

Windows Local Admin Brute Force Attack Tool (LocalBrute.ps1)

 infosecmatter.com/windows-local-admin-brute-force-attack-tool-localbrute-ps1

November 8, 2020

In this post, we will be introducing a new minimalistic tool for local privilege escalation attacks in Microsoft Windows systems. The tool is called localbrute.ps1 and it is a simple local Windows account brute force tool written in pure PowerShell.

It doesn't require any 3rd party modules and it is very small in size, which makes it a viable addition to traditional privilege escalation attacks, applicable to various penetration testing scenarios.

Why attacking local Windows accounts?

Attacking local administrative accounts such as the built-in "Administrator" account or any other account that is member of the "Administrators" local group, can be quite an interesting attack vector mainly because of the lack of account lockout policy.

We can literally try as many login attempts as we want.

If we succeed, we will have full control over the system and we will be able to do all the juicy things that we like to do as pentesters, for example:

- Disable any defenses and security controls on the system
- Extract plaintext credentials from memory and other places (files, registry etc.)
- Craft raw network packets and run exploits to attack other systems on the network
- Access protected areas of the system to locate sensitive information and many other things ..

All these things can help us with the lateral movement, moving further into the infrastructure and demonstrating impact to the customer.

Attacking local Windows accounts is nothing new, but here we DO NOT aim to do it remotely using typical pentesting tools such as Metasploit smb_login scanner, Nmap smb-brute NSE script, CrackMapExec or any other similar tools.

The localbrute PowerShell tool presented here does the brute forcing locally on the target system itself and so its usage is quite specific..

Where does this tool fit in?

This tool can be useful in cases where we have gained a low privileged user access to a Windows machine and we can run commands on it – e.g. via RDP session or via terminal services.

We could also use this tool in case when we are testing some kind of a restricted or isolated environment – e.g. a VDI environment where we were provided only a user level access and now we are supposed to do the testing from there with limited access to our favorite pentesting utilities.

Another use case would be a simulation of a disgruntled employee. Having access to a sample employee workstation, possibly hardened and protected by various security controls. Could we do something and cause a potential damage to the organization?

In all these cases, the `localbrute.ps1` tool could help us in escalating our privileges.

Tool features

In a nutshell, the `localbrute.ps1` tool performs automated login attempts locally on the system, using built-in Windows functionalities. Here are the main features of the tool:

- Performs login attacks against any selected local account using a supplied wordlist
- Small and minimalistic – can be easily typed out by hand (on the keyboard)
- Written in pure PowerShell – no additional modules needed
- Non-malicious – undetected by AV / EDR solutions

There are two versions of the `localbrute.ps1` tool currently available in the GitHub [repository](#) – the [extra-mini](#) version and the [normal](#) version. The only difference is that the normal version is slightly longer and it has the following additional features:

- Supports resuming, if interrupted
- Detects already compromised user accounts

The following sections describes how to use the tool and how does it work in detail.

LocalBrute.ps1 tool usage

1) First thing we need to do is to identify administrative user accounts on the system. These typically include:

- Members of the local Administrators group
- The local Administrator account itself

Here's how we can find members of the local Administrators group:

```
net localgroup administrators
```

```

PS C:\Users\Public> net localgroup "Administrators"
Alias name      Administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members
-----
Administrator
srvadm
The command completed successfully.

PS C:\Users\Public>

```

2) Now to run the localbrute tool, simply do:

```
Import-Module .\localbrute.ps1
```

Usage:

```
localbrute <username> <path-to-wordlist> [debug]
```

Example:

```
localbrute Administrator .\rockyou.txt
```

Here's an example:

```

PS C:\Users\Public> import-module .\localbrute.ps1
PS C:\Users\Public>
PS C:\Users\Public> localbrute "Administrator" .\rockyou.txt
Password for Administrator account found: Pa$$word123
PS C:\Users\Public>

```

Note that it can take a long time until the password is found. See below..

How it works?

The tool simply iterates through the supplied wordlist (password list) line by line and tries to authenticate as the specified user account locally on the system.

It uses internal Windows DirectoryServices.AccountManagement functionalities in the context of the local machine. In effect, this allows us to test authentication for any local account.

Here's a standalone PowerShell code snippet to validate single pair of credentials locally:

```

$u = 'Administrator'
$p = 'Pa$$w0rd!'
Add-Type -AssemblyName System.DirectoryServices.AccountManagement
$t = [DirectoryServices.AccountManagement.ContextType]::Machine
$a = [DirectoryServices.AccountManagement.PrincipalContext]::new($t)
$a.ValidateCredentials($u,$p)

```

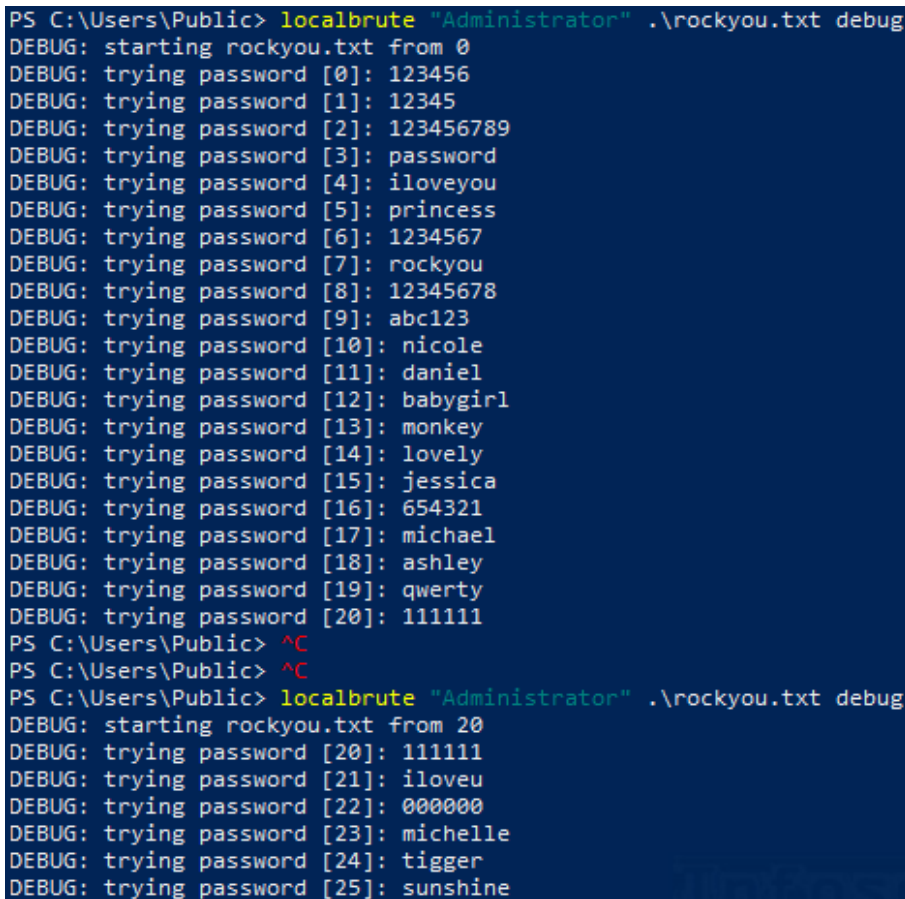
With a little bit of PowerShell scripting, we simply wrapped this code into a loop and that's exactly how the localbrute tool works.

The enhanced (longer) version has some additional functionality to improve usability when working with large wordlists. Namely it keeps a state file (localbrute.state) in the current working directory to keep track of the progress.

Upon interruption (^C), the tool will record the last password candidate that was reached from the given wordlist for the given username. This allows the tool to continue (resume) the attack once the tool is restarted.

The state file also keeps record of already compromised accounts.

You can turn on the debug mode to see exactly what the tool is doing. Here's an example:



```
PS C:\Users\Public> localbrute "Administrator" .\rockyou.txt debug
DEBUG: starting rockyou.txt from 0
DEBUG: trying password [0]: 123456
DEBUG: trying password [1]: 12345
DEBUG: trying password [2]: 123456789
DEBUG: trying password [3]: password
DEBUG: trying password [4]: iloveyou
DEBUG: trying password [5]: princess
DEBUG: trying password [6]: 1234567
DEBUG: trying password [7]: rockyou
DEBUG: trying password [8]: 12345678
DEBUG: trying password [9]: abc123
DEBUG: trying password [10]: nicole
DEBUG: trying password [11]: daniel
DEBUG: trying password [12]: babygirl
DEBUG: trying password [13]: monkey
DEBUG: trying password [14]: lovely
DEBUG: trying password [15]: jessica
DEBUG: trying password [16]: 654321
DEBUG: trying password [17]: michael
DEBUG: trying password [18]: ashley
DEBUG: trying password [19]: qwerty
DEBUG: trying password [20]: 111111
PS C:\Users\Public> ^C
PS C:\Users\Public> ^C
PS C:\Users\Public> localbrute "Administrator" .\rockyou.txt debug
DEBUG: starting rockyou.txt from 20
DEBUG: trying password [20]: 111111
DEBUG: trying password [21]: iloveu
DEBUG: trying password [22]: 000000
DEBUG: trying password [23]: michelle
DEBUG: trying password [24]: tigger
DEBUG: trying password [25]: sunshine
```

Speed limitations

Now when it comes to speed, the tool can do around 100-200 login attempts per second, depending on the system performance.

True, this is not particularly breathtaking, but it is still much faster than any SMB login capable tool directed to do remote login attack on local accounts over the network.

Here's a better overview on the attack speed based on the runtime duration:

Runtime duration	Number of login attempts
1 second	100 – 200
1 minute	6k – 12k
1 hour	360k – 720k
1 day	8.6M – 17.3M

This means that we could for example process the whole rockyou.txt wordlist (14.3M entries) between 19.9 – 39.8 hours (1-2 days). This is not so bad and in the outlined scenarios above, it's definitely realistic to have the brute forcing attack running for a prolonged period of time.

Note that if the debug mode is on, the speed is impaired by about 20-30%.

Running localbrute in parallel?

BEWARE: Running multiple instances of the localbrute script in parallel will NOT increase the speed. In fact, it will result in the following exception very soon:

```
Exception calling "ValidateCredentials" with "2" argument(s): "Multiple connections to a server or shared resource by the same user, using more than one user name, are not allowed. Disconnect all previous connections to the server or shared resource and try again."
```

So, DO NOT run the script in parallel, because at any point in time, only one instance can be calling the ValidateCredentials() system method.

Conclusion

The presented localbrute.ps1 script is a simple login brute force tool that can offer an additional method of privilege escalation attacks on Windows systems.

Due to the lack of account lockout policy on local accounts, we can use it to test the password strength of the locally privileged accounts and discover accounts configured with weak passwords.

Thanks to its compact size, it can come handy during a variety of penetration tests and offensive simulations, similarly as the other minimalistic tools released earlier:

- [Active Directory Brute Force Attack Tool in PowerShell \(ADLogin.ps1\)](#)
- [SMB Brute Force Attack Tool in PowerShell \(SMBLogin.ps1\)](#)
- [Port Scanner in PowerShell \(TCP/UDP\)](#)

Hope you will find it useful sometimes!

If you like our tools and you would like more, please do subscribe to our mailing list and follow us on [Twitter](#), [Facebook](#) or [Github](#) to not miss any new additions!

SHARE THIS

TAGS | [Brute force](#) | [Isolated environment](#) | [Login attack](#) | [Minimalistic](#) | [Penetration testing](#) | [PowerShell](#) | [Privilege escalation](#) | [Restricted environment](#) | [Scripting](#) | [Tool](#) | [Windows](#) | [Wordlist](#)
