www.phparch.com

May 2016
VOLUME 15 - ISSUE 5

FREE Article!

php[architect]

Building Laravel Shift

Mastering OAuth 2.0

Learn from the Enemy: Securing Your Web Services, Part One

An Introduction to Doctrine ORM Best Practices

Full Speed Ahead

ALSO INSIDE

Community Corner:
Gratitude

Security Corner:
Keeping Credentials Safe

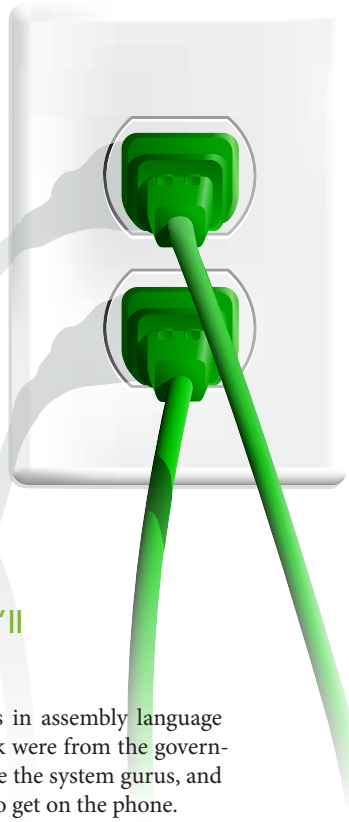
Education Station:
Directing Requests with FastRoute

Leveling Up:
You Had One Job

finally{}:
Hindsight & Planning

Learn from the Enemy: Securing Your Web Services, Part One

Edward Barnard



Knowing how to secure your web**site** does not translate into knowing how to secure your web **service**. Your website is friendly to humans. You can fend off attacks with CAPTCHA and other ways of detecting and rejecting automated traffic. Your web services, by contrast, are to be consumed by non-humans. If you have a flagship mobile app, it's not a human. It's an app! You therefore need to take a far different approach to securing your web services. I'll show you my experiences and the attitude you need to protect your own.

It Happens

On September 14, 2015, Business Wire announced[1]:

> *Kim Kardashian West, Khloé Kardashian, Kendall Jenner and Kylie Jenner today launched new Personal Media Apps – and websites – allowing them to connect more directly with their fans and provide a unique and personal look into their lives.*

Two days later, on September 16, 2015, TechCrunch published an article *Kardashian Website Security Issue Exposes Names, Email of Over Half a Million Subscribers, Payment Info Safe*[2]. The article describes the discoveries of 19-year-old software developer Alaxic Smith.

Alaxic Smith reported that the Kardashian app had a JavaScript file providing client (app) access to the website API. So long as he was logged into the website himself, he could get the web services to respond with information on the 663,270 people who had signed up for the site. The other sisters' sites behaved identically.

It happens!

Getting the Attitude

Fifteen years ago I wrote a series of articles, *How to Hack a Paysite: What the Good Guys Need to Know*, after spending time among hackers and crackers. These days I wouldn't recommend the "dark web" to anyone, not with the rise of organized crime online. But back then, I was rated "Master Exploiter" by my putative peers, was allowed in the more private "Sploiters" forums, and was made an admin of one of the larger boards.

Publishing those articles made certain people unhappy. On the other hand, a couple of billing companies changed their code as a result, and thanked me.

My own security interest stems from November 1988, when the Morris Worm[3] was released into the wild. I was teaching Cray Supercomputer operating system internals in assembly language and octal. Several of my students that week were from the government labs being hit by the worm. They were the system gurus, and as such they kept being called out of class to get on the phone.

I had virtually a ringside seat to the breaking of the Internet. It literally was torn apart, with backbones isolated from each other for a few days to stomp the worm.

Robert Morris and his Worm taught us that relatively minor mischief can cause major havoc. Learn from the experience.

Learn from the Enemy

There's good PHP security information available online. See *Additional Reading* at the end of this article. There's also information out there that's not so good. That's not necessarily the fault of the author. "Security" is a continuous battle. Techniques and needs evolve.

My best advice concerning web services comes from *Ender's Game* by Orson Scott Card:

> *You will be about to lose, Ender, but you will win. You will learn to defeat the enemy. He will teach you how.*

Your first step in securing your web services is understanding your adversarial relationship. No doubt your website is usable, friendly, inviting. That's great.

That is *not* how to view your web services. Your web services need to be *prickly and distrusting*. Don't you want to be friendly and inviting? No, you do *not!* This is the fundamental difference between your human-visible website and your invisible web services.

Guide your humans in successfully navigating your website. When errors occur—and they will—provide your humans the information they need to complete their task.

Your web services, by contrast, are aimed at computer software that already know precisely how to use your web services API. Client software does not need your guidance. Extra information would just get in the way.

Instead, remember that there is one *other* consumer of your web

1 *Busines Wire: The Kardashian/Jenner Sisters Launch Individual Personal Media Apps:* http://phpa.me/kardashian-apps
2 *TechCrunch: Kardashian Website Security Issue:* http://wp.me/p1FaB8-54RC
3 *Morris Worm:* https://en.wikipedia.org/wiki/Morris_worm

services. Your enemy, whoever that might be, is *also* consuming your web services. Do you want to *help* your enemy hack you? Of course not!

As we'll see, there may be no way to be sure if any given web service request is legitimate or an attack from the enemy. You *must* be distrusting. You *must* assume that your web services operate in a hostile environment. That's because they *do!*

Suppose the web service request is partly correct. For example, it's a properly formed request except that one parameter is out of range. Do you provide help? No. When it comes to web services, your role is to be *prickly and distrusting.* Your role is to assume you have an attacker who has almost figured you out.

In *Ender's Game,* Ender explains:

> *I've been through a lot of fights in my life, sometimes games, sometimes—not games. Every time, I've won because I could understand the way my enemy thought. From what they did. I could tell what they thought I was doing, how they wanted the battle to take shape. And I played off of that. I'm very good at that. Understanding how other people think.*

Are we planning to play games with our attackers? Absolutely not. We don't have time for that nonsense! Our strategy is to tip the odds in our favor. Once the effort to attack far outweighs the possible reward, we are far less likely to come under attack at all.

What might *your* attackers' motivations be? Cash, glory, free content downloads? Draw from your experience to date and form your own threat analysis. The attack *method* can be different via web services, but the attack *motivations* will likely remain the same.

Threat Modeling: *What motivates your attacker? What might he, she, or they be after? What might be their intrusion vector? This is threat modeling. See Threat Modeling: Designing for Security by Adam Shostack under Additional Reading.*

Web Services are Different

We've all seen "brute force" attacks before. Someone hits your website login page many times with different user name & password combinations. Alarms go off, you block a few dozen IP addresses, and it's "game over" for your attacker of the moment.

We all use CAPTCHA images during brute-force attacks to distinguish and shut down the "bots." Everything works; we've been through this before.

Do you see the attitude here? It's just another brute force attack, no big deal. Detect the bot and shut it down. It's hardly worth mentioning.

Therein lies the problem.

We need to back up and take a moment to think about how we got here. Bear with me; this is the fundamental change in thinking you need to understand.

Web Services Mirror the Site

Web services are a good thing. If you have created a well-structured RESTful API powered by PHP, you have a good thing. You have an infinitely expressive portal into your server, your business operations, your reason for existence. You'd best assume that your enemy understands this as well as you do.

We created an app for Android and iOS mobile devices. Members can now use our site through either their web browser or via the native app.

We created web services which allow our app to have the same functionality as our browser-based website. Our web services are *only* consumed by our own app. We don't publish a public API. Since nobody knows where to find our web service end points, that should make us relatively safe.

No, it does not! Learn from your enemy.

Suppose, for example, you notice that a single IP address has an

excessive number of failed login attempts. By capturing the transactions (web service requests and responses for that IP address), you realize that each request has a *different* user name / password combination. The login requests are formatted correctly.

What has our enemy taught us? That he or she has our API figured out. Our enemy is able to counterfeit our web services requests. We do not know *how* our enemy figured this out. Our enemy has taught us that our web services protocol is known, inside and out.

Our Focus

There's lots of good information out there about website security and web services security. By all means, do your homework and practice the fundamentals!

The problem is that your enemy won't feel constrained to follow *your* rules. You need to work with what your enemy *does* rather than stay within the security rules. The experts will provide you help, but only the enemy can teach you how to defeat the enemy.

Security is a continuous give-and-take. Learning is continuous; you can't "do it" and be done.

This article doesn't cover the fundamentals. Yes, the fundamentals are important, and they must be your starting point, but you won't find them here. Instead we focus on learning from the attacks, from our enemy.

An App is a Bot

Here is where the first of the problems comes in. Here is where we need to begin changing our thinking.

Much of your website security is based on telling the difference between a human attacking us and a bot attacking us.

You need to recognize *why* you might come under attack. Are you simply a target of opportunity? And if so, opportunity for what? Are you a high-profile site? Might someone attack for the glory of beating you? Can someone gain free downloads?

With a bot, it's often a series of repeated attempts. For example, if someone is running a password list, we see thousands (or millions) of login attempts. If it's blatant, we simply block it. If it's merely questionable, we use something such as a CAPTCHA to distinguish between human and bot.

By "bot," short for "robot," I mean any sort of automated mechanism for interacting with our website.

The fundamental problem is that your own app is a bot. It's a program, not a human, and by definition a bot. More to the point, any of your "are you a human?" tests will fail. It's not a human.

You may well need to "white list" your web services. That is, anything you have in place for bot detection, brute-force attacks, etc., will block normal app usage.

The problem is that an attacker can format and send an HTTPS request to your web services API which looks *exactly* like a legitimate request coming from your app. The headers are the same. There is no "secret handshake" telling you that "this is your app talking" and "this is not your app talking."

Your enemy can spoof your app. This is a startling realization. You *must* understand this! This may mean that your firewall won't help, because your firewall can't tell the difference between legitimate app traffic and your enemy's attack traffic. You need to think differently.

As developers, we blithely assume that the usual server-level protections are in place. After all, it's the same server! It's the same code base, the same load balancer, the same firewall configuration.

Assuming you use HTTPS for all web services, *and* have it correctly configured, you're covered. Right? Wrong! Your enemy will be only too happy to show you what you've missed.

Lesson: *How do you distinguish between normal users and attackers? With web services, you generally can't. That's why it's so easy for your attacker to appear as "a wolf in sheep's clothing." Attacks can go unnoticed.*

Web Services Need to be Efficient

You put your app in app stores so that people will install and use it. If a million people have your app, and they all use it, that amounts to a potential Distributed Denial of Service attack coming from a million different places. That is a widespread attack, and that is precisely the problem that we all hope to have!

Under the covers, of course, each copy of the app is making GET and POST requests to your server via HTTPS.

The web services code can be far more efficient than a normal web page load. The web services don't need to worry about HTML rendering or navigation bars. For example, you don't need to check the member mailbox if the mailbox content is not part of the current web service response.

RESTful web services are stateless. Generally speaking, the outcome of one web service request should not depend on the outcome of the previous web service request, or the next one. You probably don't even need the standard PHP session when implementing your web services.

This means we are all able to make our web service responses **fast**. Our web services have a far lighter load on our databases. We only hit the database for what we specifically need with this request.

This is all a good thing, right? Our enemy will show us otherwise. Keep this "efficiency" in mind as we look at a common example, *running the password list.*

Running the Password List

Taking a concrete example, the answer is obvious (after the fact).

When an attacker runs a password list against the main website, most large sites detect it rather quickly. The site administrators see a run of failed logins from the same IP address or series of proxies. DevOps is likewise aware of other attack possibilities and watches for them.

With web services, it's different. You expect a lot of traffic from the web service. Because legitimate app traffic looks just like bot traffic, the normal bot-detection approaches don't work.

Are members allowed to log in to your site via the app? That is, do your web services support member login? Remember that your web services are designed to be a *lot* faster than the main site pages.

Putting it together, this means that your attacker can run through their password list a *lot* faster when attacking via your web services. Your enemy will teach you that this is *not* a good thing! With the web services being so much more lightweight and efficient, your enemy can do a lot of damage before you know anything is wrong.

How do you protect your login web service from someone running a password list? We'll cover that in Part Two of this series.

An Open Portal

Continuing our example, say somebody ran a password list against your login web service. You learn to block the attack and move on. Your own efforts at writing efficient code worked against you. That's the nature of the game. What's the big deal?

Our enemy has more to teach us.

Your web services are as stateless and lightweight as you can make them. This means that a lot of what we've learned about PHP "security in depth" simply does not apply. The principles remain, so that means we need to find different ways to achieve those objectives.

One principle is to protect data by keeping it server-side. Browser cookies, for example, can be manipulated by an attacker. We would normally use the PHP session for maintaining state, but we try not to with the web services. You might cache non-sensitive information (such as which offers the user has already completed) in the app and keep everything in your database.

On the main site, a given database query might be five levels deep in the code, with input parameters long since checked, sanitized, and validated. When a web service makes a direct call to that same function, it won't be obvious what protections need to be in place.

There are a number of solutions to this "direct access" issue, such as a "bridge" which centralizes the web service requests and provides validation. Those details don't matter here. What's important is the attitude. We need to consider any such "hot path" a direct path for the enemy.

In any event, we have two (or more) paths to the same functionality. The main site has the functionality, and the web services expose that same functionality. All of this happens naturally. It's normal. You already had a website, and you later expanded your reach by creating the web services.

As we add a web services layer to expose that same functionality to the app (or AJAX or whatever), we're likely dealing with code that came before our time. "It's a trap!" (Admiral Ackbar, *Return of the Jedi*) As you add efficient access to old code, you may be unintentionally losing security that was "bolted on" years ago.

Observing HTTPS Traffic

You should force all of your web services to use HTTPS protocol. That means requests and responses are sent in encrypted form. **Incorrect HTTPS configuration is a common vulnerability.** *Get proof of your correct configuration. See the OWASP SSL/TLS Cheat Sheet[4] for a good overview.*

This should mean that even passwords can be sent in plain text across HTTPS and be safe, right? Your enemy will show you otherwise.

4 *OWASP SSL/TLS Cheat Sheet:* http://phpa.me/owasp-tlp

The problem is that your app is "out there" in the wild. Your attacker can download and install your app just like anyone else. The app can be decompiled. All copies downloaded are identical (until you update with a new app version). The app can be installed by the enemy on a test bed of their choice.

Free tools exist to capture and display encrypted web traffic. I use Fiddler by Telerik[5] for my own web services development. It allows me to see my own HTTPS app traffic, decrypted and nicely formatted.

This is one way your enemy can learn to precisely mimic your app. You can't distinguish a legitimate web service request, coming from your app, from an attack, when not one byte is different.

Do you have security tokens? Of course! But your attacker can probably harvest a live token from the current Fiddler session and use it.

Learn from the Master

What does "learn from the enemy" mean for you? Bruce Lee, possibly the greatest martial artist in living memory, stated, "Be like water, my friend." Water instantly adapts to its environment.

Bruce Lee, quoted in *The Warrior Within* by John Little[6] describes his own self-expression:

> *Jeet kune do is training and discipline toward the ultimate reality in combat. The ultimate reality is simple, direct, and free. A true jeet kune do man never opposes force or gives way completely. He is pliable as a spring and complements his opponent's strength. He uses his opponent's technique to create his own. You should respond to any circumstance without pre-arrangement; your action should be as fast as a shadow adapting to a moving object.*

Bruce's son Brandon Lee explained in the same book,

> *[The master] always talks about teaching "jeet kune do concepts." In other words, teaching someone the concepts, a certain way of thinking about the martial arts, as opposed to teaching them techniques. To me, that kind of illustrates the difference between giving someone a fish and teaching them how to fish. You could teach someone a certain block, and then they have that certain block; or you can teach someone the concept behind such a block, and then you have given them an entire area of thinking that they can grow and evolve in themselves. They can say: "Oh, I see—if that's the concept, then you could probably also perform it this way or that way and still remain true to the concept."*

In other words, one does not "do" web service security. There is no particular way to establish as the "right" way. The right way is whatever keeps your attacker at bay—for now. As your enemy grows and matures, of course, so must you.

Looking Forward

In this part, *Learn from the Enemy,* we learned that we dare not think of web *service* security the same as web*site* security. The enemy does not follow "the rules," whatever they might be. We must therefore directly learn from the enemy how to block the enemy.

5 *Fiddler:* http://www.telerik.com/fiddler

6 *The Warrior Within:* http://www.amazon.com/dp/0809231948

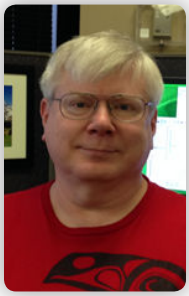
Part Two, *Security Architecture,* teaches you to meet the enemy. You've heard of Authentication and Authorization. We'll show why they do *not* work with web services. Our enemy has challenged us; we'll meet that challenge.

Part Three, *Implementing Encryption,* sounds simple. It is! The trouble is that encryption is extremely difficult to get right. In fact it's a great way to grab news headlines when you get it spectacularly wrong. We'll give you a concrete place to begin. We'll cover randomness, and how to encrypt and decrypt a string.

Additional Reading

This article serves as an introduction to securing your web services. For more advice and guidance, consult the sources collected below.

1. *Survive The Deep End: PHP Security* by Padraic Brady. Excellent survey of what you need to know about PHP security. This short online book is a good starting point. http://phpsecurity.readthedocs.org/en/latest/
2. *PHP Security Cheat Sheet* by The Open Web Application Security Project (OWASP). I include the OWASP page to point out that you should be long past dealing with these basic website security issues. But if you *are* new to PHP security, this is a good reference. https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet
3. *Web Service Security Cheat Sheet* by OWASP. Checklists are valuable. Visit this cheat sheet from time to time to ensure you still have the right things covered. https://www.owasp.org/index.php/Web_Service_Security_Cheat_Sheet
4. *Information Security* at Stack Exchange. I find the *Information Security* folks to be friendly, helpful, authoritative, and thorough. Learn to ask questions correctly and you'll be delighted with the responses. Don't be shy, but show that you've thought things through before typing out the question. http://security.stackexchange.com
5. *How to Hack a Paysite: What the Good Guys Need to Know* by Ed Barnard. This article series is old, but my exploration of attitude and motivation remains relevant. http://otscripts.com/how-to-hack-a-paysite-articles/
6. *The Art of War: Complete Text and Commentaries* by Sun Tzu, translated by Thomas Cleary. Various Twitter accounts quote this two-thousand-year-old classic, including @battlemachinne. One line at a time, this can help you retain that all-important security attitude. http://www.amazon.com/gp/product/1590300548
7. *Threat Modeling: Designing for Security* by Adam Shostack. This is the "big picture" look at formally anticipating security threats to your software. It's a tough row to hoe. But if you don't, who will? http://www.amazon.com/gp/product/1118809998
8. *Web Security: A WhiteHat Perspective*, by Hanqing Wu and Liz Zhao. This one is tough to read but worth the energy expended. I believe there were two editions of the book published, one in Chinese and one in English. A former hacker himself, the author brings a useful perspective and solid information. http://www.amazon.com/gp/product/1466592613
9. *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd Edition, by Ross J. Anderson. This thousand-page monster won't be read in one sitting. Like *Threat Modeling*, this "big picture" book will give you perspective and strategies you won't find elsewhere. http://www.amazon.com/gp/product/0470068523
10. *Cryptography Engineering: Design Principles and Practical Applications* by Niels Ferguson, Bruce Schneier, Tadayoshi Kohno. I saved the best for last. If you're planning to write security-related code, read this book first. It's a good and surprisingly fast read. You'll come away with a far better understanding of how things hold together and why. http://www.amazon.com/gp/product/0470474246



Ed Barnard has been programming computers since keypunches were in common use. He's been interested in codes and secret writing, not to mention having built a binary adder, since grade school. These days he does PHP and MySQL for InboxDollars.com. He believes software craftsmanship is as much about sharing your experience with others, as it is about gaining the experience yourself. The surest route to thorough knowledge of a subject is to teach it. @ewbarnard

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.