Rampart: Protecting Web Applications from CPU-Exhaustion Denial-of-Service Attacks

Wei Meng†, Chenxiong Qian‡, Shuang Hao*, Kevin Borgolte§
Giovanni Vigna§, Christopher Kruegel§, Wenke Lee‡

†Chinese University of Hong Kong
‡Georgia Institute of Technology
*University of Texas at Dallas
§University of California, Santa Barbara
Outline

• Background & Motivation
  • Rampart
  • Performance Evaluation
  • Mitigation Evaluation
Denial-of-Service (DoS) Attacks

• A class of attacks on availability
  • Keeping users from using a certain computing service

• Two types of DoS attacks
  • Program flaw
    • Supplying an input that can crash the target application or system
  • Resource exhaustion (focus of this work)
    • Requesting a significant amount of computing resources, e.g., CPU, memory, disk, network connections
Distributed DoS (DDoS) Attacks

- Attackers need to send traffic at a rate greater than the **bottleneck processing capacity** of the target system.

- DoS attacks are usually launched by **flooding** the target system with **excessive traffic** to impair the target’s availability.

- DDoS attackers send the traffic from **more than one single source**.

  - E.g., crafting requests from thousands of bots using many IP addresses.

  - **Higher bandwidth** + more difficult to prevent.

- Amplification techniques (e.g., DNS reflection) can be used in DDoS attacks to further increase the **bandwidth** of the attack traffic.
Low-volume Sophisticated DoS Attacks

• Attackers need to send traffic at a rate greater than the bottleneck processing capacity of the target system

• What if I do not have control over thousands of machines?

• Low-volume sophisticated DoS attacks

• Less but much more intense (computationally expensive) attack traffic

• E.g., requesting the server to compute a hash for millions of times
function HashPassword($password)
{
    $random = ''; 

    if (CRYPT_BLOWFISH == 1 && !$this->portable_hashes) {
        $random = $this->get_random_bytes(16);
        $this->gensalt_blowfish($random));
        $hash = crypt($password, $this->gensalt_blowfish($random));
        if (strlen($hash) == 60)
            return $hash;
    } 

    /* ... */

    if (strlen($random) < 6)
        $random = $this->get_random_bytes(6);
    $this->crypt_private($password, $this->gensalt_private($random));

    if (strlen($hash) == 34)
        return $hash;
}
return '*';
Conventional DDoS Attacks

https://www.smithsonianmag.com/history/seventy-years-world-war-two-thousands-tons-unexploded-bombs-germany-180957680/

Sophisticated DoS Attacks

Goals

- Protecting the back end of web applications from
- Low-volume sophisticated CPU-exhaustion DoS attacks
- While limiting impact caused by false-positives
Outline

• Background & Motivation
• Rampart
• Performance Evaluation
• Mitigation Evaluation
Threat Model

• The back-end of a web application is vulnerable against CPU-exhaustion DoS attacks

• The goal of an attacker is to occupy all available CPU resources of the server

• The attacker sends attack payload through normal HTTP requests at a low rate

• The attack requests cannot be easily distinguished from legitimate requests through statistical features

• The attacker does not flood the server with numerous requests
Approach

• Web application CPU usage modeling through context-aware function-level program profiling

• Attack detection using statistical execution model

• Probabilistic request termination

• Exploratory attack request blocking

• Performance optimizations
Call Stack

1. require_once 'lib.php';
2. function foo() {
3.     return bar(1);
4. }
5. $r = foo();  

Function Execution Records

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU time measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>a7f2943c</td>
<td>1026</td>
</tr>
<tr>
<td>1c39686a</td>
<td>1025</td>
</tr>
<tr>
<td>8009ece6</td>
<td>1025</td>
</tr>
<tr>
<td>3825111</td>
<td>1 1.1 0.9 1 ... 1</td>
</tr>
</tbody>
</table>

PID = hash(0, “a.php”)

The measured time is CPU time not wall-clock time

PID stands for the ID of the parent frame
CPU-Exhaustion DoS Attack Detection

- How to detect CPU-exhaustion DoS?
- How to detect the requests causing the DoS?
  - Setting a global timeout?
  - Finding the ones consuming the most CPU time?
- Our approach - finding the ones of which the consumed CPU time is statistically different from their past records
  - Chebyshev’s inequality:
    \[
    P(|X - \mu| > k\sigma) \leq \frac{1}{k^2}
    \]
  - Condition to label suspicious requests: \( T_C > \min(\max(\mu + k \times \sigma, T_{\text{min}}), T_{\text{max}}) \)
CPU-Exhaustion DoS Attack Detection (Cont.)

Function Execution Records

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU time measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>a7f2943c</td>
<td>1026 1055 1035</td>
</tr>
<tr>
<td>1c39686a</td>
<td>1025 1050 1031</td>
</tr>
<tr>
<td>8009ece6</td>
<td>1025 1045 1030</td>
</tr>
<tr>
<td>3825111</td>
<td>1 0.95 1 1</td>
</tr>
<tr>
<td>61c5ab22</td>
<td>700 750 730</td>
</tr>
<tr>
<td>d5d071c9</td>
<td>100 110 105</td>
</tr>
<tr>
<td>7589f636</td>
<td>50 45 51</td>
</tr>
<tr>
<td>81741924</td>
<td>500 510 495</td>
</tr>
<tr>
<td>1f6321a4</td>
<td>10 13 12</td>
</tr>
</tbody>
</table>

Rampart can detect the incident much earlier before bar() returns
Rampart may not determine it as an attack if the CPU usage is low
Probabilistic Request Termination

• Shall we kill the instances serving the suspicious requests?

• Not a good idea - false positive requests may deviate not much from the norm

• Our approach - degrading the priority of those requests by

  • Probabilistically terminating the suspicious requests

• A suspicious request would be temporarily suspended or aborted

• Depending on the current server load and the times it has been suspended
CPU-Exhaustion DoS Attack Blocking

• Is the current design good enough?
  • No, the attackers can still consume the CPU until an alarm

• We need to deploy filters to block follow-up attack requests
  • Requested URI, the request parameters, and the network address

• Are we good to go?
  • A persistent filter - What if it is a false positive filter?
  • A temporary filter - What if the attacker just waits?
The Exploratory Algorithm

• An algorithm to adaptively control the lifetime of a filter
  • Block all matched requests in a primary lifespan
  • Assume we were wrong, i.e., it was a false positive filter
  • Explore the result if it was deactivated continuously until
    • The secondary lifespan expires AND no attack is detected
      • It was a false positive filter OR the attackers had stopped
      • or, an attack is detected again before the expiry of the 2nd lifespan
        • Reset the filter with a longer primary lifespan to penalize the attacker
  • The algorithm controls the upper bound of the rate that one attacker can cause CPU-exhaustion DoS
Performance Optimizations

• Avoid unnecessary system calls
  • Disable profiling for built-in functions

• Control the profiling granularity
  • Profile up to $Max_{Prof\_Depth}$ function frames in the stack

• Improve write performance & mitigate contention
  • Batch processing measurements with dedicated daemon

• Limit the profiling rate
  • Sampling
Implementation

• An extension to the PHP Zend engine
  • 2K lines of C code
  • Linux - getrusage() for measuring CPU time
• A separate batch processing daemon
  • 400 lines of Python code
• Why PHP?
  • It is still the most popular server-side programming language
Outline

• Background & Motivation
• Rampart
• Performance Evaluation
• Mitigation Evaluation
Setup

OWASP ZAP

WordPress

Traffic Generator

Drupal

Intel Xeon quad-core CPU
16GB RAM

Trace

CVE-2014-9034
## Performance Measurements

### Baseline server performance

<table>
<thead>
<tr>
<th>Application</th>
<th>Benchmark</th>
<th>User Instances</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>96</td>
<td>128</td>
</tr>
<tr>
<td>Drupal</td>
<td>ARPT (ms)</td>
<td>277.5</td>
<td>361.8</td>
<td>398.1</td>
<td>502.4</td>
<td>607.3</td>
<td>717.5</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>19.47</td>
<td>24.83</td>
<td>32.21</td>
<td>47.18</td>
<td>59.97</td>
<td>70.53</td>
</tr>
<tr>
<td>Wordpress</td>
<td>ARPT (ms)</td>
<td>20.8</td>
<td>21.7</td>
<td>22.5</td>
<td>38.9</td>
<td>85.6</td>
<td>144.7</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>13.47</td>
<td>22.63</td>
<td>42.21</td>
<td>73.03</td>
<td>86.72</td>
<td>90.11</td>
</tr>
</tbody>
</table>

### Rampart performance

<table>
<thead>
<tr>
<th>Application</th>
<th>Benchmark</th>
<th>Max_Prof_Depth</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>Drupal</td>
<td>ARPT (ms)</td>
<td>397.6</td>
<td>389.0</td>
<td>400.9</td>
<td>393.0</td>
<td>413.6</td>
<td>412.6</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>34.53</td>
<td>34.80</td>
<td>35.62</td>
<td>36.32</td>
<td>38.52</td>
<td>40.94</td>
</tr>
<tr>
<td></td>
<td># Unique Funcs</td>
<td>12</td>
<td>76</td>
<td>567</td>
<td>1,421</td>
<td>2,473</td>
<td>4,019</td>
</tr>
<tr>
<td></td>
<td># Funcs</td>
<td>341</td>
<td>2,167</td>
<td>12,677</td>
<td>31,152</td>
<td>53,263</td>
<td>80,186</td>
</tr>
<tr>
<td></td>
<td>ARPT (ms)</td>
<td>23.7</td>
<td>23.7</td>
<td>23.5</td>
<td>24.6</td>
<td>29.1</td>
<td>36.4</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>44.25</td>
<td>43.12</td>
<td>49.08</td>
<td>56.56</td>
<td>61.60</td>
<td>69.37</td>
</tr>
<tr>
<td></td>
<td># Unique Funcs</td>
<td>17</td>
<td>199</td>
<td>846</td>
<td>3,186</td>
<td>7,909</td>
<td>13,337</td>
</tr>
<tr>
<td></td>
<td># Funcs</td>
<td>422</td>
<td>4,479</td>
<td>15,314</td>
<td>42,957</td>
<td>89,080</td>
<td>136,910</td>
</tr>
</tbody>
</table>

- **Drupal Overhead**
  - 3.41% CPU, 2.8 ms ARPT

- **Wordpress Overhead**
  - 6.87% CPU, 1 ms ARPT
CPU usage and request processing time (RPT) over time for 32 users sending requests every 0.1 seconds to Drupal
Outline

• Background & Motivation
• Rampart
• Performance Evaluation
• Mitigation Evaluation
## Performance Degradation Caused by Attacks

<table>
<thead>
<tr>
<th>Application</th>
<th>Benchmark</th>
<th>No Attack</th>
<th>PHPass [Attackers]</th>
<th>XML-RPC [Attackers]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Drupal</td>
<td>ARPT (ms)</td>
<td>398.1</td>
<td>461.2 (1.16x)</td>
<td>519.6 (1.31x)</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>32.21</td>
<td>88.95</td>
<td>95.05</td>
</tr>
<tr>
<td>Wordpress</td>
<td>ARPT (ms)</td>
<td>22.5</td>
<td>37.0 (1.64x)</td>
<td>49.0 (2.18x)</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>42.21</td>
<td>89.71</td>
<td>94.14</td>
</tr>
</tbody>
</table>
## Effectiveness of Rampart

<table>
<thead>
<tr>
<th>Application</th>
<th>Benchmark</th>
<th>ARPT-U (ms)</th>
<th>ARPT-A (ms)</th>
<th>CPU (%)</th>
<th>FPR (%)</th>
<th>ARPT-U (ms)</th>
<th>ARPT-A (ms)</th>
<th>CPU (%)</th>
<th>FPR (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drupal</td>
<td>PHPass</td>
<td>394.7</td>
<td>203.6</td>
<td>38.51</td>
<td>0.60</td>
<td>24.1</td>
<td>142.1</td>
<td>45.92</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>XML-RPC</td>
<td>427.1</td>
<td>228.3</td>
<td>38.76</td>
<td>0.00</td>
<td>26.1</td>
<td>234.4</td>
<td>51.40</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>423.4</td>
<td>148.1</td>
<td>36.30</td>
<td>0.25</td>
<td>25.6</td>
<td>205.9</td>
<td>49.89</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>460.4</td>
<td>172.2</td>
<td>37.68</td>
<td>0.00</td>
<td>26.8</td>
<td>220.5</td>
<td>50.74</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400.9</td>
<td>258.9</td>
<td>38.84</td>
<td>0.69</td>
<td>24.4</td>
<td>152.8</td>
<td>49.15</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>418.6</td>
<td>166.6</td>
<td>39.62</td>
<td>0.00</td>
<td>26.1</td>
<td>242.3</td>
<td>50.98</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>437.4</td>
<td>160.4</td>
<td>36.30</td>
<td>0.15</td>
<td>24.5</td>
<td>226.3</td>
<td>50.91</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>471.6</td>
<td>181.0</td>
<td>37.73</td>
<td>0.00</td>
<td>25.1</td>
<td>180.2</td>
<td>52.14</td>
<td>0.00</td>
</tr>
</tbody>
</table>

### Drupal baseline performance
- ARPT-U: 398.1 ms
- CPU: 32.21 %

### Wordpress baseline performance
- ARPT-U: 22.5 ms
- CPU: 42.21 %
Performance Degradation Caused by Attacks (Cont.)

CPU usage and RPT over time for 8 PHPass attackers on Drupal without Rampart
Effectiveness of Rampart (Cont.)

CPU usage and RPT over time for 8 PHPass attackers on Drupal with Rampart enabled
Summary

• Rampart performs context-sensitive function-level program profiling to learn function execution models from historical observations

• Rampart detects and mitigates CPU-exhaustion DoS attacks using statistical methods

• Rampart adaptively synthesizes and updates filtering rules to block future attack requests

• Rampart can effectively and efficiently protect web applications from CPU-exhaustion DoS attacks
Thank you!

Q & A