SIGPwny

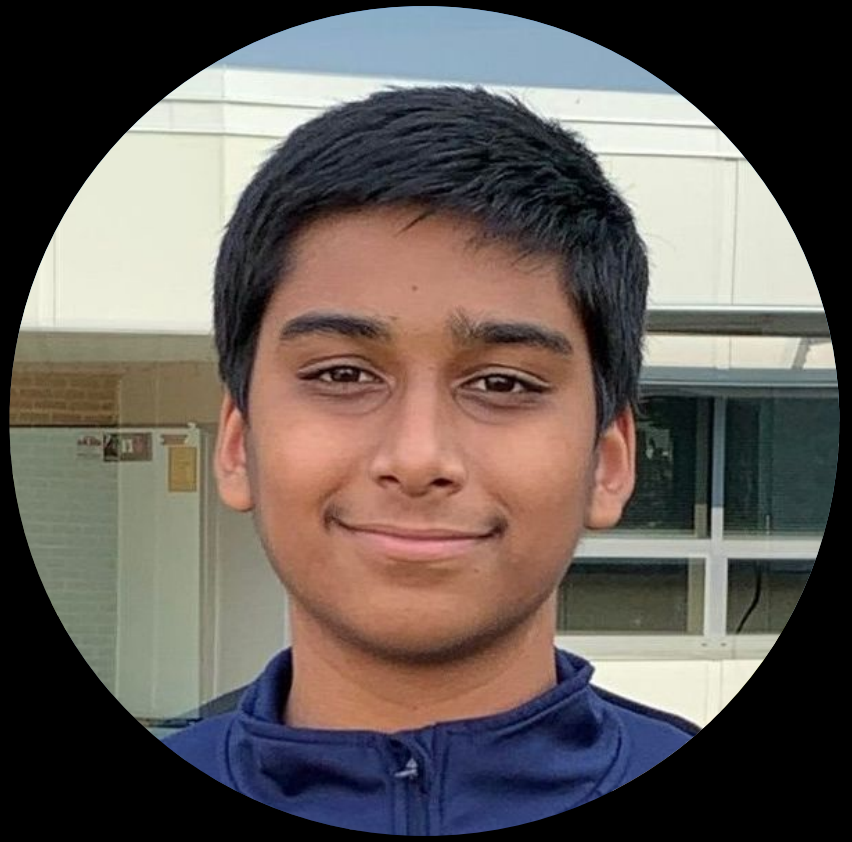
General SP2026 • 2026-03-08

Bash Jails

Krishnan Shankar, Krish Kalra

Krishnan Shankar

- SIGPwny Admin
- Computer Engineering ‘28



Krish Kalra

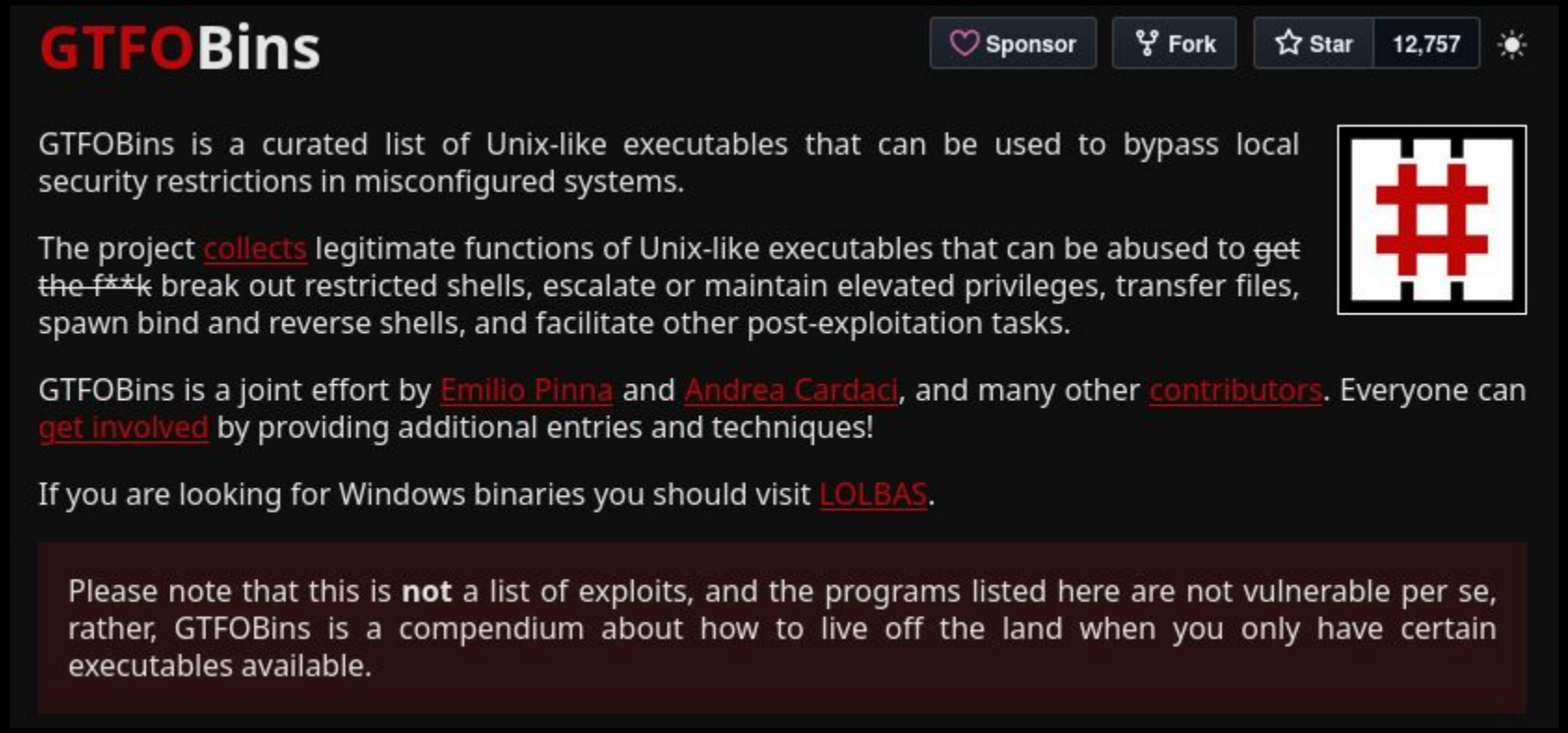
- Helper
- Computer Science

ctf.sigpwny.com

sigpwny{ju57_70uch_17}

TL;DR

- https://gtfobins.org/



What is Bash?

bash(1) - Linux man page

Name

bash - GNU Bourne-Again SHell

Synopsis

bash [options] [file]

Copyright

Bash is Copyright Â© 1989-2009 by the Free Software Foundation, Inc.

Description

Bash is an **sh**-compatible command language interpreter that executes commands read from the standard input or from a file. **Bash** also incorporates useful features from the *Korn* and *C* shells (**ksh** and **csh**).

Bash is intended to be a conformant implementation of the Shell and Utilities portion of the IEEE POSIX specification (IEEE Standard 1003.1). **Bash** can be configured to be POSIX-conformant by default.

Bash Review

File System Navigation

- pwd: print working directory
- ls: list directory contents
- ls -la: list with hidden files and details
- cd <dir>: change directory
- cd ..: go up one level
- cd ~: go to home directory
- cat <file>: print file contents
- find / -name <file>: search for files by name

Very Powerful

the horrors of “rm -rf /”

...and actual vulnerabilities are possible!

- case and point: Shellshock
  - bug in Bash that was present since 1989, announced in 2014
  - (very brief explanation): works by using environment variables maliciously
- smaller examples of attacks
  - command injection - if you don’t properly sanitize user input, an attacker can do bad things
  - including reading arbitrary files, getting root on the box, etc.

Redirection

File Descriptors

- In Linux (/Unix), everything is a file
- Including the input/output of a command

File Descriptors

- Every shell command gets three file descriptors:
  - 0: stdin
  - 1: stdout
  - 2: stderr
- Anything the command opens (using a syscall) gets an unused number
  - 3: file.txt
  - 4: flag.txt

File Descriptors

The **open**() system call opens the file specified by *path*. If the specified file does not exist, it may optionally (if **O_CREAT** is specified in *flags*) be created by **open**().

The return value of **open**() is a file descriptor, a small, nonnegative integer that is an index to an entry in the process's table of open file descriptors. The file descriptor is used in subsequent system calls (read(2), write(2), lseek(2), fcntl(2), etc.) to refer to the open file. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

By default, the new file descriptor is set to remain open across an execve(2) (i.e., the **FD_CLOEXEC** file descriptor flag described in fcntl(2) is initially disabled); the **O_CLOEXEC** flag, described below, can be used to change this default. The file offset is set to the beginning of the file (see lseek(2)).

A call to **open**() creates a new *open file description*, an entry in the system-wide table of open files. The open file description records the file offset and the file status flags (see below). A file descriptor is a reference to an open file description; this reference is unaffected if *path* is subsequently removed or modified to refer to a different file. For further details on open file descriptions, see NOTES.

Redirection

- Bash lets you “redirect” data between file descriptors
- This is very powerful!

Redirection: Summary

- `>`: write stdout to file
- `>>`: append stdout to file
- `<`: read stdin from file
- `2>`: redirect stderr
- `2>&1`: merge stderr into stdout
- `cmd1 | cmd2`: pipe stdout of one command into the next
- `ls | grep flag`: practical example — filter output

Redirection: Simple Examples

```
~ ❯ echo 'hi!' > file.txt

~ ❯ cat file.txt
hi!

~ ❯ echo 'hi!' >> file.txt

~ ❯ cat file.txt
hi!
hi!

~ ❯ echo 'overwritten' > file.txt

~ ❯ cat file.txt
overwritten

~ ❯
```

```
~ ❯ cat bee movie.txt | grep 'Buzzwell'
please welcome Dean Buzzwell.
Mr. Buzzwell, we just passed three cups and there's gallons more coming!

~ ❯ cut -f1 -d" " bee movie.txt | sort | uniq -c | sort -nr | head -n 10
    108 I
     63 You
     40 What
     37 I'm
     28 Oh,
     28 It's
     24 This
     20 That's
     18 All
     17 How

~ ❯
```

How to use this?

- Sometimes words are blacklisted in stdout, but not stderr
- `echo “hello” 1>&2`
- Redirects fd 1 (stdout) to fd 2 (stderr)

Getting Past Blacklists

Forward Slash

- `${PATH:0:1}bin${PATH:0:1}bash`
- `${PWD}bin${PWD}sh` (if you're currently in `/`)

Globbing

- “flag” is banned?
  - `cat /*.txt`

Globbing

- “flag” is banned?
  - `cat /*.txt`
- “*” is banned?
  - `cat /????.txt`

Globbing

- “flag” is banned?
  - `cat /*.txt`
- “*” is banned?
  - `cat /????.txt`
- “?” is banned?
  - `cat /[f]lag.txt`
  - `cat /[f-g]lag.txt`

More Advanced: Brace Expansion

3.5.1 Brace Expansion

Brace expansion is a mechanism to generate arbitrary strings sharing a common prefix and suffix, either of which can be empty. This mechanism is similar to *filename expansion* (see Filename Expansion), but the filenames generated need not exist. Patterns to be brace expanded are formed from an optional *preamble*, followed by either a series of comma-separated strings or a sequence expression between a pair of braces, followed by an optional *postscript*. The preamble is prefixed to each string contained within the braces, and the postscript is then appended to each resulting string, expanding left to right.

Brace expansions may be nested. The results of each expanded string are not sorted; brace expansion preserves left to right order. For example,

```
bash$ echo a{d,c,b}e
ade ace abe
```

Programming Languages

Programming Language Escapes

- `python -c 'import os; os.system("/bin/bash")'`
- `perl -e 'exec "/bin/bash";'`
- `ruby -e 'exec "/bin/bash"'`
- `php -r 'system("/bin/bash");'`
- `node -e 'require("child_process").spawn("/bin/bash", {stdio: [0, 1, 2]})'`
- etc.

Many ways to bypass character blacklists

- `php -a` spawns an interactive php shell
- `python` spawns an interactive Python repl
- and so many more…

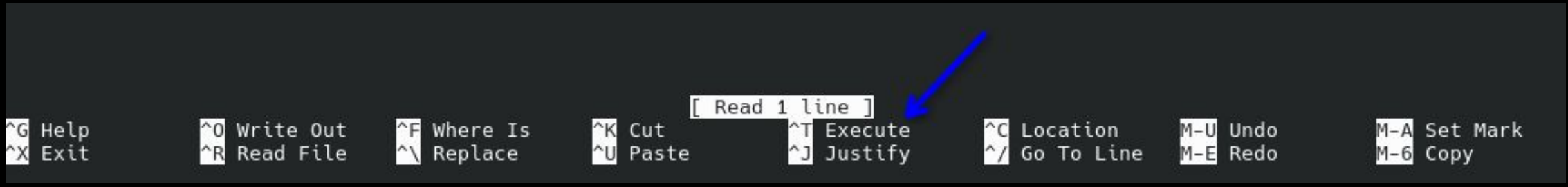
Text Editors

The Classic Example: Vim

- :!/bin/bash

Nano

- Can execute commands, grab output, and insert it into the file
- Harder to get interactive shells, but may be possible through custom spellcheck programs, etc.



Less

- !/bin/bash
- Note: Man pages use less!

```
! shell-command
       Invokes a shell to run the shell-command given.  A percent sign (%) in the command is replaced by the name of the
       current file.  A pound sign (#) is replaced by the name of the previously examined file.  "!!" repeats  the  last
       shell  command.   "!" with no shell command invokes an interactive shell.  If a ^P (CONTROL-P) is entered immedi-
       ately after the !, no "done" message is printed after the shell command is executed.  On Unix systems, the  shell
       is  taken  from  the  environment variable SHELL, or defaults to "sh".  On MS-DOS, Windows, and OS/2 systems, the
       shell is the normal command processor.

# shell-command
       Similar to the "!" command, except that the command is expanded in the same way as prompt strings.  For  example,
       the name of the current file would be given as "%f".
```

Other Utils (GNU Coreutils, etc.)

Awk

- `awk 'BEGIN {system("/bin/sh")}'`

```
system(cmd-line)      Execute  the  command  cmd-line, and return the exit status.  (This may
                      not be available on non-POSIX systems.)  See  https://www.gnu.org/soft-
                      ware/gawk/manual/html_node/I_002fO-Functions.html#I_002fO-Functions for
                      the full details on the exit status.
```

Tcl Shell (tclsh)

- tclsh
  ```
  % exec /bin/bash -i <@stdin >@stdout 2>@stderr
  ```

Zip

- `zip file.zip file -T --unzip-command="sh -c /bin/bash"`
- `file` can be any file in the system...

```
-T
--test
       Test the integrity of the new zip file. If the check fails, the old zip  file  is  un-
       changed and (with the -m option) no input files are removed.

-TT cmd
--unzip-command cmd
       Use command cmd instead of 'unzip -tqq' to test an archive when the -T option is used.
       On  Unix, to use a copy of unzip in the current directory instead of the standard sys-
       tem unzip, could use:

        zip archive file1 file2 -T -TT "./unzip -tqq"

       In cmd, {} is replaced by the name of the temporary archive, otherwise the name of the
       archive is appended to the end of the command.  The return code is checked for success
       (0 on Unix).
```

Tar

- There are many options here!
- `tar -cf file.tar file --checkpoint=1 --checkpoint-action=exec=/bin/bash`

```
Informative output
    --checkpoint[=N]
            Display progress messages every Nth record (default 10).

    --checkpoint-action=ACTION
            Run ACTION on each checkpoint.
```

SSH (!)

- You might be able to control bash before you even enter the jail
- ssh user@host "/bin/bash -i"
  - Spawns an interactive shell
- ssh user@host -t "/bin/bash --noprofile"
  - Don’t read some configuration files

```
--noprofile
       Do not read either the system-wide startup file /etc/profile  or  any
       of  the personal initialization files ~/.bash_profile, ~/.bash_login,
       or ~/.profile.  By default, bash reads these files when it is invoked
       as a login shell (see INVOCATION below).
```

```
-t      Force pseudo-terminal allocation.  This can be used to execute arbi-
        trary screen-based programs on a remote machine, which can  be  very
        useful,  e.g.  when implementing menu services.  Multiple -t options
        force tty allocation, even if ssh has no local tty.
```

Next Meetings

No general meetings until after spring break! Enjoy the rest of your week and enjoy your break!

ctf.sigpwny.com

`sigpwny{ju57_70uch_17}`

Meeting content can be found at sigpwny.com/meetings.

SIGPwny