Offensive Technologies

CCTF

Attack Presentation

Group 3

Friday 17 December 2021

The team that made the dream possible

Lorenzo Cavada

Tommaso Sacchetti

Dmytro Kashchuk

Matteo Franzil

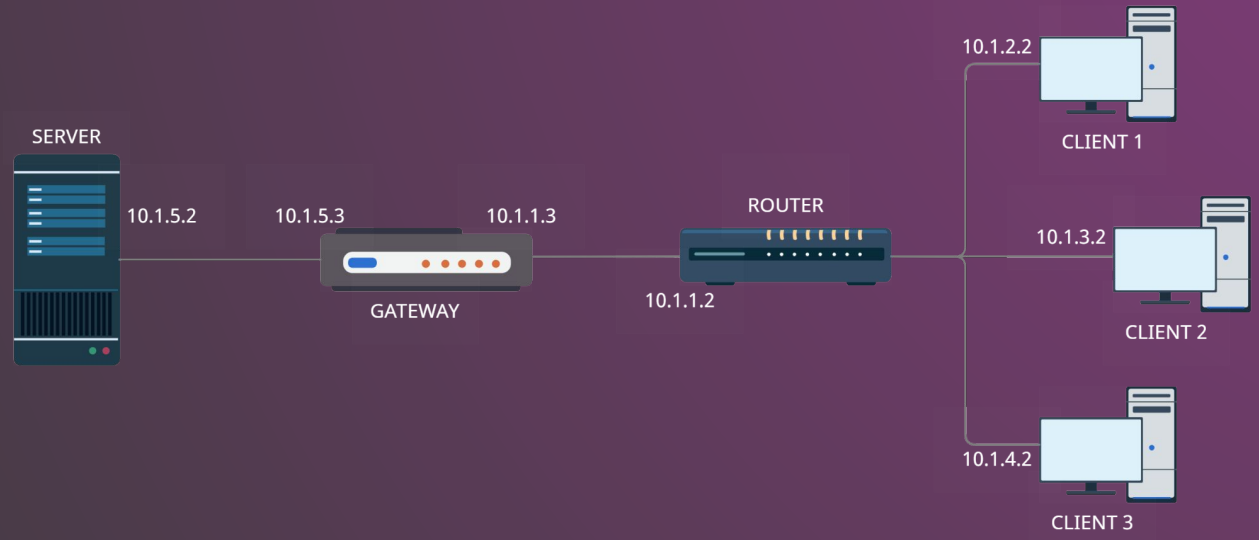
19-11-2021

CCTF Resilient

CCTF Resilient

- Context
- Goals
- Strategy
- Configuration
- 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

Goals

One of the attacker machines sends legitimate requests to the server

Exhaust server resources

Make the server waste CPU cycles, memory and disk space

Flood links

Stress router-gateway bandwidth by flooding links with crafted packets

Crash the server

With delayed connections, low CPU and memory, the server stops serving pages

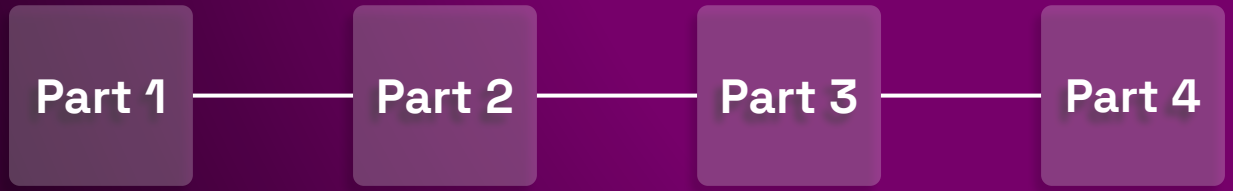
Strategy

Reconnaissance

Test opposing defenses
and capabilities

Slow start

Trick enemy team by
deploying attacks gradually



Focus

Identify key vulnerabilities
to exploit

Wafer smash

Deploy strongest tools
and take down the server

Configuration (global)

- Shared Git repository
- Cloned on each member's home directory
- Automated deployment on each client machine
- Centralization of logs and maximum synchronization

Configuration

All three clients were configured identically.

 GoldenEye

 Sockstress

 Hulk

 Targa3

 SlowLoris

 **hping3**

Loacker

A Go-written program that sends raw TCP/IP SYN packets crafted to look like legitimate curl-created SYNs using all the available bandwidth of a client.

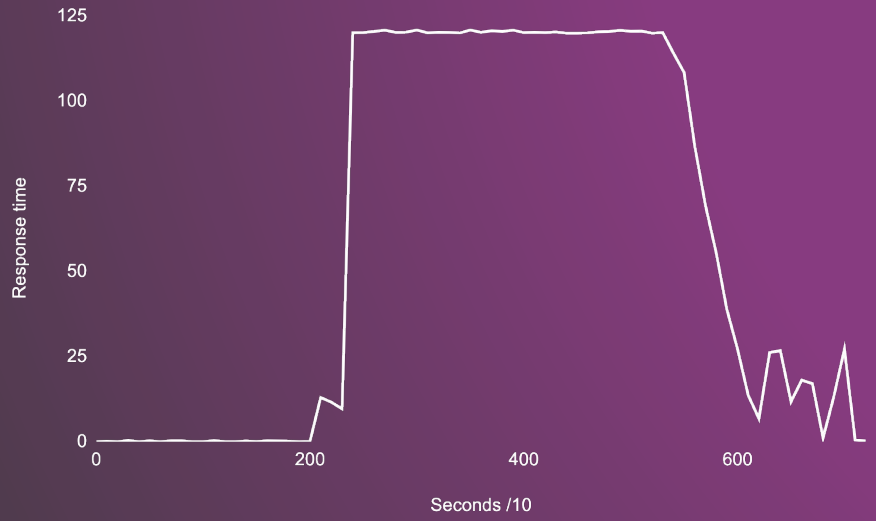
Loacker

Options set up to mock a legitimate SYN packet

Legitimate

A friendly and helpful cow that every handful of second, makes a legitimate request to the server. Comes packaged with automated logging and time measurement.

Results



Interpolated average response time from the server to the legitimate client, measured from the latter. Data aggregated in ten second batches.

Server

After T=2000 s the server started timing out, unable to serve legitimate requests

Links

The opposing gateway-server link was flooded and recovered very slowly

Reactions

The vulnerability was ultimately fixed and subsequent attacks were less effective

Improvements

The attack could be described as successful, yet:

- the defense team used a **sus**picious sliding source port technique that completely nullified Loacker
- a limited toolset and its predictable behaviour reduced the surprise factor

Some key points that could be improved include:

- the choice of tactics for spoofing the attacker machines
- the investigation on gateway vulnerabilities, overlooked in this CCTF
- the reaction time to changes in the defense team setup

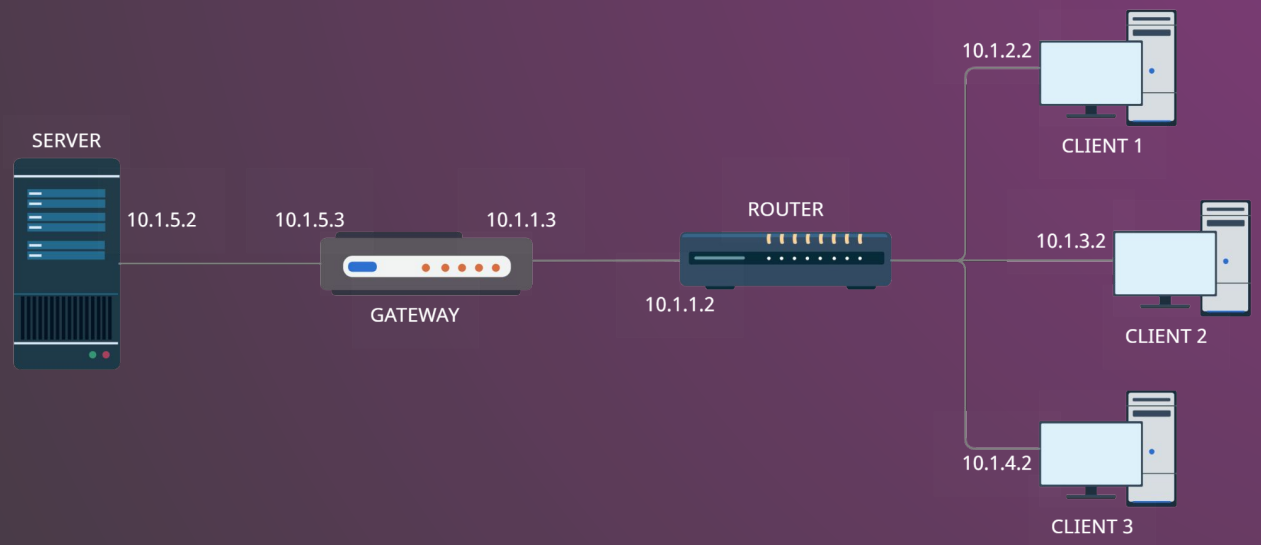
10-12-2021

CCTF Secure

CCTF Secure

- Context
- Goals
- Strategy
- Configuration
- Tools
- 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

Goals

Compromise the blue team server in order to steal information/money

Exhaust server resources

Make the server waste CPU cycles, memory and disk space (without flooding)

Steal money

Compromise the database with bogus requests in order to steal money

Steal credentials

Obtain sensitive information such as passwords and ask the blue team for a ransom

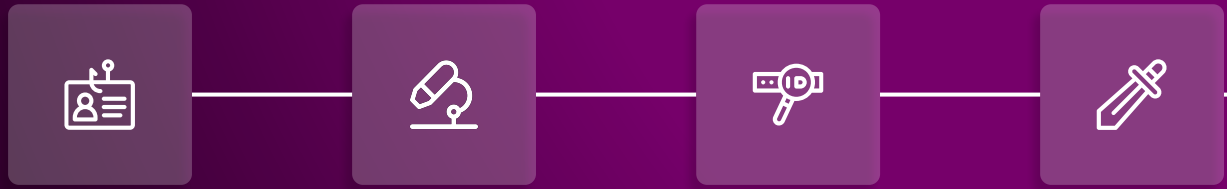
Strategy (1/2)

Phishing

Send a fake email pretending to be the TA

Focus

Identify key vulnerabilities to exploit



Reconnaissance

Test opposing defenses and capabilities

Dictionary attack

SSH passwords

User passwords

Strategy (2/2)

DB flood

Register several users

Slow down queries

Exhaust server space

Fuzzing

Feed the server with crafted query strings

Discovery

Discover deployed pages on web server

Configuration

All three clients were configured identically.

 Nmap

 Attila

 SQLmap

 BruteForcer

 SlowLoris

 emkei.cz

Attila

- Like Attila plundered his enemies with his unmatched prowess in battle, **attila.sh** plundered the enemy database, flooding it with user entries.
- An asynchronous and automated Bash function for making both legitimate and malicious server requests, with logging included.

Brute forcer

- Identify a registered user in the database by trying to register new user with a common username
- Start brute forcing the identified users with the most common passwords

Results

2400
users
created

22, 80, 111
gateway
ports open

< 5%
storage
wasted

Other “results”

No fish were caught

No input-bugs detected

All SQLi blocked

No entry points found

Improvements

The attack did not have any impact. What did go wrong?

- we completely missed the exponential log vulnerability
- the social engineering attempt was too rushed
- very strong defenses from the enemy team mitigated all our attacks

Some key points that could be improved include:

- the investigation of system constraints of the server
- choosing a different dictionary for the password attacks
- being more consistent in the database flooding
- using less improvisation and more step-by-step planning

Thank you!