

# Offensive Technologies 2021

## Group 2 - Final Report

### Executive Summary

This is the report of the two CCTFs ‘Resilient Server’ and ‘Secure Server’ by Group 2.

Considering the attack part, the main takeaway is the mindset: having appropriate tools is of course important, but even more important is how you approach the problem of breaking the security measures put in place, considering whether the defending party is aware that an attack is coming or not. In order to do so the attacking party has to make hypothesis starting from little clues being always ready to change plans on the fly.

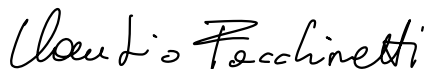
Considering the defense part, the main takeaways are to reduce the attack surface as much as possible and then properly and securely handle the remaining endpoints. In this case, it meant to filter out all the unexpected and malformed traffic, limiting and checking the permitted one. Another important aspect is to have a good monitoring system in order to detect and react immediately to potential threats.

As an overall consideration, thinking both as the defender as well as the attacker, it is a great way to assess and improve the security of a system, as it allows to really spot potential security issues and understand their severity. This double point of view is key in order to develop a secure system and is extremely valuable in high-threat scenarios.

Work submitted in partial fulfillment for the course of Offensive Technologies - Uni. Trento - a.a. 2021/2022

*This report is original work, has been done by the undersigned students and has not been copied or otherwise derived from the work of others not explicitly cited and quoted. The undersigned students are aware that plagiarism is an academic offence whose consequences include failure of the exam.*

Claudio Facchinetti 223814



Batbayar Narantsogt 222986



Gabriele Ricciardi 230371



Nicolò Vinci 220229

# 1 CCTF RESILIENT SERVER

## 1.1 Scenario

In the Resilient Server CCTF, the network consist of six machines: a server and a gateway, which are under control of the blue team, three clients, two of which are under control of the red team, and a router which is neutral, i.e. controlled by the CCTF organisers, that would drop spoofed traffic with its own IP. The links between the router and the clients are 100 Mbps, and so is the link between the server and the gateway. Noticeably, the link between the gateway and the router is 1 Gbps. See the network topology representation below.

One of the three client machines acts as a legitimate one, and cannot perform malicious actions. Each second it will make a HTTP request for a random page of the 10 present on the server: if the request is successfully served in under 500 ms, the blue team scores one point; if not, the red team does. The team with the highest score at the end of the CCTF is the winner.

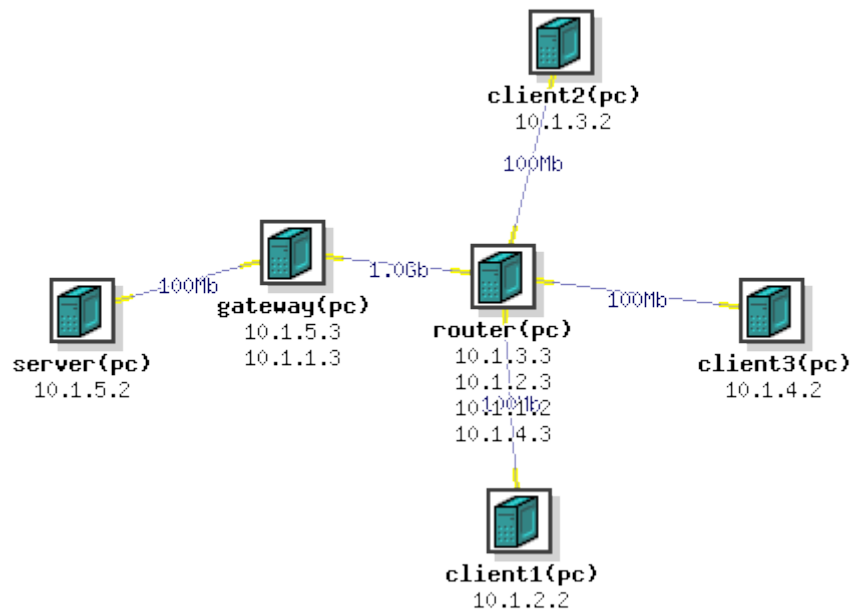


Figure 1: The CCTF network topology.

## 1.2 Attack Preparation

For this competition the attackers had control over the client machines, but only two of them were available to be used as malicious machines, as the third one had to be used to perform legitimate requests. The goal of the attackers, or red team, was to make the opponent web server unable to serve legitimate requests in a small amount of time which was set to half a second (500 ms). In order to accomplish this result the red team faced this from multiple points of view:

1. exploiting possible vulnerabilities in the network configuration;
2. exploiting possible vulnerabilities in the application server used;
3. flooding the server with a high number of TCP/UDP packets;
4. flooding the server with a high number of HTTP requests.

To achieve these goals the red team searched online for known vulnerabilities affecting the software which was provided for the CCTF. However, this gave no results since the software versions loaded on the machines were not affected by any publicly known vulnerability exploitable in this CCTF scenario. Since this was the case the red team decided to focus on the flooding part, searching online tools which allowed them to perform what specified above.

As a preparation step, the red team worked jointly with the blue team in order to test the effectiveness of both the attacks they planned as well as the defences put into place by the blue team.

Tool name	Reconnaissance	Flooding strategy	Probabilistic strategy
HPing	X	X	X
GoldenEye	X	X	
Slowloris		X	

Table 1: In the table it is shown which tool has been used in every strategy of the CCTF by the red team.

### 1.2.1 Reconnaissance

Before starting any attack, the red team needed to know something more about the opponents setup, such as which kind of traffic was allowed to go through their firewall.

To gather this kind of information the HPing[4] tool was used to send TCP and UDP packets to both the server and the gateway machines, trying to find vulnerable points for possible attacks.

HPing is a simple tool which allows to generate and send to a specified host any TCP/UDP packet providing also the possibility to spoof an IP address; this tool was selected as it really flexible, allowing to generate a packet customizing whatever is needed, such as the TCP flags.

To gather information about possible open ports the red team planned to use Nmap[12] to scan the network for open ports and known vulnerabilities available on the hosts.

Nmap[12] is a network scanning software which allows to discover useful data about a network or even a single host. This tool was selected as it is probably the most powerful available, allowing also to obtain a fingerprint of the host and to automatically discover vulnerabilities based on how the target responds to multiple probes.

### 1.2.2 Flooding Strategy

The first attack strategy is the most simple one: goal of the red team was to flood the server with as many packets as possible in order to make it unable to serve legitimate requests.

In order to achieve this the red team planned to use multiple tools at once on both malicious clients; in particular the red team planned to launch:

1. one instance of GoldenEye[3] to generate valid HTTP traffic;
2. two instances of SlowLoris[19] to perform a low-bandwidth DoS attack;
3. multiple instances of HPing[4] generating IP packets;
4. multiple instanced of HPing[4] generating TCP packets with SYN/SYN-ACK flag.

HPing[4] is a tool which allows the attacker to craft network packets and send them to the specified target; this tool was selected among others because it is flexible and powerful at the same time, as it allows the attacker to generate various types of packets with any combination of flags and to spoof any IP.

GoldenEye[3] is an HTTP flooding tool which allows the attackers to send an arbitrary number of concurrent HTTP requests as a new thread gets spawned for every single request; this powerful tool

is however also dangerous as it may be hard to stop as the number of thread spawned gets higher and higher. In order to this when needed the red team relied on the fact that it is possible to execute a command via ssh without obtaining a shell via:

```
$ ssh <host> -c "sudo pkill -9 goldeneye"
```

The basic idea behind the strategy was to flood the server, giving it so many concurrent requests making it unable to respond in the time limit frame of 500 ms; however it is also true that if the attacker does not spoof the IP address and does not configure IPtables properly then it will probably DoS itself because of the responses coming from the server.

### 1.2.3 Probabilistic Strategy

This second attack strategy is named probabilistic as it requires that the attacker is able to guess something in order for the attack to work: in particular the attacker needs to guess the correct source port of the client.

As a prescription of the TCP protocol if one of the parties receive a RST packet from the other party, then the communication is immediately dropped: in order for the attacker to impersonate the legitimate client, it must send a TCP packet spoofing the IP of the client and the correct source port number, hence the need of guessing.

In order to do so the only tool needed is HPing[4] because, as explained before, it allows to set about everything about the TCP packet to generate and send. As the red team was willing to generate some noise in the traffic it has been decided to run multiple instances of HPing[4] on the malicious clients: some of them would have generated the fake RST packets, maximizing the possibility of guessing the correct port number, while other would only generate random TCP/UDP packets to create noise.

The best thing about this strategy is that the attacker does not even to DOS itself as the RST packets do not get any response from the other party of the communication and also because the IP is spoofed.

## 1.3 Defense Preparation

In the Resilient Server CCTF, the blue team had control over the server and gateway part of the network, and their main goal was to ensure the continuity of the server operations. This meant ensuring that the legitimate client's requests would be correctly served in a short time frame (500 ms) and for as much time of the CCTF as possible. The main threat to the continuity of operations were, as the name implies, Denial of Service attacks, including flooding attacks, either fast or slow, and possibly vulnerabilities exploitation.

In order to prepare for this CCTF, the blue team carefully setup the server and the gateway, configured them properly and securely in order to resist as best as possible against DOS attacks, and setup traffic monitors and filters.

### 1.3.1 Server Setup and Modules

In order to avoid any known vulnerability exploitation from older versions (2.4.29 on Deter as of December 2021), we updated the Apache server to the latest version by installing it from source on Deter (2.4.51 as of December 2021[1]). Indeed, Deter has its own repositories so we were not sure whether all the security updates had been applied, hence the blue team decided for the aforementioned solution. The source was compiled before the actual CCTF, so that we would only need to run the installation process in the minutes before the competition.

Once the server was setup, we created the 10 html files to be requested by the clients and made them available through the server.

We then installed from source `mod_qos` (module Quality of Service for Apache)[7] and proceed to configure the Apache webserver:

- `Timeout` was set to 1, in order to limit malicious slow connections.
- `KeepAlive` was kept On, as possible exploitation of it were kept in check by `mod_qos`.
- `MaxClients` was set to 400; it is a `mod_qos` setting to limit the maximum number of simultaneous active TCP connections.
- `QS_SrvMaxConnClose` was set to 200; another `mod_qos` setting which disables the keep-alive when 50% (of 400) of the TCP connections are occupied.
- `QS_SrvMinDataRate` was set to 150 1200; another `mod_qos` setting which determines the minimum request/response speed a connection must have.

All of this setup was scripted into a bash script and run in the minutes before the start of the CCTF.

### 1.3.2 Server Hardening

The blue team decided to modify some TCP kernel parameters[22] to better handle flooding of TCP connections and timeouts. Considering the following scenario where the attacker controls client 1 and client 2 is the legitimate client:

1. Client 1 sends a legitimate TCP SYN packet spoofing its IP address with that of client 2.
2. Server responds back to client 2 with a TCP SYN ACK packet and the current TCP connection is in `SYN_RECV` state.
3. Client 2 replies with TCP RST packet to the server and the TCP connection is closed.

Now, client 1 can send a lot of legitimate TCP SYN packets. The result will be a lot of TCP SYN ACK packets from the server to client 2 and a lot of TCP RST packets from client 2 to the server. So, the attacker is able to flood the server with TCP RST packets and the legitimate client with TCP SYN ACK packets. Moreover, the server tries to retransmit the TCP SYN ACK packet multiple times when a TCP connection is in `SYN_RECV` state. The blue team decided to mitigate this attack situation in the following way:

- On the gateway: set up IPtables rules described in section 1.3.3 to reduce TCP RST packets towards the server.
- On the server: set TCP SYN ACK retransmission to 0 in order not to have the legitimate client flooded with TCP SYN ACK packets. Moreover, on the server any TCP connection in `SYN_RECV` state will not be considered anymore after a TCP retransmission timeout of 200 ms.

### 1.3.3 Gateway Setup and IPtables

The IPtables rules[5] are deployed on the gateway in order to set up a firewall in front of the server. A white-list approach was followed to design the rules. Hence, the default policy is set to DROP for the three IPtables chain INPUT, OUTPUT and FORWARD. So, the blue team has to handle only incoming TCP connection from the three clients to the server with destination port set to 80, while any other packet is dropped by default. For this purpose, a new IPtables chain called TCP-LIMIT was created to handle incoming TCP SYN packets. The new chain is matched only when a TCP packet is from one of the three clients to the server with destination port 80 and has only the SYN flag set. Then, the TCP-LIMIT chain checks the presence of TCP options[23] in order to assure that the packet is legitimate. In this way, any malformed TCP SYN DOS is detected and blocked. After that, it limits the TCP SYN packet to 5 per second considering the source IP address, using the `hashlimit`

IPtables module[6]. After a client sent 5 TCP SYN packets per second, the remaining ones are not accepted and they are dropped. There is also a rule that logs dropped TCP SYN packet at most 3 times per minute. After that, there are two rules that allow any ESTABLISHED connection from any client to the server with destination port 80, and viceversa with source port 80. The IPtables module conntrack[6] considers a connection ESTABLISHED when the gateway catches a client legitimate TCP SYN and the corresponding server TCP SYN ACK.

Then, there are rules which allow:

- SSH connections to the gateway.
- Gateway loopback traffic.
- Traffic from the gateway to the server and viceversa for monitoring purpose.
- Traffic from the router to the server and viceversa for CCTF purpose.

Finally, a rule was developed exploiting the limit IPtables module[6] to limit the TCP RST packets from the three client to the server at most 2 packets per second for the reason explained in section 1.3.2.

#### 1.3.4 Traffic Monitors

The blue team developed a monitor on the server to inspect incoming HTTP requests and collect statistics about packet received. The monitor is based on a popular Python library called Scapy which allows to send, create, modify or intercept any network packet[18]. When the Python program starts, it is able to sniff any packet flowing through a network interface. A filter rule was designed in order to intercept only TCP, ICMP or UDP traffic with destination IP corresponding to server IP. The script keeps different counters for TCP, TCP SYN, UDP and ICMP packets and bytes received. Then, it computes the TOTAL packets and bytes. It also saves the number of packets and bytes sent to the server by client IP. When a packet flows in the interface, the monitor will catch it and checks the source IP. If it corresponds to the router IP, the packet is not considered. Otherwise, the following actions are performed:

- if the packet has an HTTP layer, the HTTP request is printed.
- the TCP, TCP SYN, UDP, ICMP and TOTAL counters are updated.
- the client IP is stored and its counters are updated.
- after a period of time, all the metrics are printed and stored in a log file. Also, client IPs and their counters are stored in the same log file.

A bash script was also written in order to launch the Python monitor script targeting the correct network interface. Then, two bash scripts were developed on the gateway in order to check if the server is getting slower. In the first one, the gateway periodically pings the server and if the response time is greater than 1 ms, an alert is printed. In the second one, the gateway periodically performs an HTTP GET request to the server and if the response time is greater than 1 ms, an alert is printed. The threshold is set to 1 ms because, if the server is not under attack, it should be able to response within 1 ms to the gateway. This was observed during testing. A last monitor was developed on the gateway in order to automatically checks how many packets were dropped by the IPtables rules using the command:

```
$ sudo iptables -L -v
```

## 1.4 Execution

At the end of the reconnaissance the only traffic which was able to reach the server machine was the TCP and the UDP one, therefore the red team started with the flooding strategy.

After a few minutes in which the flooding strategy seemed to be working the opponents probably blocked the UDP packets so that the attack was no longer working. After realising this, the red team started to panic and switched to the probabilistic strategy in order to have better chances of achieving their goals. The probabilistic strategy turned out to be the most effective one, leading to 30% of the connections from the legitimate client to exceed the half second timeout making the red team gain points, leading to the red team partially achieving their goals.

During the CCTF, the blue team was monitoring the incoming HTTP requests and the TCP sockets on the server. On the gateway was checked the server status and how many packets were matched by IPtables rules.

Meanwhile, the blue team tried to understand which type of attacks the red team were running and how to best react adding new IPtables rules on the gateway. It was possible to build a CCTF timeline from 16.30 to 18.30 with attacks guesses and live mitigation:

1. From 16.30 to 17.30: SlowLoris attack. However, the blue team was able to defend successfully against it without live adjustments.
2. From 16.30 to 17.00: fast legitimate HTTP requests. They were mitigated by IPtables bottleneck.
3. From 17.00 to 17.30: malformed TCP SYN flood attack, also spoofed with legitimate client IP. The attack was blocked by checking the TCP options with IPtables rules.
4. From 17.20 to 17.50: legitimate traffic flooding spoofed with legitimate client IP. This was the heaviest attack. However, the blue team was always able to serve the router, scoring CCTF points, even if the legitimate client for a while could not reach the server. Perhaps the red team was able to flood the last link between the router and the legitimate client. The blue team decided in live to drop incoming TCP RST packets from the legitimate client in order to avoid the saturation of the link between gateway and server.
5. from 17.50 to 18.30: different DoS attacks with source port set to 20. Hence, the blue team could distinguish attacks dropping any incoming traffic with source port 20.

## 2 CCTF SECURE SERVER

### 2.1 Scenario

In the Secure Server CCTF, the network topology is the same as the one in section 1.1. This time the three clients are all under control of the red team and usable, while the server and the gateway are still under control of the blue team. The server is a LAMP server (which stands for Linux-Apache-MySQL-PHP) with PHP scripts and a MySQL database. The router is controlled by the CCTF organisers and this time it would not drop spoofed traffic with its own IP. The links between the machines are the same as in section 1.1.

The blue team plays the part of a bank webserver with a public web application which allows users to register, deposit or withdraw money, as well as check balance and transaction history. The red team, instead, plays the part of cyber robbers, trying to attack the bank to gain money.

The CCTF organisers will perform some periodic checks by validating the integrity of the database, by looking at the logs and by performing legit bank operations. Any detected problem leads to a fine for the bank, which loses money. The blue team objective is to ensure continuity of correct operations with the least money loss as possible due to fines or attacks. Vice versa, the red team objective is to disrupt as much as possible the bank operations and possibly steal some money exploiting bugs.

### 2.2 Attack Preparation

For this competition three client machines were under the control of the red team, however they only used two of them as it was forbidden to flood the server. The red team's goal is to cause severe disruption to the bank, which could include breaking into existing user's databases, manipulating the server, and crashing the server.

Before the actual CCTF, the red team had to go through some milestones, and they did it based on their experience and the Internet; in particular the red team had to:

- develop attacks to lead the server in an inconsistent state without relying on SQL injection;
- develop attacks using the SQL injections technique;
- crash the server with requests that may take too long to satisfy or that are malformed.

In order to achieve these goals, the red team went through the given code to look for possible vulnerabilities to exploit and also shared this information with the blue team. The red team also conducted experiments on the blue team servers using the same attack tools that were ready for the actual CCTF.

Before the CCTF took place, a custom script was written, namely PWNIT, which allows us to craft requests with different purposes, such as performing operations with malformed/invalid payload to analyze the behaviour of the opponent server using different character encoding as needed.

Tool name	Network vulnerabilities	Application exploits
NMAP	X	
SlowLoris	X	
CURL(SQL injection, Reflected XSS)		X
PWNIT		X

Table 2: In the table it is shown which tool has been used in every strategy adopted for the CCTF.



### 2.2.1 Network Vulnerabilities

During this phase, the goal of the red team was to probe the server and check how it responded to it, in order to identify possible vulnerabilities to exploit without any HTTP request. In particular, with this strategy the red team is interested in:

1. network misconfigurations, such as open ports;
2. scanning the network with different probes to find out known vulnerabilities.

The red team planned to use the Slowloris[19] attack tool to create some noise in the traffic, in order to make it harder for the opponents to realize the real goal of the red team attacks. Basically, Slowloris is a denial-of-service attack that allows an attacker to overwhelm a targeted server by opening and maintaining a large number of concurrent HTTP connections between the attacker and the target server. In this competition, the red team used slightly modified version of SlowLoris with 1000 sockets and with every connection delayed by 1 second to play by the rules.

The idea underneath this attack strategy was to create some noise; the red team was trying to identify vulnerabilities in the network configuration using the Nmap[12] software while the defending team was busy noticing what is going on because of the noise caused by the Slowloris taking place. Upon discovering such vulnerabilities the plan of the red team was trying to look for/develop exploits to achieve their goals.

### 2.2.2 Application Exploits

The red team planned to identify some vulnerabilities in the opponents' application and to develop some exploits for it. In particular, the red team planned to inject different malicious payloads in order to make the web server behave in a malicious manner.

Being more specific, the red team was planning to probe the web server with:

- malformed / wrong parameters such as negative amount of money;
- XSS attempts;
- SQL injection attempts;
- path traversal attempts;
- JavaScript code injection such as `| | true`.

In order to find out possible SQL injection vulnerabilities the red team planned to use the SQLMap[21] tool, which allows to automatically discover details about the database and the tables as well as possible SQL injection vulnerabilities present in the target server. A note on this tool: in order not to create random data in the database the red team planned to avoid using this on the `amount` parameter manually setting it to 1.

The basic idea of this strategy was to find out possible attack points which were not found in the Reconnaissance phase and develop exploits in order to gain advantage of vulnerabilities in the web server. Also, during this phase the red team planned to keep the Slowloris attack going in order to further hide malicious requests, just as described in the previous strategy.

## 2.3 Defense Preparation

In the Secure Server CCTF, the blue team had control over the server and gateway part of the network, and their main goal was to ensure the correctness and continuity of the server operations. This meant ensuring that the LAMP server was setup and secure against vulnerabilities exploitation and misuse.

In order to prepare for this CCTF, the blue team carefully setup the server and the gateway, as well as the PHP web application and its relative database, configured and patched them properly and securely in order to resist as best as possible against vulnerabilities exploitation and slow DoS attacks, and setup traffic monitors and filters.

### 2.3.1 Server Setup and Modules

The main idea for the server setup was to install the standard LAMP server as per Deter repository, set it up, configure it securely and add security modules. In the setup, the SQL root password was changed to a more secure one (20 random characters). In the Apache options the `Timeout` was reduced from 300 to 10, and the options `ServerSignature Off` and `Options -Indexes` were set, in order not to disclose information about the server.

The modules installed were `mod_qos`[7], setup as in the previous CCTF (see section 1.3.1), `modevasive`[8], which is used to further defend against DoS attacks, and `modsecurity`[9], which is actually something more than a module for Apache. Indeed, this last module is basically a web application firewall. It was set to actively block any detected attack with the option `SecRuleEngine On`. The attacks would also be logged and read by the blue team. All this installation and setup was scripted in a bash script to be run in the minutes before the CCTF.

### 2.3.2 Vulnerability Patching

After setting up a secure server, the blue team started to patch the PHP file and the database in order to be secure against injections or malformed requests that can lead the database into an inconsistent state. First of all, the database and any table name was changed, because the adversaries already knew the standard names. Then, the timestamp column was added to the `transfers_secure_table_random` in order to save it in the log. Moreover, a foreign key was added in the `transfers_secure_table_random` to point to the `users_secure_table`, in order to prevent an unregistered user to perform any transaction without registering before. The MySQL root password was updated to a more secure password and a new local user called `offtech` was created with limited permissions[2]. The local user was used in the PHP file and he could only read/insert users or read/insert transactions. In this way, if an attacker was able to manipulate the PHP script, he could delete nothing. An entity relationship diagram was reported in figure 2.

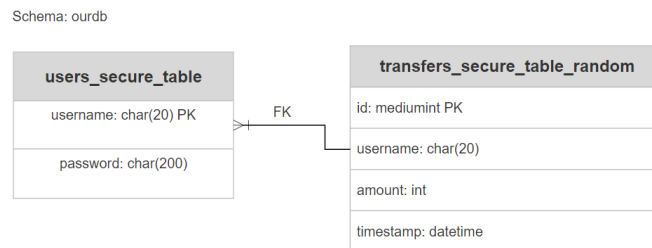


Figure 2: ER diagram

The password field has length 200 characters, because the blue team decided to store passwords hashing them with sha512 algorithm. So, even if the attackers were able to leak any hash, they need to perform some dictionary attacks to retrieve the password from the hash.

The next step was to fix the PHP file. Firstly, the blue team fixed the log by first adding the timestamp for each transaction, retrieving it from the corresponding column in the `transfers_secure_table_random`. Then, there were four query parameters to check and sanitize in order to avoid injection attacks. One of the four query parameters was an integer and it was checked with PHP function `isset()`[16] if it was set and if it contained only numbers from 0 to 9 with a custom PHP regex[17]. If both checks were passed, the parameter was sanitized with the PHP function `intval()`[15] to convert a string to integer. Otherwise, the parameter was stored with value 0. The last three were strings and each of them was checked with PHP function `isset()` if it was set and the parameter was sanitized using the PHP function `htmlspecialchars()`[14] with the option `ENT_QUOTES` that converts both double and single quotes. Otherwise, the parameter was stored as an empty string. Then, for each saved parameter different checks were done:

- **user parameter:** if the string is empty or the length is greater than 20 characters, an error was returned. This, because the user parameter is necessary in any operation and the username field in the database is stored as `char(20)`.
- **pass parameter:** if the string is empty or the length is greater than 32 characters, an error was returned as the pass parameter is necessary in any operation and its input size would be limited.
- **drop parameter:** if the string is empty or the length is greater than 9 characters, an error was returned. This, because the drop parameter is necessary to understand what operation was performed and the longest allowed choice is *withdraw* that has length 8.

Then, considering the drop parameter a switch case was built for the possible actions:

- **register:** register a new user. If the user was already registered, an error was returned to avoid duplicates. Else, a new user was created in the database hashing the password with `sha512`[13].
- **balance:** return the balance of a given user. The user was authenticated checking the input credentials. If the authentication was passed, the balance was returned to the user computing it with a single SQL query with the SQL function `SUM()`[10]. Else, an error was returned.
- **deposit:** deposit money. First, the user credentials were authenticated checking the input credentials. Then, the query parameter amount must be an integer, different from 0 and included in the PHP 32 bit int size (greater than 0 and less than 2147483647). If any of these checks failed an error was returned. Else, a transaction was created in the database with the timestamp.
- **withdraw:** withdraw money. The user must authenticate itself and the query parameter amount was checked as in the deposit case. Then, the user balance was retrieved from database and it must be different from 0 and greater than the amount parameter, otherwise an error was returned. So, a transaction was created in the database with its timestamp and a negative amount.

Any query to the database was performed using prepared statements in order to avoid SQL injections. Therefore, any time an error was returned, the database connection and the log file stream were closed to avoid multiple open connections.

### 2.3.3 Gateway Setup and IPtables

A bash script was written to install Scapy on the gateway for monitoring purpose as described in the section 2.3.4. Then, a firewall was set up on the gateway using IPtables rules. The blue team kept the rules used in the CCTF Resilient Server described in the section 1.3.3. So, a white-list approach was followed also for this CCTF, setting the default policy to DROP for the three IPtables chain INPUT, OUTPUT and FORWARD. Hence, the blue team needed to handle only incoming legitimate HTTP requests. The TCP-LIMIT chain was updated to:

- control the presence of TCP options in order to drop malformed TCP SYN packet.

- limit TCP SYN packet to 20 per second, thanks to hashlimit IPtables module.
- log dropped TCP SYN packet at most 3 times per minute.

The other rules remained the same and only the one limiting the incoming TCP RST packets was removed, hence dropping any ICMP or UDP traffic and also malformed TCP SYN packets. Moreover, snort[20] was tested on the gateway in order to deploy an intrusion detection/prevention system, but during the CCTF it was deemed not necessary, as it only introduced computation overhead for the gateway. Indeed, the logs produced by the monitors were deemed enough and malformed packets were already dropped by the IPtables rules.

### 2.3.4 Traffic and Database Consistency Monitors

The blue team developed a monitor to inspect incoming HTTP requests both on the server and gateway. The monitor is based on the Python library called Scapy like in CCTF Resilient Server. The Python program simply intercepts any packet that flows in the network interface. A filter rule was designed to intercept only TCP packets intended for the server and with destination port 80. If the packet has an HTTP layer, it prints the HTTP request and stores it in an external txt file. Moreover, a bash script was developed in order to launch the monitor program passing the desired network interface. So, the blue team was able to monitor incoming HTTP requests on the gateway and on the server in order to understand what attacks were running by adversaries.

Then, another Python script was written to monitor the database consistency. It uses the library mysql.connector[11] to perform queries. Periodically, the monitor performs two checks, i.e. no user can have a negative balance or can perform a transaction without being registered. If the two checks are passed, the database is in a consistency state. Otherwise, the attackers were able to tamper the database. In any case, the monitor prints the results and logs them in an external txt file. A bash script was developed in order to launch the monitor passing the check time interval. So, the blue team was able to react immediately to maintain the database in a consistency state if the attackers managed to tamper it.

## 2.4 Execution

At the beginning of the CCTF the red team started to create noise as described before in order to confuse the opponents on what they were really trying to investigate and achieve. As a result, this was enough to take down the opponent server leaving the red team with no possibility to try exploiting any vulnerability.

Apart from this the red team noticed something strange: they managed to register some users which were supposed to be already registered with another password: it is still unclear if this was because of a timing issue or because of a vulnerability patched on the fly.

After a while the opponent blue team realized that something was wrong with their configuration and probably decided to reboot the server application, making our attack fade away. After restarting it, it took about 10 minutes to make their server go down again.

The opponent blue team then realized that they were under SlowLoris attack and finally deployed a fix. At this point the red team switched to the second attack strategy but with little to no results. The only positive result was given by the JS injection, as the web server responded with a blank page every time a JS injection was present in the parameters, so the red team made the hypothesis that the web server was Node-based.

The red team also started thinking that the opponents were using a variant of the LEXN stack (Linux, Express, X-whatever, Node.js): with this in mind the red team started developing some NoSQL injections. Since the server has been down for a long time the NoSQL injections have actually been

prepared but no result was achieved, probably because they were not perfect and there has been no time to make them work.

On the other side, the blue team was monitoring the incoming HTTP requests and the TCP sockets on the server, as well as monitoring the database consistency. On the gateway was checked the incoming HTTP requests and how many packets were matched by IPtables rules.

Meanwhile, the blue team tried to understand which type of attacks the red team were running and how to best react adding new IPtables rules on the gateway, and possibly patching previously undetected vulnerabilities. Also in this CCTF it was possible to build a timeline from 15.30 to 17.30 with attacks guesses and live mitigation, although of the latter there were none as no attack was successful from the opposing red team:

1. From 15.30 to 17.30: SlowLoris attack. However, the blue team was able to defend successfully against it without live adjustments.
2. Some minutes after 15.30: Nmap scan.
3. 16.00: SqlMap scan and requests with malformed amount.
4. From 16.20 to 17.30: registering of new user at the rate of 2 per second in order to exhaust the server resources and render the logs unreadable efficiently. However this did not work and the operations could continue regularly.
5. From 16.30 to 16.40: many deposit of 1€ were done in order to further aid the disruption of the server memory and logs. Also this attack did not work.

As a result the red team managed to make the opponents miss a lot of transactions while the blue team had no missed/disputed transactions: the combination of these made the group win the CCTF.

### 3 TEAM ROLES

The team was composed of four members, divided in two on the red team and two on the blue team. Their names, roles and contribution for the two CCTF are listed below:

- Claudio Facchinetti: Member of the red team, developer of the PWNIT tool and creator of the probabilistic strategy during the Resilient Server CCTF and looked for tools for both the CCTFs. Supporting the blue team with vulnerability identification, possible patching ideas and testing of proposed patches in both the CCTFs.
- Batbayar Narantsogt: Member of the red team, researched software and tools that could detect vulnerabilities in the opposing team's servers and attack those servers for both the CCTFs.
- Gabriele Ricciardi: Member of the blue team, setup scripting for defense, server initial install and setup, mod configuration, IPtables testing, traffic monitors development, general defense testing and fine-tuning. On the red team side, attacks testing and suggestion of possible attacks.
- Nicolò Vinci: Member of the blue team, setup scripting for defense, server setup, mod configuration, IPtables setup and testing, webapp PHP patching, database rehaul, traffic and database integrity monitors development, general defense testing and fine-tuning. On the red team side, scripting of base attacks development and testing.

## References

- [1] *Apache webserver download page*. URL: <https://httpd.apache.org/download.cgi>.
- [2] *Create new MySQL user and set permissions*. URL: <https://www.digitalocean.com/community/tutorials/how-to-create-a-new-user-and-grant-permissions-in-mysql>.
- [3] *GoldenEye repository*. URL: <https://github.com/jseidl/GoldenEye>.
- [4] *HPing3 documentation*. URL: <https://www.kali.org/tools/hping3/>.
- [5] *IPtables documentation*. URL: <https://linux.die.net/man/8/iptables>.
- [6] *IPtables modules*. URL: <https://ipset.netfilter.org/iptables-extensions.man.html>.
- [7] *mod\_qos main page*. URL: <http://mod-qos.sourceforge.net/>.
- [8] *modevasive GitHub page*. URL: [https://github.com/jzdziarski/mod\\_evasive](https://github.com/jzdziarski/mod_evasive).
- [9] *modsecurity GitHub page*. URL: <https://github.com/SpiderLabs/ModSecurity/>.
- [10] *MySQL SUM() function*. URL: [https://www.w3schools.com/SQL/func\\_mysql\\_sum.asp](https://www.w3schools.com/SQL/func_mysql_sum.asp).
- [11] *mysql\_connector usage*. URL: [https://www.w3schools.com/python/python\\_mysql\\_getstarted.asp](https://www.w3schools.com/python/python_mysql_getstarted.asp).
- [12] *Nmap main page*. URL: <https://nmap.org/>.
- [13] *PHP hash() function*. URL: <https://www.php.net/manual/en/function.hash.php>.
- [14] *PHP htmlentities() function*. URL: <https://www.php.net/manual/en/function.htmlentities.php>.
- [15] *PHP intval() function*. URL: <https://www.php.net/manual/en/function.intval.php>.
- [16] *PHP isset() function*. URL: <https://www.php.net/manual/en/function.isset.php>.
- [17] *PHP regex*. URL: [https://www.w3schools.com/php/php\\_regex.asp](https://www.w3schools.com/php/php_regex.asp).
- [18] *Scapy documentation*. URL: <https://scapy.readthedocs.io/en/latest/introduction.html>.
- [19] *SlowLoris documentation*. URL: <https://github.com/gkbrk/slowloris>.
- [20] *Snort documentation*. URL: <https://www.snort.org/>.
- [21] *SQLMap documentation*. URL: <https://sqlmap.org/>.
- [22] *TCP kernel parameters*. URL: <https://sysctl-explorer.net/net/ipv4/>.
- [23] *TCP options*. URL: [https://en.wikipedia.org/wiki/Transmission\\_Control\\_Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol).