www.phparch.com

September 2015
VOLUME 14 - ISSUE 9

Security Boot Camp

Basic Intrusion Detection with Expose

Is Your Website Secure from Hackers?

Keep Your Passwords Hashed and Salted

Leveling Up: DeLoreans, Data, and Hacking Sites

php[architect]

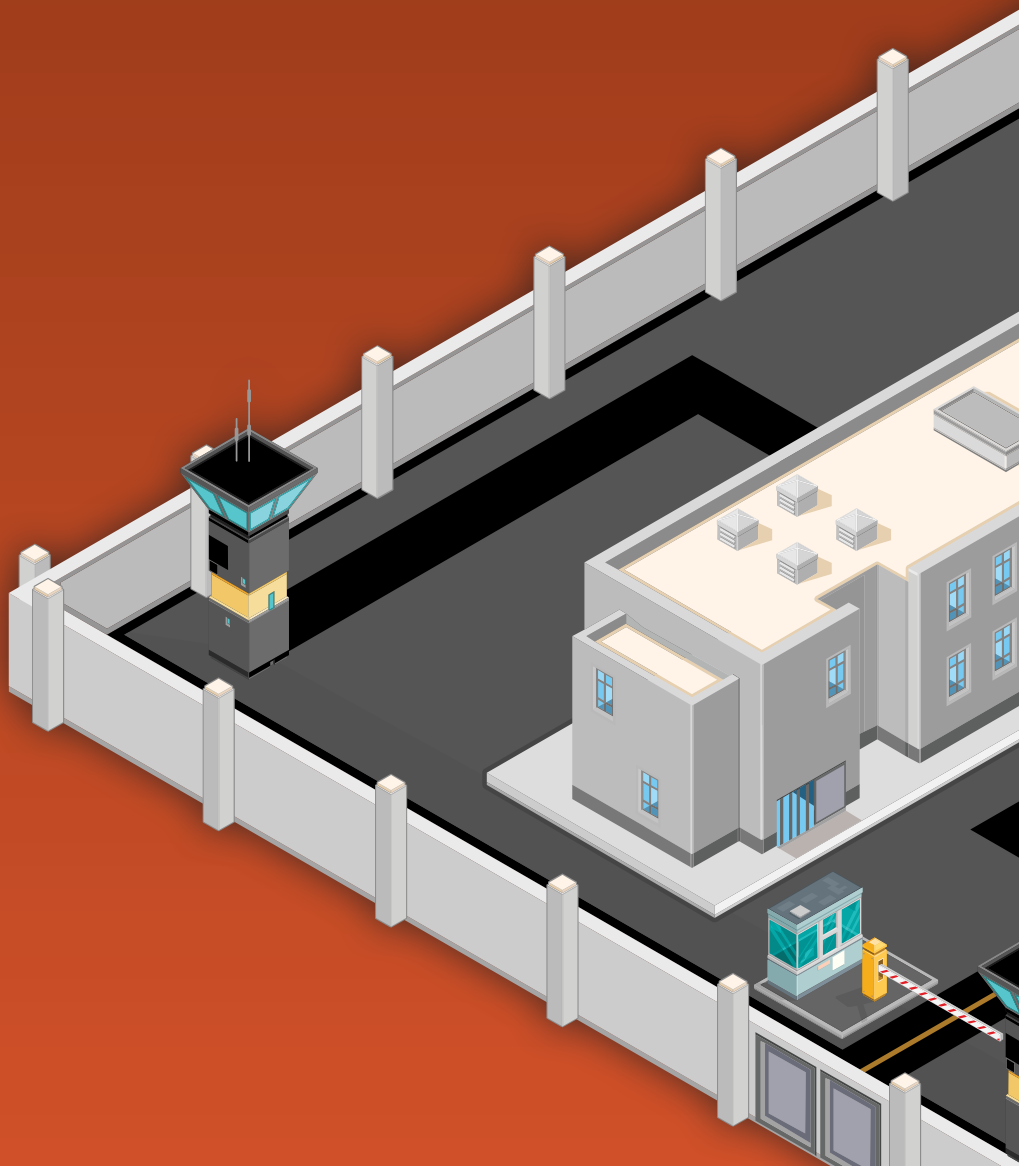
ALSO INSIDE

FREE Article!

Basic Intrusion Detection with Expose

Greg Wilson

The recent high-profile hacks to major retailers and governments reveal that being hacked is not an if—it is a when. It is time for you to go beyond simple input filtering and into the world of Intrusion Detection Systems. Let's start preventing the pollution.

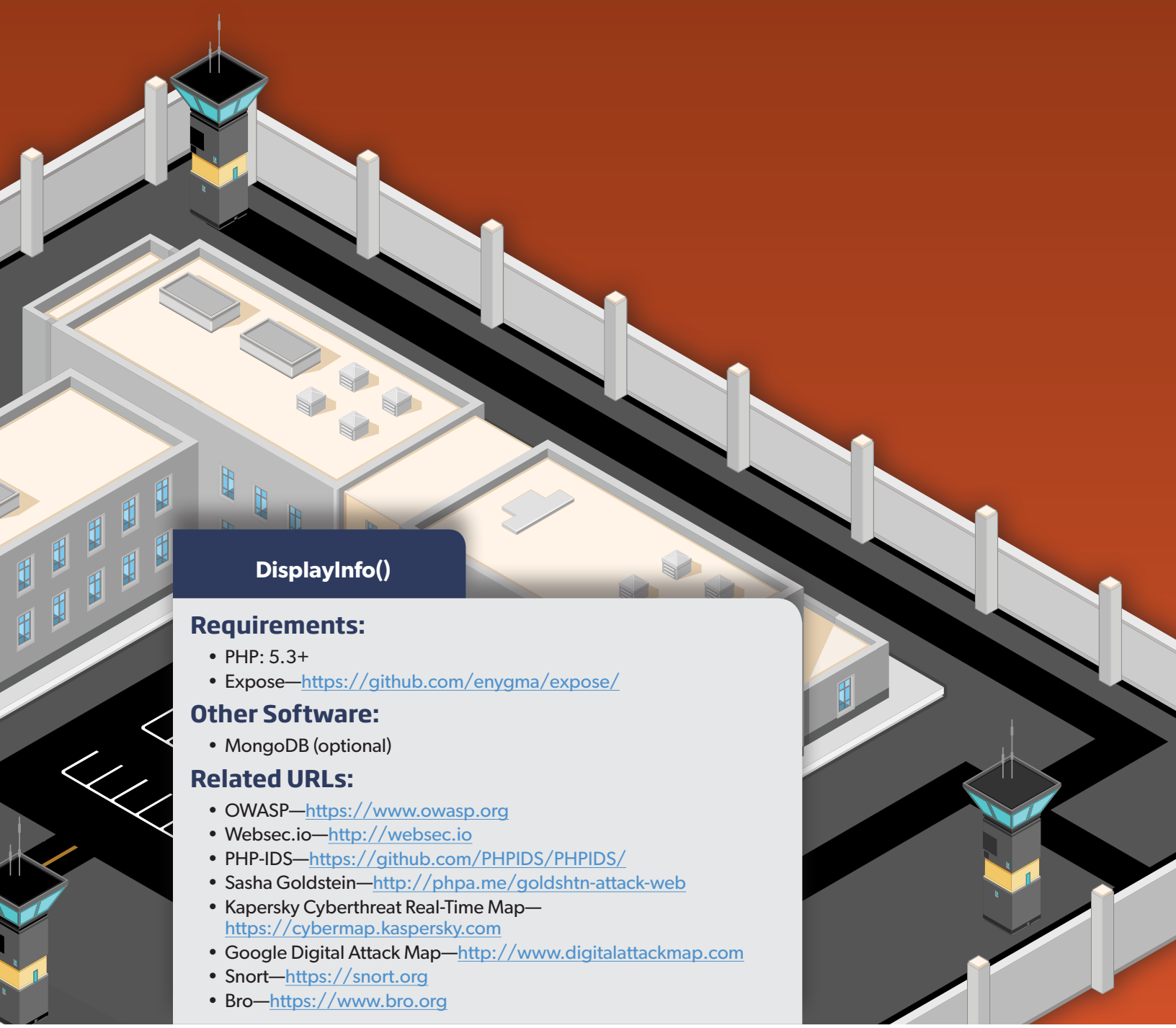


What Is an IDS and Why You Should Use One

An Intrusion Detection System (IDS) at its simplest level monitors for malicious activities or policy violations. They are designed to identify bad behavior before a foothold can be established on your system.

Web servers, on the other hand, are designed to be friendly and amiable. You can ask them a question and expect an answer in return. They do their best to fulfill your request. They do this job so well that they even fulfill your bad request. In fact, they can even fulfill tens of thousands of bad requests, sometimes in a massive barrage from a script kiddie, other times slowly over the span of days.

If you are routinely reviewing your system logs (and you should be), you may notice these hits from all over the globe. Your web server will not, by default, provide you with more than the basic connection information. As such, they cannot tell you if the data being sent back and forth contain malicious or benign content.



DisplayInfo()

Requirements:

- PHP: 5.3+
- Expose—https://github.com/enygma/expose/

Other Software:

- MongoDB (optional)

Related URLs:

- OWASP—https://www.owasp.org
- Websec.io—http://websec.io
- PHP-IDS—https://github.com/PHPIDS/PHPIDS/
- Sasha Goldstein—http://phpa.me/goldshtn-attack-web
- Kapersky Cyberthreat Real-Time Map—https://cybermap.kaspersky.com
- Google Digital Attack Map—http://www.digitalattackmap.com
- Snort—https://snort.org
- Bro—https://www.bro.org

```
10.1.1.1 "GET /webmanage/fckeditor/asp/connector.asp HTTP/1.1"
10.1.1.1 "GET /admin/fckeditor/asp/connector.asp HTTP/1.1"
10.1.1.1 "GET /editor/fckeditor/asp/connector.asp HTTP/1.1"
```

As a PHP system, we don't even have `.asp` files, and these are most likely from a routine script kiddie looking for a known vulnerability. We know something is going on here, but PHP would be immune, leading one to easily dismiss it. With "normal" traffic, however, we sometimes can't tell.

```
10.2.3.4 "GET / HTTP/1.1" 200 5194
10.2.3.4 "GET /js/app.js HTTP/1.1" 304 -
10.2.3.4 "POST /login/ HTTP/1.1" 302 18
10.2.3.4 "GET /user/3 HTTP/1.1" 200 3135
10.2.3.4 "POST / HTTP/1.1" 302 127749
```

Perhaps 10.2.3.4 is a valid user. From the logging, it appears that he entered the application successfully, but what did he post to our base route, and why was so much data returned? Simple Apache logs won't tell us.

The Threat

There are currently quite a few very pretty online visualizations of the global threat environment, from Kaspersky's Cyberthreat Real-Time Map to Google's Digital Attack Map, see Related URLs.

External attackers will try to penetrate the standard outer defenses of your system. Given enough time, they will likely bypass your firewall and go after your application. Who are these people? They range from script kiddies and state actors to jilted lovers and former employees.

Internal attackers possibly pose a greater risk. They may already have login rights. They might already know the innards of the application. Curious employees might copy-paste something they found on the Internet just to see if it will do anything. Inept employees might stumble upon a bug. Disgruntled employees may want to sell off information to crash the whole accounting database.

Types of IDS

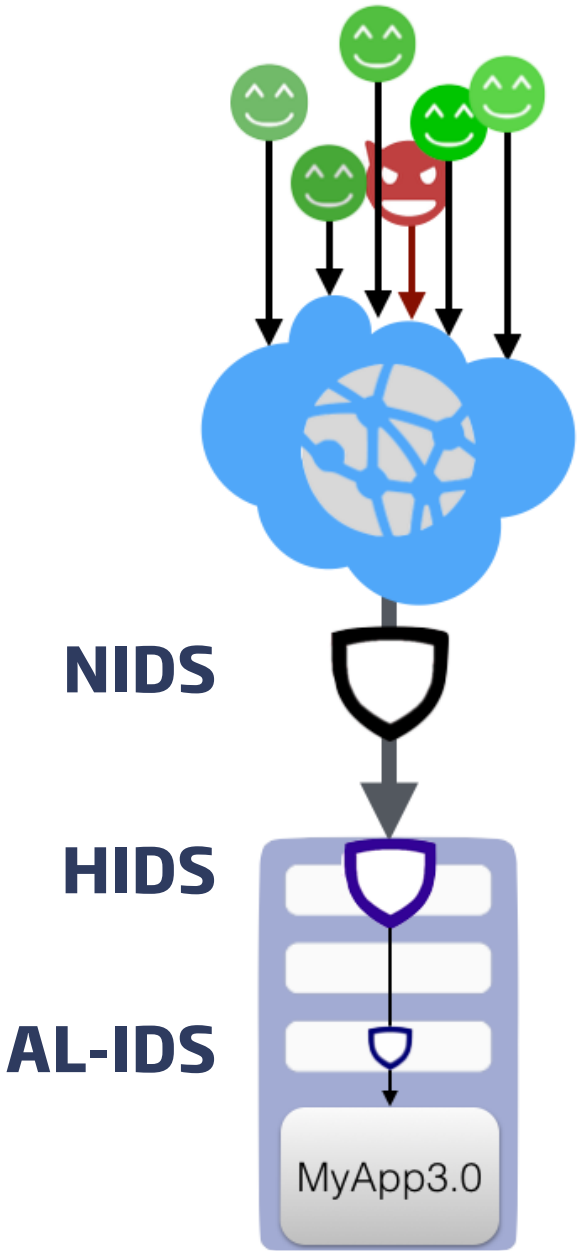
There are three main positions of Intrusion Detection Systems: Network (NIDS) , Host (HIDS), and Application Layer (AL-IDS).

NIDS sit between all of your server and the pipe to the outside world. Popular OS examples include Bro and Snort, see Related URLs.

A HIDS would monitor the packets going in and out of your particular server. It will often maintain a known good state of the machine, alerting if key files are modified. Other capabilities often include RAM and log file monitoring. Open-source examples include OSSEC and Samhain, see Related URLs.

AL-IDS sit on top of your application, attempting to thwart abuse. In the case of Expose, it will monitor REQUEST data for bad input. It could additionally be set up to monitor output back to the user.

IDS Positions **FIGURE 1**



Why Use an IDS?

- **Defense in depth.** Redundancy is good. The more layers an attacker has to get through, the more likely they will either fail or be slowed down enough that countermeasures can be put into place. If one portion of your perimeter fails and is bypassed, the other layers remain in place to shield the core.
- **Don't trust the source code.** The longer you are a developer, the greater the chance of inheriting a rat's nest of spaghetti code from your predecessor. A full security audit will take time or money—and probably both. Additionally, all code may contain zero-day vulnerabilities, leaving you exposed until you patch.
- **Don't trust the plugins.** Wordpress and Drupal have gone through tons of security audits, and when bugs are found, they are patched quite quickly. Some of their plugins, however, may not get patched quickly, if at all.
- **Don't trust the database.** If a database exploit is developed, there is a reasonable chance that the corresponding drivers will allow that exploit to be passed directly through the application.

- **Don't trust the drivers.** The database driver developers do an amazing job making sure that we can securely access our data. Prepared queries greatly reduce the application attack surface for PHP. Sometimes, though, the boss wants FancySkyMallDatabase2018(c), and the drivers are still in alpha.
- **Don't trust yourself.** I make mistakes all the time. Sometimes I am in a rush; other times I suffer from chronic caffeine deficiency syndrome. Sometimes there isn't someone else to audit my code. A healthy dose of humility in our abilities is a good thing.

None of this is to denigrate the countless hours that other developers have put into their craft. It is only to point out that we are all flawed, and we write flawed code now and then.

Excuses

WhiteHat Security reported that "86% of all websites had at least one serious vulnerability during 2012." Whether that is an exaggeration or an understatement does not matter. We should know the code for which we are responsible.

Excuses abound before and after security breaches, and very often, a few heads roll. Common poor security excuses include:

1. My framework validates everything.
2. We have nothing a hacker would want.
3. Our systems are internal and therefore not at risk.
4. It would cost too much.
5. We don't have time.

Each of these excuses makes explicit assumptions.

1. The frameworks validation routines will continue to be perfect into the future. Bad people won't find workarounds.
2. We won't ever have something an attacker might want. We can't be a stepping stone to attacking others.
3. I trust my team. There is no such thing as corporate espionage. Edward Snowden was an anomaly.
4. We have explored every option available.
5. It will be a long and laborious process.

These are not good assumptions.

Hopefully, by now, you know how to validate input and filter output, especially from untrusted sources. If you don't, I recommend checking out the #validation articles at Websec.io.

> *"But I validate all my input," said no one, ever.*

Alas, escaping input these days is not enough. Sasha Goldshtein, @goldshtn, has reported seventy different ways of encoding just the greater-than > symbol. This means that there are billions of ways to encode even the simplest of exploit vectors.

Advantages, Limitations, and Disadvantages of Expose

Before we get into the nitty-gritty of installing Expose, you need to keep in mind some of the pros and cons of Expose.

Advantages

1. Defense in Depth

A firewall is not enough these days. The attacks are targeting all layers of your system. An IDS can help protect your application layer by providing one more block before they get to your code.

2. Protection from Known Vectors in Multiple Categories

These include:

1. Cross-site scripting (XSS)
2. SQL injection
3. Header injection
4. Directory traversal
5. Remote file execution
6. Local file inclusion

3. Block Script Kiddies

Script kiddies want an easy win. Rarely are they targeting just you, but you don't want to be hit by their shotgun blast. With current toolsets like Metasploit, it is easy to hit any system in the world with every published vulnerability. Scanning 24 hours a day, they leave a trail of ugly Apache logs in their wake.

4. Cover over Framework Holes

No web framework is perfect. Flaws will be eventually found and exploited. An IDS will provide one more layer that an attacker will have to get through.

5. 0-day Lag

An IDS may give you that little extra time that you need to patch your server when a 0-day exploit is revealed.

6. Copious Logging

Most IDSs can alert you via email depending on the thresholds you have set. Instead of information languishing in an access log somewhere, you can have near-instant notification of an attack occurring. The faster you know, the faster you can respond.

7. Cloud & Budget Friendly

You might not have the in-house budget or expertise to run Snort, FireEye, and other expensive systems. As a application layer IDS, Expose can be dropped in place without needing to involve system operations (if you have them).

8. Detect Malicious Users

Once a person has logged in, they have access to the internal attack surface of your application. Make sure they don't have an enhanced ability to post malicious content by testing their POSTs with an IDS.

9. Works over HTTPS

Network-based IDS often do not have access to the certificates to inspect https traffic. Because Expose is an application layer IDS, it receives the data after webserver https decryption.

10. Easy to Install

Using Composer, you can be up and running in ten minutes.

11. Free as in Freedom

Expose is not only free as in no-cost, but it is free in the sense of freedom. You can be sure that any logging that takes place remains on your system. Unlike some of the big players, your clients' data is not broadcast to an external company to boost their research and stats.

Limitations

No system is perfect. IDSs do have a few weak points.

1. Advanced Persistent Threat (APT) Detection

Expose does a great job discovering well-known malicious and suspicious content. Unfortunately, it cannot (as of this writing) track the extremely patient hacker who only probes your system once a week. Nation state and professional hackers have the money and resources to eventually get past an IDS.

2. Rules-based Signature Detection

Life is full of rules. And rules are meant to be obeyed. Well, we know that doesn't exactly happen. Alas, the bad guys are smart and keep finding ways to bend and break the rules. Expose's rule set is large, and it covers many known and possible attack vectors. If a novel technique is developed, the IDS will not be able to detect it.

> *No one expects the Spanish Inquisition!*

3. Upstream Bugs

All IDS systems rely on an underlying technology stack and hence are vulnerable when that stack has flaws. Expose may not be able to detect and block underlying PHP bugs or exploits. Likewise, attacks against flaws in the webserver itself may go undetected (e.g., Heartbleed).

Disadvantages

I'll be honest. There are some disadvantages of using an IDS. The three largest are the performance hit, privacy, and a false sense of security.

1. Performance Hit

An IDS will inspect every request you send it. This takes additional CPU and memory, resulting in performance degradation. It is something you will need to take into account when implementing. Depending on how large your user base is, you may want to spool up another server to compensate.

2. Privacy

Because the IDS inspects the requests, it also knows every secret bit of data your users are submitting to the application. If you are logging the content of the bad requests, that private data may show up in the logs. You need to be very careful what you log, where you log it, and who has access to those logs.

3. Noise

You will receive false positives. The more data, the more noise. The more noise, the more logs. The more logs, the less likely you will review the logs for security policy violations.

4. Possible False Sense of Security

We put locks on our doors as a medium barrier of entry, but that doesn't mean a robber can't get in through a window. There are numerous other vectors that can be exploited: poor credential hashing or encryption, exposed session IDs, and Insecure Direct Object References to name a few (see OWASP site). Expose will only provide protection from injection and XSS attacks, not faulty application logic. An IDS is only one layer in our security profile and shouldn't be considered as the golden ticket to a life of ease.

Expose Installation Run Through

Expose is the new hotness when it comes to PHP intrusion detection. Although it shares the same signature ruleset as PHPIDS, it is a clean, ground-up rewrite. We are going to go through the basics of Expose installation. As of this writing, there are no plugins yet available for the major frameworks and applications. If you need something yesterday, have a look in the ending notes to see if your library has an old PHPIDS plugin.

Recently, I have been using the Flight micro-framework http://flightphp.com, but to keep things dead simple, we are going to skip frameworks altogether.

composer.json

Following current conventions, we'll be using Composer to bring in the dependencies. Below is what our `composer.json` file looks like.

```
{
    "require":{
        "enygma/expose":"2.*"
    }
}
```

With that in place, and after running the Composer install, all necessary dependencies will be in the newly created vendor directory (Figure 2).

index.php

Create an `index.php` file in your root directory with code in Listing 1.

Composer Install **FIGURE 2**

```
$ composer install
Loading composer repositories with package information
Installing dependencies (including require-dev)
  - Installing symfony/console (v2.5.7)
    Loading from cache

  - Installing psr/log (1.0.0)
    Loading from cache

  - Installing monolog/monolog (1.11.0)
    Loading from cache

  - Installing twig/twig (v1.13.2)
    Loading from cache

  - Installing enygma/expose (2.1)
    Loading from cache
```

LISTING 1

```
01. <?php
02. require 'vendor/autoload.php';
03. require 'vendor/enygma/expose/tests/MockLogger.php';
04.
05. ini_set('display_errors', 0);
06.
07. // load in the default signatures file, based upon PHPIDS
08. $filters = new \Expose\FilterCollection();
09. $filters->load();
10.
11. // register a PSR-3 compatible logger
12. $logger = new \Expose\MockLogger();
13.
14. // build the main processor
15. $manager = new \Expose\Manager($filters, $logger);
16.
17. // feed expose with the gooey bits
18. $manager->run(array(
19.     'GET' => $_GET,
20.     'POST' => $_POST,
21.     'COOKIE' => $_COOKIE
22. ));
23.
24. // return how bad the input was
25. $impact = $manager->getImpact();
26. ?>
27. <form action="/" method="POST">
28.     <label for="badstuff">Bad Stuff:</label>
29.     <input name="badstuff"/>
30. </form>
31. <hr>
32. <p>
33.     Results: <input value="<?=$_POST['badstuff']?>"/>
34. </p>
35. <p>
36.     Impact: <?=$impact?>
37. </p>
```

You should notice a deliberate flaw on lines 27. Not only are we not validating our input, we are not filtering our output. **DO NOT DO THIS ON A PRODUCTION BOX!** It is only here to show how Expose detects injection and XSS attempts.

When you ran the composer install, you noticed a couple of extra packages come down, as well, including monolog. Expose requires that a logger be defined, so for now, we will use a built-in one from its own test suite. You could easily pop in your own PSR-3 compatible logger, instead.

Start a Webserver

The easiest way to test is to start up PHP's built-in server, from the directory where `index.php` is located:

```
php -S localhost:8000 -t .
```

Now, navigate to http://localhost:8000 in your favorite browser. Our interface is not the prettiest in the world, but it will suffice.

Hack the Site

Probably the most trivial test would be to break out of the HTML. Try submitting the following:

```
">Vulnerable<a="
```

You will see that our text breaks out of the input box onto the page, but Expose detects this attempt and rates it at an impact level 11. There are several variations on this type of attack.

```
">Vulnerable<script>alert('Yo!')</script>
```

Some browsers will prevent this snippet from harming you, but others won't. Expose notices the attempt to inject a script and increases the impact to 27.

How about SQLi?

```
union select from
```

A standard attempt to gain more data from a database table, which Expose gives us an impact of 20.

Let's Get Nasty

Although a simple typo might occur (if you are not using a WYSIWYG editor), yielding a moderate impact level, malicious attempts will almost always generate an impact level above 12.

LISTING 2

```
01. <?php
02. set_time_limit(0);
03. $shell = 'unset HISTFILE; unset HISTSIZE; uname -a; w;'
04.        . ' id; /bin/sh -i';
05. if (function_exists('pcntl_fork')) {
06.     $pid = pcntl_fork();
07.     if ($pid == -1) {
08.         printit("ERROR: Can't fork"); exit(1);
09.     }
10.     if ($pid) {
11.         exit(0);
12.     }
13.     if (posix_setsid() == -1) {
14.         printit("Error: Can't setsid()"); exit(1);
15.     }
16. } else {
17.     printit("WARNING: Failed to daemonise.");
18. }
```

Listing 2 is a cleaned-up snippet of an attempt upon one of my systems. This is only the beginning of the code, but you can expect to see similar items in your logs if you run Expose long enough.

As you can see, the code attempts to see if it can fork itself so that it can run, regardless of the status of the web server. The attacker would then open a high port on your box and send themselves an email to let them know they had access.

Thankfully, Expose easily catches such shenanigans with an impact of 37.

Logging, Alerting, and Thresholds

Speaking of these impact levels, none of this will do you any good unless you are tracking the results and acting on them.

Logging

The above examples used the MockLogger included in the Expose test suite. You will want to replace that with your own logger. Have a good look at all of the suggestions monolog makes to see if one makes sense for your needs.

Alerting

In addition to your logger gathering data, Expose can notify you of the events via email.

```
$notify = new \Expose\Notify\Email();
$notify->setToAddress('watcher@example.com');
$notify->setFromAddress('expose@example.com');
$manager->setNotify($notify);
$manager->run($data, false, true);
```

You will need to make sure the third param of `run` is set to true for the notification system to work.

If you should happen to create a `Notify\Twitter`, please let me know!

Thresholds

Sometimes the input is obviously malicious. Sometimes you don't really care if they tried every trick in the kitchen sink. Information overload can easily contribute to security breaches. Expose will pick up a certain amount of noise in the traffic sees. If you:

- Don't need to see the full extent of what they were up to
- Don't want be alerted every time someone includes a ' in their post
- Want to save some CPU

What you can do is set a threshold:

```
$manager->setThreshold($int);
```

Now, any input that generates an impact level less than `$int` will be silently logged. But what should that magical threshold be? I have found out that because most attacks will need to include at least one technique, the impact level will be at least 12. After watching your logs for a few days, you will have a better idea for your system's threshold.

Next Steps

Expose is installed. You are monitoring your logs. What comes next?

Keeping Up-to-Date

A little while ago, Expose's progenitor, PHPIDS, was shown to have a flaw, whereby it was possible to craft special injection code that bypassed the detection routines. You will want to keep an eye out for updates to the signature ruleset. When instances like this are found, a new rule will usually be added to the default PHPIDS signature set. You will want to keep your eyes out to make sure Expose's ruleset gets updated, as well.

Code Audits

Because Expose will not completely defend you from bad code, ensure that you perform regular security audits on your code. One service that might help you out in that process is CodeClimate (http://codeclimate.com), which is free for open-source projects. If you handle highly sensitive or personal data, ask around for professional code auditing services.

Deeper Defense

If you can access more of your network or are in good graces with your netops staff, you can add additional perimeter fencing. You may want to consider adding Snort or Bro for network intrusion detection and prevention. The lower-level analysis they perform will make it that much harder for a malicious actor to "pop" or compromise your application.

Conclusion

Whether you like it or not, as a developer or manager, security is your job and responsibility. Using an IDS adds one more defensive layer onto your system and places one more feather in your cap. It is not a bullet-proof solution, but in the constantly escalating war for control of the Internet, it is a great option to reduce risk and increase your security profile.

GREG WILSON has been building PHP applications since 1998, from university labs to the two largest HIV/AIDS clinical trial networks in the world. Currently he is a Senior Security Software Engineer at Redport Information Assurance. He enjoys triathlons, astrophotography, and running his family's Minecraft server. Yes, this photo is of him and not his evil twin.

Twitter: @Awnage

Want more articles like this one?

Keep your skills current and stay on top of the latest PHP news and best practices by reading each new issue of php[architect], jam-packed with articles.

Learn more every month about frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

php[architect]

magazine

books

conferences

training

www.phparch.com

Get the complete issue for only $6!

We also offer digital and print+digital subscriptions starting at $49/year.