Backporting Security Patches of Web Applications: 
A Prototype Design and Implementation on Injection Vulnerability Patches

Youkun Shi\textsuperscript{1}, Yuan Zhang\textsuperscript{1}, Tianhan Luo\textsuperscript{1}, Xiangyu Mao\textsuperscript{1}, Yinzhi Cao\textsuperscript{2}, Ziwen Wang\textsuperscript{1}, Yudi Zhao\textsuperscript{1}, Zongan Huang\textsuperscript{1}, Min Yang\textsuperscript{1}

\textsuperscript{1}Fudan University
\textsuperscript{2}Johns Hopkins University
A large number of websites are still running old vulnerable applications\(^1,2\)

Critical Drupal Core Vulnerability: What You Need to Know

Drupal is popular, free and open-source content management software. On March 28, the Drupal security team released patches for CVE-2018-7600, an unauthenticated remote code execution vulnerability in Drupal core. The vulnerability affects Drupal versions 6, 7 and 8. Patches have been released for versions 7.x, 8.3.x, 8.4.x and 8.5.x.

\textbf{CVE-2018-7600}, a high-risk unauthenticated RCE vulnerability in Drupal core

After three months since the patch release, there are still about 115,000 unpatched websites.

\(^1\) https://threatpost.com/drupalgeddon-2-0-still-haunting-115k-sites/132518/
\(^2\) https://www.tenable.com/blog/critical-drupal-core-vulnerability-what-you-need-to-know
Vulnerability Patching Practice

1. Patch command
   • Directly applies the official patch (for a specific version) to a vulnerable version
   • **Limitations**: Highly susceptible to code conflicts
     • In our dataset, 1,049/1,526 target versions report code conflicts when applying the patches

2. Auto-upgrade APIs
   • Uses auto-upgrade APIs provided by (some) web applications
   • **Limitations**: Requires significant developer efforts and is prone to compatibility issues
     • In our dataset, 624 / 1,526 target versions do not have auto-upgrade APIs
     863 / 1,526 target versions report compatibility issues
Running Example

- OpenEMR 5.0.0.5 and 5.0.0.6 are affected by CVE-2018-10572
  
  1. OpenEMR doesn’t provide auto-upgrades API
  
  2. Directly apply patch command will fail on the old version due to code conflicts

```php
<?php
+ require_once("./globals.php"); // patch modification
+ require_once($GLOBALS['srcdir'] . "/patient.inc"); // patch modification

use OpenEMR\Core\Header; // the anchor for patch modification location

- include_once("./globals.php"); // patch modification
- include_once($GLOBALS['srcdir'] . "/patient.inc"); // patch modification

$template_dir = $GLOBALS["OE_SITE_DIR"] . "/letter_templates";
...

- $fh = fopen("$template_dir/".$_GET["template"], 'r'); // patch modification
+ $fh = fopen("$template_dir/" . // patch modification
  convert_safe_file_dir_name($_GET["template"]), 'r');
...

?>
```

Code Snippet of OpenEMR 5.0.0.5

- Code conflicts hinder patch apply!

Official Patch for CVE-2018-10572 on OpenEMR 5.0.0.6
Patch Backporting

• **Problem Definition:** Given a patch for a vulnerable version, backport the patch to fix the same vulnerability on another vulnerable version.

• **Challenges:** How to automatically backport patches to old versions with guaranteed compatibility and security?
  • Can the patch be compatible (not affecting normal functionality) with another vulnerable version?
  • Can the patch fix the vulnerability on another vulnerable version?
  • Can the patch be automatically applied to another vulnerable version?
Problem Understanding

• Three Mismatches among <vulnerability, patch, target>

  • <Patch, Vulnerability> Mismatch → break compatibility
    • The patch may contain vulnerability-irrelevant modifications, which may affect the functionalities of a web application.
  
  • <Target, Vulnerability> Mismatch → break security
    • The target version may have a different vulnerability logic to the one that the patch aims to fix, thus requiring a new patching logic.
  
  • <Patch, Target> Mismatch → break automation
    • The patch may not be easily applied to a target version due to cross-version code location changes
This Work: Patch Backporting

• **Scope:** Injection-based vulnerability patches
  - Key Insight: injection-based vulnerabilities are fixed by restricting the capability of the sink function
  - **Sink Capability:** all the user inputs that can go to the sink functions

• **Key Idea:** backport the safe sink capability across different vulnerable versions
  - **Safely Backportable Patch (SBP):** the patch only restricts the capability of the sink function
    - Filter our irrelevant patch modifications: ✗ <Patch, Vulnerability> Mismatch & ✓ Compatibility
  - **Safely Backportable Version (SBV):** the target has the same sink capability as the pre-patch version
    - Select only a part of backportable versions: ✗ <Target, Vulnerability> Mismatch & ✓ Security
  - Deploy SBP upon on SBV: replace the vulnerable sink with the safe sink
    - Only requires minimal source code modifications: ✗ <Patch, Target> Mismatch & ✓ Automation
Running Example: Patch Backporting

```
<?php
function convert_safe_file_dir_name($label)
{
    return preg_replace('/[\^A-Za-z0-9\._-]/', '', $label);
}

function safe_fopen($bp_globals_oe_site_dir, $bp_get_template)
{
    $template_dir = $bp_globals_oe_site_dir . '/letter_templates';
    return fopen($template_dir . '/' .
        convert_safe_file_dir_name($bp_get_template), 'r');
}
```

Official Patch

```
<?php
+ require_once("../globals.php");
+ require_once(SGLOBALS['sdir'] . "/patient.inc");

use OpenEMR\Core\Header;

+ include_once("../globals.php");
+ include_once(SGLOBALS['sdir'] . "/patient.inc");
+ $template_dir = $GLOBALS['OE_SITE_DIR'] . "/letter_templates";

+ $fh = fopen("$template_dir/$GET['template'], 'r');
+ $fh = fopen("$template_dir/" .
    convert_safe_file_dir_name($GET['template']), 'r');

...?
```

Deployed Safely Backportable Patch on OpenEMR 5.0.0.5

```
<?php
+ $bp_get_template = $GET['template'];
+ $bp_globals_oe_site_dir = $GLOBALS['OE_SITE_DIR'];
+ sanitize_all_escapes = true;
+ $fake_register_globals = false;

+ include_once("../globals.php");
+ include_once($GLOBALS['sdir'] . "/patient.inc");
+ $template_dir = $GLOBALS['OE_SITE_DIR'] . "/letter_templates";

+ $fh = fopen("$template_dir/$GET['template'], 'r');
+ $fh = safe_fopen($bp_globals_oe_site_dir, $bp_get_template);

...?
```

Safely Backportable Patch for CVE-2018-10572

```
<?php
+ function convert_safe_file_dir_name($label)
+    return preg_replace('/[\^A-Za-z0-9\._-]/', '', $label);
+
+ function safe_fopen($bp_globals_oe_site_dir, $bp_get_template)
+    $template_dir = $bp_globals_oe_site_dir . "/letter_templates";
+    return fopen($template_dir . '/' .
        convert_safe_file_dir_name($bp_get_template), 'r');
```

convert

SBP

SBV

safely backport with guarantee

deploy

Page 8
Sink Capability

• The Representation
  
  • Sink Flow: a control-flow path leading to the sink function
    
    • The inputs that reach the sink along each path are represented as $<flow_1, flow_2, ...>$
  
  • Each sink flow consists of $<RC_{flow}, DE_{flow}>$
    
    • Reaching Condition ($RC_{flow}$): a set of the control-flow conditions in the flow
    • Data Expression ($DE_{flow}$): the symbolic expression of the critical sink parameters in the flow

  • Thus, the sink capability can be represented as
    
    • $\{<RC_{flow1}, DE_{flow1}>, <RC_{flow2}, DE_{flow2}>, ...\}$
Sink Capability Example

- The Sink Capability (SC) can be represented as \(<flow_1, flow_2, flow_3>\)

```
<?php
    $input = $_GET['inp'];
    if ($condition1)
        $value = $input."bar1";
    else{
        if($condition2){
            $value = $input."bar2";
        }else{
            $value = $input."bar3";
        }
    }
    $value = 'foo'.$value;
    sink_func($value);
?>
```
• Safely Backportable Patch (SBP) Properties: \textit{pre-patch} vs \textit{post-patch}

1. \( P_{\text{SBP-a}} : R_{\text{flow}_k}^{\text{post}} \) is a subset of \( R_{\text{flow}_k}^{\text{pre}} \)

2. \( P_{\text{SBP-b}} : D_{\text{flow}_k}^{\text{post}} \) is a subset of \( D_{\text{flow}_k}^{\text{pre}} \)

3. \( P_{\text{SBP-c}} : R_{\text{flow}_k}^{\text{post}} \) and \( D_{\text{flow}_k}^{\text{post}} \) are deterministically computable for every \( \text{flow}_k \)

• \textbf{Compatibility Guarantee:} SBP deployment will not affect the functionality of the target application

• Safely Backportable Version (SBV) Properties: \textit{pre-patch} vs \textit{target}

1. \( P_{\text{SBV-a}} : R_{\text{flow}_k}^{\text{pre}} \) is same as \( R_{\text{flow}_k}^{\text{target}} \) for every \( \text{flow}_k \)

2. \( P_{\text{SBV-b}} : D_{\text{flow}_k}^{\text{pre}} \) is same as \( D_{\text{flow}_k}^{\text{target}} \) for every \( \text{flow}_k \)

• \textbf{Security Guarantee:} SBP deployment can fix the vulnerability of the target application
Approach Overview

• Three Steps

1. SBP Identification & Generation
   • Analyze whether a patch is backportable and if so, transform it to SBP

2. SBV Verification
   • Verify whether a target version is an SBV (aka, can apply the SBP)

3. Patch Deployment
   • Automatically deploy SBP on an SBV

• Automatic Tool: SKYPORT, based on PHPJoern
SKYPOR T Workflow

Four Modules

- Patch Affection Analysis (M1), Sink Capability Extraction (M2)
- Backportable Analysis (M3), Patch Deployment (M4)

Step I: SBP Verification and Generation

- Official patch
- Post-patch version
- Pre-patch version
- Target version

Step II: SBV Verification

- Sink Extraction
- All sink functions

Step III: Patch Deployment

- SCpost
- SCpre
- STarget
- SBP Analysis
- Backportable Analysis
- SBV Analysis
- Patched target version

SBP

SBV
### Evaluation & Dataset

<table>
<thead>
<tr>
<th>CMS Name</th>
<th># CVEs</th>
<th># Versions</th>
<th># &lt;CVE, Version&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordPress</td>
<td>34</td>
<td>187</td>
<td>430</td>
</tr>
<tr>
<td>PHPMyAdmin</td>
<td>29</td>
<td>108</td>
<td>257</td>
</tr>
<tr>
<td>Prestashop</td>
<td>11</td>
<td>34</td>
<td>101</td>
</tr>
<tr>
<td>RoundcubeMail</td>
<td>8</td>
<td>48</td>
<td>76</td>
</tr>
<tr>
<td>MantisBT</td>
<td>24</td>
<td>74</td>
<td>198</td>
</tr>
<tr>
<td>Piwigo</td>
<td>11</td>
<td>37</td>
<td>108</td>
</tr>
<tr>
<td>OpenEMR</td>
<td>11</td>
<td>20</td>
<td>70</td>
</tr>
<tr>
<td>phpipam</td>
<td>3</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>MISP</td>
<td>9</td>
<td>55</td>
<td>118</td>
</tr>
<tr>
<td>LimeSurvey</td>
<td>15</td>
<td>82</td>
<td>155</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>155</strong></td>
<td><strong>651</strong></td>
<td><strong>1,526</strong></td>
</tr>
</tbody>
</table>

**Selection Criteria**

1. The Web application with more than 1k stars in the GitHub
2. Injection vulnerability patches
3. Patches that fix the vulnerability by restricting the sink functions
Evaluation & Dataset Statistics

• Patches
  • 98 / 155 security patches contain vulnerability-irrelevant modifications
    • E.g., functionality modifications, variable or function name modifications
    • May lead to backward compatibility or patch deployment issues

• Target versions
  • 563 / 1,526 target versions do not have the same vulnerable logic as the pre-patch
    • These versions are not SBVs, thus not being backportable (aka, requiring a new patch)
  • 1,071 / 1,526 target versions have code location changes around the patch
    • May lead to code conflicts when directly applying the original patch via patch command

These results show that patch backporting is non-trivial!
1. **Effectiveness:** How effective is SKYPORT in patch backporting?
   • SKYPORT successfully backport 98 SBPs to 750 SBVs with 100% success rate

2. **Efficiency:** How efficient is SKYPORT in patch backporting?
   • SKYPORT takes 6459.75 seconds on average for an end-to-end case

3. **Comparison:** How does SKYPORT compare to existing practices?

<table>
<thead>
<tr>
<th></th>
<th>Patch Command</th>
<th>Auto-upgrade/Strict</th>
<th>Auto-upgrade/Relaxed</th>
<th>SKYPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>455</td>
<td>39</td>
<td>149</td>
<td>750</td>
</tr>
</tbody>
</table>
Evaluation & SKYPORT-patched Apps

• Evaluating SKYPORT-patched Apps involves significant human efforts
  • We evaluate a subset of SKYPORT-patched Apps, covering <11, 27> CVE-version pairs

1. **Security**: Can the SBPs defend against vulnerability-related attacks?
   • SKYPORT-patched apps successfully defended against all the collected exploits

2. **Compatibility**: Do the SBPs incur functionality issues?
   • SKYPORT-patched apps with 100% test pass ratio for compatibility with single or multiple SBPs

3. **Performance**: What is the performance overhead introduced by SBPs?
   • The SBPs introduce negligible overhead when compared with the official patches
Conclusion

- Methodology for automatic patch backporting with guaranteed compatibility and security.
- Formulation for safely backportable patches (SBP) and safely backportable versions (SBV), which enable safe patch backporting.
- Tool for automatically backporting injection-based PHP patches to old vulnerable versions.
- Evaluation results that demonstrate the effectiveness and efficiency of the proposed approach.
THANKS
Q&A

ykshi21@m.fudan.edu.cn