HOW TO PROTECT AGAINST THE OWASP TOP 10 AND BEYOND

THINK APP SECURITY FIRST

INTRODUCTION

Application security is hard. We see headlines about data breaches almost daily. According to F5 Labs, web application attacks are the #1 single source entry point of successful data breaches. Additionally, the fallout from application attacks come with high costs. What is it about web applications that makes them so precarious?

Insecure software.

The challenges of web application security are well known, but they're still all too common out in the real world. Age-old vulnerabilities are still exploited as bad actors use automation to probe the Internet looking for potential exploit victims. In a fast-moving digital economy where speed to market is key, most development teams don't have the resources to sufficiently protect against the onslaught of attacks at each vector—or the level of expertise needed to address every vulnerability quickly and accurately. The need to address these vulnerabilities over and over, in every application that goes out the door, is a significant blocker in your path to production. And because applications are the pathway to your data, protecting your business means it's critical that you understand the potential risks.

These vulnerabilities are typically persistent, long-standing problems created by technical debt or the existence of layers of complex dependencies—often third party and open source software—which require dedicated time and resources to remedy. As a result, the vulnerabilities are often never addressed: When the main objective of application development is to push out new features, it's difficult to build remediations into every new application update that is shipped.

Fortunately, there are options. Having the right tools and third-party controls in place can go a long way toward mitigating risk—and speeding development of your applications at the same time.

72%

VERACODE NOTED THAT 72% OF ALL APPLICATIONS HAD AT LEAST ONE VULNERABILITY.

THE OWASP
TOP 10

VULNERABILITIES AND MITIGATIONS

THE OWASP PROJECT: CAN EDUCATION REDUCE VULNERABILITIES?

The pervasive nature of web application security shortcomings has not gone unnoticed. In 2001, a number of security professionals banded together to create the Open Web Application Security Project (OWASP) to educate developers and security professionals with the goal of reducing these security shortcomings. OWASP is a nonprofit international group that produces publicly available methodologies, documentation, tools, and training addressing many aspects of web application security.

THE OWASP TOP 10: A TAXONOMY OF RISK

The Open Web Application Security Project publishes the OWASP Top 10, which represents a broad consensus on the ten most critical web application security risks. Many are well known vulnerabilities but remain difficult to defend against. While each organization's risks are different, the OWASP Top 10 is a perfect way to encourage secure coding within an organization since these are the most pervasive threats to software security today. The goal of the OWASP Top 10 is to provide a basic taxonomy of risk with respect to web application vulnerabilities.

Future versions of the OWASP Top 10 are slated to be more closely aligned with widely accepted risk frameworks, such as ISO 31000:2015[1]. This change is intended to boost the credibility and applicability of the project, which is increasing uptake and adherence to its philosophy.

OWASP also publishes the API Security Top 10, the Mobile Top 10, the IoT Top 10 and the Automated Threats list.

PROTECTING YOUR APPLICATIONS: AN OVERVIEW OF THREATS

If you are responsible for the development, security, or operation of a web application, becoming familiar with the OWASP Top 10 can help you better protect that app. In addition, security testing against the OWASP Top 10 is a core requirement of numerous industry and regulatory standards such as the PCI DSS. The OWASP site also lists other relevant international security standards that reference the OWASP Top 10.[2] By delving into the most common web app security problems and learning about effective mitigations, you can boost your organizational security posture, protect your critical applications, and help ensure the confidentiality, integrity, and availability of your data.

[1] https://www.iso.org/iso-31000-risk-management.html
[2] https://www.owasp.org/index.php/Industry:Citations

A1 INJECTION

Injection is a broad category of attacks that can manifest differently, depending on context. These attacks occur when there's insufficient user input validation, and an attacker secretly adds—or injects—their own instructions into an existing authorized application execution process. This alters the normal or expected operation of the application process. Because the web application is not properly filtering the input, it allows injected commands to be passed through to either the local system or a dependent one.

What's the goal of that injection payload? It varies widely. Some injection exploits let attackers execute commands within the operating system. Some are used to circumvent authentication, and some inject code—such as malware—or other objects, including code for escalation of privilege.

Here's how it works:

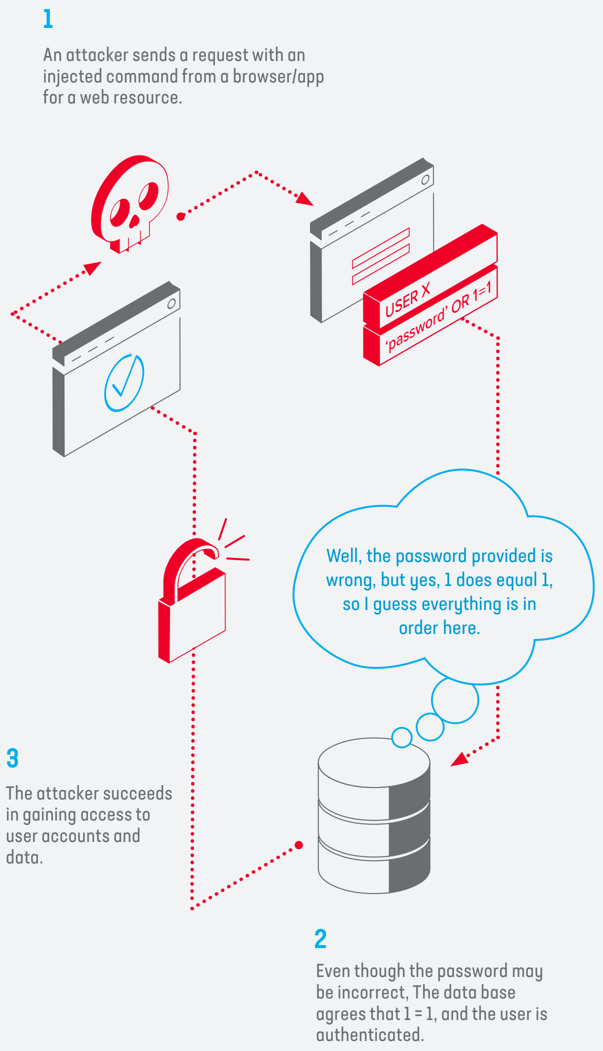
- An attacker injects a type of bogus code, or a malicious command such as SQL, LDAP, or XPath, into a vulnerable element of the application.
- The application's interpreter processes the injection.
- The attacker then gains unauthorized access, causes a denial of service, steals sensitive data, or some other nefarious outcome.

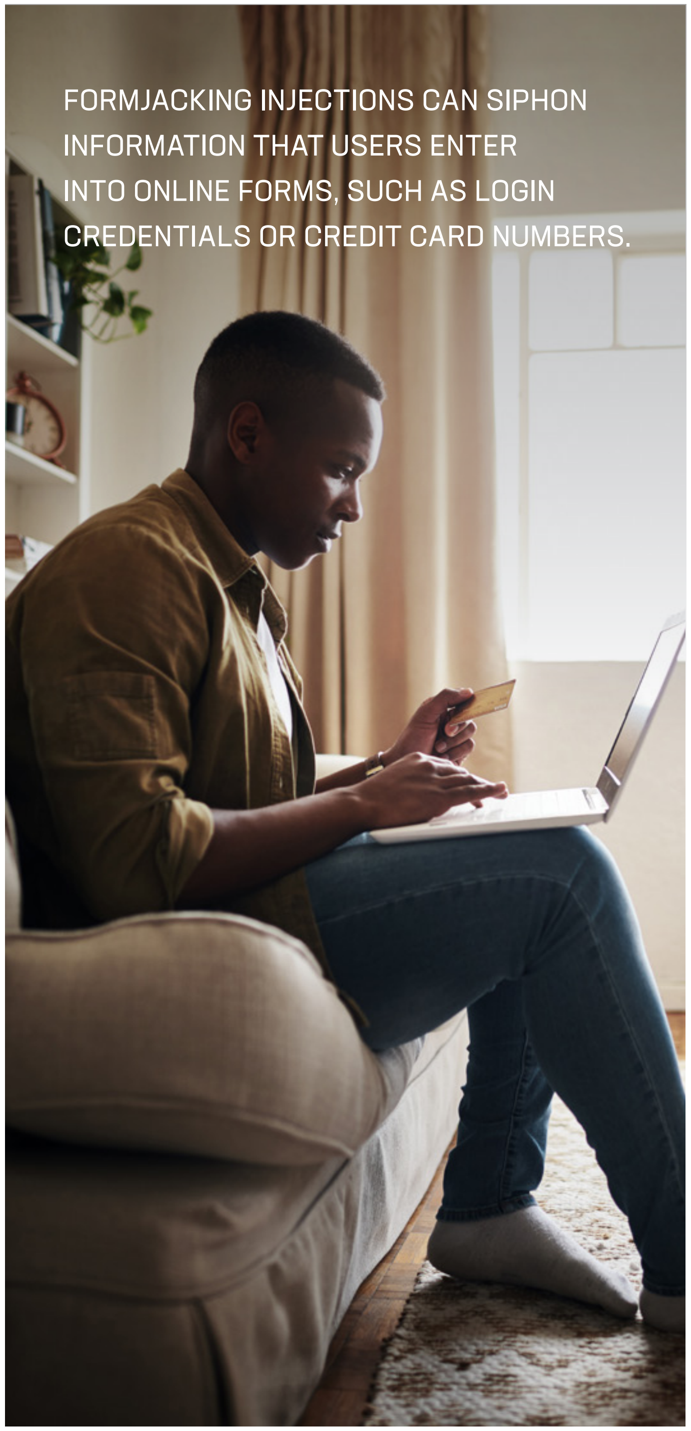
A common example is the SQL injection attack. Many applications rely upon user input to build SQL statements to fetch information or to log them in. For example: select * from USERTABLE where USERID = '[userid-from-web-form]' and PASSWORD = '[passwd-from-web-form]'

Under normal circumstances this could match an entry in the data table USERTABLE, and if so, then the statement evaluates to "True" and the login succeeds. But what if the user puts in "password' OR 1=1" as the password on the web form: select * from USERTABLE where USERID = '[userid-from-web-form]' and PASSWORD = 'password' OR 1=1;

But what if without proper sanitization and escaping, the SQL server will evaluate the 1=1 conditional as TRUE and log the user in with the username provided regardless of what password was supplied. This is a very simple, and targeted, example. SQL injection attacks can get far more sophisticated and malicious, and have been used successfully to delete entire databases, modify records, and exfiltrate sensitive data.

SQL ATTACKS HAVE BEEN USED TO DELETE ENTIRE DATABASES, MODIFY RECORDS AND EXFILTRATE SENSITIVE DATA.


1
An attacker sends a request with an injected command from a browser/app for a web resource.
USER X
'password' OR 1=1
Well, the password provided is wrong, but yes, 1 does equal 1, so I guess everything is in order here.
3
The attacker succeeds in gaining access to user accounts and data.
2
Even though the password may be incorrect, The data base agrees that 1 = 1, and the user is authenticated.


FORMJACKING INJECTIONS CAN SIPHON INFORMATION THAT USERS ENTER INTO ONLINE FORMS, SUCH AS LOGIN CREDENTIALS OR CREDIT CARD NUMBERS.

A1: INJECTION, CONT'D

Currently, one of the most effective types of injection is formjacking. A formjacking attack siphons information that users put into online forms such as login screens or payment forms, and delivers it to a location under the attacker's control. Most of the time attackers seek login credentials, or financial information such as payment card numbers. You may have heard of Magecart attacks, a type of formjacking attack.

MITIGATING INJECTION RISKS

Injection vulnerabilities aren't new, and mitigating them is simple—in theory. With respect to data input and security, you cannot inherently trust any data from the user. Injection vulnerabilities can be detected during development, but are more difficult to spot in deployed systems. Because injection flaws can be exploited in any stage of an attack, finding and evaluating their impact depends on context.

All input must be examined, escaped, sanitized, and filtered. Injection attacks can occur in normal user input forms as well as in hidden web fields. Leveraging parameterized SQL[3] can go a long way toward mitigating this risk by compartmentalizing input data and distinguishing it from code, regardless of user input. It is also advisable to monitor outbound responses returned to the user in an effort to detect information leakage resulting from a successful injection attack. A Web Application Firewall (WAF) provides the defense in depth needed to protect against vulnerabilities too expensive to fix in code, ones not caught by other tools in the pipeline, and new attack permutations.

Watch a video about: OWASP Top 10: Injection Attacks

[3] https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet#Defense_Option_1:_Prepared_Statements_.28with_Parameteized_Queries.29

61% OF FORMJACKING BREACHES WERE IN THE RETAIL INDUSTRY

F5 Labs Threat Intelligence found 145 breaches attributable to formjacking attacks on web payment forms in 2019, up from 83 in 2018. Sixty-one percent of those formjacking breaches came from the retail industry[4].

[4] https://www.f5.com/labs/articles/threat-intelligence/application-protection-research-series-executive-summary

UK'S NATIONAL CYBER SECURITY CENTRE (NCSC) THE TOP 10 MOST COMMON PASSWORDS IN 2019[5]

1. 123456
2. 123456789
3. qwerty
4. password
5. 111111
6. 12345678
7. abc123
8. 1234567
9. password1
10. 12345

A2 BROKEN AUTHENTICATION

Accurately knowing who a user is (authentication) and what they are allowed to do (authorization) are foundational concepts of security that complement each other. Since we're talking about web application authentication, we should begin with passwords. Despite their rudimentary nature and the inherent risks and inconvenience that come with them, passwords are unfortunately still far and away the most common way of authenticating a user.

Stolen credentials are the predominant method of web application compromise.[5] The insecure nature of user passwords makes attacks like credential stuffing both easy and remarkably successful. A credential stuffing attack occurs when an attacker has gotten their hands on a large database of stolen user credentials. They then use automated tools to test those passwords against a variety of other sites and services to see what works. By some estimates, as much as 90% of all login attempts on web-based applications at Fortune 100 firms are credential stuffing attempts rather than legitimate logins.[6] In addition, F5 Labs found that applications and identities are initial targets in 86% of breaches.[7] Attacks like credential stuffing are made possible by our persistent refusal to move past passwords, or more broadly adopt federated authentication.[8]

MITIGATING BROKEN AUTHENTICATION

One way to avoid the problems with passwords is not to use them. Client certificates, token-based two-factor and federated authentication are great ways to reduce reliance on passwords. Robust authentication can be difficult to build, secure, and maintain, and can negatively impact user experience, so consider leveraging federation to speed development and delivery of your application—and make users happy at the same time. Remember: no one wants yet another account and password to manage, no matter how cool your app may be.

Watch a video about: OWASP Top 10: Broken Authentication

[5] https://www.teamsid.com/worst-passwords-2017-full-list/
[6] http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/
[7] http://www.darkreading.com/attacks-breaches/credential-stuffing-attacks-take-enterprise-systems-by-storm/d/d-id/1327908
[8] https://f5.com/about-us/blog/articles/credential-stuffing-what-is-it-and-why-you-should-worry-about-it-24784

A3 SENSITIVE DATA EXPOSURE

Sensitive data exposure is an information leakage problem. The sensitivity of what is leaked can vary, but divulging any information to an attacker about a web application's design is a bad idea. This kind of information is low-hanging fruit for automated scanners and ripe for exploitation.

Some examples of the kinds of information commonly leaked by web applications that attackers find useful include the following:

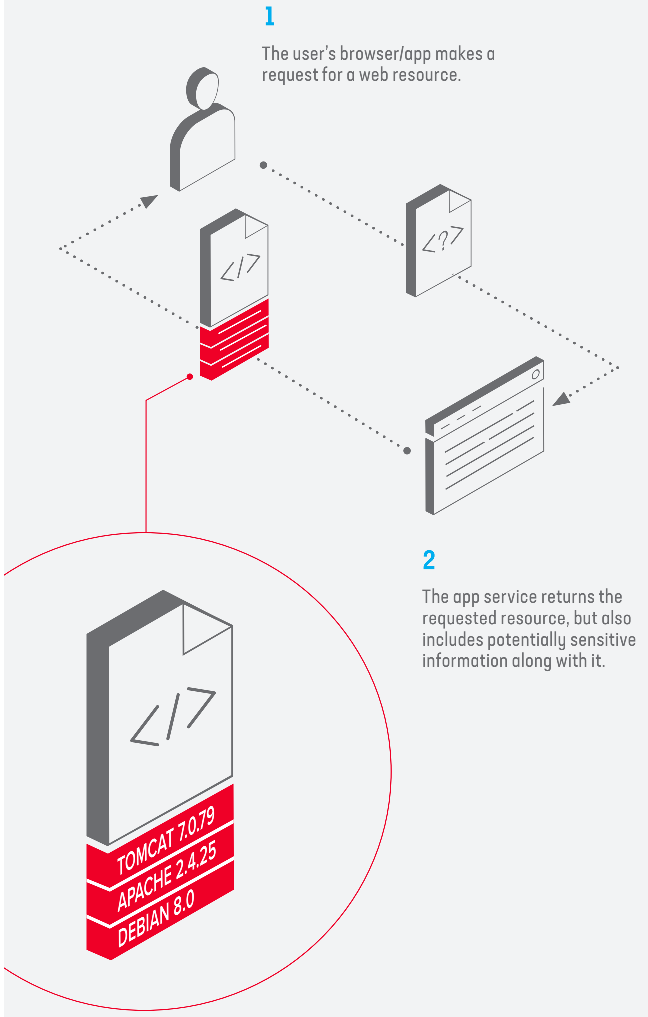
- Error messages detailing how unexpected input is handled
- Physical locations of files on server
- Specific versions of components and libraries
- Stack traces from failed functions (can be decompiled and examined)
- "Forgot password" function error messages that reveal user ID validity, which can be used for discovery and brute-force attacks on user accounts and passwords

MITIGATING SENSITIVE DATA EXPOSURE

There are several steps you can take to minimize your risk for data leakage. It is very common for web servers to report vendor and version information, among other things.[9] Make sure that usernames cannot be validated from server response codes: an incorrect username error and incorrect password should generate the same error message. Ensure browser security directives are used to help protect sensitive data in transit. Avoid old and known weak cryptographic algorithms and methods. Transport Layer Security (TLS) is easy to use and is becoming the universal norm on the Internet.[10] All sensitive data stored within a web application should be rendered unreadable—using techniques such as encryption or tokenization[11]—in case an attacker gains access through the application.

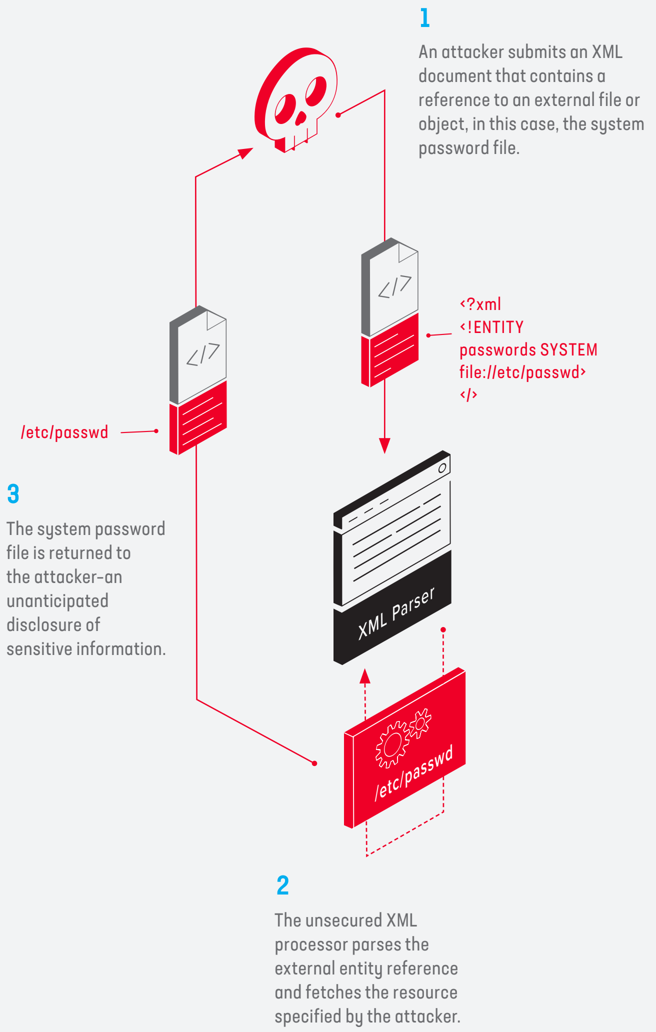
Watch a video about: OWASP Top 10: Sensitive Data Exposure

WEBSITES OFTEN RETURN MORE DATA THAN IS NECESSARY, WHICH GIVES ATTACKERS ADDITIONAL INFORMATION TO USE AND EXPLOIT.


1
The user's browser/app makes a request for a web resource.
</?>
</>
2
The app service returns the requested resource, but also includes potentially sensitive information along with it.
</>
TOMCAT 7.0.79
APACHE 2.4.25
DEBIAN 8.0

[9] https://f5.com/labs/articles/threat-intelligence/identity-threats/phishing-for-information-part-4-beware-of-data-leaking-out-of-your-equipment
[10] https://f5.com/Portals/1/PDF/labs/R065%20-%20REPORT%20-%20The%202016%20TLS%20Telemetry%20Report.pdf
[11] https://www.owasp.org/index.php/File:Reducing_Your_Data_Security_Risk_Through_Tokenization.pptx

APPS THAT ACCEPT XML INPUT CAN INADVERTENTLY ALLOW EXTERNAL COMMANDS THAT CAUSE XML PROCESSORS TO DIVULGE DATA.


1
An attacker submits an XML document that contains a reference to an external file or object, in this case, the system password file.
<?xml
<!ENTITY
passwords SYSTEM
file://etc/passwd>
</>
/etc/passwd
3
The system password file is returned to the attacker–an unanticipated disclosure of sensitive information.
XML Parser
/etc/passwd
2
The unsecured XML processor parses the external entity reference and fetches the resource specified by the attacker.

A4 XML EXTERNAL ENTITIES

Insufficiently hardened or misconfigured XML parsers or processors evaluate external entity references with XML documents. Apps, such as SOAP services, that accept XML input can inadvertently allow for inclusion of unanticipated external references and commands that cause XML processors to divulge data on internal file shares, execute code, initiate internal port scanning, and perform denial of service (DoS) attacks.

MITIGATING XML EXTERNAL ENTITIES

As digital transformation increases organizations' reliance on third party integrations that improve speed to market, the API becomes the focal point of the business. API security requires richer context, analytics, identity, and correlation capabilities to be effective. Carefully consider which APIs to expose in the first place, and ensure that vulnerable XML processors are upgraded, patched, or otherwise hardened. Once that is done, ensure that HTTP message sizes, versions, and methods are all enforced. Using a web application firewall (WAF) gives you the option to whitelist appropriate requests or block malicious ones from being sent to your XML processor in the first place. Also, a WAF can learn and validate JSON, XML, and SOAP requests, and apply protections against known attacks that can cause denial of service and unauthorized information disclosure. Finally, a network firewall and traffic management can limit outbound requests to other endpoints, internal or external.

Watch a video about: OWASP Top 10: XML External Entities

A5 BROKEN ACCESS CONTROL

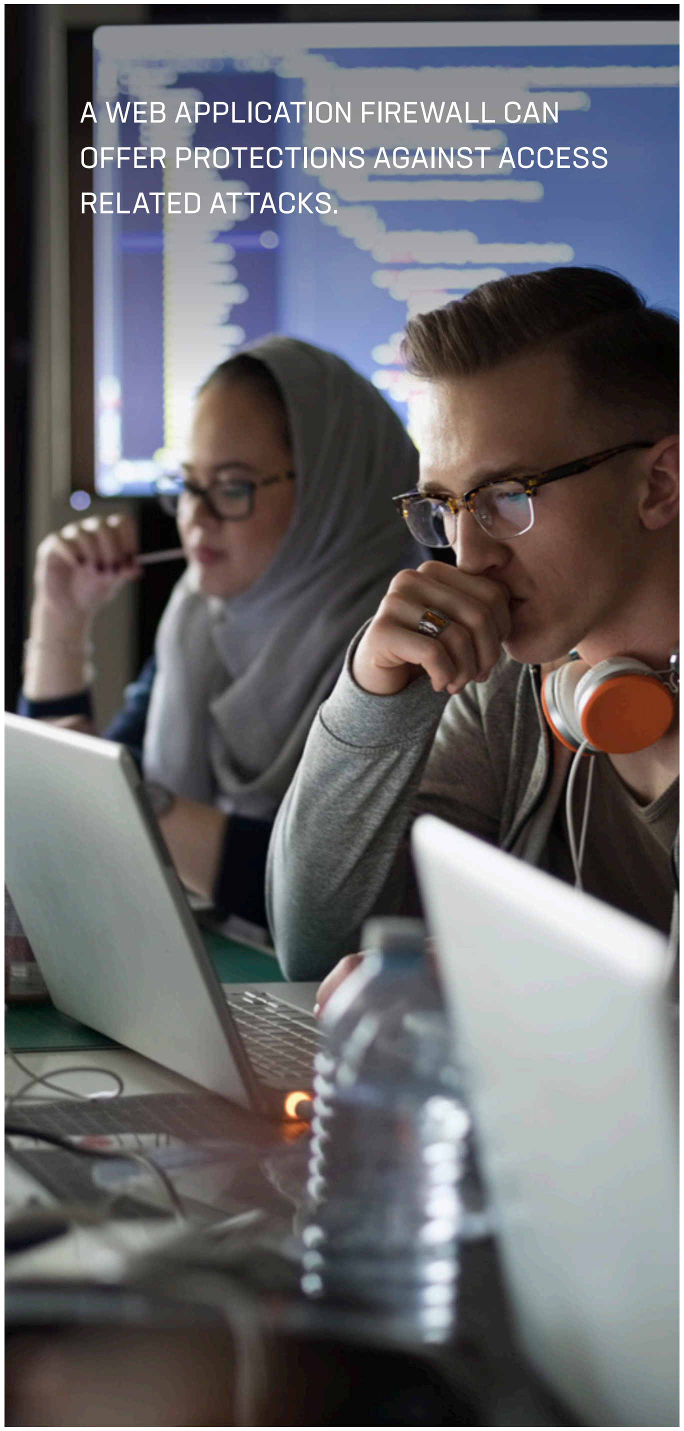
A broken access control vulnerability refers to a flaw in the design of the web application where unauthorized access to a sensitive object (like a directory or record) is improperly or insufficiently enforced. For example, it could be that any anonymous user could view certain files on a website simply by knowing what URL to request; or the application could execute a function that assumes some level of authentication or authorization has occurred without first validating that that is the case.

MITIGATING BROKEN ACCESS CONTROL

All web application objects and pages should have an enforcement mechanism that denies access by default. From there, the system should enforce explicit rights and only grant them to users associated with specific user roles.

A web application firewall can offer protections against access related attacks. Examples include directory traversal attacks, where an attacker attempts to access files and directories that are stored outside the web root folder, attacks that attempt to modify the URL, or attacks that use an API attack tool.

Watch a video about: OWASP Top 10: Broken Access Control


A WEB APPLICATION FIREWALL CAN OFFER PROTECTIONS AGAINST ACCESS RELATED ATTACKS.

93%

OF ATTACKS ON WEB APPLICATIONS WERE FINANCIALLY MOTIVATED, PERPETRATED BY ORGANIZED CRIMINAL GROUPS.[12]

A6 SECURITY MISCONFIGURATION

Security controls can be considered misconfigured for a variety of reasons, but a common cause stems from errors or omissions in user-facing documentation that result in gaps in controls or missed steps. It could also be oversight and/or mistakes made by systems administrators who are, after all, only human. Dependent software and infrastructure can also be overlooked. Most web applications depend on other software (such as Apache, IIS, or Nginx) and may leverage other applications, libraries, and databases (such as PHP, ASP, or SQL).

This problem has been exacerbated by a multi-cloud world, where the security level of default configurations (e.g. web servers, access control rules, etc.) can vary by cloud provider.

MITIGATING SECURITY MISCONFIGURATION

To properly secure a web application, not only must the software itself be properly locked down, but so must all the other integral components. It is also important to conduct regular and thorough audits to ensure that controls have been implemented properly and remain firmly in place.

Proper configuration management can also help alleviate potential one-off misconfigurations, but continuous vulnerability scanning is the key to ensuring nothing is missed. Results can be automatically fed into a WAF to provide immediate protection rather than waiting on a time intensive fix. Also, because any modification to code—whether to resolve a vulnerability or fix a defect—can introduce additional vulnerabilities, a WAF is a critical component to reduce risk and improve remediation time.

Watch a video about: OWASP Top 10: Security Misconfiguration

[12] http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/

A7 CROSS-SITE SCRIPTING (XSS)

Cross-site scripting (XSS) refers to an input validation vulnerability that lets attackers run their own malicious scripts in a victim's browser within the trusted context of a site they're visiting. XSS can be used to steal session tokens, initiate hidden transactions, or display falsified or misleading content. More sophisticated XSS scripts can even load key loggers to monitor victims' passwords as they type them in, relaying that information to command-and-control servers operated by the attackers.

XSS can occur anywhere an external user can contribute content to a website, which makes it one of the most common types of vulnerabilities. XSS is also hard to identify and eliminate because it uses the same HTML commands required by a website to render its pages. Furthermore, XSS attacks can be encoded in a variety of ways. For example, a basic attack script to pop up the message "XSS" on a page could look like this:

<script>alert('XSS')</script>

But encoded, it could also look like this: %253Cscript%253Ealert('XSS')%253C%252Fscript%253E

or this: <IMG SRC="jav&#x0D;ascript:alert('XSS');">

or even this: <IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>

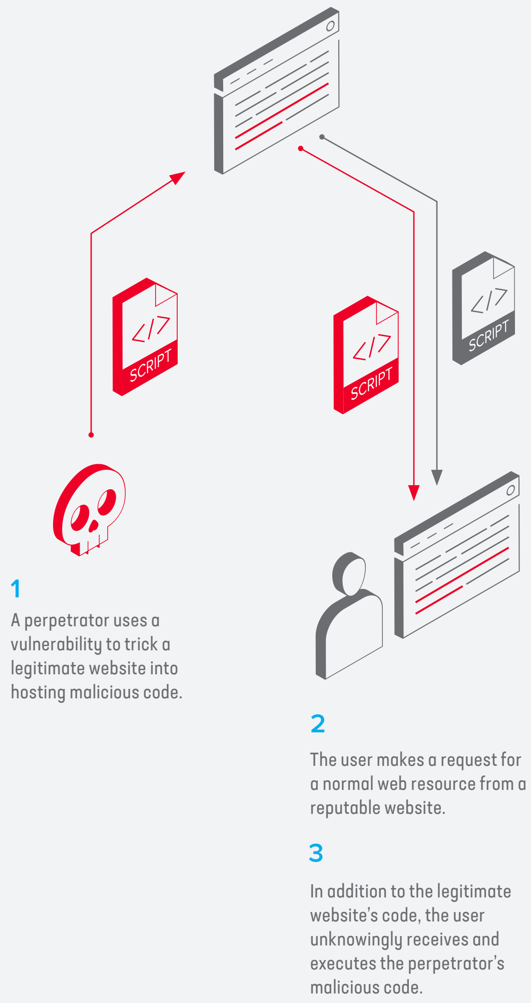
With all these encoding possibilities supported by native browsers, it's easy to see how XSS vulnerabilities can slip through. Complicating things further is the fact that there are a number of different XSS delivery methods that can be employed, making this a very flexible attack vector.

MITIGATING CROSS-SITE SCRIPTING

Stopping XSS goes back to the idea that any external data supplied by the user is untrustworthy. Since any data input can have a malicious payload embedded in it, it is imperative to filter everything coming in, although this can be very difficult to accomplish comprehensively on your own. Fortunately, a good WAF can do it for you, freeing up time and resources with infrastructure that is reusable. Other tools like XSS static code analysis can be a first line of defense within a CI/CD pipeline.

Watch a video about: OWASP Top 10: Cross-Site Scripting (XSS)

XSS CAN OCCUR ANYWHERE AN EXTERNAL USER CONTRIBUTES CONTENT TO A WEBSITE, MAKING IT ONE OF THE MOST COMMON TYPES OF VULNERABILITIES.



</>
SCRIPT
1
A perpetrator uses a vulnerability to trick a legitimate website into hosting malicious code.
2
The user makes a request for a normal web resource from a reputable website.
3
In addition to the legitimate website's code, the user unknowingly receives and executes the perpetrator's malicious code.

MANY PROGRAMMING LANGUAGES OFFER NATIVE SERIALIZATION OR ALLOW CUSTOMIZATION OF THE SERIALIZATION PROCESS, WHICH CAN IN TURN BE USED MALICIOUSLY.

A8 INSECURE DESERIALIZATION

This is a new addition to the top 10, and deals with object serialization, which is the process of turning an object into a data format that can be restored later. Think about how files are saved to a disk or how data is transferred over a network. The data is saved in a given structure using formats such as JSON or XML. Deserialization is the opposite—reading this structured data and building an object from it.

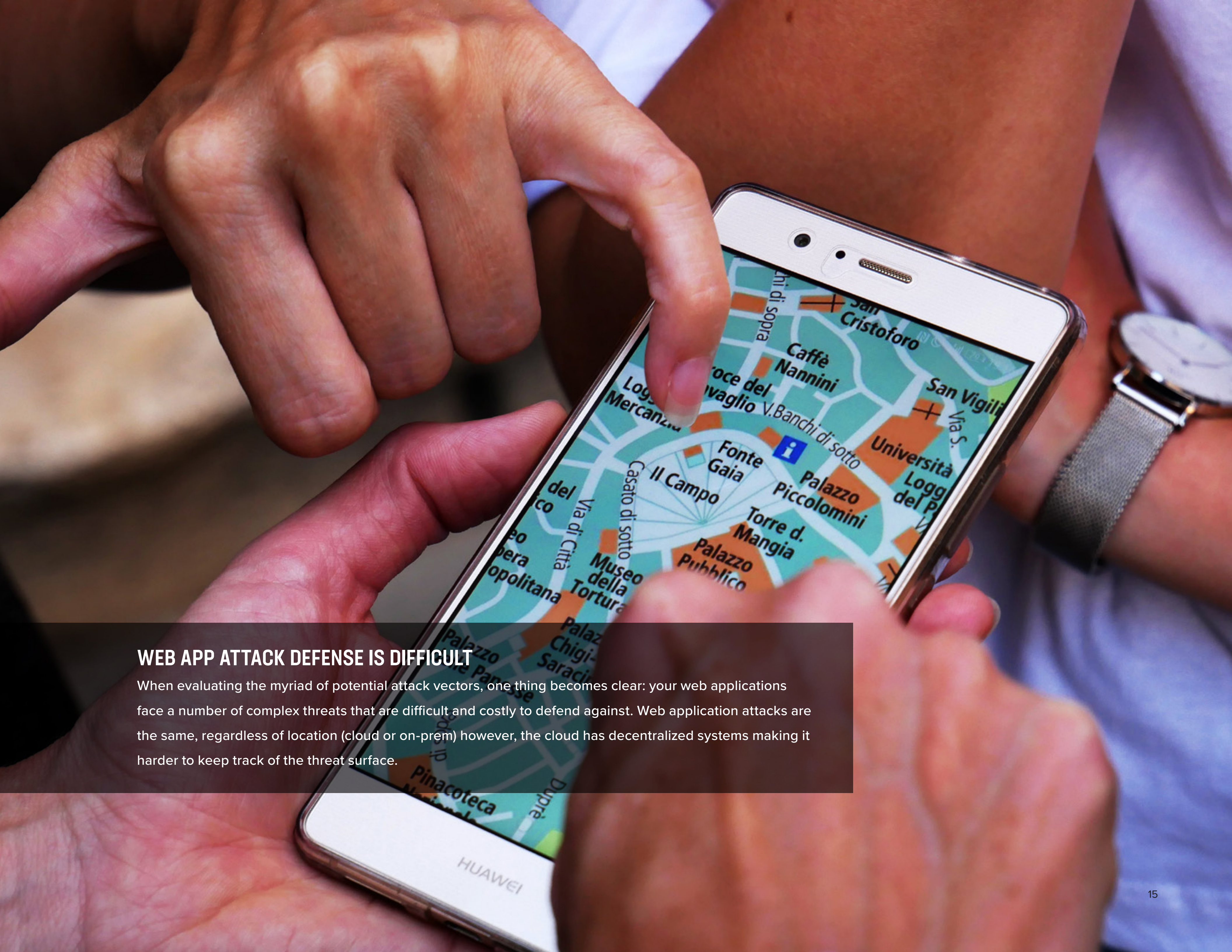
Many programming languages offer native serialization or allow customization of the serialization process, which can in turn be used maliciously. Insecure deserialization has been found to allow remote code execution, denial-of-service, replay, injection, and privilege escalation attacks.

One recent example involved one of the most popular web forum software packages on the market. An anonymous bug hunter shared an 18-line python script with which attackers can remotely execute commands on a target server with no authentication. From there, they can steal data, tamper with information, and launch attacks on other systems. The simple HTTP POST request to a vulnerable host allows full control depending on the service’s user permissions. The published exploit code returns its successful execution in a JSON formatted response. The payload is sent, the server runs the command, and it replies with whatever the attacker asks for.

MITIGATING INSECURE DESERIALIZATION

To protect against a deserialization attack, you should not accept serialized objects from untrusted sources. When that isn’t possible, you can enforce HTTP message sizes, versions, methods, and required headers. Also, an advanced WAF can validate JSON, XML, and SOAP requests against known attack signatures as well as validate data length, value, and structure. Your WAF should also allow you to customize handling of data designated as sensitive.

Watch a video about: OWASP Top 10: Insecure Deserialization



WEB APP ATTACK DEFENSE IS DIFFICULT

When evaluating the myriad of potential attack vectors, one thing becomes clear: your web applications face a number of complex threats that are difficult and costly to defend against. Web application attacks are the same, regardless of location (cloud or on-prem) however, the cloud has decentralized systems making it harder to keep track of the threat surface.

IF YOU'RE DEPLOYING APPS IN THE CLOUD, VULNERABILITIES INCREASE IN COMPLEXITY AND BECOME MORE COMPLICATED TO MANAGE BECAUSE OF ALL THE DEPENDENCIES.

A9 USING COMPONENTS WITH KNOWN VULNERABILITIES

This is another one of those risk areas that may seem obvious, but it is worth addressing since many software components are chosen solely for their utility in fulfilling some basic operational requirement. Such components may not have known vulnerabilities when implemented, and it is common for there to be resistance to upgrades out of fear of breaking functionality or losing a valuable legacy feature.

These are valid concerns, but a successful exploit against a known security vulnerability can result in a significant loss of service or breach of customer confidence, so this risk must be weighed against the perceived risks of upgrading.

If you're deploying in the cloud, you should be aware that vulnerabilities are increasing in complexity, and are becoming incredibly difficult to manage due to the sheer number of dependencies. The breach of a small, seemingly insignificant vuln can have a massive impact because of the level of abstraction in clouds, flatter networks, and poor access control.

Recent research has found three times more high/critical vulnerabilities in external providers—including cloud service providers and other services such as co-location and managed hosting—compared to on-premises.

MITIGATING USING COMPONENTS WITH KNOWN VULNERABILITIES

The key to a secure environment is to keep components updated wherever you can. Where you can't, you can use compensating controls such as a strong WAF to block the exploitation of known vulnerabilities, while keeping your software intact and operational. Software composition analysis tools provide an open source component inventory along with their associated vulnerabilities, and a WAF can buy you time while patches are developed and rolled out—and it also protects from any subsequent attack variants that are common when exploit code hits the web.

Watch a video about: OWASP Top Ten: Using Components With Known Vulnerabilities

A10 INSUFFICIENT LOGGING AND MONITORING

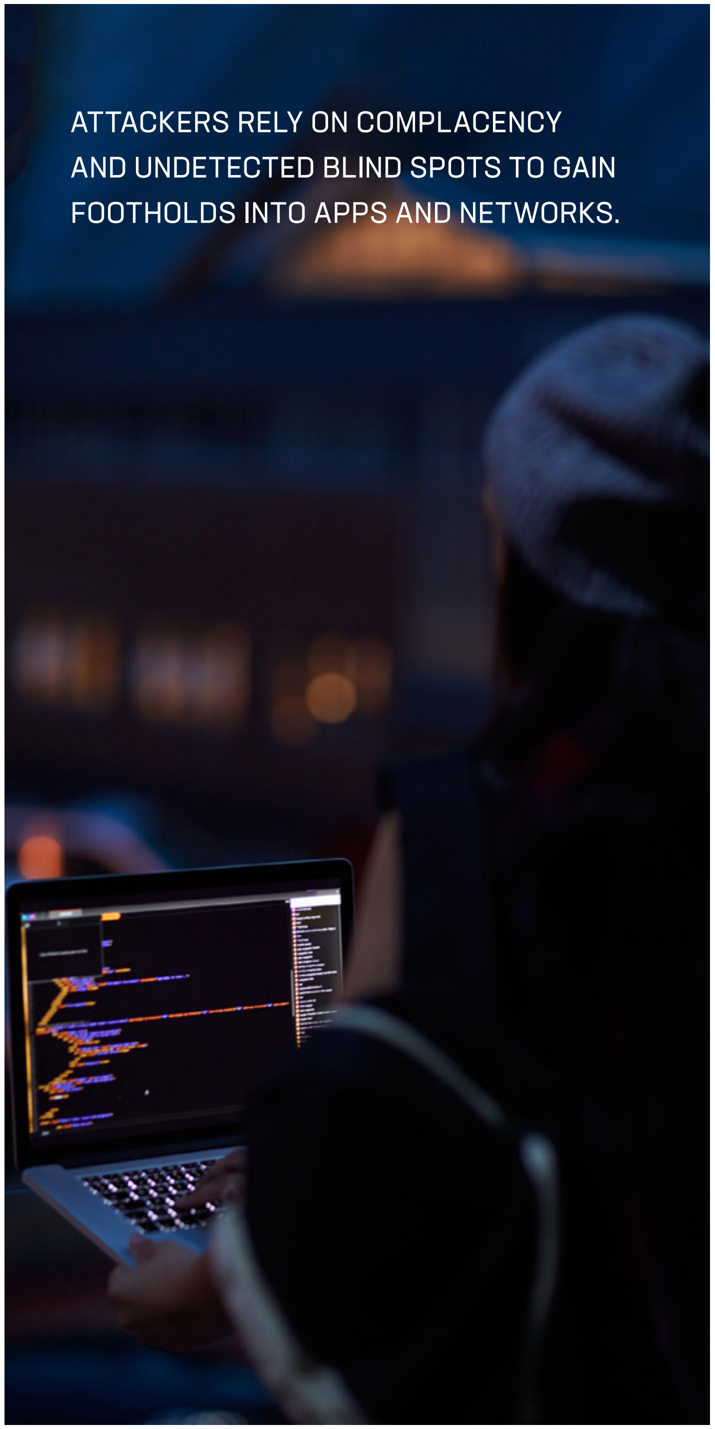
While most applications are built to log some level of access and authorization information, many are not, or do not do so by default. The risk here is that log data and access information that is not collected or not analyzed will not help detect a breach, or facilitate expedient service recovery in the event of an incident.

Attackers rely on complacency and blind spots to remain undetected, and gain footholds into apps and networks. Diligent monitoring of this data can help detect attacks early, allowing you to build or further fortify your defensive posture, and ultimately, minimize damage or impact to your organization.

MITIGATING INSUFFICIENT LOGGING AND MONITORING

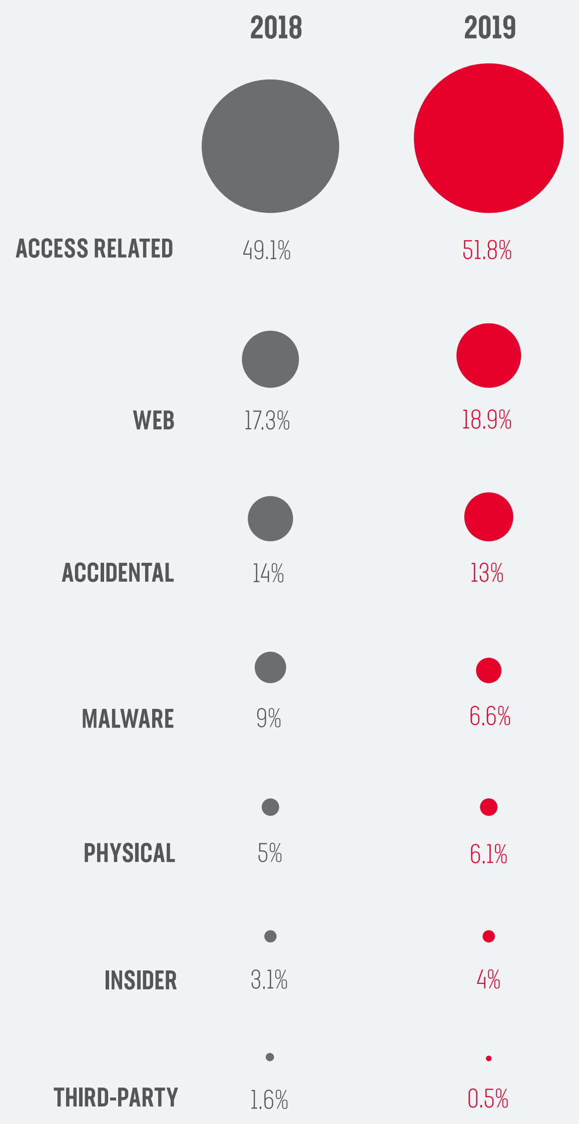
Start by identifying which apps and endpoints are likely to generate the most useful logging information and can be relied on to provide early warning of anomalous behavior. You will need to define what must be collected and how it should be analyzed and monitored. Good WAF solutions will allow you to standardize logging for all of your web applications—and to log that information off-box for further analysis and reporting.

Watch a video about: OWASP Top Ten: Insufficient Logging and Monitoring


ATTACKERS RELY ON COMPLACENCY AND UNDETECTED BLIND SPOTS TO GAIN FOOTHOLDS INTO APPS AND NETWORKS.

2018–2019 U.S. BREACHES BY CAUSE

Distribution of causes of U.S. breaches in 2018, by breach count. The lack of detail in the breach reports means that there is partial overlap between many of these categories.



THE OWASP TOP 10: **ONLY ONE PIECE OF THE APP SECURITY PUZZLE**

When evaluating the myriad of potential attack vectors, one thing becomes clear: your web applications face a number of complex threats that are difficult and costly to defend against. Most development teams simply do not have the resources to sufficiently protect against the variety of attacks that are relevant to each vector—and the level of expertise required is such that it will be difficult to come by even if your project has the time and budget for it.

The good news is that advanced WAF technology is more accessible and affordable than ever before. Modern, full-featured WAFs can help organizations of all sizes defend their critical apps—whether they're deployed in data centers or hybrid cloud environments. Unique and flexible deployment options can simplify implementation and make it easy to customize protection for your app.

While you're addressing your security needs to protect against the OWASP Top 10, it's also essential to consider all the other threats to your applications: DDoS attacks, bot-enabled attacks, intellectual property theft, and fraud, just to name a few.

JUST BECAUSE A THREAT DIDN'T MAKE THE OWASP TOP 10 DOESN'T MEAN IT IS NO LONGER RELEVANT.

Take, for instance, cross-site request forgery (CSRF), which involves tricking a victim using a browser into clicking on a benign-looking link that actually performs a malicious action, such as initiating a transfer within your banking app. CSRF doesn't appear on the list this year, but you can be sure that criminals are still out looking for opportunities to exploit it—as well as innumerable clever gambits—to compromise your apps. Defending against the OWASP Top 10 is a great—and necessary—step, but it's only one piece of a comprehensive, defense-in-depth strategy that will help you protect your apps, your data, and your business.

Full overview: OWASP Top Ten - 2017

For more information on the threats that affect your applications and your organization—as well as what you can do to defend against them—visit f5.com/security.

THINK APP SECURITY FIRST

Always-on, always-connected apps can help power and transform your business—but they can also act as gateways to the data beyond the protections of your firewalls. With most attacks happening at the app level, protecting the capabilities that drive your business means protecting the apps that make them happen.

Find more security resources at f5.com/solutions