whiteprocess
Whitelist Threads Filter for UNIX Servers
Documentation and Analysis

Author: Stefano Gorresio
Software: whiteprocess PyELF 0.8 beta

Date: 22 July 2017
Last Update: 3 February 2018

Contents

1 Premise

This documentation will be updated with the development of the various versions.

In previous papers, I have recommended the use of ”whitelist systems” for process/threads and connection security.

I have noticed that a system for filtering in white list executable is not easily present (solutions that I found are not completely exhaustive).

I also found interesting articles that discussed the lack of many solutions on UNIX systems.

Since I could not find a software that can ”brutally” filter executables on UNIX... I decided to create it I.

Note: If software how this is already present… please let me know.

2 Testing Environment

All tests are made on servers running the follow operating systems:

- Debian 9
- FreeBSD
- Ubuntu Server 16.04
- CentOS 7
- Alpine Linux (see troubleshooting)

Client used for ”security test” is a laptop running Kali Linux.

3 General Functioning of whiteprocess

whiteprocess is a simple software (for now it is in beta version) programmed in Python 2.7 and compiled in ELF for UNIX systems which is used to filter threads in whitelist.

Its purpose is to optimize the work that antivirus offers, performing a filtering of all process/threads that the user did not declare to be ”allowed” to run.

whiteprocess replaces antivirus on server.

Its objectives are:

- Blocking malwares and unwanted threads (executables and scripts)
- Blocking buffer overflow attacks
- Regularly check the running threads
- Offer a user-friendly security system

Its functioning is very simple:

When started, it read various lists (executables, arguments, ecc.) and various parameters from configuration file (default **/etc/whiteprocess.conf**).

Periodically, it control if each thread respects the conditions declared in config file. If a thread doesn’t respect the conditions, whiteprocess kills it.

The kernel threads will not be taken into account.

Time from one control to another and what controls should be done, are managed by configuration file.

All operations are recorded on log file (default **/var/log/whiteprocess.log**).

whiteprocess must be run as root and in background.

As you can easily notice, it runs in user mode.

4 Installation and Uninstallation

The installation process create:

- configuration file: **/etc/whiteprocess.conf**
- directory: **/usr/share/whiteprocess/**
- main executable: **/usr/bin/whiteprocess**
- daemon launcher: **/usr/bin/whiteprocessd**
- whiteprocess tools: **/usr/bin/whiteprocess_tools**
- status file: **/usr/share/whiteprocess/whiteprocess.status**
- template config file: **/usr/share/whiteprocess/whiteprocess.conf**

The uninstallation process remove all except log file.

4.1 Installation

For install whiteprocess, download the correct package for your OS and in package directory type **./install** command with root permission.

```
utente@server-generico:~/whiteprocess-package$ sudo ./install
'whiteprocess.conf' -> '/usr/share/whiteprocess/whiteprocess.conf'
'whiteprocess.conf' -> '/etc/whiteprocess.conf'
'whiteprocess' -> '/usr/bin/whiteprocess'
'whiteprocessd' -> '/usr/bin/whiteprocessd'
'whiteprocess_tools' -> '/usr/bin/whiteprocess_tools'

whiteprocess installed.
utente@server-generico:~/whiteprocess-package$
```

4.2 Uninstallation

For uninstall whiteprocess, in package directory type **./uninstall** command with root permission.

```
utente@server-generico:~/whiteprocess-package$ sudo ./uninstall
removed '/usr/share/whiteprocess/whiteprocess.conf'
removed '/usr/share/whiteprocess/whiteprocess.status'
removed directory '/usr/share/whiteprocess'
removed '/usr/bin/whiteprocess'
removed '/usr/bin/whiteprocess_tools'
removed '/usr/bin/whiteprocessd'
removed '/etc/whiteprocess.conf'

whiteprocess removed.
utente@server-generico:~/whiteprocess-package$
```

5 Compilation

For compile whiteprocess from sources, *PyInstaller* must be installed.

For manual compilation, download the source code from **https://github.com/Gorresio/whiteprocess** and execute **./build** script.

As output, the script will generate a **./whiteprocess-package** folder containing the package ready for installation.

Linux systems and BSD systems generate a different package (different ELF).

6 Use Python Interpreter instead of ELF

For use whiteprocess using the python interpreter follow the instructions below:

1. Install dependencies *python2.7* and *python-psutil*
2. Download the sources.
3. Put ***.py** files in **/usr/share/whiteprocess/**
4. Execute script with the command **python2.7 *<python script>*** (ex. python2.7 /usr/share/whiteprocess/whiteprocess.py help)

For daemon mode, run the follow command:

```
nohup python2.7 /usr/share/whiteprocess/whiteprocess.py start >/dev/null 2>&1 &
```

7 Use of whiteprocess

whiteprocess must be run as **root**.

whiteprocess could be execute through **whiteprocess** or **whiteprocessd** command.

whiteprocessd is a script that launch **whiteprocess start** in daemon mode.

7.1 Configuration

Before start the service, configuration file must be configured.

Warning: Bad configuration of config file may cause a DoS on server.

For configure whiteprocess there are two options: **manual configuration** and **guided configuration**.

7.1.1 Guided Configuration

For execute the guided configuration run **whiteprocess autoconf** with root permission and follow instructions.

```
utente@server-generico:~$ sudo whiteprocess autoconf

        ** whiteprocess Guided Configuration **

Do you want reset config file? (yes/no): yes
/etc/whiteprocess.conf resetted.

Do you want insert in executables allowed (EXE_FILTER) current executables running? (yes/no): yes
Found 14 executables running:
/lib/systemd/systemd
/lib/systemd/systemd-journald
/lib/systemd/systemd-udevd
/lib/systemd/systemd-timesyncd
/usr/bin/dbus-daemon
/usr/sbin/rsyslogd
/usr/sbin/cron
/lib/systemd/systemd-logind
/sbin/agetty
/usr/sbin/sshd
/bin/bash
/usr/bin/sudo
/usr/bin/whiteprocess

Do you want insert executables in '/bin/' in whitelist? (yes/no): no

Do you want insert executables in '/usr/bin/' in whitelist? (yes/no): yes

Do you want enable Arguments Filter? (yes/no): no

Do you want enable After Exec Filter? (yes/no): yes
After Exec Filter enabled... AFTER_EXEC_KILL must be filled manually.

Time in second from one control to another? (Default: 5): 0.5
Time setted to 0.5 seconds.

Do you want enable Password for stopping service? (yes/no): no

Do you want enable Remote Control? (yes/no): yes

Insert TCP port to listening? (Default: 2357):
TCP port setted to 2357.

Saving configuration...
utente@server-generico:~$
```

Note: For now, some configurations must be done manually.

7.1.2 Manual Configuration

The first step is declare what controls should be made:

- **EXE_FILTER** Executable Filter: kill unlisted executables.
- **ARGS_FILTER** Arguments Filter: kill executables that have unlisted arguments.
- **AFTER_EXEC_FILTER** After Execution Filter: kill executables started after whiteprocess (listed).

The second step is fill the white list EXE_ALLOW with executables that have to run (services, system-logger, ecc.).

To find executables running type **whiteprocess check_exe** command with root permission.

```
utente@server-generico:~/whiteprocess$ sudo whiteprocess check_exe
Found 14 executables running:
/lib/systemd/systemd
/lib/systemd/systemd-journald
/lib/systemd/systemd-udevd
/lib/systemd/systemd-timesyncd
/usr/sbin/cron
/lib/systemd/systemd-logind
/usr/bin/dbus-daemon
/usr/sbin/rsyslogd
/sbin/agetty
/usr/sbin/sshd
/sbin/dhclient
/bin/bash
/usr/bin/sudo
/usr/bin/whiteprocess
utente@server-generico:~/whiteprocess$
```

The third step is complete the whitelist with executables and commands that could be used.

Most commands are performed in less than half a second, but there is a little chance that an any command could be killed before to finish correctly... So it is better include commands in whitelist, with priority commands that require more than two second to finish (personally, in several tests there were no problems in using commands how **ls**, **echo**, **cd**, etc. in ”blacklist”).

This is the simplest configuration.

7.1.3 Other Manual Configurations

In config file, the form of a **ARGS_FILTER**'s element is:

```
<executable>:<arg1>,<arg2>,<arg3> ,...
```

Executable and arguments are split by '**:**' character and arguments are separed by '**,**' character.

For run **/home/utente/script.py** on GNU/Linux without allow python for other things (except script.py):

```
/usr/bin/python2.7:/home/utente/script.py
```

if **After Execution Filter** is enable, you can fill **AFTER_EXEC_KILL** list with executable to kill if them starting after whiteprocess execution.

You can configure a password for stopping service. The password will be saved in configuration file in **sha256**.

Configuration file ignores **space** and **tab**.

7.2 Start and Stop whiteprocess

Once the configuration file has been completed, you can start the daemon with command **whiteprocessd start** how root.

Note: whiteprocess can only be run with one instance

For terminate the service is sufficient execute **whiteprocess stop** how root.

If whiteprocess is killed forcedly without appropriate command, it is necessary execute **whiteprocess reset_status** how root and execute again the service.

7.3 whiteprocess Log

whiteprocess logs now include the following information:

- When the service started
- When the service stopped
- What filters are running
- Various parameters (own PID, time check, remote check TCP port)
- When and what command with executables is been killed
- Connection received

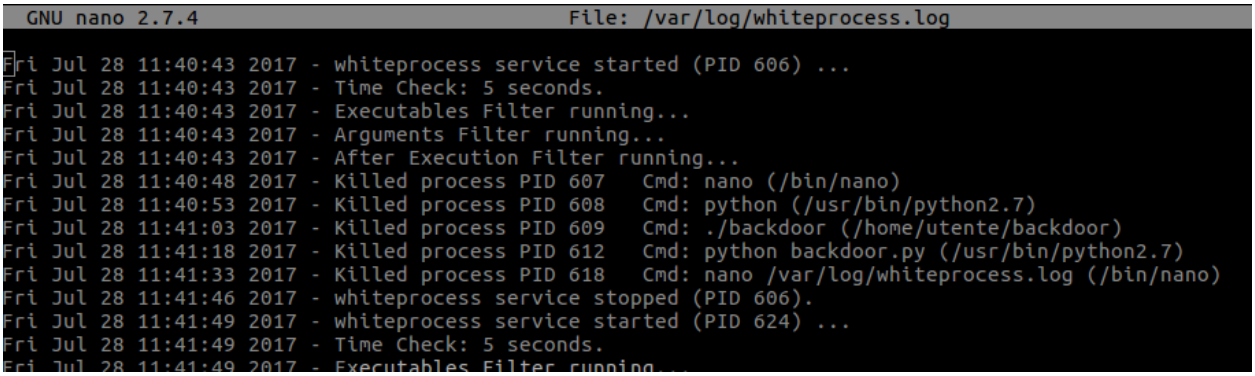


Figure 1: Log file example.

8 whiteprocess Network Protocol

To provide whiteprocess status monitoring on multiple servers, a network protocol has been developed to allow remote monitoring of whiteprocess.

On whiteprocess, you can enable the **remote check** option to allow remote monitoring.

whiteprocess_tools is the client for communicate remotely with whiteprocess, API for develop custom script is **source/whiteprocess_socket.py**

8.1 Details

This protocol (Layer 7 of ISO-OSI) works with TCP.

Its structure is as follows:

```
wp\x00<payload>\xff
```

In the payload, lists are encoded:

```
element1\x00element2\x00elementN
```

Communication consists of one request and one response.

Low ISO-OSI layers are interchangeable.

Between the request and the response there is a wait of 0.5 seconds fixed.

8.2 Implementation

The options for now implemented are as follows:

8.2.1 Check Alive

Check if whiteprocess is running.

```
Request:    wp\x00ALIVE?\xff
Response:    wp\x00Y\xff
```

8.2.2 Get Last Logs

Show last whiteprocess logs generated.

```
Request:    wp\x00LAST<rows>LOG?\xff
Response:    wp\x00<'logs rows>\xff
```

8.2.3 Executables Allowed

Show executables in whitelist.

```
Request:    wp\x00EXEALLOWED?\xff
Response:    wp\x00<executables list>\xff
```

9 Logic of whiteprocess

If an executable is present in EXE_ALLOW but arguments are filtered in ARGS_ALLOW, the filter ”work logically” for ARGS_ALLOW.

If an executable is present in EXE_ALLOW but not declare in ARGS_ALLOW, all arguments are allowed.
If an executable is not present in EXE_ALLOW but arguments are filtered in ARGS_ALLOW, the executable will be always killed. After Execution Filter is last filter that logically will filter allowed threads (Executable Filter and Arguments Filter passed).

A thread has logically one executables and optionally one or more arguments. Arguments is a set of all arguments of a executables, but since arguments are all analyzed together, arguments can be considered logically how a single element.

In ARGS_ALLOW have listed the first arguments... any arguments next unlisted will be accepted.

Ex.

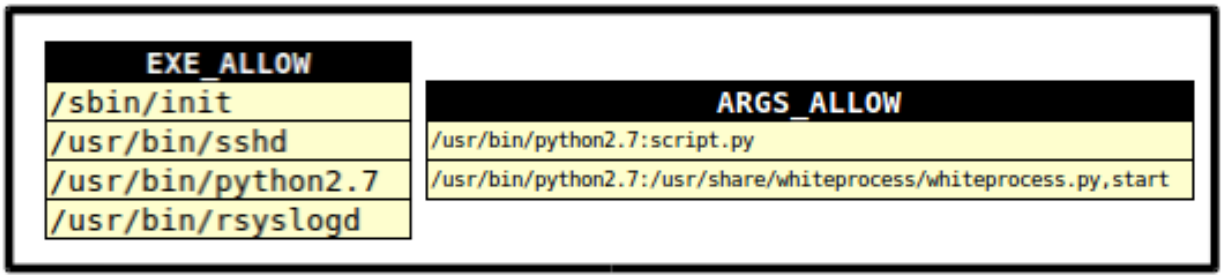
```
/usr/share/python2.7:scrypt.py,stop
python2.7 scrypt.py start                    Killed
python2.7 scrypt.py stop                     Allowed
python2.7 scrypt.py stop -f shut             Allowed
python2.7 scrypt.py stop <any arguments>     Allowed
```

Example

PID	Executable	Command
1	/sbin/init	/sbin/init
576	/usr/bin/sshd	/usr/bin/sshd -D
675	/usr/bin/python2.7	python
676	/usr/bin/python2.7	python script.py
416	/usr/bin/rsyslogd	/usr/bin/rsyslog -n
416	/bin/nano	/bin/nano

Figure 2: Threads (executables and arguments).

The various lists generate the logical whitelist.



EXE_ALLOW
/sbin/init
/usr/bin/sshd
/usr/bin/python2.7
/usr/bin/rsyslogd

ARGS_ALLOW
/usr/bin/python2.7:script.py
/usr/bin/python2.7:/usr/share/whiteprocess/whiteprocess.py,start

Whitelist	
Executables	Arguments
/sbin/init	*
/usr/bin/sshd	*
/usr/bin/python2.7	/usr/share/whiteprocess/whiteprocess.py start *
/usr/bin/python2.7	script.py *
/usr/bin/rsyslogd	*

PID	Executable	Command	Killed
1	/sbin/init	/sbin/init	NO
576	/usr/bin/sshd	/usr/bin/sshd -D	NO
675	/usr/bin/python2.7	python	ARGS_FILTER
676	/usr/bin/python2.7	python script.py	NO
416	/usr/bin/rsyslogd	/usr/bin/rsyslog -n	NO
416	/bin/nano	/bin/nano	EXE_FILTER

Figure 3: Threads status with whiteprocess running.

10 Testing

10.1 Security Test

whiteprocess is not a perfect ”thread filter” (kernel threads are not filtered and a potential malware can execute in a lesser time of time passing between control and the other).

A command executed with root permission can stop the service forcedly, but one or more banal script can quietly launch an alarm in case of anomalies (to be sure, the attacker should know all the processes running before attacking) or use the external monitor system.

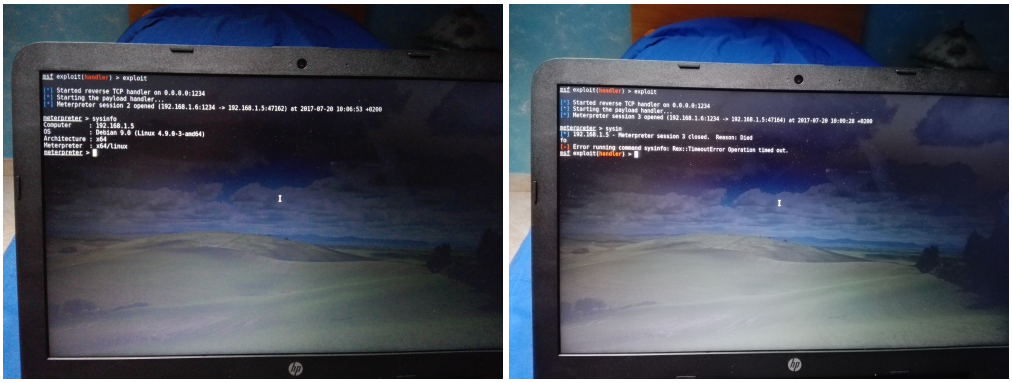
Despite this, this program working well and its purpose reaches it.

10.1.1 Executable Malwares

Most executable malwares do not work on a server running this filter.

The most interesting malwares are typically spyware and backdoor, but require a very high execution time.

In a normal condition, a backdoor how Meterpreter (Metasploit Framework) work without problems.



(a) Without whiteprocess running (b) With whiteprocess running

Figure 4: Comparison whiteprocess running with check time every 5 seconds and not

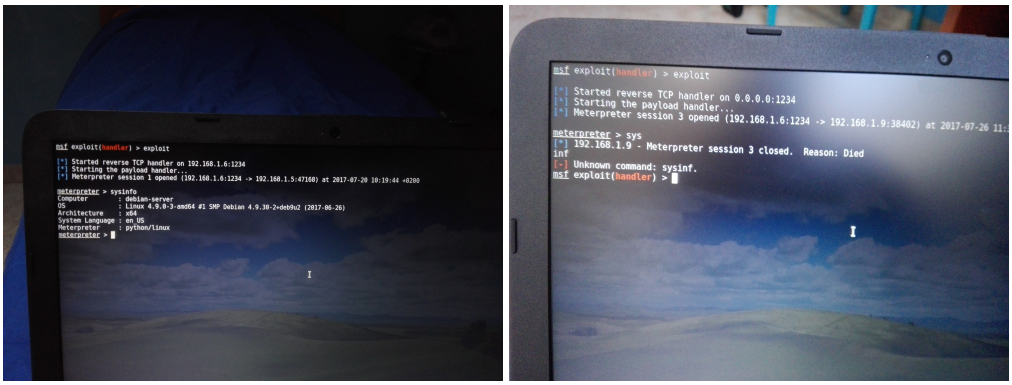
For evading, a malware must be performed in a very short time: insufficient time to steal files, but enough to steal small information such as content folders, threads information (interesting) and so on.

Less than a second between one control and the other would prevent also only the connection with a server C&C and the malware.

10.1.2 Script Malwares

If arguments filter is disabled and intepreter of a script malware is in the whitelist, the malware will run smoothly.

To solve this problem is enough control arguments passed.



(a) Without whiteprocess running (b) With whiteprocess running

Figure 5: Comparison whiteprocess running with check time every 5 seconds and not

10.1.3 Buffer Overflow Attacks

If a buffer overflow attack create a thread with AFTER_EXEC_FILTER running, the thread will be killed.

The buffer overflow flaws are complicated to manage, but with whiteprocess configured well, the probability of failure of an attack increases.

10.2 Performance Test (Still to Do)

10.3 Code Robustness Test (Still to Do)

11 Conclusions

11.1 Notes

Whiteprocess has been developed with the aim of increasing the chances of not having a computer attack, don't make a server invulnerable… because it is practically impossible.

whiteprocess is still an incomplete project, and I felt it correct to show its current flaws (I've always felt right to show the problems of your software).

As common sense implies that it is important not to use only a security software to ensure a minimum of IT security. whiteprocess is not an exception. Even when it is finished, whiteprocess should not be considered a perfect system.

11.2 Future Development

The following points could be developed in the future:

- Perform complete and exhaustive performance and robustness tests.
- Insert TLS in whiteprocess network communications.
- Divide agents into different executables to lighten primary memory.
- Prevention execution instead of kill after execution.
- Improve ”user experience”.
- Improve ”remote management”.
- Simplification and organization of source codes.

11.3 Source Code Note

The source code consists of multiple **.py** files.

Each file contains the source of a software's component.

For now the source code is a bit disorganized... fault the too many changes of idea (whiteprocess originally was supposed to be just a simple PoC) and the little time available.

Some pieces of code have been reorganized into classes and some parts have been simplified.

In future updates, the code will be completely overhauled (without changing the functioning of the software).

11.4 Projectual Choices

whiteprocess has been programmed in python for simplicity and timing.

In the future, agents (executable code that needs efficiency) could be programmed in C.

PyInstaller compilation eliminates all dependencies necessary for the functioning of whiteprocess alpha versions.

For reasons of safety, simplicity and stability, it was decided to store the state of the current execution of whiteprocess on secondary memory, in OS filesystem in location
/usr/share/whiteprocess/whiteprocess.status

12 Troubleshooting

12.1 whiteprocess doesn’t start

Execute **whiteprocess start** (no daemon mode) and/or analyze log file.

12.2 Some services don’t work when whiteprocess is running

Make sure that all the executables necessary for system operation are entered in the whitelist.

12.3 Login doesn’t work when whiteprocess is running

Make sure that all the executables necessary for system operation are entered in the whitelist.
When whiteprocess is stopped, try execute **whiteprocess check_exe** and compare with your whitelist.

12.4 line 1: syntax error: unterminated quoted string

Make sure the operating system is 64-bit.

This message may appear if you run whiteprocess compiled for x86_64 on a GNU/Linux x86 (32-bit).

12.5 Standard glibc absent

On operating systems like **Alpine Linux**, libraries to run whiteprocess are absent.
In these cases, you must install the appropriate libraries before running the program.

13 License

This software is distributed under **GPLv3** license.