owasp十大_OWASP十大安全风险的三点替代方案

翻译

weixin_26722031 于 2020-07-30 10:38:52 发布 1127 收藏
文章标签： java
原文链接：https://medium.com/better-programming/a-3-point-alternative-to-owasps-top-10-security-risks-4e659d1b7b80

owasp十大

For those who don't know, the OWASP Top Ten is a list of common (web) application security concerns that are frequently referenced within the infosec community. If you're applying for a position in the industry, more often than not, one of the requirements listed on the job posting will be "familiarity with the OWASP Top Ten," and you can expect to be quizzed on them during a job interview.

对于那些不知道的人， OWASP十佳是信息安全社区中经常提到的常见(Web)应用程序安全性问题列表。 如果您要申请该行业的职位，通常，职位发布中列出的要求之一是“熟悉OWASP十佳”，并且您可以期望在面试时被问到这些问题。 。

I started out wanting to write this article as an accessible primer featuring clear examples of the Top Ten. While doing research and reviewing the list, I found I didn't necessarily concur with their breakdown and organization of vulnerabilities and thought it could perhaps be streamlined and modernized.

我开始是想写这篇文章作为入门文章，其中包含十大例子。 在研究和查看列表时，我发现我不一定同意它们的细分和漏洞组织，并且认为可以简化和现代化它。

Specifically, I saw some redundancy between points #2, #3, #5, and #6; some that maybe didn't deserve a dedicated spot ,such as #4 (XML External Entities); plus some things that appeared to be missing or understated. It seemed like there was some room for a different take on things, at least.

具体来说，我看到了＃2， ＃3， ＃5和＃6点之间的一些冗余； 一些可能不值得关注的地方，例如＃4 (XML外部实体)； 加上一些似乎缺少或低估的内容。 似乎至少有一些空间可以采取不同的态度。

So instead of writing a primer, I've produced my own version of the list in the form of three high-level areas to cover, each with a list of specific threats and steps for mitigation underneath them.

因此，我没有写底稿，而是以三个高级领域的形式制作了自己的列表版本，每个领域都包含了特定威胁的列表以及在其下的缓解措施。

I believe this approach is a practical and accessible one that acts as a checklist to ensure one's bases are covered. Even if you won't be asked the trivia-type question of listing them off during a job interview, they'll help you give a more holistic answer about how you'd secure a site when asked what your approach would be. Maybe you can impress your interviewer by saying, "Sure, I know the Top Ten, but have you heard of Dawe's Three?" (working title). One can dream.

我认为这种方法是一种实用且可访问的方法，可以用作清单以确保覆盖基础。 即使在面试过程中不会被问到将它们列出来的琐事式问题，它们也会帮助您提供一个更全面的答案，当您被问到采用哪种方法时，如何保护网站。 也许您可以通过说：“当然，我知道前十名，但是您听说过Dawe的三强吗？”打动您的面试官。 (工作名称)。 一个人可以梦想。

Let's assess some threats.

让我们评估一些威胁。

1.恶意输入 (1. Malicious Input)

Photo by Kira auf der Heide on Unsplash

图片由 Kira auf der Heide 摄于 Unsplash

它是什么？ (What is it?)

This is a broad category — it covers OWASP #1, #4, #7, #8, plus some additional concerns. Input comes in many forms (including literal form submission) and is the primary way in which users interact with your application. Malicious user input, often in the form of code injection or file interaction, is likely the most often exploited attack vector when it comes to web apps.

这是一个广泛的类别-它涵盖OWASP ＃1， ＃4， ＃7， ＃8，以及一些其他问题。 输入有多种形式(包括文字形式的提交)，是用户与您的应用程序交互的主要方式。 当涉及到Web应用程序时，通常以代码注入或文件交互的形式进行的恶意用户输入可能是最常利用的攻击媒介。

There are many types of attacks that fall under this heading, but they all follow the same pattern: unsanitized, unexpected, or mishandled inputs result in a system doing something that it wasn't intended to, with negative results. Here's a list of common concerns, which can act as a jumping-off point for further research (just Google the cited item of interest):

有许多类型的攻击属于此类别，但是它们都遵循相同的模式：未经过滤，意外或处理不当的输入会导致系统执行原本不希望的操作，但结果是负面的。 以下是一些常见问题，可以作为进一步研究的起点(仅Google列出感兴趣的项目)：

简单攻击 (Simple attacks)

Input provided into obvious sections of an application, such as input elements, forms, and URL parameters — GET and POST variables — that contain snippets of malicious code that's then executed on the target system or by clients using the service.

提供给应用程序明显部分的输入，例如输入元素，表单和URL参数(GET和POST变量)，其中包含恶意代码段，然后这些恶意代码段在目标系统上或由使用该服务的客户端执行。

> **SQL injection:** Database queries are manipulated through user-supplied input to extract or tamper with a site's database
>
> **SQL注入：**通过用户提供的输入来操作数据库查询，以提取或篡改站点的数据库
>
> **Cross-site scripting (XSS) family:** Client-side code (such as HTML elements and JavaScript) gets added to a site that's then executed by visitors, often to directly steal their information or trick them into giving it up — or to trigger operations on their behalf
>
> **跨站点脚本(XSS)系列：**客户端代码(例如HTML元素和JavaScript)被添加到网站上，然后由访问者执行，通常是直接窃取他们的信息或诱使他们放弃信息-或触发操作代表他们

复杂的攻击 (Complex attacks)

These are easy to overlook because the exploitation path is less obvious. Instead of providing a simple malicious input to a site, the attacks are embedded in things like markup, encoded data, or other more obscure areas of interaction. These types of attacks can lead to lower-level compromises of the underlying system since the involved operations are often more trusted than simple user inputs, and code executed as a result can be very damaging.

这些很容易被忽略，因为开发路径不太明显。 攻击不是向站点提供简单的恶意输入，而是嵌入到标记，编码数据或其他更晦涩的交互区域中。 由于所涉及的操作通常比简单的用户输入更受信任，因此这些类型的攻击可能导致底层系统受到较低级别的破坏，并且所执行的代码可能会造成很大的破坏。

(Un)serialization: Complex (non-primitive) data passed between the client and server or between services is often serialized ("packed" and "unpacked") to facilitate the ease of transmission. Because the data, especially in its serialized form, is unreadable to humans, it's often overlooked as a potential attack vector. Malicious inputs and instructions can be hidden in serialized data that then are processed and executed upon deserialization.

(非序列化)：客户端和服务器之间或服务之间传递的复杂(非原始)数据通常被序列化("打包"和"解包")，以简化传输。 由于数据(特别是序列化形式的数据)是人类无法读取的，因此通常被视为潜在的攻击媒介而被忽略。 恶意输入和指令可以隐藏在序列化数据中，然后在反序列化后进行处理和执行。

XML/JSON attacks: Similar to the above point, XML and JSON (the language of the web) can be manipulated by attackers to achieve their goals. Capabilities of the formats can also be exploited in unintended ways — they're not just dumb data wrappers. OWASP dedicates an entire point (#4) to XML-related XXE attacks, which personally I find to be a very fringe consideration not necessarily worthy of a dedicated spot, but as a whole, these types of attacks must be considered. Beware of what may be lurking in this data.

XML / JSON攻击：与上述观点类似，攻击者可以操纵XML和JSON(网络语言)以实现其目标。 格式的功能也可以以意想不到的方式加以利用-它们不仅是愚蠢的数据包装器。 OWASP将与XML相关的XXE攻击的重点放在整个(＃4)位置，我个人认为这是一个非常重要的考虑因素，不一定值得专门研究，但总的来说，必须考虑这些类型的攻击。 当心什么可能潜伏在此数据中。

Cookies, session data, and HTTP headers: These are all forms of input that get processed by clients and servers and thus act as an additional attack vector, albeit an obscure and rarely fruitful one. Keep it in mind during parsing.

Cookie，会话数据和HTTP标头：这些都是客户端和服务器处理的所有形式的输入，因此尽管是一种晦涩且很少有成果的方法，但仍可作为一种附加的攻击媒介。 在解析时请记住这一点。

以文件为中心的攻击 (Filecentric attacks)

Files (and file access requests) are another type of input that can pose a significant threat to an application. Abusing storage and retrieval functions can lead to a total compromise of a system or disclosure of its contents.

文件(和文件访问请求)是另一种输入，可能会对应用程序构成重大威胁。 滥用存储和检索功能可能会导致系统完全受损或系统内容泄露。

Directory traversal: In an app that provides file retrieval, especially from the underlying file system (and not a database), the `filename`, `path`, and related parameters are vectors for malicious input. If you have a script that downloads a file of a given name from a fixed directory, providing `../../../../secret.txt` can be a way for attackers to access things they shouldn't. Some misconfigured webservers can also expose this type of threat in a generic sense — i.e., without specifically relating to file-retrieval logic.

目录遍历：在提供文件检索功能的应用程序中，尤其是从底层文件系统(而非数据库)中检索文件的`filename`， `path`和相关参数是恶意输入的向量。 如果您有一个脚本从固定目录中下载了给定名称的文件，则提供`../../../../secret.txt`可能是攻击者访问他们不应访问的内容的一种方式。 一些配置错误的Web服务器也可以从一般意义上暴露这种类型的威胁-即，与文件检索逻辑没有特别关系。

Uploading scripts and files: Like above, attackers may target files for replacement when given the ability to upload things to a server. Alternatively, they may upload malicious scripts (such as web shells) to a location above the webroot which they can then run through their browser. Watch those extensions and MIME types!

上载脚本和文件：像上面一样，如果攻击者可以将文件上载到服务器，则攻击者可以将文件替换为目标。 或者，他们可以将恶意脚本(例如Web Shell)上载到Webroot上方的某个位置，然后他们可以通过其浏览器运行该脚本。 观看那些扩展名和MIME类型！

Zip bomb: A malicious ZIP file that extracts to a near-infinite size which can cause denial-of-service and resource-exhaustion type attacks

Zip炸弹：恶意的ZIP文件，提取到接近无限的大小，可能导致拒绝服务和资源耗尽型攻击

Zip slip: A malicious ZIP file that replaces files elsewhere on the system during extraction, relying on similar principles to directory-transversal attacks

邮编：恶意的ZIP文件，在提取过程中会替换系统上其他位置的文件，并依赖于与目录遍历攻击类似的原理

如何防范恶意输入 (How to protect against malicious input)

Repeat after me: Sanitize your inputs. Sanitize *all* your inputs. Don't trust them. Get paranoid. Put on a tinfoil hat.

在我之后重复：清理您的输入。 清理*所有*输入。 不要相信他们。 变得偏执。 戴上锡箔帽子。

I've given you a list to work from, and researching specific vectors will yield advice on mitigation — just make sure you're considering everything that applies to your system. Put yourself in the hacker's shoes, and think of ways in which input could possibly be abused — then compensate for it.

我给了您一个工作清单，研究特定的矢量将为缓解提供建议-只要确保您正在考虑适用于您系统的所有内容即可。 让自己陷入黑客的困境，想想可能会滥用输入的方式，然后加以补偿。

It's important to remember, anything that's passed to the server counts as user input, even if there's no section on the front end for it and it's processed behind the scenes. Out of sight shouldn't mean out of mind — one quick look at the network panel reveals a wealth of potential attack vectors not immediately obvious when practicing defensive security.

请记住，重要的是，传递到服务器的所有内容都将视为用户输入，即使前端没有任何部分并且已在后台对其进行处理。 视线不应该意味着失去理智-快速浏览一下网络面板会发现大量潜在的攻击媒介，在实施防御安全性时不会立即显现出来。

Once you're screening your inputs like an overzealous TSA worker, it's time to move on to shoring up the next core function of your system.

一旦像狂热的TSA工作者一样筛选输入内容，就该着手加强系统的下一个核心功能了。

2.访问控制不足 (2. Insufficient Access Control)

Locked gate

Photo by Jose Fontano on Unsplash

Jose Fontano在 Unsplash上 拍摄的照片

它是什么？ (What is it?)

Trust is a difficult and fragile thing to achieve over the web — and this challenge scales as more moving parts and systems get involved.

通过网络获得信任是一件困难而脆弱的事情，而且随着越来越多的活动部件和系统参与其中，这一挑战也日益严峻。

In an ecosystem populated by SaaS products, third-party APIs, cross-domain traffic, shared authentication providers, and mixed devices and platforms, there's a lot of room for a missing authentication check or rule in the mix. You have to ensure each resource is only accessible by whoever should be accessing it, and that the communication channels themselves are also secure. This section roughly covers OWASP #2, #3, #5, and #6.

在由SaaS产品，第三方API，跨域流量，共享的身份验证提供程序以及混合的设备和平台组成的生态系统中，混合身份验证检查或规则缺失的空间很大。 您必须确保每个资源只能由应该访问它的任何人访问，并且通信通道本身也是安全的。 本节大致涵盖OWASP ＃2， ＃3， ＃5和＃6 。

标准和威胁 (Standards and threats)

It's all about how users are allowed to interact with your system. Any point of interaction, if not properly controlled, is a potential source of information disclosure or unauthorized operations.

这与如何允许用户与您的系统进行交互有关。 如果没有适当控制，则任何交互点都是潜在的信息泄漏或未经授权的操作。

> **Protocol security:** The obvious stuff — and really a precursor to this category. Communications need to be secured using a modern TLS configuration. Your site redirects HTTP to HTTPS appropriately, and there isn't room for SSL-strip type attacks. Cookies should use the `secure` flag where applicable. Without this, attackers can intercept and manipulate your traffic, and efforts taken to enforce authentication and access control will be in vain.
>
> **协议安全性：**显而易见的东西—实际上是该类别的先驱。 需要使用现代的TLS配置保护通信的安全。 您的站点将HTTP适当地重定向到HTTPS，并且SSL-strip类型的攻击没有空间。 Cookies应在适用的地方使用`secure`标志。 没有这些，攻击者就可以拦截和操纵您的流量，而用于实施身份验证和访问控制的努力将徒劳无功。
>
> **Authentication and access control:** The core concept of the category from which specific threats extend from. Users should be authenticated and resources protected by an appropriate ruleset — basically, if established identity *x* requests resource *y,* determine if they're permitted access it. An oversight here can lead to information disclosure or worse.
>
> **身份验证和访问控制：**特定威胁源自的类别的核心概念。 应该对用户进行身份验证，并使用适当的规则集保护资源-基本上，如果已建立的标识*x*请求资源*y，请*确定是否允许他们访问它。 此处的疏忽可能导致信息泄露或更糟。
>
> **Lack of server-side checks:** Attackers can make calls to your server without using your client (web, mobile, etc). This means auth and AC has to be done on the server. A user may have permission to update their bio, but what happens if they post `{"admin": true}` as part of their update submission, even if there's no "admin" checkbox displayed client-side? All endpoints and actions need to be secured (and sanitized, re: malicious inputs) as if the client doesn't exist.
>
> **缺乏服务器端检查：**攻击者可以在不使用客户端(Web，移动设备等)的情况下呼叫服务器。 这意味着必须在服务器上完成身份验证和AC。 用户可能有权更新自己的个人资料，但是，即使没有在客户端显示“admin”复选框，但如果他们在提交更新时发布了`{"admin": true}`，会发生什么情况呢？ 就像客户端不存在一样，需要保护所有端点和操作(并进行清理，重新：恶意输入)。

Scraping: Modern APIs are very semantically consistent and don't rely on security by obscurity. Because of this, it's easy for someone to write a script to request `/record/1`, `/record/2`, …. If an attacker can guess or infer the URL for sensitive information or iterate/brute-force their way to it, you're at risk for information disclosure.

爬网：现代API在语义上非常一致，并且不因模糊而依赖于安全性。 因此，很容易有人编写脚本来请求`/record/1`, `/record/2`, … 如果攻击者可以猜测或推断出敏感信息的URL或对其进行迭代/强行处理，则您面临信息泄露的风险。

Bot mitigations: A source of brute-force and credential stuffing attacks, as well as spam. Use captchas and IP blacklisting where appropriate, like on login and submission forms.

缓解僵尸病毒：暴力和凭据填充攻击以及垃圾邮件的来源。 在适当的地方使用验证码和IP黑名单，例如在登录和提交表单上。

Cross-site request forgery (CSRF): Imagine you have an endpoint `https://example.com/user/1/delete` to delete a user — what happens if one of your users is tricked into clicking that link through another website or other source? Any action that can be directly called in this manner is at risk of being exploited by attackers working outside your system. Tokenized operations are one way to mitigate this.

跨站点请求伪造(CSRF)：假设您有一个端点`https://example.com/user/1/delete`删除用户-如果一个用户被诱骗通过另一个网站或其他网站单击该链接，会发生什么情况资源？ 可以这种方式直接调用的任何操作都有被系统外的攻击者利用的风险。 令牌化操作是缓解这种情况的一种方法。

Cross-origin resource sharing (CORS) and cross-domain attacks: Browsers and apps can make resource requests behind the scenes. When these requests are to a different site (between subdomains or domains, to an API, etc.), certain security rules apply. A misconfiguration of these rules can mean when one of your users visits a malicious site, that site can automatically trigger a request to your own site using your user's permissions. These rules are tricky to configure, so it's tempting to make them extremely permissive — to your peril. They should be as restrictive as possible while still allowing your application to function. Also look into implementing `X-Frame-Options` if you haven't already.

跨域资源共享(CORS)和跨域攻击：浏览器和应用程序可以在后台提出资源请求。 当这些请求发送到其他站点(在子域或域之间，到API等)时，将应用某些安全规则。 这些规则的错误配置可能意味着当您的一个用户访问恶意站点时，该站点可以使用用户的权限自动触发对您自己的站点的请求。 这些规则配置起来很棘手，因此很容易使它们变得极端宽容-带来危险。 它们应尽可能严格，同时仍允许您的应用程序运行。 如果还没有，请考虑实施`X-Frame-Options`。

Unnecessary exposure: Anything that doesn't need to be publicly accessible shouldn't be. This can include admin panels, testing files and folders, directory indexing, `robots.txt` entries for uncrawlable sections, internal services, testing or internal sites and subdomains, and so forth. For hackers performing reconnaissance, the more information, sites, and resources they can glean, the better. Use IP whitelists and password protection on any sensitive internals that are public-facing but need to remain exposed, and get rid of everything else.

不必要的暴露：不需要公开访问的任何内容都不应存在。 这可以包括管理面板，测试文件和文件夹，目录索引，不可`robots.txt`部分的`robots.txt`条目，内部服务，测试或内部站点和子域等。 对于进行侦查的黑客来说，他们可以搜集的信息，站点和资源越多越好。 在面向公众但需要保持公开状态并摆脱其他所有问题的任何敏感内部组件上，使用IP白名单和密码保护。

如何保护您的服务和资源 (How to secure your services and resources)

After considering the above, ask yourself the following questions:

考虑上述因素后，请问自己以下问题：

1. Are all necessary sections of your application restricted to authenticated users only?
   您的应用程序的所有必要部分是否仅限于经过身份验证的用户？
2. Are all actions and resources properly restricted to the appropriate users/roles/origins?
   是否将所有操作和资源适当地限制在适当的用户/角色/来源中？
3. Are your endpoints secured independently of the client-side logic?
   端点的安全性是否独立于客户端逻辑？
4. Are your communication channels encrypted and secured against eavesdropping, and do your servers enforce proper protocol use?
   您的通信通道是否经过加密和安全保护以防窃听，并且服务器是否强制使用正确的协议？
5. Are you exposing resources to the public internet that you shouldn't be?
   您是否在向公共互联网公开不应该的资源？

It's about reviewing your system, spotting gaps, addressing them, and then repeating. Think about how information flow is controlled from the protocol level up for all involved systems and resources. When possible, actually simulate testing the threats versus looking at code and configurations — something may be behaving differently than expected.

这是关于检查系统，发现差距，解决差距然后重复的问题。 考虑一下如何从协议级别控制所有相关系统和资源的信息流。 在可能的情况下，实际模拟测试威胁而不是查看代码和配置-某些行为可能与预期不同。

It's a lot to cover, and making your system airtight is an art. One must also consider what happens when an item is overlooked and things go wrong.

涉及很多，使系统密闭是一门艺术。 人们还必须考虑当某件物品被忽略且出现问题时会发生什么。

3.操作失败 (3. Operations Failures)

People sitting in front of lots of monitors

Pixabay@Pexels 来自Pixaba @ Pexels

这些是什么？ (What are they?)

Under the broad umbrella of *ops* — DevOps, sysadmin, policies, and procedures — bad practices and lack of coverage can lead to disaster or make a disaster worse. Every moving part and its (proper or improper) implementation is a potential source of threat, and the plans and practices put into place by an organization serve to mitigate and handle such threats should the need arise. This is where you prepare for success or failure.

在*OPS*的广泛伞- DevOps的，系统管理员，政策和程序-不良做法和覆盖不足可能导致灾难或使灾难雪上加霜。 每个活动的部分及其(适当或不适当的)实施都是潜在的威胁源，组织制定的计划和实践可在需要时缓解和应对此类威胁。 这是您为成功或失败做准备的地方。

威胁与担忧 (Threats and concerns)

This will cover OWASP #9 and #10 — if you're keeping track, those were the two remaining points. It also includes some of the softer considerations around policies and practices that are often underserved when discussing the security of a system. Neglected best practices develop into latent then active dangers to an organization.

这将涵盖OWASP ＃9和＃10-如果您要跟踪的话，这就是剩下的两个要点。 它还讨论了围绕策略和实践的一些较软的考虑因素，而这些因素在讨论系统的安全性时往往没有得到足够的重视。 被忽视的最佳实践会给组织带来潜在的潜在危险。

Dependency count: Can you list all the dependencies and technologies your product uses? Having a high number is neither inherently good or bad, but I'm of the opinion that they should be minimized when possible and within reason. What's more important is that they're identified and their risks considered through a security lens. For example, If you're using WordPress — a ubiquitous, useful, and regularly-targeted CMS — it (and its plugins) needs regular attention to be made and kept secure.

依赖性计数：您能否列出产品使用的所有依赖性和技术？ 拥有大量数字既不是天生的好坏，但我认为应该在可能的情况下并在合理的范围内将其最小化。 更为重要的是，通过安全镜头可以识别它们并考虑风险。 例如，如果您正在使用WordPress(一种无处不在，有用且定期定位的CMS)，则需要经常注意它(及其插件)并保持其安全性。

Patching: Using out-of-date software with known vulnerabilities is one of the easiest ways to get compromised. Apply your patches! Automatically is preferred, but for production systems, if you're taking the manual approach, don't let anything slip through the cracks. Stay up to date on security bulletins for technology in use throughout your organization.

修补：使用具有已知漏洞的过时软件是最容易受到威胁的方法之一。 应用您的补丁！ 首选自动，但对于生产系统，如果您采用手动方法，请不要让任何东西从裂缝中溜走。 随时了解整个组织中正在使用的技术的安全公告。

Securing software and services: Lack of firewall/IDS/IPS coverage and access control gives hackers extra opportunities to breach your systems. Internal and public systems should be isolated or put into a DMZ (when applicable), in case of a partial breach. Lock down anything that doesn't need to be publicly accessible, and secure what does.

保护软件和服务的安全：缺乏防火墙/ IDS / IPS覆盖和访问控制，使黑客有更多机会破坏您的系统。 如果发生部分违反，则内部和公共系统应隔离或放入DMZ(如果适用)。 锁定不需要公开访问的所有内容，并保护需要的内容。

Logging: A lack of proper logging means two things: the inability to detect potential threats and patterns of malicious actors and the inability to properly respond to a security incident. Logging should be robust, centralized, regularly reviewed, and available when needed to help trace problems.

日志记录：缺少适当的日志记录有两件事：无法检测到恶意行为者的潜在威胁和模式，以及无法正确响应安全事件。 日志应该健壮，集中，定期检查，并在需要时可用以帮助跟踪问题。

Backups (and RAID): An obvious one. You need a copy, in fact, multiple copies (follow the 3–2–1 rule) of your important data — otherwise, you face potential data loss, which can be devastating to an organization. Offline backups are a must. Be prepared for physical disasters, hardware failure, accidental deletion, intentional deletion, corruption, ransomware and cryptolocker attacks, and so on. Backups must be routinely verified, and the restoration process practiced so you're prepared should the time come.

备份(和RAID)：显而易见。 实际上，您需要一份重要数据的副本(遵循3–2–1规则)–否则，您将面临潜在的数据丢失，这可能对组织造成破坏。 脱机备份是必须的。 为物理灾难，硬件故障，意外删除，故意删除，损坏， 勒索软件和cryptolocker攻击等做好准备。 必须定期验证备份，并执行恢复过程，以便在需要时做好准备。

Disaster recovery plan (DRP): Have you considered what'd happen if there's a fire in your office, your production server gets hacked, your database is leaked to the public, or all your systems get locked down in a ransomware attack? Thinking about and discussing such threats helps you mitigate them before they happen. Having a documented plan means you're able to respond quickly and efficiently during an emerging crisis when time is of the essence. Creating and regularly maintaining this document is key for organizational security (and make sure you can access it when systems are down).

灾难恢复计划(DRP)：您是否考虑过办公室发生火灾，生产服务器被黑客入侵，数据库泄漏给公众或者所有系统都被勒索软件攻击锁定的情况？ 考虑和讨论此类威胁可帮助您在威胁发生之前将其缓解。 制定文件化的计划意味着您能够在时间紧迫的新兴危机中快速有效地做出响应。 创建并定期维护此文档是组织安全的关键(并确保在系统停机时可以访问它)。

做有需要的 (Doing the needful)

Secure systems, apply patches, and keep and monitor backups and logs. Put policies into place to ensure this is being done. Brainstorm potential threats and disasters to help develop a DRP, and treat it as a living document. Get creative — ask as many "what happens if …" questions as possible. Also, ask "is [this technology] safe to use?" and "are we following best practices?" out of habit.

保护系统安全，应用补丁以及保存和监视备份和日志。 制定适当的政策以确保做到这一点。 集思广益潜在的威胁和灾难，以帮助开发DRP，并将其视为活动文档。 发挥创造力-尽可能多地询问"如果……会发生什么"问题。 另外，问" [这项技术]使用安全吗？"和"我们遵循最佳做法吗？"出于习惯。

For routine maintenance, such as checking logs and applying patches, simply using a repeating calendar event and shared reporting document can make sure nothing is left to slide. Assign responsibility to an individual, and follow up with them to ensure things are being done.

对于例行维护(例如检查日志和应用补丁程序)，只需使用重复的日历事件和共享的报告文档即可确保没有滑动。 将责任分配给个人，并跟进他们，以确保事情已经完成。

Due diligence and thoughtful preparation will provide security and peace of mind in your organization.

尽职调查和周到的准备将为您的组织提供安全和安心。

回顾 (Recap)

As you can see, there are a lot of bases to cover. At a high level, three things need to be done to ensure an app is secure:

如您所见，有很多基础可以覆盖。 从高层次上讲，需要做三件事来确保应用程序的安全性：

1. Inputs must be sanitized and securely handled.
   输入内容必须经过清理和安全处理。
2. Communications and resource access must be appropriately implemented and secured.
   通信和资源访问必须得到适当实施和保护。
3. Best practices for operating a system of technologies need to be followed.
   需要遵循操作技术系统的最佳实践。

Those get broken down into the related steps and considerations documented above. This by no means an exhaustive list — but rather a framework to get you thinking about securing an application from the most common threats. Given the complexity and interplay of technologies in use today, it's impossible to cover all the potential ways a system may be exploited.

这些细分为以上记录的相关步骤和注意事项。 这绝不是一个详尽的清单，而是一个框架，可让您考虑保护应用程序免受最常见威胁的侵害。 考虑到当今使用的技术的复杂性和相互作用，不可能涵盖开发系统的所有潜在方法。

When securing your application, in particular, think about what you're doing that isn't standard, and in that difference, if there are any unique dangers that you may face.

特别是在保护应用程序安全时，请考虑一下您正在做的事情不是标准的，并且如果您可能面临任何独特的危险，则应区别对待。

结论 (Conclusion)

Relating this back to OWASP, I've covered all the individual points in the Top Ten, plus some additional areas.

与OWASP相关，我涵盖了前十名中的所有各个要点，以及其他一些方面。

I think three high-level points, each with significant detail underneath, may be an easier-to-remember and more practical framework in comparison.

我认为，相比之下，三个高级要点，每个要点下面都有重要的细节，可能会更容易记住和更实用。

I do believe the way I've presented this may serve as a more comprehensive checklist-style approach to actually securing an application, versus a simple catalog of certain threats. Ultimately, I think this should act as a useful reference for someone who's found themselves responsible for securing an application/system — it's built from my experience in doing just that for over 10 years, put to paper.

我确实相信我所介绍的方法可以作为一种更全面的清单样式方式来实际保护应用程序，而不是简单地列出某些威胁。 最终，我认为这对于那些发现自己负责保护应用程序/系统的人应该是一个有用的参考-它是基于我10多年的经验构建而成的。

And if you liked this writeup, I also produced a ransomware-focused mitigation guide for organizations earlier this year.

如果您喜欢这篇文章，我还将在今年早些时候为组织制作针对勒索软件的缓解指南 。

> 翻译自: https://medium.com/better-programming/a-3-point-alternative-to-owasps-top-10-security-risks-4e659d1b7b80

owasp十大