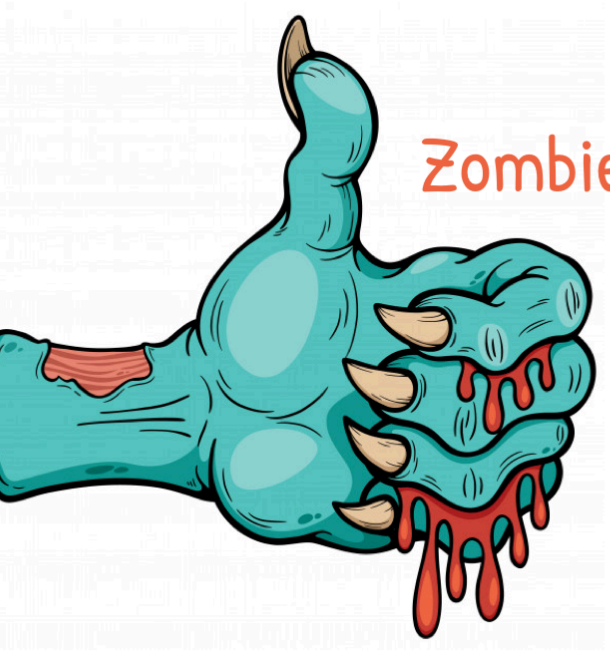
ZombieVPN
Breaking That Internet Security

The little story of CVE-2020-12828

Discovered by 0xSha



Meet Zombie VPN

What ?
It's SYSTEM level a code execution on a special VPN SDK (anchorFree), which mostly used by privacy and Antivirus vendors.

Yes!, A name and logo because you'll see it deserve it! Well, I'll spare you with the separate domain though 🙄

A 0day ?
No. The issue is patched; CVE-2020-12828

Introduction

I know you may think, seriously? A name and logo? I know, but I want your attention. In the past, a lot of researchers, notably Alex wheeler, Joxean Koret, Tavis Ormandy proven endpoint security in a lot of cases, makes your system more vulnerable. The journey is a long and exciting one. So sit tight.

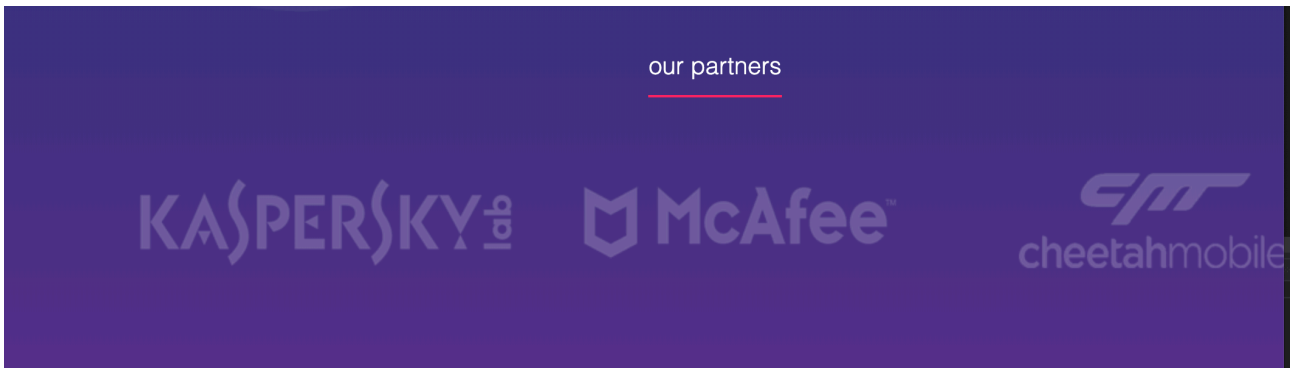
It starts by downloading a total internet security.

Before I start looking for security issues, I checked around to see what kind of programs are out there, so I ended up downloading the latest BitDefender total security. I installed it on a windows virtual machine, A few hours later, I find out a thing I didn't know: a rush and substantial competition between endpoint security and privacy tools sellers to win the VPN client market. I didn't use any personal internet security tool for a long while, So I had no idea. They evolve so much. The competition makes competitors buy or outsource VPN part of their endpoint security; in some cases, they even didn't rename demo SDKs and runs them with SYSTEM and SERVICE access. So result? They ship untested software. But why they do this? You be the judge.

Why did I tell you all of this? Because this is the **very first vulnerability**, the rush means less time to test code. Less tested code = vulnerable, so there is a very high chance you can find similar vulnerabilities on the less reviewed portion of codes.

Yet, Another Serialization Vulnerability

While auditing the VPN of Bitdefender total security 2020, I found out they use VPN SDK from AnchorFree. You can find it on Github. Next, I visit their developer website to make sense of their API and saw this.



Oh my god! So it means one vulnerability on this SDK can exploit hundreds of millions of VPN users worldwide. What an exciting find, already.

Deep Dive In SDK Code

It's 2020, and manual code auditing is still amazing and the most satisfying! Put your code audit hats on. I'm going to take you on tour. Its called cake tube SDK tour and it's free! This file coded with .net framework. It's a sample file, and it's supposed to be modified by each customer.

I have successfully exploited a case (Bitdefender total security 2020) as a proof of concept. At the time of this writing, the vulnerability exists in 1.2.13.81, which was the latest).

In our case study, I found a file called vpnservice.exe on this path.

C:\Program Files\Bitdefender\Bitdefender VPN\

This so-called VPN server starts and listens on a TCP port.

Let's dive in code

```
    private void listenCycle()
        {
            for (;;)
            {
                TcpClient client = this._tcpListener.AcceptTcpClient();
                Task.Run(delegate()
                {
                    this.processClientRequest(client);
                });
            }
        }
```

Now let's look at how it processes unauthenticated user requests

```
private void processClientRequest(TcpClient client)
        {
            NetworkStream stream = client.GetStream();
            try
            {
                for (;;)
                {
```

```
                    byte[] array = new byte[client.Available];
                    int num = stream.Read(array, 0, array.Length);
                    if (num == 0 && !stream.DataAvailable)
                    {
                        break;
                    }
                    if (num != 0)
                    {
                        string @string = Encoding.UTF8.GetString(array);
                        this.OnDataReceived(@string, client);
                    }
                }
                client.Close();
            }
            catch (Exception)
            {
                client.Close();
            }
        }
```

Nothing special here; it will read user input as stream and passes it to OnDataRecived function.

```
protected virtual void OnDataReceived(string data, TcpClient client)
        {
            ReceivedDataHandler dataReceived = this.DataReceived;
            if (dataReceived != null)
            {
                dataReceived(data, client);
            }
        }
```

So this is wrapper for dataRecivied function and there is few more interesting functions here.

```
        public void SendResponse(string result, TcpClient client)
        {
            Stream stream = client.GetStream();
            byte[] bytes = Encoding.UTF8.GetBytes(result);
            stream.BeginWrite(bytes, 0, bytes.Length, null, null);
        }
```

and start function

```
     public void Start(int tcpPort)
        {
            this._managementTcpServer = new ManagementTcpServer(tcpPort);
            this._managementTcpServer.DataReceived +=
this.managementTcpServerOnDataReceived;
            this._managementTcpServer.Start();
        }
```

this function has an exciting information for us the data received by TCP Server

```
    // Token: 0x0600004F RID: 79 RVA: 0x0000307C File Offset: 0x0000127C
        private void managementTcpServerOnDataReceived(string data, TcpClient
client)
        {
            ManagementCommand command =
this._commands.FirstOrDefault((ManagementCommand managementCommand) =>
managementCommand.CanHandle(data));
```

```
            if (command != null)
            {
                Task.Run(delegate()
                {
                    string result = command.Handle(data);
                    this._managementTcpServer.SendResponse(result, client);
                });
            }
```

This seems to accept commands, and most importantly, it can handle them.

```
ManagementCommand command = this._commands.FirstOrDefault((ManagementCommand managementCommand) => managementCommand.CanHandle(data));
```

Handle function is overridden three times by three different commands check, connect and disconnect.

here is what will happen

```
ManagementCommand command = this._commands.FirstOrDefault((ManagementCommand managementCommand) => managementCommand.CanHandle(data));
protected string GetJsonString(string data)
        {
            return data.Replace("\\", "\\\\");
        }

        // Token: 0x06000040 RID: 64 RVA: 0x00002A38 File Offset: 0x00000C38
        [return: Dynamic]
        protected dynamic GetJsonObject(string data)
```

```
        {
            JavaScriptSerializer javaScriptSerializer = new
JavaScriptSerializer();
            string jsonString = this.GetJsonString(data);
            return javaScriptSerializer.Deserialize<object>(jsonString);
        }
```

Alright, we can see the data passed to javaScriptSerializer, and the only defense mechanism seems to be data. Replace("\\", "\\\\") to get JSON string, I felt like I already found a vulnerability, and rushed to exploit this using ysoserial; after too many tries, I found it JavaScriptSerializer is only exploitable if the function initiated with SimpleTypeResolver(). I even contacted the ysoserial creator and asked him if there is any trick. Here is his reply.

No, sorry, if there is no type resolver configured, you won't be able to instantiate arbitrary types.

Oh, I felt terrible for a moment because I thought I had a moment, and I have broken the internet, but now it seems even though it passes the user input to unserialize its a safe unserialize, or is it now?

Data Only Attack For Everything

I can't explain the idea of data only attack in even a single separate article, but relatively speaking when you try to exploit any "kind" of vulnerability; you always can "use" what you "have" this concept used to exploit memory corruptions, logical bugs, web vulnerabilities, and ...

Our vulnerability is one good real-world example I thought it okay we couldn't initiate an object using this insecure deserialization, but we still can speak to this endpoint unauthenticated from any privilege, and it runs as SYSTEM. What if we can exploit deserialization by making a valid "malformed" object to make the program misbehave somehow.

Reversing and more Auditing

Remember tree commands; it turns out those three are the objects we can forge. I analyzed all three, and the most exciting one from the attacker's perspective is the connect function.

```
    // Token: 0x06000049 RID: 73 RVA: 0x00002B9B File Offset: 0x00000D9B
        public override bool CanHandle(string data)
        {
            return base.IsCommandHasName(data, "connect");
```

```
        }
```

On Handling connect commands, it will launch a process, and we can partially control it !.

```
          public async Task<VpnProcessLaunchResult> TryLaunchAsync(string
vpnExecutablePath, string protocol, string ip, string port, string authFilename,
bool enableLog)
        {
            VpnProcessLaunchResult result;
            if (this.isLaunched(this._vpnProcess))
            {
                result = new VpnProcessLaunchResult(true, this._vpnProcess.Id);
            }
            else
            {
                try
                {
                    ProcessStartInfo info =
this._processInfo.GetInfo(vpnExecutablePath, protocol, ip, port, authFilename,
enableLog);
                    ConfiguredTaskAwaitable<bool>.ConfiguredTaskAwaiter
configuredTaskAwaiter =
this.tryLaunchProcessAsync(info).ConfigureAwait(false).GetAwaiter();
                    if (!configuredTaskAwaiter.IsCompleted)
                    {
                        await configuredTaskAwaiter;
                        ConfiguredTaskAwaitable<bool>.ConfiguredTaskAwaiter
configuredTaskAwaiter2;
                        configuredTaskAwaiter = configuredTaskAwaiter2;
                        configuredTaskAwaiter2 =
default(ConfiguredTaskAwaitable<bool>.ConfiguredTaskAwaiter);
                    }
```

```
                    if (configuredTaskAwaiter.GetResult())
                    {
                        return new VpnProcessLaunchResult(true,
this._vpnProcess.Id);
                    }
                }
                catch (Exception)
                {
                }
                result = new VpnProcessLaunchResult(false, 0);
            }
            return result;
        }
```

This function will check if there is an openvpn.exe running or not. If not, It will try to spawn one using user-controlled arguments. We can forge all three commands to introduce new exploitation vectors. Finally, Look at GetInfo() function.

```
public ProcessStartInfo GetInfo(string vpnExecutablePath, string protocol,
string ip, string port, string authFilename, bool enableLog)
        {
            string fileName = Path.Combine(vpnExecutablePath, "openvpn.exe");
            string text = Path.Combine(vpnExecutablePath, "cert.pem");
            string text2 = Path.Combine(vpnExecutablePath, authFilename);
            string str = Path.Combine(vpnExecutablePath, "ovpn.log");
            string text3 = string.Concat(new string[]
            {
                "--management 127.0.0.1 40742 --service ",
                this.SignalEventName,
                " 0 --client --dev tun --proto ",
                protocol,
```

```
                " --redirect-gateway def1 --remote ",
                ip,
                " ",
                port,
                " --ca \"",
                text,
                "\" --auth-user-pass \"",
                text2,
                "\" --verb 3 --ping-exit 15"
            });
            if (enableLog)
            {
                text3 = text3 + " --log \"" + str + "\"";
            }
            return new ProcessStartInfo
            {
                CreateNoWindow = true,
                UseShellExecute = false,
                FileName = fileName,
                WindowStyle = ProcessWindowStyle.Hidden,
                Arguments = text3
            };
        }
```

Now let's go back one more time on this part of code .Now I understand why this function makes sense.

```
protected string GetJsonString(string data)
        {
            return data.Replace("\\", "\\\\");
        }
```

Now let's go review it one more time. JsonString is saving the day by making UNC path injection harder.Oh, it looks just like a dead end. So this is it? Not at all, we can always obfuscate our payloads or find more issues by reading the API internal codes.

Case Study Exploiting Bitdefender 2020 total security

The exploit developed tested on Bitdefender Total Security 2020 and windows 7 and 10

Exploitation is an art; there are so many creative ways to exploit a vulnerability. There are at least a few paths I thought about when I was trying to exploit this issue. I'm sure these are not the only paths you can always think of something clever.

- Spoof openvpn.exe and ignore parameters (privilege-ESC, RCE)
- Find a bypass for UNC path (unauthenticated, RCE)
- Rough Open-VPN server (remotely sniff all clients packets, this idea appears to me later)
- …

Practical Exploit Scenario One

The crafted payload is all you have to consider in this exploit we simply can execute arbitrary payloads how marvellous.

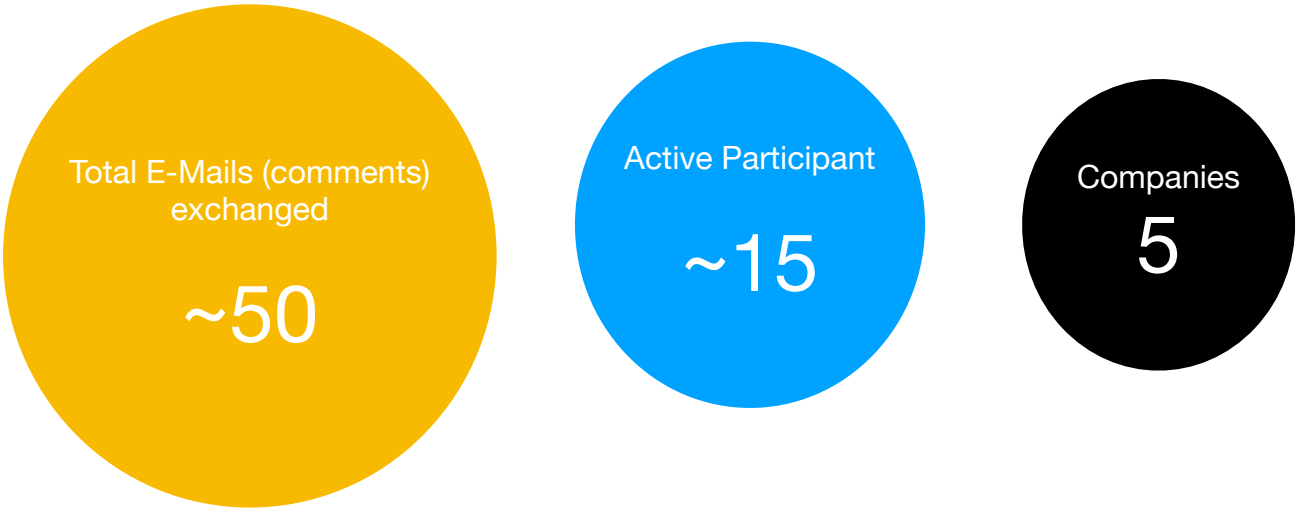
Here are steps to replicate

Step I: run netstat -anb to find out vpnservice.exe (remotely guessable)
Step II: change the port with your vpnservice.exe port
Step III: change C:\Users\LowPriv with the path you want to run the privileged command from
Step IV: copy your file as openvpn.exe in C:\Users\LowPriv
Step V: Run the PoC and your payload will be executed as SYSTEM.

Victory dance.

Timeline:

- I found the vulnerability (5 months ago)
- The same day, the issue reported to Bitdefender. (5 months ago)
- The issue also reported to HackerOne internet bug bounty like a rookie. (because I didn't pay attention, I should've waited for the patch.)
- Bitdefender triaged vulnerability also paid a bounty for it. (3 months ago, thanks <3)
- I asked for update a month after it (3 months ago)
- I heard the issue is still there. (2 months ago)
- HackerOne BOT closed the issue as N/A (2 months ago)
- Bitdefender referred the contact to the vendor. (1 month ago)
- I contacted them and asked them about the update. (1 month ago)
- They replied warmly with this: CVE-2020-12828 has been assigned (25 days ago)
- AnchorFree made an exception for their bug bounty program and offered a prize. (23 days ago, thanks <3)
- HackerOne re-opened the Issue (20 days ago)
- CVE released by MITRE , Acknowledged by Pango (22 days ago)
- HackerOne triaged the vulnerability (23 days ago)
- And the story goes on …



PoC and Demo

Given the number of affected devices, this is probably one of the world's shortest exploit ever created that works reliably on modern systems and affects a massive amount of machines 😑. Basic PoC at github.com and lame demo at https://www.youtube.com #PoC||GTFO

Conclusion

It's 2020, and it's possible to use classic techniques to find underlying vulnerabilities in security vendors themselves. Statically speaking, this issue affected hundreds of millions of devices. Because of the nature of this vulnerability, it's possible to make 100% reliable exploits for it, not being a memory corruption bug means all the memory mitigations (ASLR, DEP, ETC) are inefficient.It took around six months from discovery to releasing this paper. Responsible disclosure is a long, even sometimes frustrating path, that’s why we need professional teams working as a bridge to handle such a long process.

Finally, I genuinely need to appreciate many people and companies worked with me in this case, Especially people at Bitdefender, BugCrowd, Pango Inc, and HackerOne.

I hope you enjoyed reading this story.

References:

https://www.blackhat.com/presentations/bh-usa-05/bh-us-05-wheeler.pdf
www.joxeankoret.com/download/breaking_av_software-pdf.tar.gz
https://lock.cmpxchg8b.com/sophailv2.pdf
https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-Json-Attacks.pdf

Jun 2020 0xSha