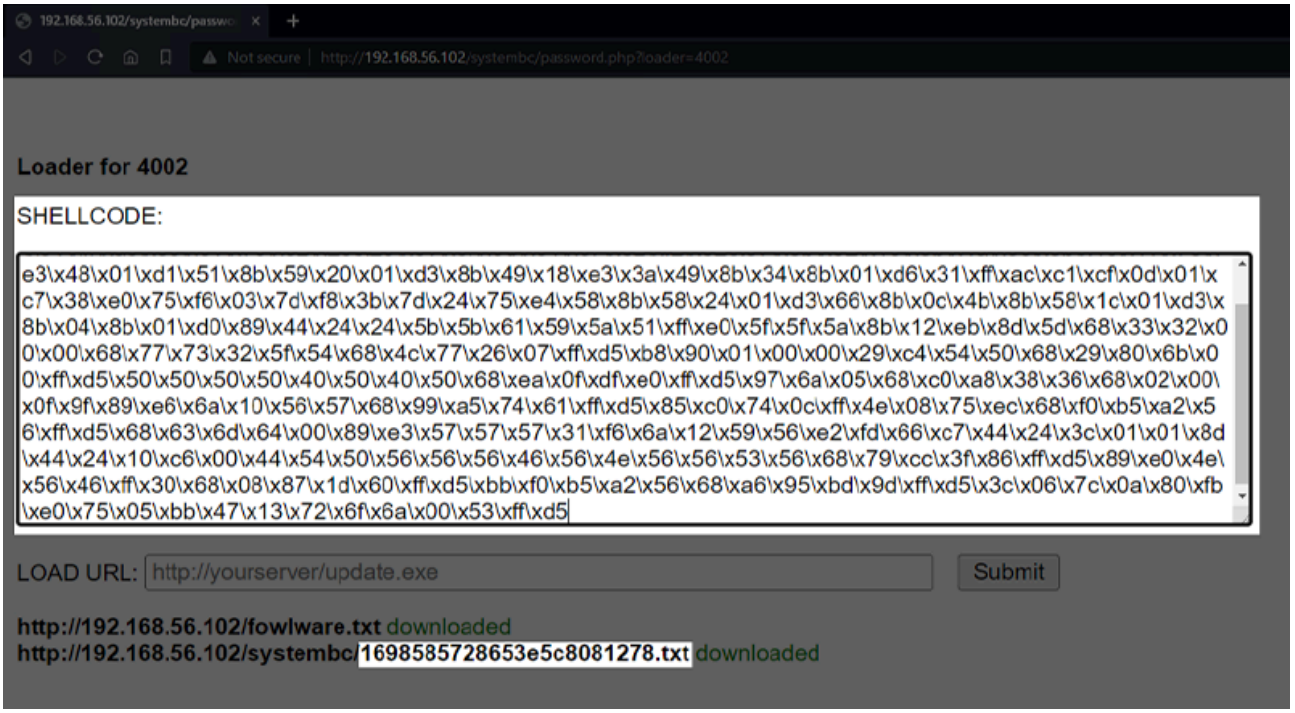

Figure 21 – Example Shellcode for a Reverse TCP Shell Being Entered into C2 Panel

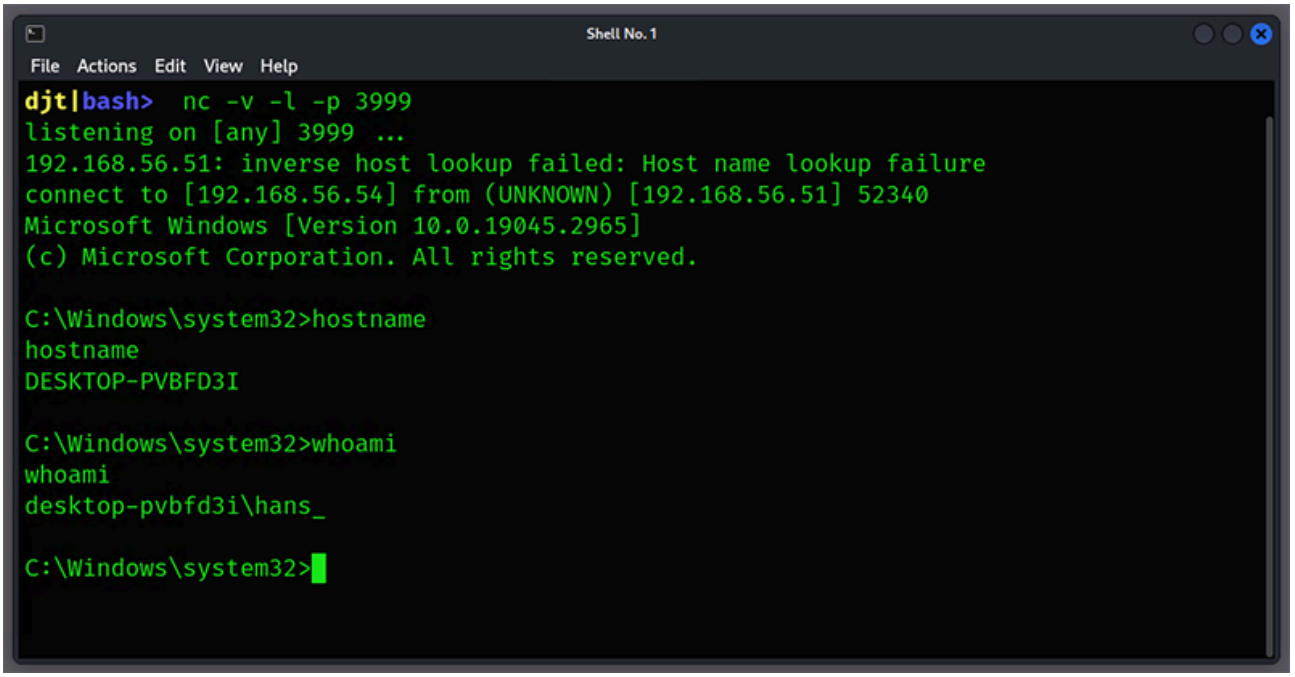

Figure 22 – Reverse TCP Shell Running

The PHP control panel creates a text file contain the shellcode string utilizing the uniqid function seeded with the current time for filename purpose. This gives the file a somewhat predictable filename as highlighted in figure 23, the first part of the filename will be the 10-character UNIX timestamp, followed by 13 hexadecimal characters and a ".txt" extension.

```php
if ($_POST['shellcode'])
    $name = uniqid(time()).'.txt';
    $_POST['url'] = 'http://'.$_SERVER['HTTP_HOST'].str_replace(basename(__FILE__), $name, $_SERVER['SCRIPT_NAME']);
    file_put_contents($name, $_POST['shellcode']);
```

Figure 23 – PHP Code Used to Generate Distinctive Shellcode Filename on Server

The implant will download this file directly into memory. It then sets the area of memory via the VirtualProtect Windows API call to executable by passing in 0x40, and then executes it via the CreateThread call.

```
PUSH    EAX
PUSH    0x40
PUSH    dword ptr [EBP + param_2+0x4]
PUSH    dword ptr [EBP + param_1+0x4]
CALL    KERNEL32.DLL::VirtualProtect        ;BOOL VirtualProtect(LPVOID lpAddress...
PUSH    0x0
PUSH    0x0
PUSH    0x0
PUSH    dword ptr [EBP + param_1+0x4]
PUSH    0x0
PUSH    0x0
CALL    KERNEL32.DLL::CreateThread          ;HANDLE CreateThread(LPSECURITY_ATTRI...
MOV     EAX,0x1
```

Figure 24 – Code Within the Implant for Directly Executing Shellcode from Server

Unlike the loader, the implant does not write the downloaded shellcode file to disk before execution. This can be seen when tracing the implant execution with the procmon utility. As such, the implant can avoid detection based on filesystem-scanning AV tools, which will hinder obtaining malicious samples from disk forensics.

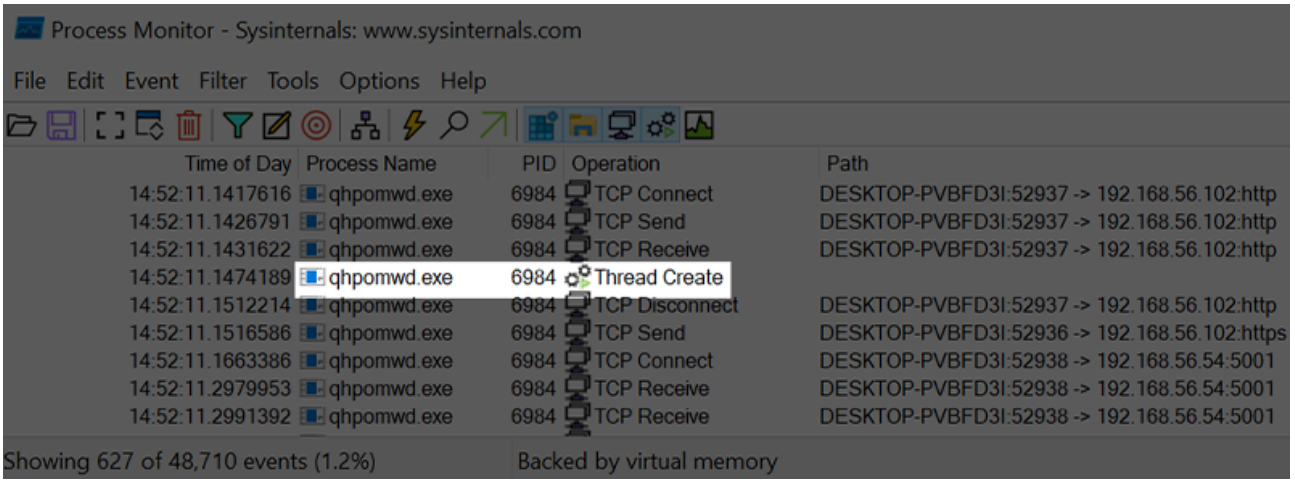

Figure 25 – Procmon Output Showing No File-Writes Around Time of Shellcode Execution

However, because the shellcode runs inside the implant itself, malicious activity might be able to be linked back to the implant if detected at runtime. For example, with our reverse shell, we can see cmd.exe has spawned from our implant process. An attacker will have to choose when to use the loader functionality and when to use code injected directly into the implant itself to avoid detection, along with understanding what activities might draw attention to the implant itself while utilizing the direct-code injection.

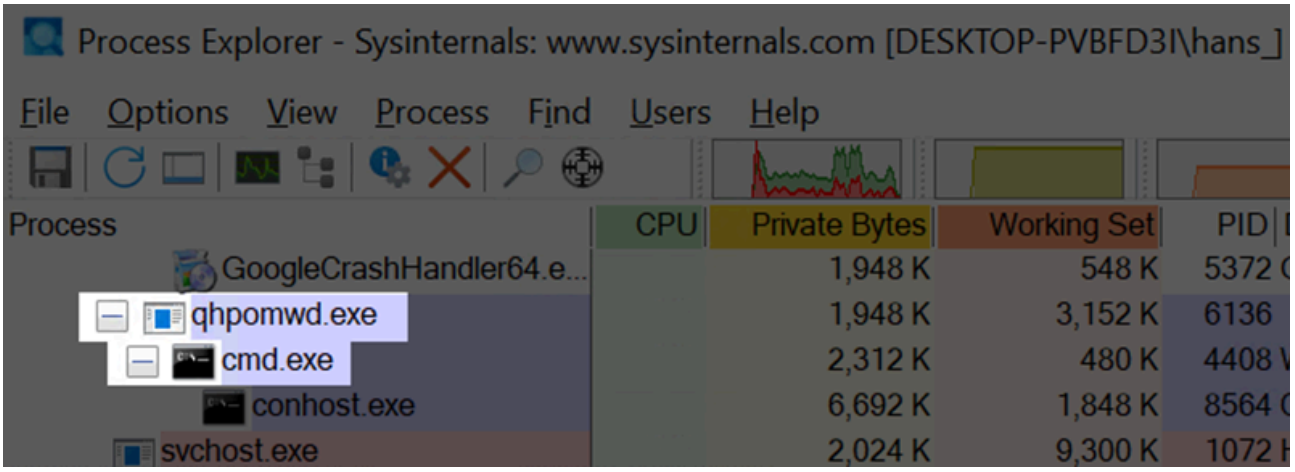

Figure 26 – Reverse Shell Spawning as Child of Implant When Running as Shellcode

The shellcode functionality is not only limited to a reverse shell, but also has full remote capabilities that can be injected into the implant at runtime, while being less obvious than spawning cmd.exe for a reverse shell. For example, figure 27 shows a Meterpreter instance running inside the implant, which provides a full remote toolset for performing a variety of post-exploitation activity. Because the remote functionality in Meterpreter is contained within the code loaded directly into memory, the implant no longer has a suspicious cmd.exe.

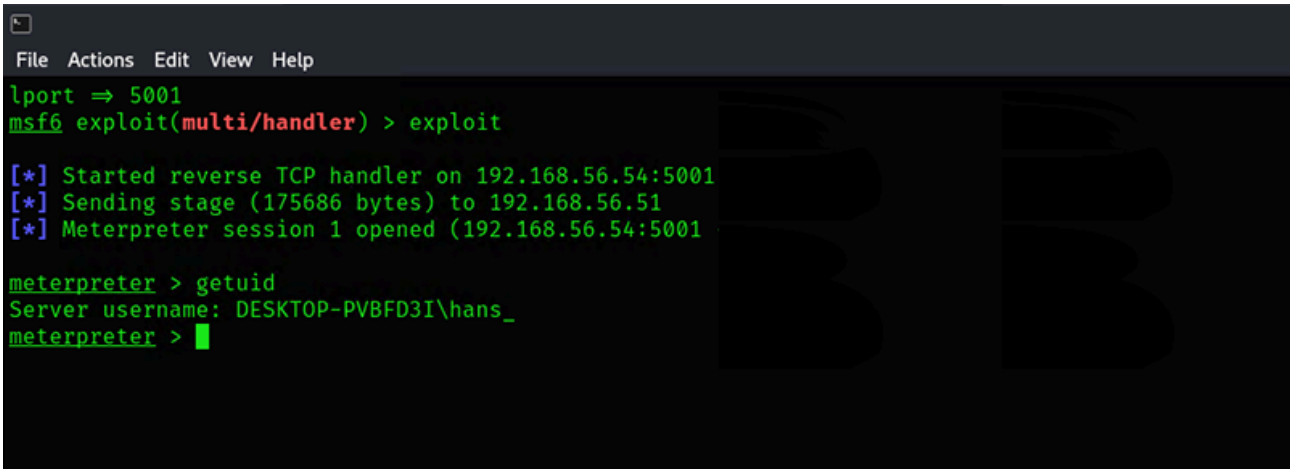

Figure 27 – Meterpreter Running After Being Injected via Shellcode Functionality

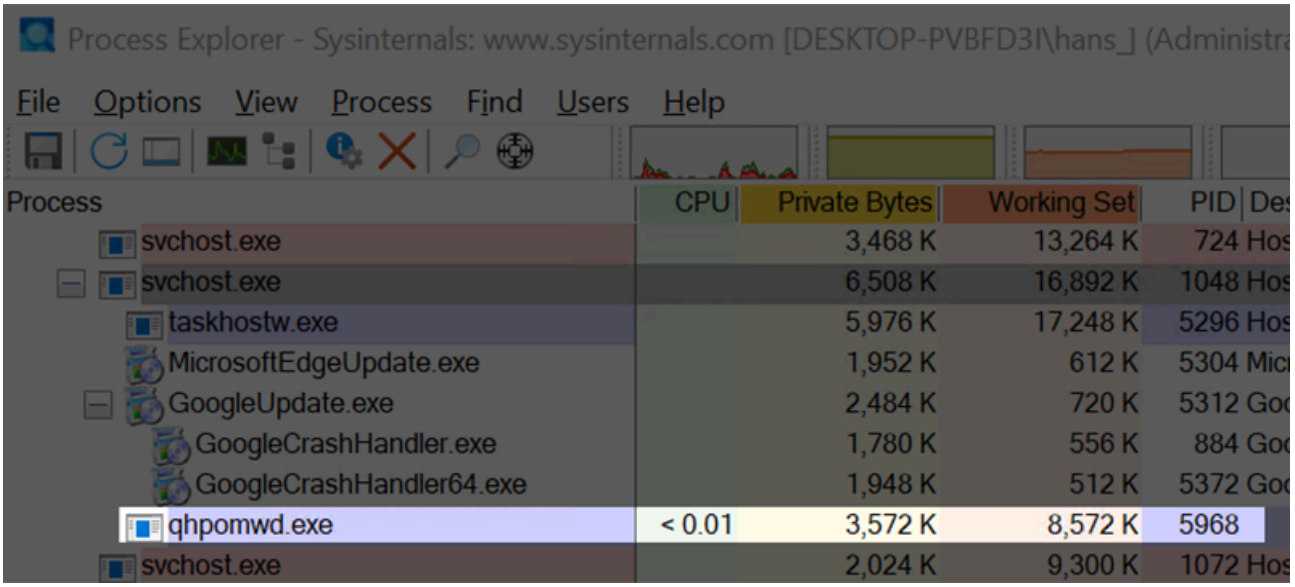

Figure 28 – Process Listing Showing No Obvious Sign Meterpreter is Running Within Implant

The loader, shellcode and ability to turn on SOCKS authentication functionality are duplicated in the settings page of the control panel. The difference is that on the settings page, any submissions will be sent to all active implants, with an option of setting a “repeat” timer.

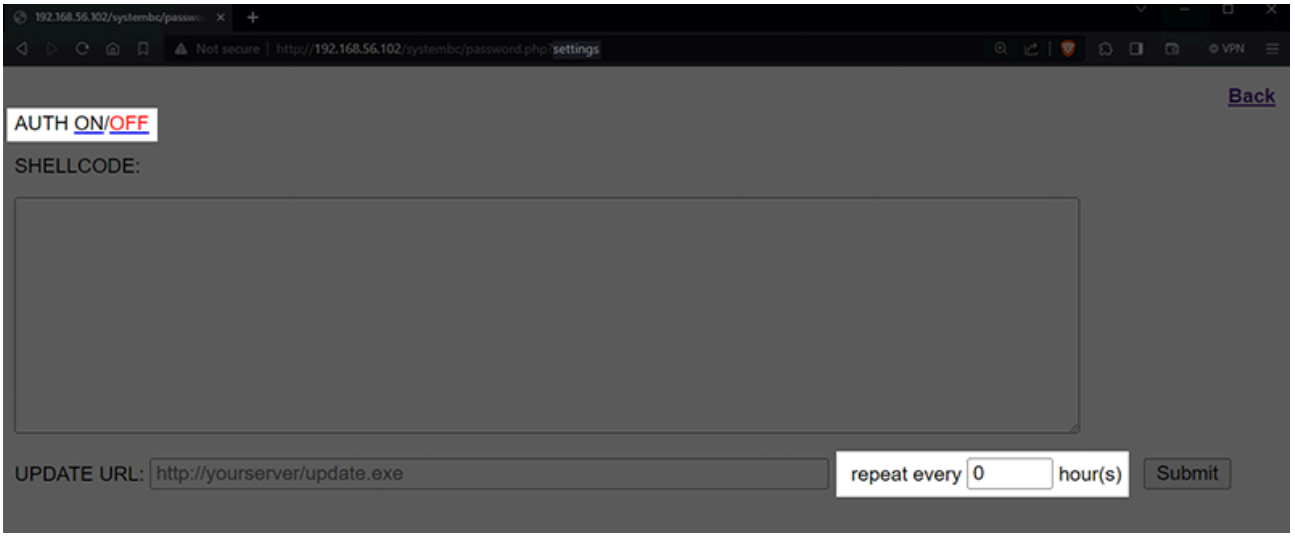

Figure 29 – Loader, Shellcode and SOCKS Functionality Duplicated for Global Deployment, with Repeat Scheduling

Conclusion

SYSTEMBC represents a real and significant threat to organizations, as cybercriminals typically deploy it as a method of maintaining access to an environment after initial compromise. Kroll has identified that SYSTEMBC is favored by RHYSIDA ransomware operators. In one such case impacting a large health care organization, threat actors accessed the system using compromised credentials coupled with a vulnerability in the client’s Citrix NetScaler environment. Shortly after access, the threat actors deployed SYSTEMBC, which established their beachhead into the victim environment, and allowed the actors to begin deploying additional tooling to further the

attack. The actors used multiple tools during the incident, including Advanced Port Scanner for network discovery, AnyDesk for remote access and MegaSync for exfiltration. After files were successfully encrypted, the actors changed passwords to the system so that IT employees could not access the network.

When looking at a compromised device from the threat-actor point of view, the Kroll CTI team was able to simulate the workflow of an attacker to recreate the launch of an attack using the SYSTEMBC toolset. Our experts were able to leverage the tool as a backdoor into a victim's network, demonstrating the tool's utility and ease of use. The program is commonly used for persistent access to a victim network or left behind as a secondary ingress point in case the primary is discovered and remediated.
