RIPS: A static source code analyser for vulnerabilities in PHP scripts

By Russ McRee – ISSA member, Puget Sound (Seattle), USA Chapter

Prerequisites

Linux platform – installation methodology described is specific to Ubuntu

While I've often focused on run-time web application security assessment tools, I've paid absolutely no attention to static analysis tools. That changes now starting with Johannes Dahse's RIPS, a static source code analyzer for vulnerabilities in PHP scripts.

Johannes informed me via email that he uses RIPS whenever he audits PHP code; he's been using and improving the tool for three years now. He states that RIPS is capable in detecting flaws in well-known open source products as well as his "own security-aware written code. ;)" RIPS currently works very well for small PHP apps and "low hanging fruit" in large apps, but does have some significant issues with large apps largely due to missing Object-Oriented Programming (OOP) support (everything is OOP these days :/).

Johannes indicates that he really likes RIPS for his own use as it helps quite a bit during manual audits. Although RIPS does not find everything or may have false positives, the interface allows quick identification of sensitive code blocks. It also helps understand the code by jumping through the highlighted lines, several listings of sources, sinks, graphs, etc., all part of the reconnaissance and mapping phase for any good penetration test or security review. In summary, "it's not a perfect vulnerability detector (yet), but it helps analyzing PHP code a lot."

While you should consider RIPS a work in progress, it's already quite effective as described below and is in a state of active development.

Johannes mentions in his paper, "RIPS - A Static Source Code Analyser for Vulnerabilities in PHP Scripts,"[1] that his motivation and personal intent was to find security vulnerabilities quickly during Capture The Flag (CTF) contests. During a CTF contest vulnerable source code is analyzed by participating teams and security vulnerabilities have to be identified, patched, and exploited. Exploiting security vulnerabilities is the equivalent to capturing flags from opposing teams resulting in awarded points. The team with the most captured flags wins. A number of web applications distributed by CTF organizers have been written in PHP, leading to PHP's popularity as a scripting language in CTFs too.

Graduate from such CTFs to penetration testing engagements and you can see the value of reviewing web application source code for vulnerabilities, performed by either the developer or penetration testers. As large applications often include many thousands of lines of code coupled with financial and resource constraints, manual source code review is often untenable leading to incomplete reviews. Tools such as RIPS can help penetration testers and developers reduce time and cost by automating time-intense processes inherent to source code reviews.[2]

Installation

RIPS installation is really straightforward. I utilize it with LAMP[3] stacks on Ubuntu 10.10 and 11.04 servers, but RIPS has no real dependencies other than a local web server with a PHP interpreter and a web browser; it should work just as well on a WAMP/WIMP[4] stack or others such MAMP, SAMP, or OAMP (really, look them up).

Download the packages from SourceForge[5] and be sure to include Johannes' RIPS paper (rips-paper.pdf) as it is essential reading as reference material. Unpack `rips-0.40.zip` in `/var/www`, or whatever is the appropriate webroot for your preferred system. Be sure to then unpack the optional `rips-0.40-patch_SaveGraph.zip` and merge the `js` directory and `main.php` into the RIPS install directory you just populated.

Static code analysis with RIPS

Johannes' paper describes "sensitive sinks" as potentially vulnerable functions (PVF) that execute critical operations and should only be called with trusted or sanitized data.

RIPS currently scans for 233 PVFs separated into the following vulnerability types:

- Cross-Site Scripting (4)
- SQL Injection (54)

1 http://www.php-security.org/downloads/rips.pdf or download with RIPS package from SourceForge below.

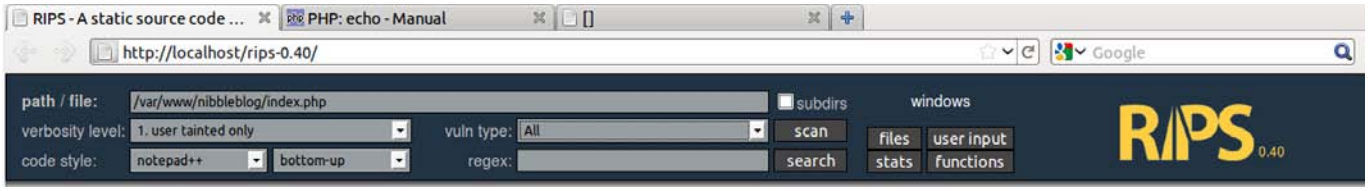
2 http://sourceforge.net/projects/rips-scanner/files/rips-paper.pdf/download.

3 http://en.wikipedia.org/wiki/LAMP_%28software_bundle%29.

4 http://en.wikipedia.org/wiki/WAMP.

5 http://sourceforge.net/projects/rips-scanner/files/.

Figure 1 – RIPS UI



- File Disclosure (39)
- File Manipulation (20)
- File Inclusion (8)
- Remote Code Execution (47)
- Remote Command Execution (8)
- Header Injection (27)
- XPath Injection (3)
- LDAP Injection (5)
- Unserialize / POP (1)
- Other (17)

The RIPS UI is simple, but there are some nuances. The first is specific to verbosity levels. Experiment with this a bit; you'll be most satisfied with results from verbosity levels 1 and 2. As the level increases though, so, too, do the false positives. Always start with level 1 as it "scans only for PVF calls which can be tainted with user input without any detected securing actions in the trace." Tainting in this scenario refers is defined as follows:

"A web application security vulnerability can occur when data supplied by the user (e.g. GET or POST parameters) is not sanitized correctly and used in critical operations of the dynamic script. An attacker might be able to inject code that changes the behavior and result of the operation during script execution in an unexpected way. These kind of vulnerabilities are called taint-style vulnerabilities because untrusted sources such as user-supplied data is handled as tainted data that reaches vulnerable parts of the program called sensitive sinks. Any kind of data that can be modified by the user such as GET or POST parameters as well as cookie values, the user agent, or even database entries or files have to be assumed as tainted." [6]

Also important in the RIPS UI is the vuln type. I almost always select *All*, but you can narrow the approach to server-side (SQLi, file inclusion, etc) or XSS. Note that you can also tweak code style rendering to your preference; I favor notepad++.

I'll explore a PHP-based web application for which I've already published (via coordinated disclosure) an advisory for discovered vulnerabilities. As the disclosed vulnerability findings are very specific they provided ample opportunity to compare them against the application source code for the flaws that I discovered via run-time analysis with Burp Suite.

6 Johannes Dahse , rips-paper.pdf, pg.4.

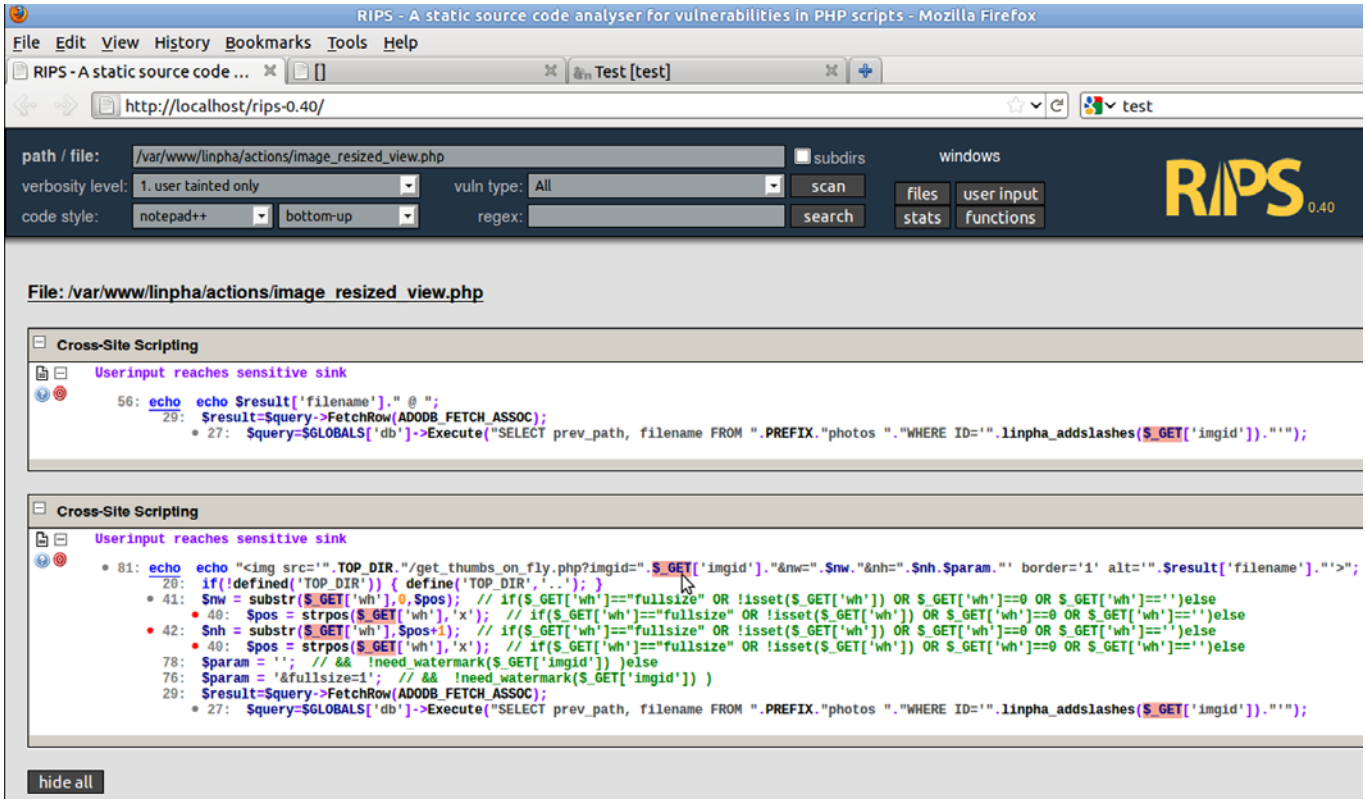


Figure 2 – RIPS scan

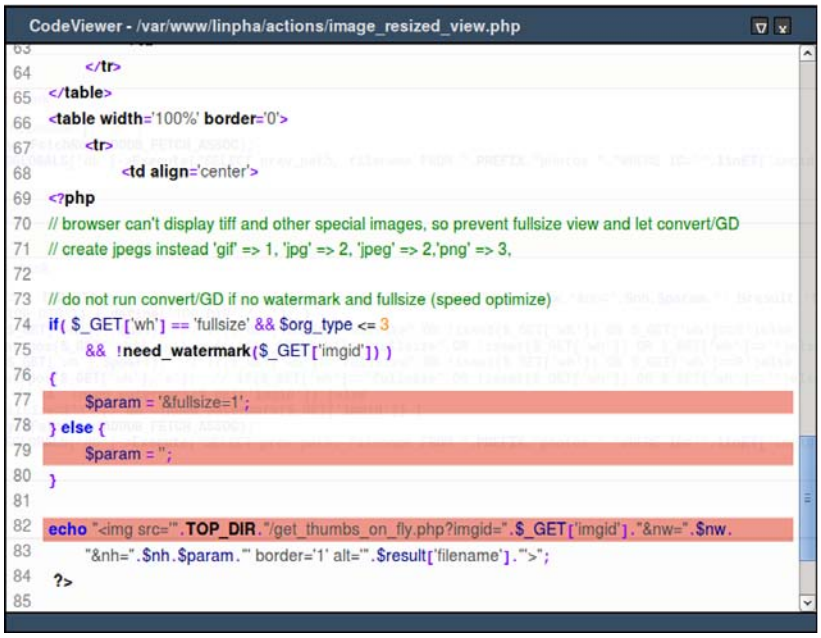


Figure 3 – RIPS CodeViewer

I chose LinPHA 1.3.4, a photo/image gallery that is no longer supported or maintained, having reported[7] cross-site request forgery (CSRF) and cross-site scripting (XSS) vulnerabilities in March 2009.

Run-time analysis of LinPHA quickly determined that input passed via GET to the "imgid" parameter is not properly sanitized by the `image_resized_view.php` script before being returned to the user. This vulnerability can be exploited to execute arbitrary HTML and JavaScript code in a user's browser session in the context of an affected site.

To compare this finding to source code analysis with RIPS, I navigated to `/var/www/linpha/actions/image_resized_view.php` in the RIPS UI and clicked *scan*.

The results were immediate and clearly identified in source code the same vulnerability I'd discovered at run-time, as seen in Figure 2.

You'll note four small icons in the upper left-hand corner of each XSS finding in the RIPS UI. Clockwise, in order from the upper left, these buttons provide *review code, minimize, exploit,* and *help.*

The review code feature is an essential tool during RIPS use. Clicking it will pop-up a CodeViewer window that will mark vulnerable lines in the source code as seen in Figure 3.

Use the minimize button when the developer walks up behind you and shoulder surfs you while you pick apart his code. ;-) You've probably noticed the reference to the "imgid" parameter, validating our comparison between run-time and static analysis.

The exploit feature is also very useful; Exploit-Creator will generate a PHP script that can be used as proof-of-concept code for your penetration test results. Running the script as part of a demonstration for the client is an effective manner of exhibiting the many facets of the security development lifecycle.

Designed to use curl, an edited snippet of script generated for my test scenario follows:

```
#!/usr/bin/php -f
<?php
# image_resized_view.php curl exploit
$target = $argv[1];
$ch = curl_init();
curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch, CURLOPT_URL, "http://
localhost/linpha/actions/image_resized_view.
php/image_resized_view.php?imgid=%22%3E%3CSCRIP
T%3Ealert(document.domain)%3C%2FSCRIPT%3E");
curl_setopt($ch, CURLOPT_HTTPGET, 1);
curl_setopt($ch, CURLOPT_USERAGENT,
"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT
5.0)");
```

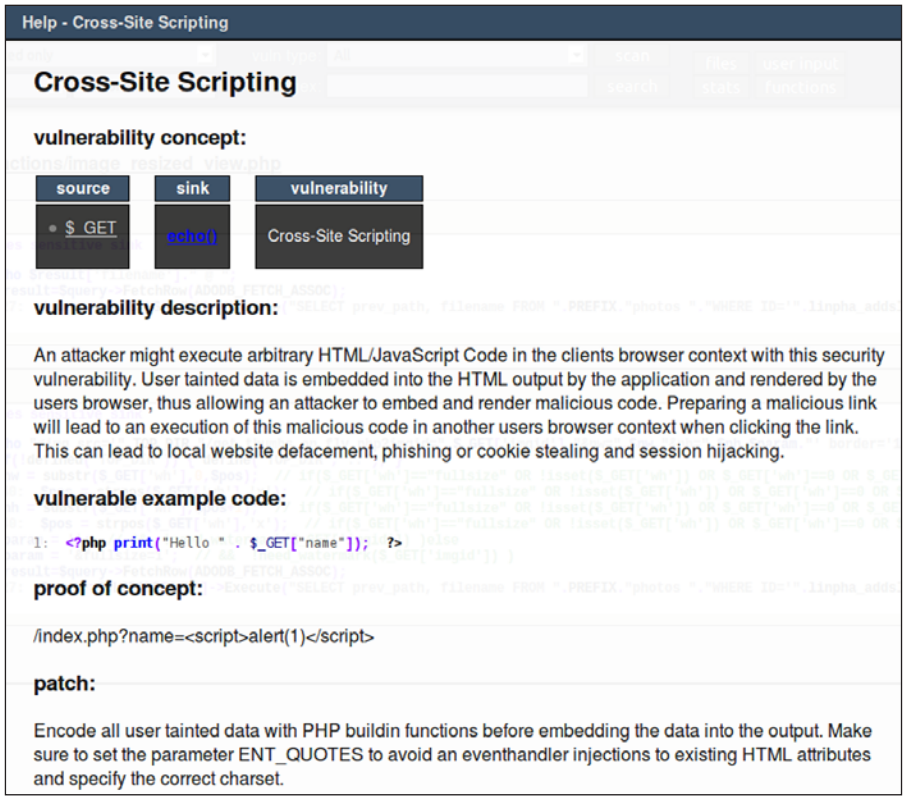


Figure 4 – RIPS help

7 http://secunia.com/advisories/34130/.

Target URL, GET parameter, and even a cookie to maintain state are all definable.

The *hotpatch* button leads to a help file useful in providing guidance for the mitigation and remediation of the specific vulnerability detected in source as seen in Figure 4.

Recently added RIPS features include:

- Graphical representations of user defined functions and calls including color coded sources, sensitive sinks, and vulnerabilities via the *functions* button
- Listing of all related user input via the *user* input button
- Scan results are displayed via *stats* as seen in Figure 5

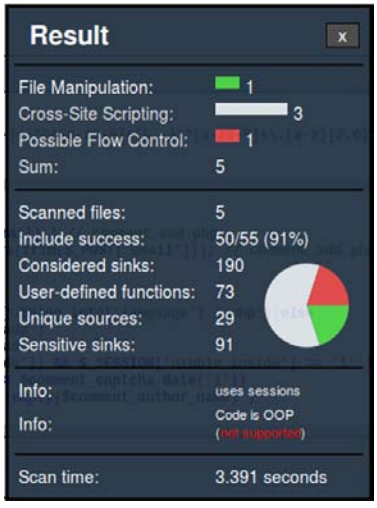


Figure 5 – RIPS results

- Graphical representations of scanned files and includes (my favorite) with representation of "how files are connected to each other, what files accept sources (userinput) and what files have sensitive sinks or vulnerabilities"[8] as seen in Figure 6.

The *functions* and *files* features both allow clickable listings to drive into specific source code references, and the graphs can be saved for use in reports.

8 https://websec.wordpress.com/2011/06/04/project-rips-status/

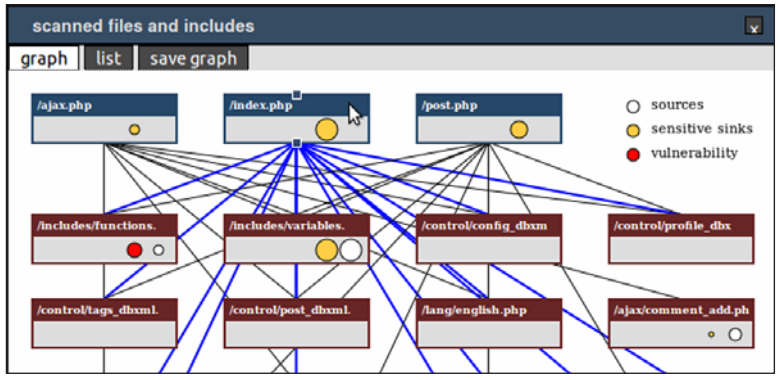


Figure 6 – RIPS files

I found myself wandering way down the rabbit hole with RIPS, slipping off on vulnerability exploration tangents I had not expected; refreshing indeed.

In conclusion

I'm excited by the prospects for this project and hopeful that Johannes will keep up the great work on RIPS. There aren't enough truly useful open source, freely available static code analyzers and RIPS really starts to close that gap for PHP applications.

Download and install RIPS, and give the related paper a thorough read as it goes into far more depth than covered here.

Ping me via email if you have questions (russ at holisticinfosec dot org).

Cheers…until next month.

Acknowledgements

—Johannes Dahse, RIPS developer and project lead

About the Author

Russ McRee, GCIH, GCFA, GPEN, CISSP, is team leader and senior security analyst for Microsoft's Online Services Security Incident Management team. As an advocate of a holistic approach to information security, Russ' website is holisticinfosec.org. Contact him at russ at holisticinfosec dot org.