PHPIDS: Attack my website, please!

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

PHPIDS
WEB APPLICATION SECURITY 2.0

Prerequisites

PHP 5.1.6 or above
LIBXML 2.6.21 or above
A PHP-based web application

Similar Projects

ModSecurity[1]
HTMLPurifier[2]

Want to know how to make me happier than a flea at a dog show? Probably not, but I'll tell you anyway. Merge two of my favorite disciplines in one freakishly good application. For the ultimate blend of web application security and intrusion detection, it pleases me to no end to introduce you to PHPIDS.

PHPIDS exists as a project to protect web applications while remaining transparent via monitoring and logging. "PHPIDS (PHP-Intrusion Detection System) is a simple to use, well-structured, fast, and state-of-the-art security layer for your PHP-based web application. PHPIDS doesn't strip, sanitize, or filter any malicious input; it simply recognizes when an attacker tries to break your site and reacts in exactly the way you want it to."[3]

I engaged Mario Heiderich, lead developer/co-founder of PHPIDS immediately upon choosing to focus on the project for *toolsmith*, and was treated to presentation slides, whitepapers, and lots of details about PHPIDS.

Mario's road map includes several minor tasks like eradicating false positives. The filter rules will likely receive some optimization, as will the converter module. Version 0.5 now makes PHPIDS WYSIWYG ready. This translates to a reduced and specialized rule set that has been created to be capable of exclusively detecting malformed HTML tags and event handlers, as well as other common attack patterns suitable for WYSIWYG editors. The developer/administrator will be able to flag certain fields to be checked with these rules via the `Config.ini` file to be discussed in "Working with PHPIDS."[4]

PHPIDS is often misunderstood as a web application firewall (WAF), which is also meant to block or manipulate requests. This is not the case and is not on the roadmap for the core system in the immediate future. Someday we may look forward to PHPI**P**S, but there is much to gain from PHPIDS in its current form.

PHPIDS rules have been tested by dozens of recognized security experts, ranging from SQL experts from Germany and the UK and JavaScript experts from the all over the world. As PHPIDS is a security system built on blacklisting, it was necessary to solicit as much help as possible, and the contributor community made the project what it is now.

Another special feature, not likely available in comparable tools, is the ability to detect attacks generically. Refer to the white paper, "PHPIDS - Monitoring attack surface activity,"[5] for more information on these techniques and indepth details. The "PHPIDS Centrifuge" section below will clarify generic detection as well.

I'm always drawn to open source projects, and as such, PHPIDS is released under the LGPL.[6] It goes without saying, it is entirely free and community-driven.

Working with PHPIDS

Why use PHPIDS? PHPIDS can be used by any web application written in PHP that meets the necessary requirements. PHPIDS provides detailed information about malicious traffic directed at the protected web application. When a site owner, developer, or system administrator wishes to know more about those types of requests without tediously parsing through log files, using PHPIDS will save a lot of time.[7]

PHPIDS works well with various PHP web apps like CakePHP, Symphony, ZF, or even WordPress (with a modified version of PHPIDS).

It includes extensive tagged regex rules, and you'll see reference to tags like XSS, SQLI, RCE, LFI, DT, CSRF, LDAP Injections, and DoS. Refer to the "Glossary" for any acronym you may not be familiar with.

1 http://www.modsecurity.org.

2 http://htmlpurifier.org.

3 http://php-ids.org.

4 Notes contributed to the article by Mario Heiderich.

5 PHPIDS: Monitoring attack surface activity, http://docs.google.com/View?docid=dd7x5smw_17g9cnx2cn.

6 http://www.gnu.org/licenses/lgpl.html.

7 PHPIDS: Monitoring attack surface activity, http://docs.google.com/View?docid=dd7x5smw_17g9cnx2cn.

The XML filter rule set is heavily tested and updated regularly. If you'd like to see examples of obfuscated and regular injections detected by the PHPIDS, refer to reference.[8]

PHPIDS works step by step, as follows:

1. User generated input coming in
2. Test to determine if the whole detection process is necessary
3. Conversion process
4. Detection process
5. Reporting and optional logging

The conversion process includes normalization of the user's inputs from several formats including JavaScript oct, hex, Unicode, and CharCode as well as UTF7, endless entities, obfuscation and concatenation patterns, evil characters, nullbytes, on and on, ad infinitum.[9]

Allow me to offer you a few key pointers to configuring PHPIDS to work with your PHP application. Most importantly, be sure `/lib/IDS/tmp` is writeable; in particular `phpids_log.txt`, which will likely be your primary logging mechanism.

You'll find core configuration settings in `/lib/IDS/Config/Config.ini`.

Note: If you choose to log to a database, you'll establish the connection string in `config.ini`. It is therefore critical that you don't place `config.ini` in a readable directory.

Path references are critical in `config.ini`; be sure you're clear on your directory and file hierarchy.

The most essential directory included in the PHPIDS installation package is *phpids-0.x/lib*; often installation can be as simple as placing said folder in your application or include folder. Because PHPIDS is configured to run immediately you can unpack it in your webroot and explore *phpids-0.x/docs/examples* to get a good sense of how things work. With it unpacked in your webroot you can browse included files easily to learn more. Also investigate the PHPIDS FAQ as a good starting point.

As you'll quickly learn in `config.ini`, PHPIDS results can be written to a flat file, a database, and email alerts.

To give you an idea of how you can use PHPIDS for yourself, I built a demo implementation on my website for you to go nuts on. What's that, you say? Yes indeed, attack me please!

Figure 1 – String

`http://attackmeplease.holisticinfosec.org` is at the ready for you to wear out with your favorite attack strings. It's built from version 0.5, so all the new functionality is available. There's no data behind it so you shouldn't be able to lift any nuggets, but you will see the impact rating PHPIDS applies to your attack and can review the possible reported data and logs. Properly configured, PHPIDS will capture attacks against any inputs on your application; this demo is a very simplified example.

The resulting report view you see after submitting an attack string is what can be provided as output in the above mentioned formats, or more, as you can customize reporting to your liking with relative ease.

Allow me to provide you a quick example.

Consider the string `qwerty"><'<>` (see Figure 1). This is a useful test for SQLi issues, so it should result in a response from PHPIDS, yes? I've submitted it to my demo below.

The result (see Figure 2) is a comprehensive data set including total impact, affected tags, and detailed descriptions of each detection, including the expected:

```
Description: Detects classic SQL injection
probings 2/2 | Tags: sqli, id, lfi.
```

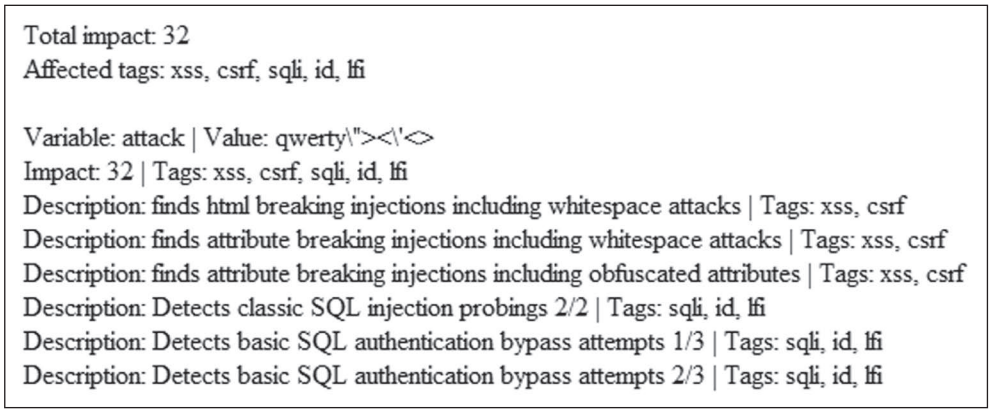

Figure 2 – Result

On my demo, I provide the option to see the output as written to `phpids_log.txt`.

The attack submitted above looks like this when written out to the log or your database.

```
"71.35.118.165",2008-06-07T18:07:23-04:00,32,"xss csrf
sqli id lfi","attack=qwerty%5C%22%3E%3C%5C%27%3C%3E"
,"%2Fattackme.php"
```

I think you'll agree that detection like that offered by PHPIDS can bring great benefit to site administrators and developers alike.

8 https://trac.php-ids.org/browser/trunk/tests/IDS/MonitorTest.php#L884.

9 Mario Heiderich, Generic Attack Detection: Avoiding blacklisting traps with the PHPIDS.

PHPIDS Centrifuge

Generic attack detection is provided by the Centrifuge module. Blacklisting alone is useless, given the unlimited ways to obfuscate payloads. PHPIDS assumes that special characters characterize an attack. That's web app security humor ;-), in case you missed it. With this assumption at its core, almost all real-world attacks, JS worms, SQL injection exploits and others are detected by the PHPIDS Centrifuge. Does this create opportunity for false positives? Of course, but like any good IDS, it can be trained. To simplify the concept, the Centrifuge normalizes and weighs standard programming language elements.

Early in development, when PHPIDS was based purely on blacklists, the developers realized that the project would generate bloated rules, decrease performance, and generate false positives. The first attempt to correct that was the creation of the converter class, which at first performed basic normalizations and is now capable of converting almost any reasonable format and encoding to what it should be before hitting the rules. Through a number of versions the conversions were enough to keep the rules slim but, in the face of further obfuscated JS and SQLi, the developers planned on something new. Mario's idea for the centrifuge came to him, ironically, after much partying, but when developing the code some days later it worked out really well. With the help of a huge quantity of attack strings and vectors (500+) they were able to fine tune the algorithm for great results. Later, they added the second component of the centrifuge, the ratio calculation, and currently, plans for a third tier to judge input are underway.

Again, conceptually, the premise of the first centrifuge module was that any programming language uses similar patterns such as operators, variables, parenthesis, delimiters, etc. Via normalization and unification of the results, which after additional intermediary steps, results in a 90% likely match to a certain pattern. If this pattern appears, PHPIDS treats the string as an attack.[10]

PHPIDS 0.5 and future releases

As this column was being written the PHPIDS project announced they are skipping 0.4.8 and are going right for PHPIDS 0.5. By the time you read this, PHP 0.5.x or later will be available, and though I largely tested 0.4.7, there are significant improvements for you in 0.5. Among the highlights to expect:

1. "PHPIDS 0.5 uses the HTMLPurifier[11] to compare the original user input with the purified one to determine the differences and analyze them with the rules and the centrifuge. You can, of course, choose freely which fields you want to monitor the traditional way and which are allowed to contain valid HTML – just have a look at the packaged `Config.ini` to see how it works."[12]
2. False positives have been removed as usual and some very interesting rule and Centrifuge circumventions were fixed. Those vectors can be found at sla.ckers.org.[13]
3. The converter was optimized too, and several problems with the conversion method order have been fixed.[14]

Benefits and drawbacks

This project is young and you may find it takes some tweaking to make it work with your app, but the community is strong, as indicated on the forum, where you'll likely find the answers you need.

The information you'll acquire resulting from your implementation of PHPIDS will far outweigh any perceived challenges. This is simply an invaluable project.

In conclusion

You'll find PHPIDS in use on dozens of real, high traffic sites including neu.de, shoppero.com, astalavista.com, ormigo.com, doccheck.com, and sevenload.de.

Suggestions and input are always welcome. You can contact the developers via the project Google Group or forum, and always via email and PHPIDS.[15] I hope you find this project as valuable as I have.

Cheers, until next month…

Acknowledgments

Mario Heiderich, lead developer/co-founder of PHPIDS, for his extensive contributions to this article.

Glossary

CSRF – cross-site request forgery
DoS – denial of service
DT – directory traversal
ID – information disclosure
LDAP Injections – lightweight directory access protocol injection
LFI – local file inclusion
RFE – remote file execution
SQLI – SQL injection
XSS – cross-site scripting

About the Author

Russ McRee, GCIH, GCFA, CISSP, is a security analyst working in the Seattle area. As an advocate of a holistic approach to information security, Russ' website is holisticinfosec.org. Contact him at russ@holisticinfosec.org.

10 Notes contributed to the article by Mario Heiderich.

11 http://htmlpurifier.org.

12 http://php-ids.org/2008/06/07/phpids-05-has-landed.

13 http://sla.ckers.org/forum/read.php?12,8085,22761,page=20.

14 Mario Heiderich, mailing list announcement.

15 http://php-ids.org/contact.