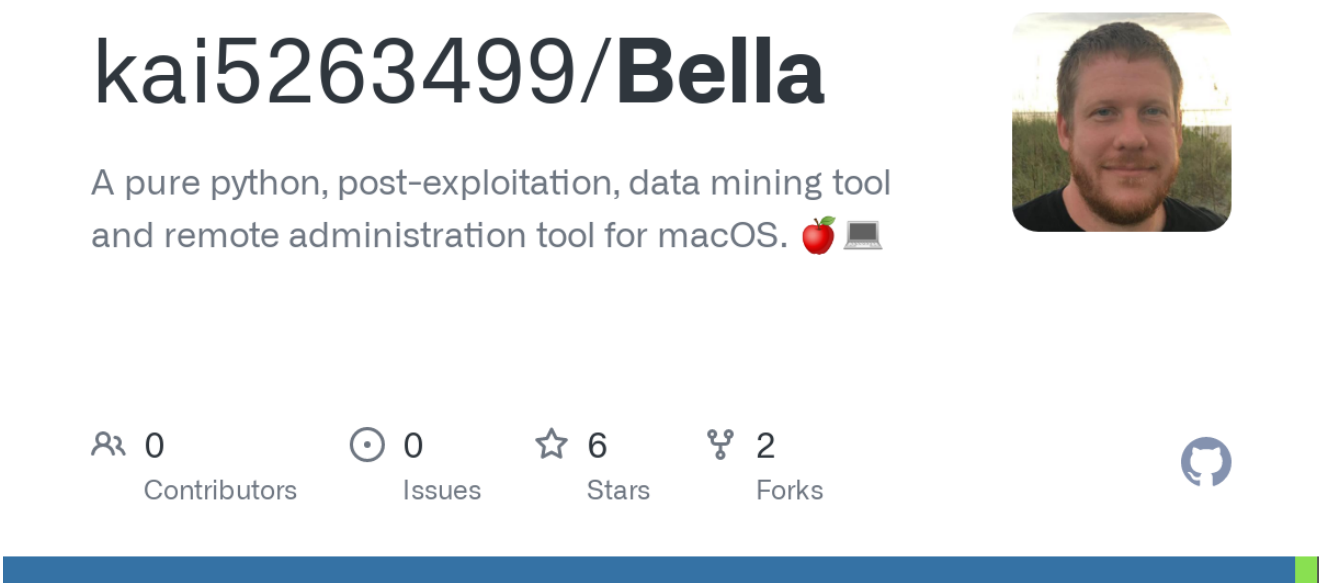
kai5263499/Bella: A pure python, post-exploitation, data mining tool and remote administration tool for macOS. 🍎💻

github.com/kai5263499/Bella

kai5263499



kai5263499/**Bella**

A pure python, post-exploitation, data mining tool and remote administration tool for macOS. 🍎💻

0 Contributors

0 Issues

6 Stars

2 Forks

master

This branch is 27 commits ahead of Trietptm-on-Security/Bella:master.

Contribute

README.md

#Bella Bella is a pure python post-exploitation data mining tool & remote administration tool for macOS. 🍎💻

##What is it? Bella is a robust, `pure python`, post-exploitation and remote administration tool for macOS.

`Bella` a.k.a. the `server` is an SSL/TLS encrypted reverse shell that can be dropped on any system running macOS >= 10.6. `Bella` offers the following features:

1. `Pseudo-TTY that emulates an SSH instance` [CTRL-C support for most functions, streaming output, full support for inline bash scripting, tab completion, command history, blocking command handling, etc].

2. `Auto installer!` Just execute the binary, and Bella takes care of the rest - a persistent reverse shell in a hidden location on the hard drive, undetectable by anti-viruses.

3. `Upload / Download any file[s]`

4. `Reverse VNC Connection.`

5. `Stream and save the computer's microphone input.`

6. `Login / keychain password phishing through system prompt.`

7. `Apple ID password phishing through iTunes prompt.`

8. `iCloud Token Extraction.`

9. `Accessing all iCloud services of the user through extracted tokens or passwords.`

   `This includes: iCloud Contacts, Find my iPhone, Find my Friends, iOS Backups.`

10. `Google Chrome Password Extraction.`

11. `Chrome and Safari History Extraction.`

12. `Auto Keychain decryption upon discovery of kc password.`

13. `macOS Chat History.`

14. `iTunes iOS Backup enumeration.`

15. `Extensive logging of all Bella activity and downloaded files.`

16. `VERY comprehensive data storage.` All information that Bella discovers [tokens, passwords, etc] is stored in an encrypted SQL database on the computer running Bella. This information is used for faster function execution, and a "smarter" reverse shell.

17. `Complete remote removal of Bella`

18. `An interactive shell for commands such as nano, ftp, telnet, etc.`

19. `A lot of other great features!` Mess around with it to see it in action.

These are some of the features available when we are in the userland. This shell is accessible at any time when the user has an internet connection, which occurs when they are logged in and the computer is not asleep.

If we get `root` , Bella's capabilities greatly expand.

Similar to the `getsystem` function on a meterpreter shell, Bella has a `get_root` function that will attempt to gain root access through a variety of means, including through a phished user password and/or local privilege escalation exploits if the system is vulnerable.

Upon gaining root access, Bella will migrate over to a hidden directory in /Library, and will load itself as a LaunchDaemon. This now provides remote access to the Bella instance **at all times**, as long as the computer has a network connection. Once we get root, we can do the following:

1. `MULTI-USER SUPPORT!` Bella will keep track of all information from any active users on the computer in a comprehensive database, and will automatically switch to the active computer user. All of the aforementioned data extraction techniques are now available for every user on the machine.

2. `Decrypt ALL TLS/SSL traffic and redirect it through the control center!` [a nice, active, MITM attack]

3. `Disable/Enable the Keyboard and/or Mouse.`

4. `Load an Insomnia KEXT to keep a connection open if the user closes their laptop.`

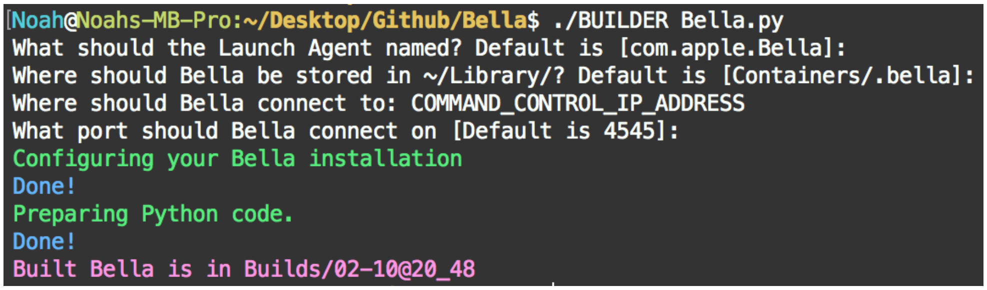
5. `Automatic dumping of iCloud Tokens and Chrome passwords` [leverages keychaindump and chainbreaker if SIP is disabled]

6. `A lot of behind the scenes automation.`

##HOW TO USE

Bella's `power` lies in its high level of automation of most of the painstaking tasks that one faces in a post-exploitation scenario. It is incredibly easy to **setup and use**, requires no pre-configuration on the target, and very little configuration on the Control Center. It leverages the *incredible* behind the scenes power of macOS and Python for a fluid post-exploitation experience.

1. Download / clone this repository onto a macOS or Linux system.

2. Run ./BUILDER and enter the appropriate information. It should look something like this:


```
[Noah@Noahs-MB-Pro:~/Desktop/Github/Bella$ ./BUILDER Bella.py
What should the Launch Agent named? Default is [com.apple.Bella]:
Where should Bella be stored in ~/Library/? Default is [Containers/.bella]:
Where should Bella connect to: COMMAND_CONTROL_IP_ADDRESS
What port should Bella connect on [Default is 4545]:
Configuring your Bella installation
Done!
Preparing Python code.
Done!
Built Bella is in Builds/02-10@20_48
```

3. That's it! Bella is all ready to go. Just upload and execute `Bella` on your macOS target.

4. Now run `Control Center.py` on your macOS or Linux control center. It requires no-dependencies [except for mitmproxy if you want to MITM]. It will do some auto-configuration, and you will see something like this after a few seconds. ![](Screenshots/Found Clients.png) The Control Center will constantly update this selection, for up to 128 separate computers.

5. Press `Ctrl-C` to choose from the selection, and then type in the number of the computer that you want. You will then be presented with a screen like this. ![](Screenshots/Command entry.png)

6. Start running commands! `bella_info` is a great one. Run `manual` to get a full manual of all of the commands. Also, you can hit tab twice to see a list of available commands. ![](Screenshots/Bella Info.png)

VERY IMPORTANT DISCLAIMER: USE BELLA RESPONSIBLY. BY USING BELLA YOU AGREE TO THE MIT LICENSE CONTAINED IN THIS REPOSITORY. READ THE LICENSE BEFORE USING BELLA.

Little note: Bella works across the internet, if you do some configuration. Configure your firewall to forward Bella's port to your Control Center. Other important ports to forward: 1) VNC - 5500. 2) Microphone - 2897. 3) MITM

- 8081. 4.) Interactive Shell - 3818

Also, VNC relies on the RealVNC application for macOS, as it is one of the few clients that supports a reverse VNC connection. It is free to download and use.

VNC and Microphone streaming are not yet supported for Linux control centers.

##Other Information This project is being **actively** maintained. Please submit any and all bug reports, questions, feature requests, or related information.

Bella leverages keychaindump, VNC, microphone streaming, etc, by sending base64 encoded C binaries over to the Bella server / target. I have included pre-compiled and encoded files in the Payloads/payloads.txt file. If you wish to compile your own version of these payloads, here is what to do after you compile them:

1. Encode them in base64 and put them in the payloads.txt in the following order, each one separated by a new line.
2. vnc, keychaindump, microphone, rootshell, insomnia, lock_icon, chainbreaker.

payload_generator in the Payloads directory should help with this.

Please let me know if you have any issues.

###HUGE thanks `https://github.com/juuso/keychaindump`

`https://github.com/n0fate/chainbreaker`

`https://github.com/richardkiss/speakerpipe-osx`

`https://github.com/semaja2/InsomniaX`

`https://github.com/stweil/OSXvnc`

`https://bugs.chromium.org/p/project-zero/issues/detail?id=676&redir=1`

###TODO

1. `Reverse SOCKS proxy to tunnel our traffic through the server.`

2. `Firefox password decryption / extraction`

3. `Keystroke logging with legible output [80% done]`

4. `VNC and Microphone functionality for a Linux Control Center`

####Some design points

1. As previously stated, Bella is a pseudo-TTY. By this, the base socket and remote code execution handling of Bella is a fairly abstracted version of a very simple request-response socket. Bella receives a command from the server. If the command matches a pre-programmed function (i.e chrome history dump), then it will perform that function, and send the response back to the client. The client will then handle the response in the same way. After processing the response, it will prompt the client for another command to send.

2. Issues with a low-level socket are numerous, and not limited to: 3. Program execution that blocks and hangs the pipe, waiting for output that never comes (sudo, nano, ftp) 4. Not knowing how much data to expect in the socket.recv() call. 5. Not being able to send ctrl-C, ctrl-Z and similar commands. 6. No command history 7. A program that crashes can kill a shell. 8. One-to-one response and request.

3. Bella address the above by: 4. recv() and send() functions that serialize the length of the message, and loop through response/requests accordingly. 5. Readline integration to give a more 'tty' like feel, including ctrl-C support, command history, and tab completion. 6. Detecting programs that block, and killing them. 7. Allowing multiple messages to be sent at once without the client prompting for more input (great for commands like ping, tree, and other commands with live updates).

For full information on the pre-programmed functions, run the `manual` command when connected to the server.

--