Offensive Technologies

CCTF

Defense Presentation

Group 3

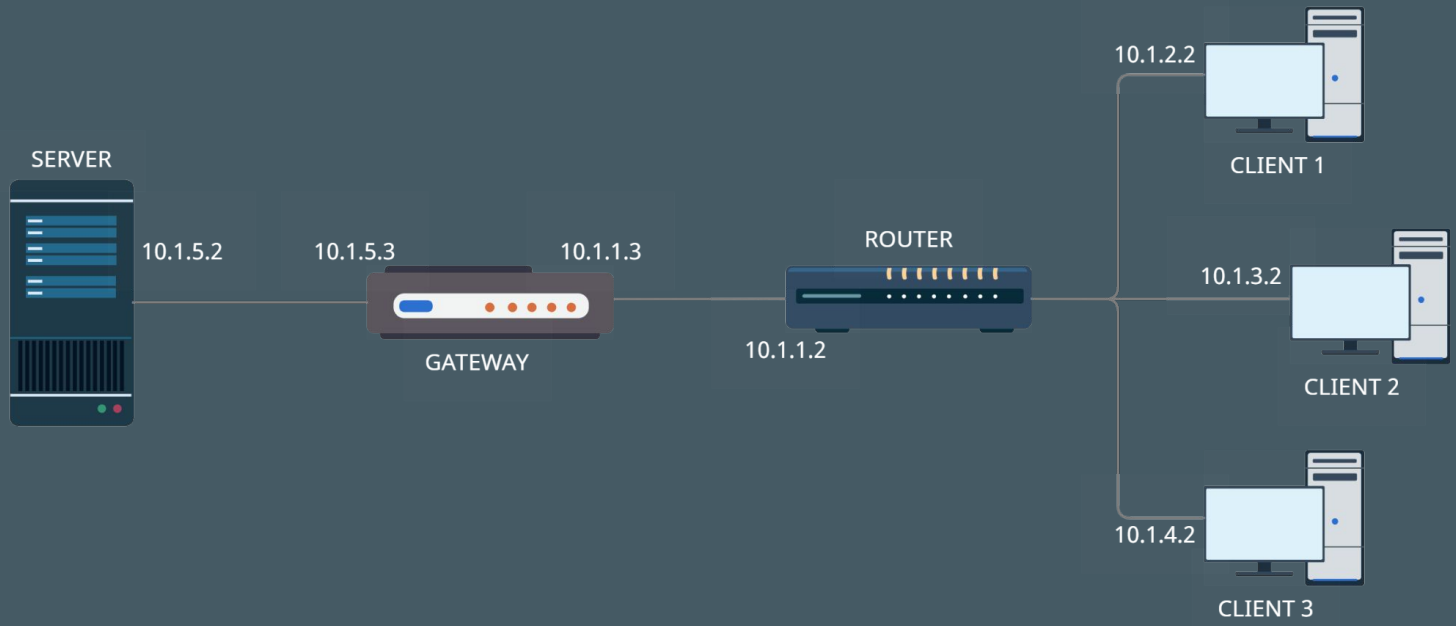
Monday 20 December 2021

19-11-2021
CCTF Resilient

CCTF Resilient

- Context
- Goals
- Strategy
- Configuration
- Tools and scripts
- Results
- Improvements

Context


SERVER
10.1.5.2
10.1.5.3
10.1.1.3
GATEWAY
ROUTER
10.1.1.2
10.1.2.2
CLIENT 1
10.1.3.2
CLIENT 2
10.1.4.2
CLIENT 3

Goal

Guarantee server response time < 500ms

Block unexpected traffic

Traffic as UDP, ICMP and other unwanted traffic should be blocked

Block unauthorized access

The access to the server and the gateway must be protected

Monitor the network

Constantly check what is happening on the network to detect any attack

Strategy

Gateway hardening

Implement a set of rule on the gateway to create a robust firewall

Server hardening

Update Apache, install Varnish and configure them properly

Network monitoring

Continuously check incoming traffic and periodically check if the server was up and running

Configuration

Server

- TCP/IP stack hardening
- Up-to-date **Apache**
- Up-to-date **Varnish**
- Live traffic monitoring

Gateway

- **Firewall** rules (iptables)
- TCP/IP stack hardening
- Live traffic monitoring

Hardening (1/2)

- Tweaking TCP memory parameters following CISCO guidelines for 1Gbps connection
- Reduced TCP timeouts and retries
- Bigger TCP network queue
- Different TCP congestion control algorithm (BBR)

TCP/IP stack

Hardening (2/2)

Firewall

- Static RAW table rules to block
  - ICMP
  - UDP
  - Fragmented IP packets
  - Bogus TCP flags
- Mangle table rules to block
  - Invalid packets
  - Weird MSS values
- Rate limiters

Varnish

HTTP accelerator

Caching static content

Efficient on memory and CPU

Doubles as application level firewall

(filter only HTTP GET requests)

Monitoring (1/3)

`traffic_logger.py`

Where? server, gateway

What? checks the incoming traffic, identifying protocol and source IP

How? using the pyshark module and listening on the network interface

`apache_logger.py`

Where? server

What? check the response times of requests filtering ones above a given threshold

How? looking at the log of Varnish which was set to save such times

Monitoring (2/3)

`server_logger.py`

Where? server

What? checks incoming requests identifying the requested page, the number of time it was requested and the source IP

How? using the pyshark module and listening on the network interface

```
                                    SERVER
------------------------------------ 2021-11-22 10:18:26.391689 ------------------------------------------------
  1.html
  2.html
  3.html
  4.html
  5.html
  6.html
  7.html
  8.html
  9.html
  10.html
  others
```

```
UPDATE EVERY: 0.5s
                                                                SERVER
------------------------------------ 2021-11-22 10:16:57.155976 ------------------------------------------------
SOURCE     | TCP     | B_TCP   | SYN     | B_SYN   | UDP     | B_UDP   | ICMP    | B_ICMP  | OTHER   | B_OTHER
-----------------------------------------------------------------------------------------------------------------
CLIENT_1   | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0
CLIENT_2   | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0
CLIENT_3   | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0
GATEWAY    | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0
ROUTER     | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0
SERVER     | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0
OTHERS     | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0
-----------------------------------------------------------------------------------------------------------------
VARIANCE   | +0      |         | +0      |         | +0      |         | +0      |         | +0      |
```

Monitoring (3/3)

What worked	What may be improved
• All the log worked as expected • The output was clearly readable • A log of the output was saved for further analysis • The first CCTF simulation allowed us to improve the script	• The traffic logger weighed heavily on the performance • A successful SYN flood was not clearly detected • A better analysis of which traffic to filter needed to be done

Results

We were able to identify and automatically stop most of the incoming attacks

Attackers were not able to DoS the server for most of the time

Varnish caching was a game changer and drastically improved page serving times

Improvements

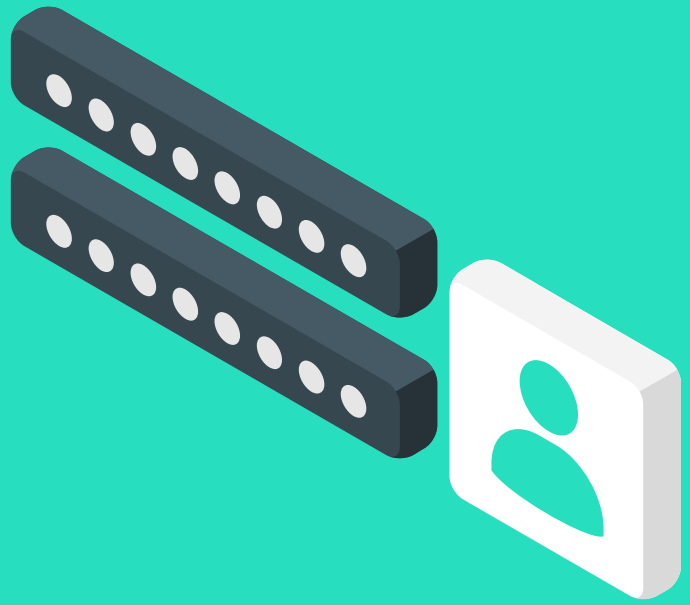
The defense could be described as successful, yet:

- The iptables were misconfigured, we blocked any new incoming SSH connection to the gateway because of the MSS value
- So for some time we were unable to access the gateway reducing our monitoring possibilities

Some key points that could be improved are:

- Adaptive defense based on the attacker's behaviour
- Better configuration testing

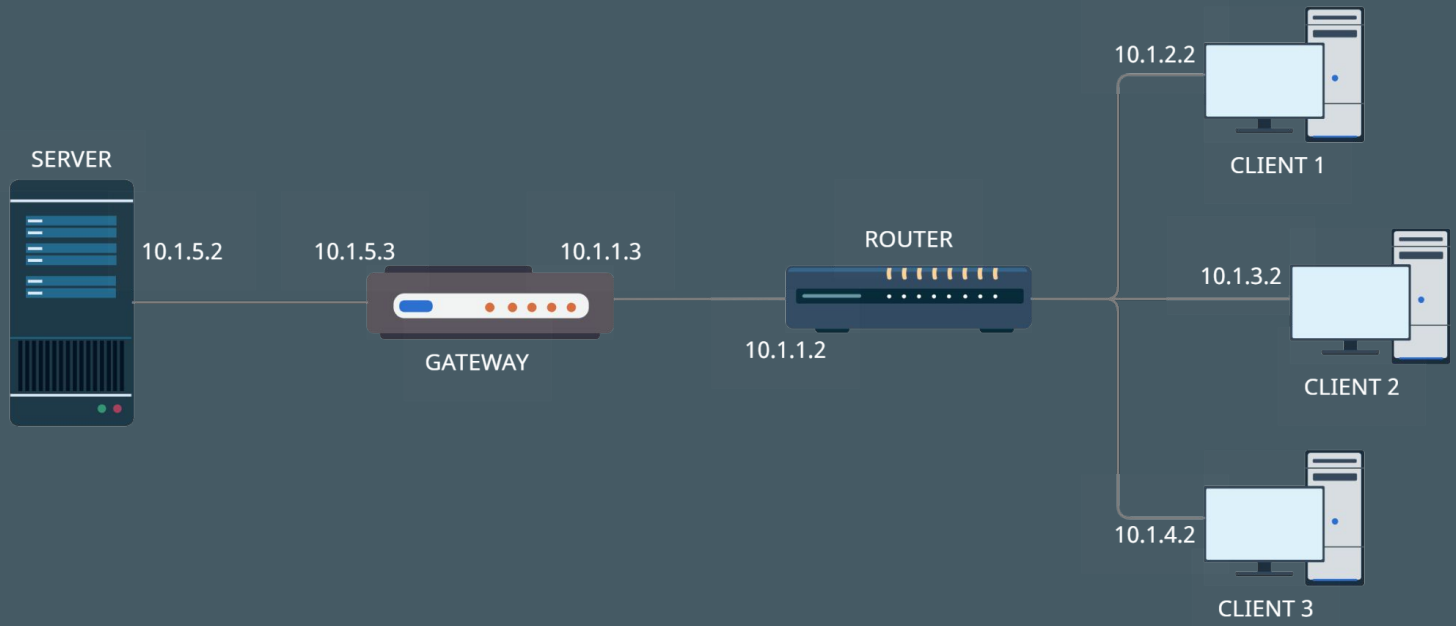
10-12-2021
CCTF Secure Server



CCTF Secure Server

- Context
- Goals
- Strategy
- Configuration
- Tools and scripts
- Results
- Improvements

Context


SERVER
10.1.5.2
10.1.5.3
10.1.1.3
GATEWAY
ROUTER
10.1.1.2
10.1.2.2
CLIENT 1
10.1.3.2
CLIENT 2
10.1.4.2
CLIENT 3

Goal

Guarantee server response time < 500ms

Fix the code

Fix the PHP code against SQL injection, XSS attack, invalid parameters or anything else

Block unauthorized transaction

Avoid unauthorized bank transaction

Monitor database integrity

The database must always have a consistent state. We must record every changes made.

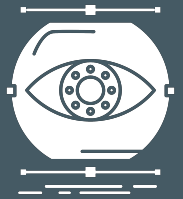
Strategy

Patch the application

The PHP source code was vulnerable to various attacks

Server hardening

Configure the web-server and the database to improve security



Monitoring the database

Periodically check the database for inconsistencies

Starting configuration

Server

- Up-to-date **Nginx**
- PHP code **refactored**
- Hardened Mysql
- Live DB monitoring

Gateway

- **Firewall** rules (iptables)
- **Pyshark** and dependencies installed
- Live traffic monitoring

NGINX

Efficient web server

Hardened configuration

Application level firewall

Source code

String validation

Integer validation

Balance consistency check

Password hashing

Password policy enforcing

MySQL Hardening

Changed root password

Allowed only local access

Enforced the Principle of Least Privilege
using different users

Delay on failed login

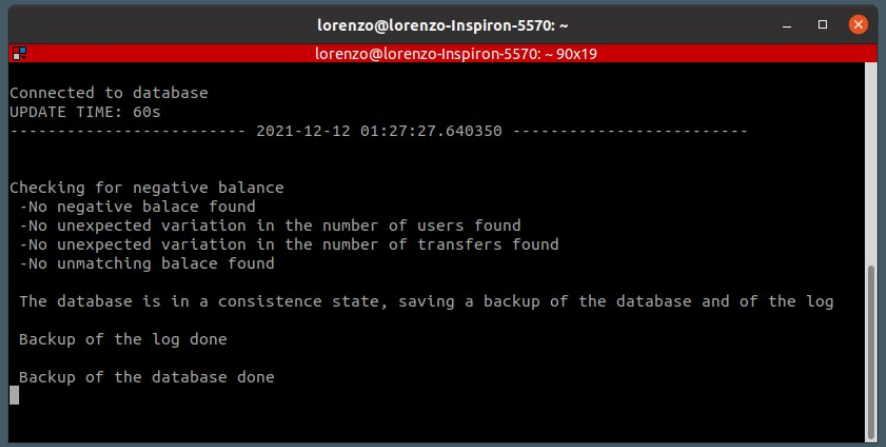
Enabled MYSQL global log

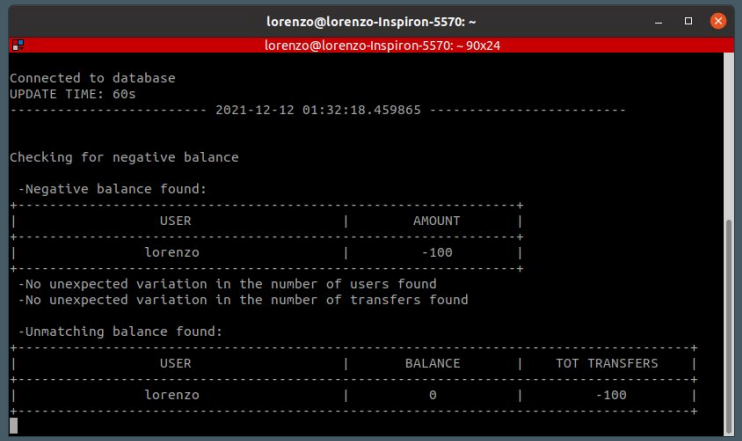
Scripts 1/2

`db_check.py`

Where? Server

What? Periodically check the consistency of the database
Perform a dump of the database and a copy of the `request.log` file





Scripts 2/2

`traffic_logger.py`

Where? Server and Gateway

What? Check the incoming traffic identifying the protocol and the source IP

How? using the pyshark module and listening on the network interface

What worked	What may be improved
• Same as before • No data was compromised	• DPI for HTTP requests • Automatize attacks identification • Use Snort

Results

We always had updated backup of the most valuable file and information

Attackers were not able to compromise our server or our database

We were able to patch almost all the vulnerabilities in the PHP code

Improvements

The defense could be described as successful, yet:

- We missed the exponential log file DoS vulnerability
- We blocked the gateway DNS traffic, cutting it out of the network
- We noticed too late the NAT happening on the router

Some key points that could be improved include:

- Incremental defense based on the behaviour of the attacker
- Better testing of our defense

Thanks!