

# How we analyzed and built an exploit PoC for CVE-2022-24086, a Magento RCE

Catalin Filip

November 10, 2022

# On today's agenda

- CVE-2022-24086 - a quick overview
- Magento - a TL;DR
- Product implications
- Why we chose this CVE
- What the impact for Magento is
- Live demo



# Whoami

- The name is: **Cătălin Filip**
- I've studied at:



ATM Ferdinand I



UPB

# Whoami ++

My passions are:

- Pwn
- Reverse Engineering
- Exploits
- Formula 1 (Hamilton Fan)

And currently I'm working at



# Let's unpack CVE-2022-24086

It's all about an improper  
input validation vulnerability,  
which leads to...

## **Remote Code Execution**

during the checkout process  
in Magento shop



It all started with a lot of fakes...

POC

Notice: This Not The True POC ...

Send Cancel < > Follow redirection

**Request**

Pretty Raw Hex \n ☰

```
1 POST /checkout/cart/add/product/14/uenc?0=cat+/etc/passwd
  HTTP/1.1
2 Host: testlab.com
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

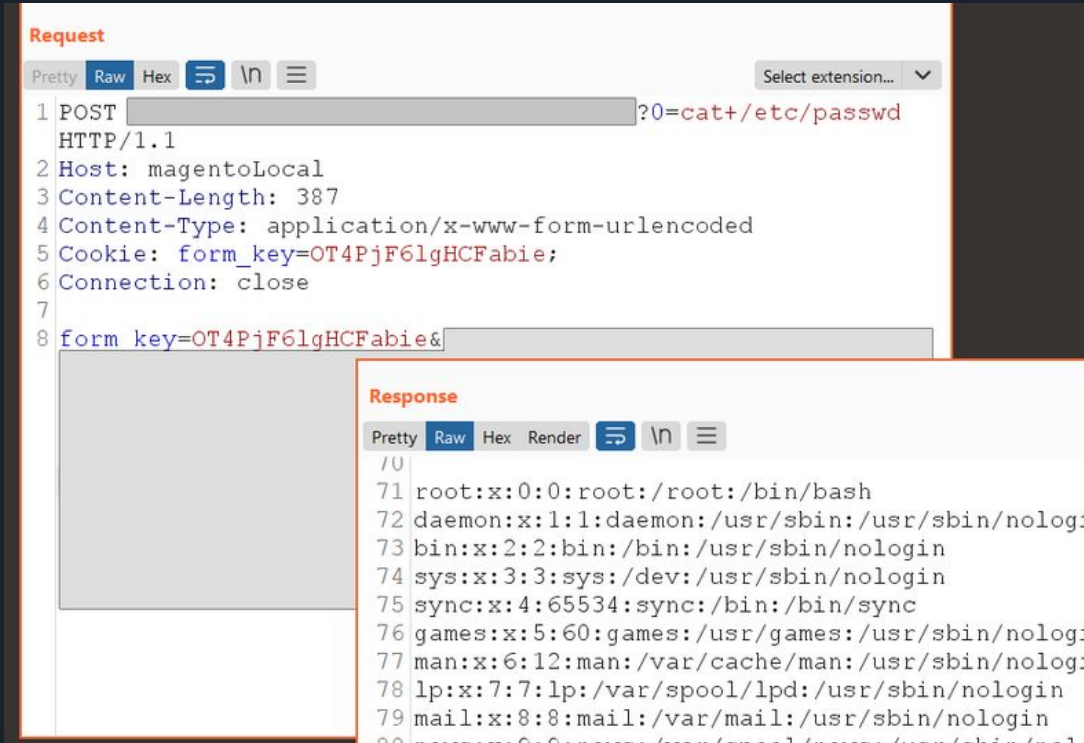
**Response**

Pretty

```
1 HTT
2 Dat
3 Ser
4 X-I
5 Set
6 Set
```

Fake Endpoint

# Or content not publicly available



The screenshot displays the 'Request' and 'Response' sections of a web browser's developer tools. The request is a POST to a hidden endpoint, and the response is a root shell.

```
Request
Pretty Raw Hex [ ] [ ] [ ]
1 POST [redacted]?0=cat+/etc/passwd
  HTTP/1.1
2 Host: magentoLocal
3 Content-Length: 387
4 Content-Type: application/x-www-form-urlencoded
5 Cookie: form_key=OT4PjF6lgHCFabie;
6 Connection: close
7
8 form key=OT4PjF6lgHCFabie&[redacted]

Response
Pretty Raw Hex Render [ ] [ ] [ ]
/0
71 root:x:0:0:root:/root:/bin/bash
72 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
73 bin:x:2:2:bin:/bin:/usr/sbin/nologin
74 sys:x:3:3:sys:/dev:/usr/sbin/nologin
75 sync:x:4:65534:sync:/bin:/bin/sync
76 games:x:5:60:games:/usr/games:/usr/sbin/nologin
77 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
78 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
79 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
80 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

Hidden Endpoint

Ok, this is frustrating -\_-

### After all, what is Magento?

- Magento is an e-commerce platform written in PHP
- It's open-source
- It was acquired by Adobe
- It has a 2.3% market share of the global ecommerce platforms



# The deal with Adobe

In May 2018, Magento was acquired by Adobe.

After that, in 2021, Magento Commerce was integrated in Adobe Commerce and they released several versions:

- Adobe Commerce 2.3.7
- Adobe Commerce 2.4.5

Both versions had 3 types of users:

- Guest
- Customer
- Admin

We chose CVE-2022-24086 because



Widely used



**THE**  
**CHALLENGE**



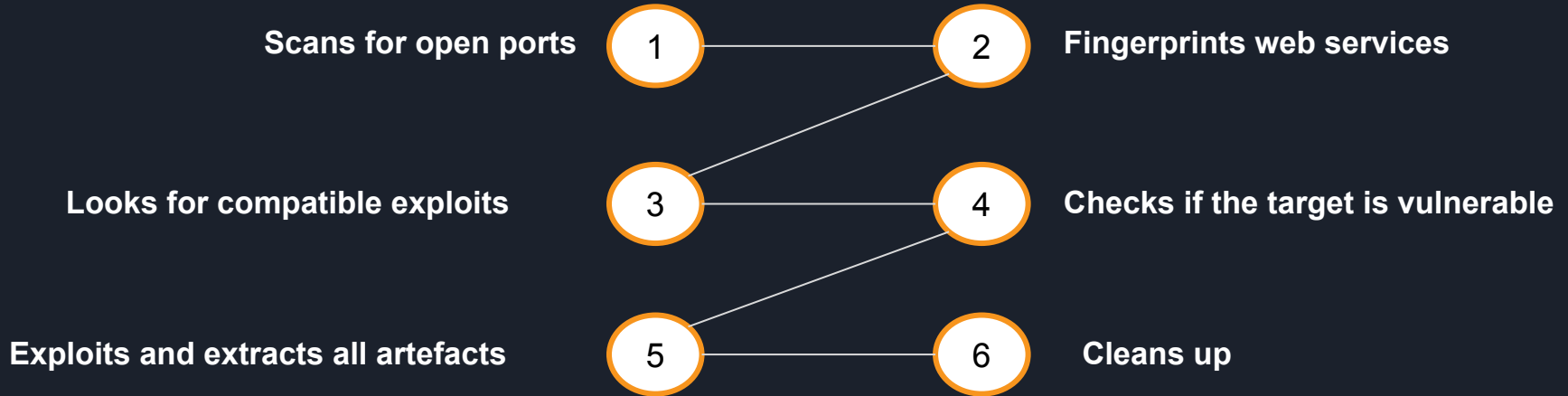
A mention before we dig in

All of this was possible with the  
commitment of the **Sniper**  
**Auto-Exploiter** team that handles  
security research at [Pentest-Tools.com](https://pentest-tools.com)





# What is Sniper?

**Sniper is the automated vulnerability exploitation tool that helps you validate the real impact of critical, widespread CVEs**



# We love automated reporting

 **Exploitation summary** ▼

 At least one service running on the target system was found **vulnerable** and it was **successfully exploited**.


The successful exploits were:

- 80 Magento - Remote Code Execution (CVE-2022-24086)

Sniper managed to obtain remote code execution as user `daemon`.

Current working directory: `./bitnami/magento/pub`.

---

 **Computer:** debian

---

**IP address:** 80.240.21.20

---

**OS:** Linux 4.19.0-18-amd64

---

**Architecture:** x86\_64

The following TCP ports have been fingerprinted on the target machine:

OPEN PORT	SERVICE NAME	SERVICE VERSION	WEB FINGERPRINT	EXPLOIT STATUS
80	http	HTTP	App title: Home page	<b>SUCCESSFULLY EXPLOITED</b> <ul style="list-style-type: none"><li>Magento - Remote Code Execution (CVE-2022-24086)</li></ul>

# The steps we took

Magento deployment



Store configuration



Setting the  
debugging environment



Research, testing and  
crashing



# Following the patch

```
3 + --- a/vendor/magento/module-email/Model/Template/Filter.php
4 + + + + b/vendor/magento/module-email/Model/Template/Filter.php
5 + @@ -618,6 +618,12 @@ public function transDirective($construction)
6 +         }
7 +
8 +         $text = __($text, $params)->render();
9 + +
10 + +     $pattern = '/{{.*?}}/';
11 + +     do {
12 + +         $text = preg_replace($pattern, '', (string)$text);
13 + +     } while (preg_match($pattern, $text));
14 + +
15 +         return $this->applyModifiers($text, $modifiers);
16 +     }
17 +
```

# The regex gave up its secret

The image shows a debugger interface with two main panels. The left panel displays the 'VARIABLES' window, and the right panel shows the source code of a PHP file.

**VARIABLES Panel:**

- Locals:**
  - `$construction: array(3)`
    - 0: `"{{trans "%name," name=$order_data.customer_name}}"`
    - 1: `"trans"`
    - 2: `"%name," name=$order_data.customer_name"`
  - `$directive: "%name," name=$order_data.customer_name"`
  - `$modifiers: "escape"`
  - `$params: array(1)`
    - `$text: "{{we could put every char here .*?}} {{we could put every char here .*?}}..."` (highlighted with an orange box)
- Superglobals:**
- User defined constants:**

**WATCH Panel:**

- `$construction: array(3)`
  - `$arguments: error evaluating code`

**CALL STACK Panel:** Paused on step

**Source Code Panel:** `Filter.php`

```
603 * {{trans "string to translate"}}
604 * {{trans "string to %var" var="$variable"}}
605 *
606 * The |escape modifier is applied by default, use |raw to override
607 *
608 * @param string[] $construction
609 * @return string
610 */
611 public function transDirective($construction)
612 {
613     list($directive, $modifiers) = $this->explodeModifiers($construction[2], 'escape');
614
615     list($text, $params) = $this->getTransParameters($directive);
616     if (empty($text)) {
617         return '';
618     }
619
620     $text = __($text, $params)->render();
621     return $this->applyModifiers($text, $modifiers);
622 }
623
624 /**
625 * Parses directive construction into a multipart array containing the text value and key/value
626 *
```



# Exploit flow - part I

```
611 public function transDirective($construction)
612
613     list($directive, $modifiers) = $this->explodeModifiers($construction[2], 'escape');
614
615     list($text, $params) = $this->getTransParameters($directive);
616     if (empty($text)) {
617         return '';
618     }
619
620     $text = __($text, $params)->render();
621     return $this->applyModifiers($text, $modifiers);
```

```
134 ): void {
135     if (method_exists($stackArgs[$i - 1]['variable'], $stackArgs[$i]['name'])
136         || substr($stackArgs[$i]['name'], 0, 3) == 'get'
137     ) {
138         $stackArgs[$i]['args'] = $this->getStackArgs(
139             $stackArgs[$i]['args'],
140             $filter,
141             $templateVariables
142         );
143
144         $stackArgs[$i]['variable'] = call_user_func_array(
145             [$stackArgs[$i - 1]['variable'], $stackArgs[$i]['name']],
146             $stackArgs[$i]['args']
147         );
148     }
149 }
```

# Exploit flow - part II


```
private function handleObjectMethod(Template $filter, array $templateVariables, int $i, array &$stackArgs): void
{
    $object = $stackArgs[$i - 1]['variable'];
    $method = $stackArgs[$i]['name'];
    if ($this->isMethodCallable($object, $method)) {
        $args = $this->getStackArgs($stackArgs[$i]['args'], $filter, $templateVariables);
        $stackArgs[$i]['variable'] = call_user_func_array([$object, $method], $args);
    }
}
```

```
public function filter($value)
{
    foreach ($this->directiveProcessors as $directiveProcessor) {
        if (!$directiveProcessor instanceof DirectiveProcessorInterface) {
            throw new \InvalidArgumentException(
                'Directive processors must implement ' . DirectiveProcessorInterface::class
            );
        }

        if (preg_match_all($directiveProcessor->getRegularExpression(), $value, $constructions)) {
            foreach ($constructions as $construction) {
                $replacedValue = $directiveProcessor->process($construction, $this->templateVariables);
                $value = str_replace($construction[0], $replacedValue, $value);
            }
        }
    }


    $value = $this->afterFilter($value);
}
```

# Exploit flow - part III



```
public function addAfterFilterCallback(callable $afterFilterCallback)
{
    // Only add callback if it doesn't already exist
    if (in_array($afterFilterCallback, $this->afterFilterCallbacks)) {
        return $this;
    }

    $this->afterFilterCallbacks[] = $afterFilterCallback;
    return $this;
}
```



```
protected function afterFilter($value)
{
    foreach ($this->afterFilterCallbacks as $callback) {
        $value = call_user_func($callback, $value);
    }

    // Since a single instance of this class can be used to filter content
    // prevent callbacks running for unrelated content (e.g., email subject)
    $this->resetAfterFilterCallbacks();
    return $value;
}
```

In the end

Thank you for your attention!

Twitter: @CatalinFilip15

LinkedIn: [linkedin.com/in/catalin-filip-b54456175](https://www.linkedin.com/in/catalin-filip-b54456175)

Gmail: filipcatalinilie@gmail.com