



SECURITY PHOENIX LTD

NSC42

STATE OF APPLICATION SECURITY

AN UNBIASED VIEW ON THE STATE OF APPSEC INDUSTRY

NSC42 LTD
Kemp House, 152 City Road, EC1V 2NX, London

Intro

This document is a collaborative document that aims to include the thoughts of modern appsec leaders and leading organisation to show the state of application security.

The document is targeted to both executives and practitioners. Like the [Verizon data breach report](#), this report will have an executive summary and a more detailed section.

This document is an open contribution from outstanding expert around the industry if you would like to contribute more reach out to info at <https://www.nsc42.co.uk> ro appsec@nsc42.co.uk

Why Read this report

This report is community-driven with a data-driven approach to application security. The report will refer to other reports currently on the subject (e.g. Forrester, Sonatype, Snyk, WhiteSource and other reports) as well as broader subject (Verizon data breach report)

Thank You notes

A massive thank you to all the authors, contributors, editors, and reviewers of the document. This document truly embodies the power of the information security community full of exceptional selfless and inspirational professionals.

Document Licence- Creative Common

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 License. Kindly refer to this work if you want to add a reference.

All images under copyright shall not be reproduced. Any image with attribution shall be shared in the same way.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/legalcode> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA

Executive Summary

Application security is a growing concern for Board and organisations. We've seen a rise in focus on application security as more and more elements in the organisation are becoming code-driven.

According to a recent survey carried out on C-suite users, a total of 53% of respondents indicated "cybercrime and data breaches" as the number one concern when it comes to cybersecurity. [[IBM Study](#)]

So why criminals (not a hacker) attack an organisation? Well mostly for financial reason, even though there are exceptions, (see later in the report).

[Verizon's Data Breach Investigations Report \(DBIR\)](#) finds that 86% of data breaches are financially motivated—up 15% over the previous year. In contrast, espionage—the second-highest motive—declined from 2018 to 2020.

So how much of the vulnerabilities being exploited are due to web applications, code or similar are being exploited?

This report focuses on an industry view of application security. The report takes from various sources (cited) and aims to be a point of contact for application security. Application security, code security seems to have particular attention in the latest few years.

The Verizon report also reveals that web applications show up as a vector to which cybercriminals have increasingly turned their attention over the past year: 43% of all data breaches analysed by Verizon this past year were the result of a [web application vulnerability](#).

With all the things becoming code (see infrastructure as code, containers being part of the build, serverless) the focus of the security team is right...but how we can extend this knowledge to the engineering teams? Why the lack of code security? Is it because of experience?

This is a worrying statistic, as over 54% of the world's organisations have experienced some sort of significant cyber-attack in the past year. [[IBM](#)]

In 2018-2019, almost 53% of organisations reported a problematic shortage of cybersecurity skills. [[Security Intelligence](#)]

With the complexity of code coming from different sources (stack traces), different teams developing different part of the application is always difficult for a security team to scale. The report focus on data breaches statistics and how they are linked to application security and further dive into the potential methodologies (the HOW) and solution (the WHAT).

The first part of the report aims to address the high-level statistics that could be used to justify an application security programme or the introduction of tools to perform application security in a development pipeline. The time for the developer to fix code is key as the later a vulnerability is fixed the more it will cost.

In 2019, 64% of companies that allocate more than 10% of their budget towards cybersecurity experienced at least one breach. 34% of the companies indicated that they experienced a data breach last year. [[Helpnet Security](#)]

This report aims to enable the C-suite team to allocate focus and found appropriately and with the hope that data breach will be less and less of a problem and development teams will be given the time, tool and knowledge to fix vulnerabilities before they enter and threat the organization clients.

Data Breaches over time

Author - Francesco Cipollone

Data breaches are the modern cybersecurity problem of those days. We have malicious actors of all sorts attacking organizations for different reasons, but mostly those can be categorized in

- Hactivism – damaging an organization for a cause
- Direct financial gain – damaging an organization to steal easily usable information (e.g. credit card)
- Indirect financial gain – stealing information to then resell them on the dark web for a price or attack other organizations or subsidiaries.

Well Known/Hactivism



Anonymous, Lizard Squad,
LulzSec, Chaos Comp Club,
Syrian Elect Army

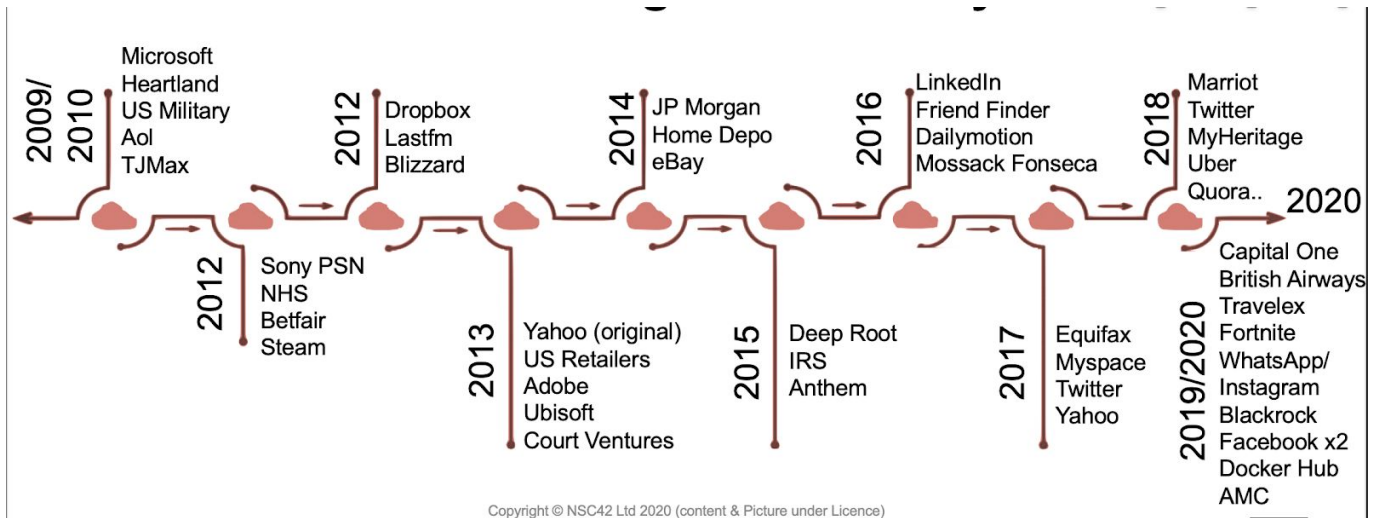
Financially motivated

Gold Southfield -
ReEvil, Magecart,
Lazarus group ...

Less visible more
dangerous

Copyright © NSC42 Ltd 2020 (content & Picture under Licence)

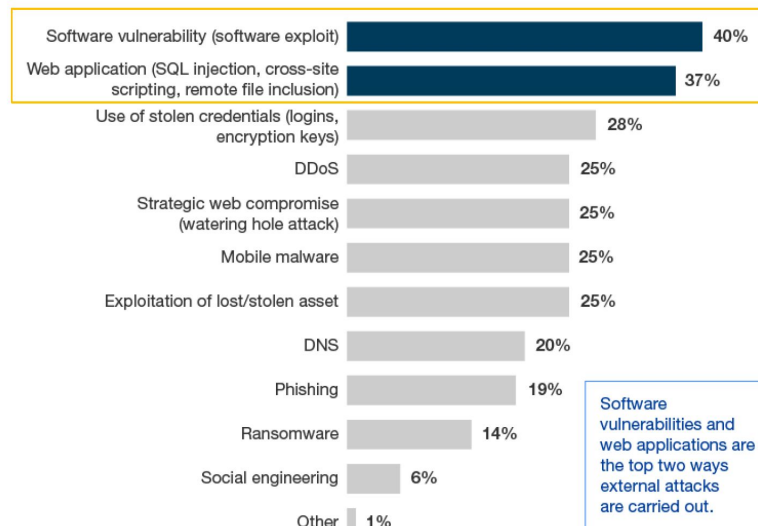
Naturally, there are more nuanced than this but we could write otherwise a book talking about nuances. The picture below displays just a quick timeline of data breaches over time. Nonetheless, the data breaches are not limited only to those



According to the Forrester report, 40% of successful data breaches are due to Application security vulnerabilities of some sort. While some other reports highlight the prevalence of Phishing and social engineering the application security element of data breaches is undeniable

“How was the external attack carried out?”

(Multiple responses accepted)



Base: 283 Enterprise global network path security decision makers who experienced an external attack when their company was breached

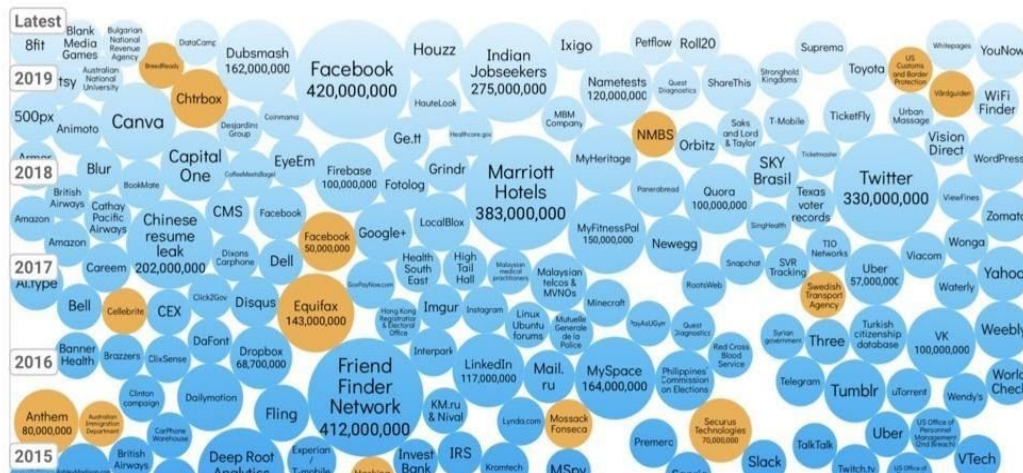
Sources: Forrester Analytics Global Business Technographics® Security Survey, 2019

Without fully diving in the latest [Verizon data breach report](#), report some of the highlights:

- **56% of breaches took months or longer to discover”**
- **“Errors were causal events in 21% of breaches”**
- **“breaches with compromised payment cards are becoming increasingly about web servers”**

- “The web application compromises are no longer attacks against data at rest. **Code is being injected to capture customer data as they enter it into web forms.**”

Data breaches have a huge impact on an organization. The picture below from Information is beautiful provides a good indication of how widespread those data breaches are



Source: <https://informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/>

See the data:

<https://docs.google.com/spreadsheets/d/1i0oIJJMRG-7t1GT-mr4smaTTU7988yXVz8nPlwaJ8Xk/edit#gid=2>

The monetary impact of data breaches is high, without taking into consideration regulators or GDPR fines the average data breach costs roughly 3 ml dollars. Financial impacts are not limited to corporations. Figures from the European Union Agency for Network and Information Security (2016) showed Germany suffered losses estimated at 1.6% of Gross Domestic Product (GDP). The Netherlands suffered an estimated 1.5% loss of GDP. Insecure code has morphed from a localized business problem to a threat to national financial stability.

	Av Cost of data Breach	
	3.03 m	
Adversaries don't need many vulnerabilities - ONE is enough.	Every 36 minutes	N. Of Vuln per year 1400 vuln
Is your business equipped with the right tools to react fast enough?	A new security vulnerability is identified	As disclosed vulnerabilities
	It takes 150-180 days	It takes 3-15 days
	To fix a vulnerability	To exploit a vulnerability

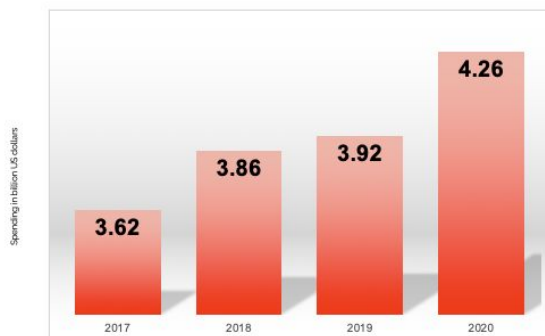
Source: [NSC42 & Security Phoenix statistics](#)

The reason why cybersecurity struggles as a field are that the application vulnerabilities get released. Naturally, those costs vary depending on the elements considered in a data breach. But overall the picture above displays a significant impact of a data breach and the relative speed at which the vulnerabilities (Appsec and not) get released and exploited (3-15 days as average).

Considering some historic data breaches you can see that the 3m average number can be easily surpassed, nonetheless consider that number to be a pure indicative reference as many statistics diverge from that number.

- Experts agree that by the year 2020, the average cost of a data security breach for a major business would be over \$150 million. This estimate is due to the higher level of digitalization and connectivity that the world has experienced over the last few years. [[BigCommerce](#)]
- The average total cost per data breach worldwide in 2019 amounted to a total of \$3.92 million and \$3.5 million in 2014. [[IMB](#)]
- The average price for a Business Email Compromise hack is \$24,439 per case, according to a 2019 report by Verizon. [[Verizon](#)]

Av Cost of data Breach
3.03 m



AVERAGE COST OF DATA BREACH IN MILLION OF DOLLARS

The number of data breaches reported since 2017 to regulators has increased 480%.

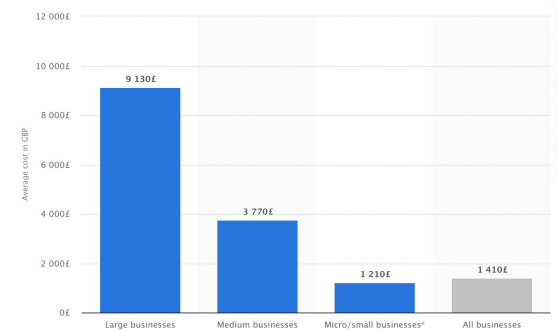
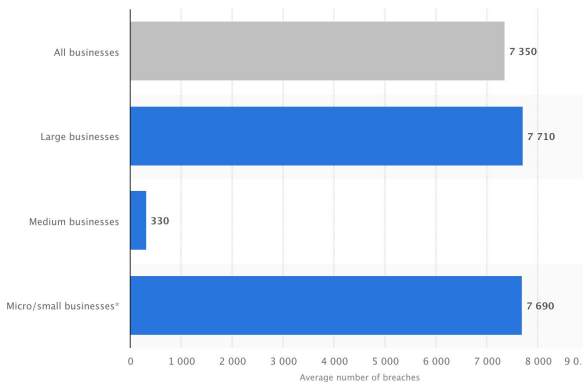
71% of breaches in 2019 were financially motivated

Company	Net Loss (\$US)	Settlement/Regulatory Fine	No. of Customer Affected
eBay	2.33 billion, \$1.82/share	\$650million	147million
Equifax	\$555.9 million	\$700million	147million
Capital One	\$150million	TBD	106million
Travelex	£4.6m ransom + 30 days inactivity	TBD	
Average	the average total cost of a breach now stands at £3.03m		

Source: [NSC42 & Security Phoenix statistics](#)

In the UK the data breaches numbers are lower but not insignificant by any means...

Large business and small businesses are the one more affected according to [Cyber Security Breaches Survey 2019, page 45](#)




Details: United Kingdom; Ipsos MORI; Institute of Criminal Justice Studies; October 10, 2018 to December 20, 2018; Respondents are UK businesses that identified a breach or attack in the last 12 months; Telephone interview

Source: [Cyber Security Breaches Survey 2019, page 51](#)

The average cost for the UK for data breaches might be lower than the overall average nonetheless it's still quite an impacting number. The cost does not consider potential GDPR fines (2-4%) overall revenue as well as brand image and customer impact

Some of the Recent Data Breaches

Top breaches by misconfiguration



Common Mistakes

The common theme around all those breaches are

1. Misconfiguration
2. Open Source Libraries Injection/Vulnerabilities
3. Common Vulnerabilities (OWASP Top 10)
4. Public Storage (data left unauthenticated)
5. Easy to guess credentials (cred Stuffing) - Collection X anyone?
6. No MFA on critical accounts
7. Unchecked Open source software
8. Break of logic (API abuse)

Scale of Equifax breach

The Equifax data breach was one of the largest in history. The company announced the data breach in September 2017, eventually reporting that 147 million consumers were affected. Hackers were able to get access to a multitude of consumer private information, including names, Social Security numbers, dates of birth, credit card numbers and even driver's license numbers.

Source: [NSC42 & Security Phoenix statistics](#)

eBay

E-Bay in 2014: The data breach was carried out using stolen login credentials from a small number of employees. A total of 145 million eBay accounts were compromised. [[Business Insider](#)]

Marriot

Marriott International in 2014/2018: The breach occurred due to unauthorized access to the guest's information database. As a result, over 500 million user accounts were compromised. [[Forbes](#)]

Capital One - Misconfiguration

106 million records exposed due to misconfiguration

Capital One is the 10th largest bank in America on the basis of assets. It uses the revered Amazon Web Services (AWS) as its cloud solution. Here is the chain of fateful events leading to the breach.

Access token was then used to fetch data from AWS storage.

700 folders and data packets containing customer info were copied to an external location.

British Airways (2018)

RiskIQ supported the Magecart claim by showing that the British Airways website had a third-party (but self-hosted) script which appeared to have been modified to include code to exfiltrate payment data from the payment page.

Facebook via 3rd party app

Third-Party Facebook App Data Exposure – 540 Million Records

Facebook in 2018: This data breach was caused after hackers exploited a vulnerability in Facebook's "View As" code. They were left with 50 Million compromised accounts. [[The Guardian](#)]

Special Mentions:

Equifax (Special mention) – Breach of the database via Apache Struts

months old vuln

Equifax in 2017: The data breach occurred as a result of a vulnerability in the open-source software used to access its servers. As a result, the personal information of 143 million consumers was exposed.

[[Forbes](#)]

hackers used an Apache Struts vulnerability, [a months-old issue that Equifax knew about but failed to fix](#), and gained access to login credentials for three servers. They found that those credentials allowed them to access another 48 servers containing personal information.

Slow exfiltration of records (76 days)

Nintendo

Nintendo – Credential Stuffing - April 27, 2020:

A credential stuffing attack using previously exposed user IDs and passwords of popular video game company, [Nintendo](#), granted hackers access to over 160,000 player accounts. With unauthorized access to the accounts, the fraudsters may have purchased digital items using stored cards as well as view personal information including name, date of birth, gender, country/region and email address.

Collection 1

January 17, 2019: Security researcher [Troy Hunt discovered a massive database](#) on cloud storage site, MEGA, which contained 773 million email addresses and 22 million unique passwords collected from thousands of different breaches dating back to 2008. The information was shared on a popular hacking forum where they could be shared with other cyber thieves. If you're concerned your credentials may have been compromised, visit [Have I Been Pwned?](#)

MGM

February 20, 2020: Over [10.6 million hotel guests](#) who have stayed at the MGM Resorts have had their personal information posted on a hacking forum. The data dump exposed includes names, home addresses, phone numbers, emails, and dates of birth of former hotel guests.

Main Section

Table of Contents

Why Read this report	1
Thank You notes	1
Document Licence- Creative Common	1
Data Breaches over time	3
Some of the Recent Data Breaches	7
eBay	7
Marriot	8
Capital One - Misconfiguration	8
British Airways (2018)	8
Facebook via 3rd party app	8
Special Mentions:	8
Equifax (Special mention) – Breach of the database via Apache Struts	8
Nintendo	8
Collection 1	9
MGM	9
Appsec Leader Quote Section	12
Tanya Janca - Shehackspurple	12
Francesco Cipollone - NSC42	12
Grant Ongers - Secure Delivery	13
Andrew Peterson - Signal Sciences	13
Dr. Philippe De Ryck - Pragmatic Web Security	13
Methodology:	14
APPSEC programme	16
DevOps Intro	16
DEVSECOPS	17
Immature and Slow SDLC:	18
Good Reference guide:	19
Security Governance for the DESIGN phase	19
Security for the DESIGN phase	19
Good practices:	20

Security in Build/Test	20
Good practices:	20
Maturity Models	21
Roadmap to evolution	21
OWASP SAMM	23
BSIMM	24
NSAMM	25
What are the key pillars of NSAMM (NSC42 adaptation of SAMM) and SAMM?	25
SDLC Security	27
Intro	27
Why Secure code is important and how to achieve it	28
Create a feedback loop between activities	29
Gradually push appsec further	30
Adopt a defense-in-depth strategy	31
Agile Testing Quadrants	31
Documented Benefits of a Secure SDLC	33
Type of Testing	34
Static code analysis	35
Benefits	36
Drawbacks	37
Static Code Analysis - Why should I?	37
Web Application Testing - a form of DAST	39
Automated vs Pentesting - differences between pen testing and automated testing	39
Web Application Testing - why should I	39
Documentation	41
Libraries and open source vulnerability analysis	42
Dynamic code analysis	42
Benefits,	43
Drawbacks	43
DAST Why should I?	44
Modern Application Security in Production	44
Introduction	44
Focus on bugs that matter	45
Build feedback loops	45
Deploy proactive web defense	46
Middle Ground	48

Closing Remarks	49
Thank You notes	49
Contributors	49
Bios	50
Francesco Cipollone	50
Dr. Chris Sellards	51
Tanya Janca	52
Nicole Becher	52
Dr. Philippe De Ryck	53
Vandana Verma Sehgal	53

Working Group & Sections

Appsec Leader Quote Section



Tanya Janca - Shehackspurple

The current state of application security is that we do not have enough qualified individuals, with relevant training and experience, to do all of the work that we need doing. According to the Verizon breach report (2016, 2017, 2018, & 2019), insecure software is the number one cause of data breaches, meaning we are far from succeeding at protecting our applications and information. I currently see a trend of westernized countries prioritizing profit over the safety and security of our data and systems. There is a lot of gatekeeping of the information and training required to become an application security engineer, and this is hurting us, not helping. I feel we must make education easier to obtain, and higher quality, if we are ever going to succeed at protecting ourselves against that ever-improving attacks from criminal and other online threat actors.



Francesco Cipollone - NSC42

“don’t be used by a tool, use the tool as a tool and or you’ll become a tool”

Developers are our first line of defence and it is our responsibility to give them the tools to make informed decisions based on risks.



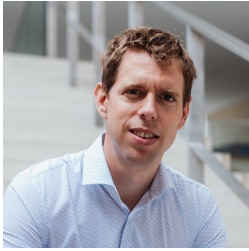
Grant Ongers - Secure Delivery

Because most breaches can be traced back to code and we have the data to show this, it’s clear that security is a non-functional requirement for good code and a question of code quality. The only way to improve the quality of that code is to ensure that developers know what good looks like (through awareness and education) and that they are empowered (through tooling and processes) to produce code that meets the mark.



Andrew Peterson - Signal Sciences

It would not be hard to argue that AppSec is the most difficult part of infosec today. Security needs to get out of our organizational silos and be proactive, helpful partners to the Application development teams who are in the midst of navigating a generational change in SDLC process and architecture. Ensuring that we have an awareness of how, where, and what attackers are doing to apps in production as well as having a clear bug identification and remediation strategy are both fundamental to building an effective defensive strategy that both development and security teams can carry out.



Dr. Philippe De Ryck - Pragmatic Web Security

“The key to building secure software is knowledge”

Even the most automated security pipelines rely on someone to interpret the results and take proper action, which boils down to security knowledge.

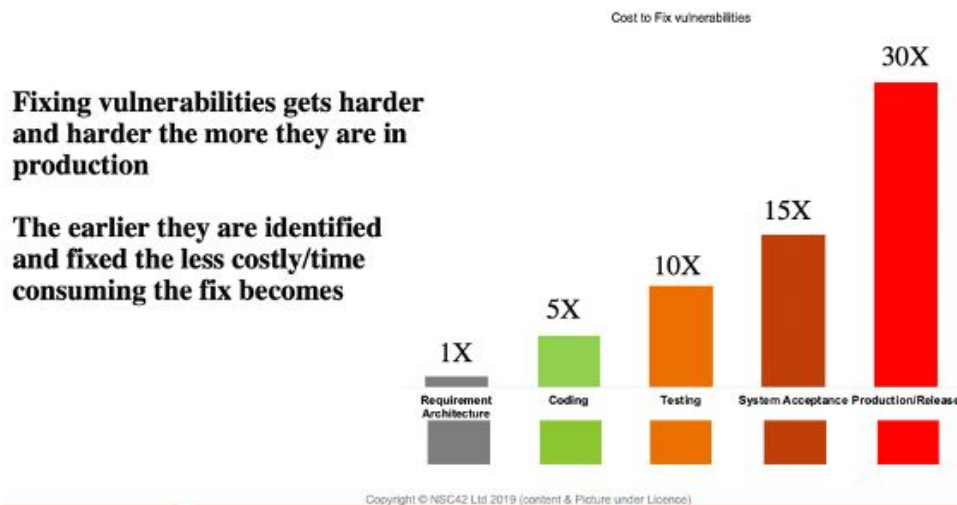
Section 1 - the HOW

This section focuses more on methodology and tools to insert in the pipeline to secure code, testing applications. The take on the code vs vendors will be unbiased. There is a lot of free and open-source software and the adoption of it depends on the complexity of the journey you want to embark on.

Methodology:

Authoring: Francesco Cipollone

Appsec and Devsecops are fundamentally linked but different with nuances. Application security (Appsec in short) is a collection of methodologies to fix code, Dev Sec Ops, in my opinion, is about having security early in the lifecycle but address different tools, methodologies and processes at different stages. The key point is fixing vulnerabilities as early as possible



Source: [NSC42 Appsec Program & Security Phoenix statistics](#)



Source: [WhiteSource](#)

if you need any other reason that the cost above here top 3 reasons:

1. Saving Money and Time - In my view one the biggest reasons to track security bug, before they become vulnerabilities, as early is because little problems can turn into bigger
2. Interdependence - the closer the code goes to production and the more it stays in production the more people will rely on it. Ultimately the more dependencies on your API, Data sets and code the harder and longer the testing will get.
3. Fixing security bugs sets intention and demonstrate excellence - others will follow suit and example if they see fixing security is not an impossible and insurmountable task

The real power of a leader is in the number of minds he can reach, hearts he can touch, souls he can move, and lives he can change.” — Matshona Dhliwayo

APPSEC programme

Francesco Cipollon and Kim Crawley

DevOps Intro

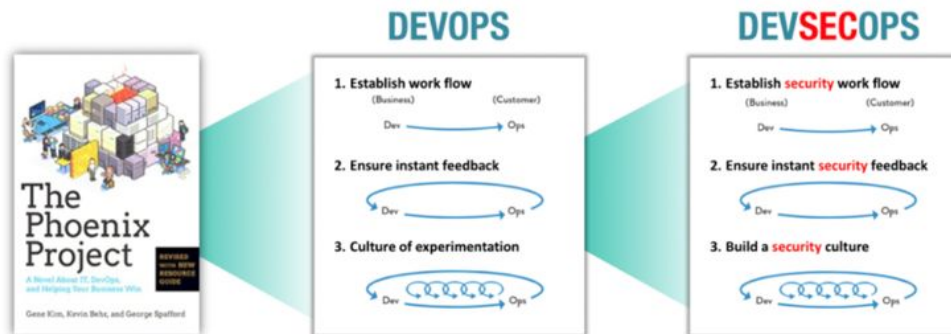
Author - Francesco and Kim

What kind of animal is the DEV-SEC-OPS?

Integrate security into the OPS team (and add a spark of RISK)



DevOps is the preferred way to develop applications, now that cloud platforms have given organizations better scalability and flexibility than ever before. By integrating development with operations, DevOps is the most responsive way to adapt to your organization's ever-changing needs.



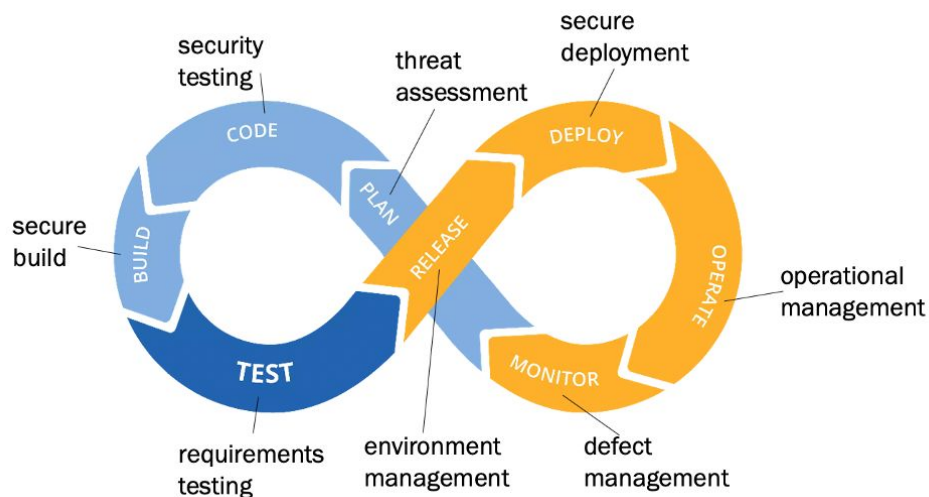
Source: DevOps to DevSECops All Day DevOps talk

Synopsys launched the Building Security in Maturity Model (BSIMM) in 2008.

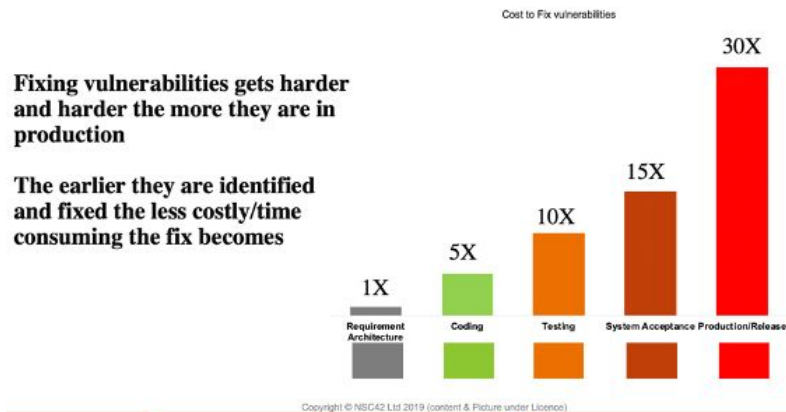
How do application security programs pertain to DevSecOps?

DevSecOps is a methodology and your app sec program implements it.

[John Allspaw & Paul Hammond \(Velocity 2009\)](#)



Without going too in-depth on DEVOPS process the following represents the security element in the famous CI/CD DevOps loop



So, we combine development and operations from the ground up. Therefore, your security must be built into your applications every step of the way! Not only is it more expensive to apply security after-the-fact, but it's also ineffective. Also having security as separate function/process ends up frustrating everybody, the developer that needs to wait for security to approve controls/user stories, project and programme managers.

DEVSECOPS

Author: Francesco Cipollone &

Many have debated what is devsecops. We have debated what the meaning is in this document as well... ultimately we landed in defining DevSecOps as a principle of considering security as early as possible in the lifecycle.

The rationale is to detect and determine security bugs before they become big and risky vulnerabilities as well as too complex to be fixed.

The picture below displays the macro area of operation for the security programme to identify the stream of work. Those areas are also aligned with [OWASP Open SAMM](#) and [BSIMM](#) described in the later section on [Maturity Models](#)



Source: [NSC42 Appsec Program](#)

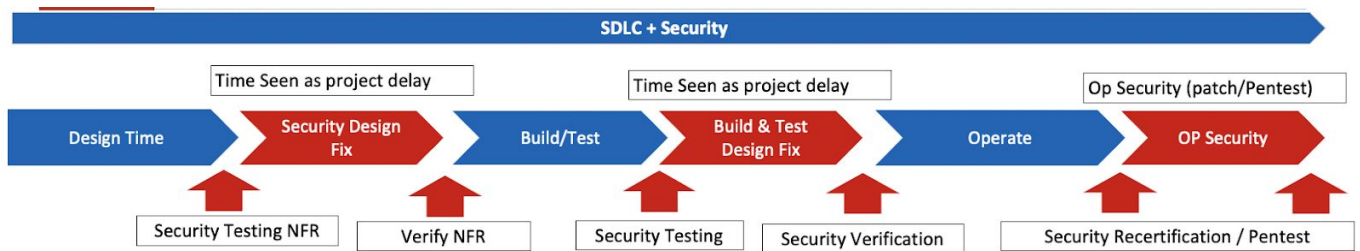
The picture below displays a traditional process of security testing in code that you might or might not have experienced before in traditional organizations.

The less mature organization deem security as testing when something gets released into production. or when the building is on fire...that is generally a bad approach because at that point is either too late (someone has used a vulnerability and you were lucky enough to notice) and too complex to fix.

In maturing organizations the process to test applications, code and op security

- Pentesting to test an application
- Testing some of the applications with the internal red team
- Probing some production with internal red teams.

Immature and Slow SDLC:



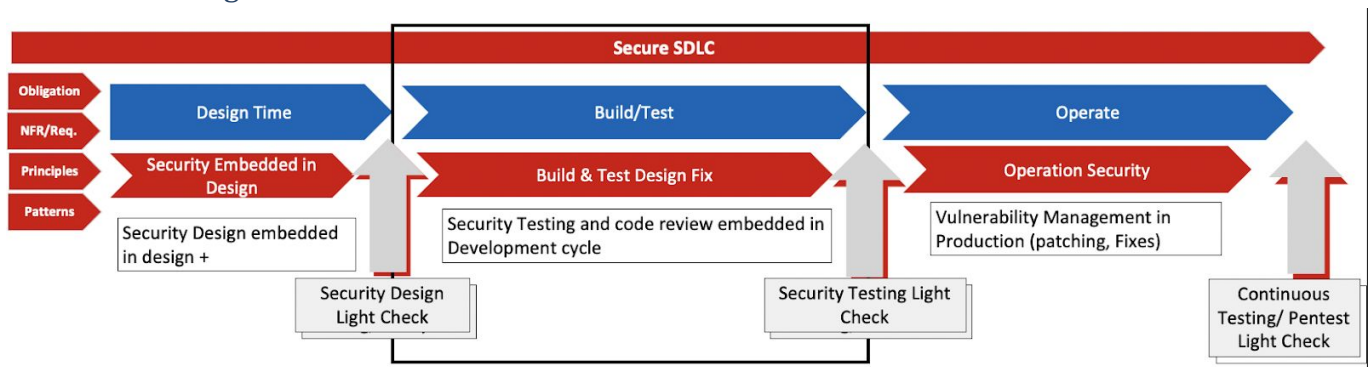
Source: [NSC42 Appsec Program/Training & Security Phoenix statistics](#)

The organization that undergoes maturity assessment using either the tools (NSC42 Assessment) or the matrix (BSIMM and SAMM) will introduce elements of security in each part of the application development lifecycle. A mature SDLC process would look similar to the one illustrated in the picture below.

The key element in this SDLC process is the security elements as soon as possible and security methodologies (like threat modelling) aligned to the various phases of the SDLC (Design, Build, Test, Operation).

Security testing and checks will always be present, but instead of lengthy processes, gate would act just as a validation fo the parallel security processes.

Good Reference guide:



Source: [NSC42 Appsec Program/Training & Security Phoenix statistics](#)

The more mature organizations aim to determine the following:

I can probably end up writing an entire book on this section, but to shorten the discussion I'll mention below some of the key points that I use to coach and train dev team in [NSC42 Training](#)

Security Governance for the DESIGN phase

- Verification that policy and procedures, as well as standard, are applied (those could be as simple as automated Jira ticket/security use stories)
- Policy Standards and Procedure with feedback loops and implementable.
- Blueprints - documents that provide guidance on what pieces of security control (driven by policy and standard) need to be implemented. E.g. authenticating with a specific method, using a specific library

Security for the DESIGN phase

- Threat Modelling [e.g. [Microsoft Threat Modelling](#)]
- Attack Surface Reduction [[Microsoft SDL- Attack Surface Reduction](#)]
- Security Functional Requirements/Non Functional Requirements - take the security policy, procedure and most important standard into something closer and implementable in code (e.g. libraries to use for a specific level of encryption) [[Open Security Architecture SNFR](#)]
- More advanced Rapid threat modelling and prototyping [[Rapid Threat Modelling and Prototyping](#)]

Good practices:

- Create standardized libraries and patterns for accessing system: e.g. have a reference design to specify which control to apply when interfacing with the corporate authentication system
- Specify the security requirements as default use stories in the Jira template (or other use story system) to have them implemented by default.

Security in Build/Test

- Scan your code for vulnerabilities [e.g. [NSC42 Appsec](#)]
- Implement a network of security champion (trained developers) to help development teams to fix security issues
- Centralize the approval and the scan for open source libraries
- Centralize the creation of code libraries to access common part of the infrastructure
- Automate the deployment and remove the human factor
- Create a list of approved tools and libraries
- If possible use standardize pipeline and testing methodologies
- Test components using a minimum library of the test (develop the test basing them on the OWASP good practice guidance)
- Validate API
- Create lists of security tests

Good practices:

- Deprecate functionality/methods, don't use them and keep updating the reference libraries
- Create centralized methods (code libraries) to access corporate system (e.g. authentication, authorization, databases)
- Use key management systems
- Use [OWASP reference good practices](#) area to consider are:
 - Input Validation
 - Output encoding
 - Authentication and Password management
 - Session Management
 - Access Controls
 - Cryptographic practices (don't invent cryptography)
 - Error handling and logging
 - Encryption and secure communication
 - System configuration
 - Database Security
 - File management
 - Memory management

There are tons of reference guide like [good practices from OWASP](#): and the guidance in the [Maturity Models](#) section

We will cover methodologies and other tools in the following sections

Maturity Models

Why the maturity Model?

"The most that can be expected from any model is that it can supply a useful approximation to reality: All models are wrong; some models are useful."

– George E. P. Box

Why a maturity model?



[OWASP SAMM Presentation at All Day Dev Ops](#)

So what's a maturity model, and how does it apply to DevSecOps?

Maturity models are how application security can improve over time, usually for years to come. A few different maturity models have become popular, including [OWASP SAMM](#) and [BSIMM](#). Applying maturity model standards can help guide your DevSecOps to continuous improvement. But implementing the appropriate maturity models is key!

Your organization should probably choose to apply both OWASP SAMM and BSIMM. But even having just one of those models should noticeably improve your application security as time goes on.

For this reason, we apply both methodologies in the [Application security programme, coaching and training in NSC42](#).

If your DevSecOps team is unsure how to proceed, we recommend that you start to implement OWASP SAMM and then consider integrating BSIMM later on. Then map the two maturity models together.

How to utilize those models

1. Do an assessment
2. Map the roadmap of actions and related KCI
3. Measure progress
4. Assess again

Roadmap to evolution

One of the key elements of the maturity model is to define a baseline (where you are), a target state (or several steps) and what are the actions to achieve the target state.

AS-IS					
Time	2019				
	Level 1	Level 2	Level 3	Level 4	Level 5
Security Design	Initial	Managed	Defined	Quantitatively Mat	Optimized
Security Design Governance	AS-IS				
Security Build & Test	AS-IS				
Security Operate		AS-IS			
Security Education		AS-IS			
Application Security Risk Management		AS-IS			

TO-BE					
Time	2020				
	Level 1	Level 2	Level 3	Level 4	Level 5
Security Design	Initial	Managed	Defined	Quantitatively Mat	Optimized
Security Design Governance		TO-BE			
Security Build & Test		TO-BE			
Security Operate		TO-BE			
Security Education		TO-BE			
Application Security Risk Management		TO-BE			

[Application Security Roadmap Part Of NSC42 Appsec Programme](#)

The Image above defines a simple solution we deploy to define a roadmap, an alternative is [OWASP SAMM](#) that enable to define a roadmap based on interviews questions.

The key to the roadmap tough is a series of tasks to go from one maturity level to the next (called maturity steps in the following image).

The other key indicator of the velocity (the speed of evolution) from one state to the next is KCI or Key Change Indicators (alternatively just maturity metrics).

Those are represented in the rightmost part of the Build/Test section of the maturity model with the frequency of measurements as well as the mapping to the maturity step. One of the key element is the

latter, the mapping to the maturity stage. Is pointless to measure how fast we can run if we haven't learned how to walk first right? One of the first exercise to do in measuring the maturity is to define what to measure at each stage. E.g. at the very beginning what you want to measure is how many people get on board with a maturity programme, at stage 2 that metric is becoming obsolete or not anymore an indicator of change towards a higher maturity level. The metrics that you want to consider at that stage are for example the number of teams onboarded on security tools and the number of metrics reported or repository scanned and so on.... Nonetheless, the element below is just examples of best practices! what really works for your organization to drive a higher maturity level from a cybersecurity and application security only you can define.

Overall Maturity

	Level 1	Level 2	Level 3	Level 4	Level 5
	Initial	Managed	Defined	Quantitatively Managed	Optimized
Security Design	AS-IS->TO-BE				
Security Design Governance	AS-IS	TO-BE			
Security Build & Test	AS-IS	AS-IS->TO-BE			
Security Operate					
Appsec Security Education	AS-IS	TO-BE			
Application Security Risk Management	AS-IS	TO-BE			

Maturity Steps

Maturity Task	How much
1 No Peer Review	
1 No Code Scanning	
1 No library update	
1 No risk management	
1 No visibility on vulnerabilities	
1 No knowledge of pods	
1 No team to pod mapping	
1 No Documentation of Fixes	
2 Peer Review	
2 Team to Pod to Stash recorded	
2 Onboarded application on code scanners	
2 Library scanning	
2 triage of vulnerabilities (base) - Consider only high medium and low	
2 Manual Evaluation of team and Allocation of Licence to Operate	
2 No SLA	
2 No Documentation	
2 Adoption Dashboard	
3 Peer Review with Toolset	
3 Updated Teams and asset register	
3 Basic Triage of vulnerabilities	
3 Code Scan with Pipeline Break & Basic SLA	
3 Adoption Dashboard (advanced) Per A.C. and Per Region	
3 Risk Assessment from code scanning with record in Risk management Register	
3 Automated Licence to operate: Code Scanning, Libraries, Internal Training	
4 Fix time of vulnerabilities recorded	
4 T-Shirt Sizing of fixes and Adaptation of SLA based on fixes	
4 Visualization of pod to fix	
4 segmentation dev and prod	
4 Fix ticket in Jira & Build vs Fix Concept	
4 Fuzzing (basic with generic per app)	
4 Automated Licence to operate: Code Scanning, Libraries, Internal Training, Build Vs Fix	
5 Build Vs Fix	
5 Automated Fuzzing & Library of tests	

KCI

	KCI Maturity	Reporting Frequency
Prereq ->	0 Who is working on which repository	Monthly
	1 Team On-boarded on scanners (per pod)	Monthly
	1 Code Scanning Frequency per project (min 1 per week)	Monthly
	1 Dashboard for Scanners created	Monthly
	2 Number of vulnerabilities ticket recorded	Weekly
	2 Dashboard for vulnerabilities - Onboarded Projects	Weekly
	2 Vulnerability Fixed (quarter)	Monthly/Quarterly Checks
	3 Project vulnerabilities integrated in Vulnerability management programme	Monthly
	3 Projects breaching the Build vs Fix target	Monthly/Quarter Checks
	4 Fixes per thematic in SLA	Monthly
	4 SLA for Fixes (breached/achieved)	Monthly
	4 Team Achieving Licence to operate and Out of the licence	Monthly
	5 Build vs fix	Monthly
	5 Licence to operate	Monthly

[Application Security Roadmap Actions and Metrics part of NSC42 Appsec Programme](#)

OWASP SAMM

OWASP SAMM is a maturity model that's developed by OWASP, an organization that focuses on improving web application security. [From their official site](#): "The prime maturity model for software assurance that provides an effective and measurable way for all types of organizations to analyze and improve their software security posture."

OWASP SAMM is more prescriptive than BSIMM. SAMM is not yet influenced by measurement data resulting from its maturity assessments. But there are advantages that SAMM has over BSIMM, as noted by [Jason Morrow](#):

"OpenSAMM may not have descriptive assessment data, but since it is prescriptive it can provide information the BSIMM cannot. For each of the maturity levels, OpenSAMM provides a stated objective. If you have not already defined objectives in your software security group these can be helpful examples to draw from in creating your own. I really like those at objectives have been included."

OWASP SAMM provides a point-in-time assessment of the maturity of one's application security program as a whole, not just the tools involved in application security. OWASP SAMM is a process providing a subjective, self-assessment of one's application security program. SAMM focuses on governance, design, implementation, verification, and operations. When the assessment is complete, a maturity score and heat map are generated. Heat maps provide leadership with a quick view of the maturity of the application security program. It should be noted SAMM does not provide a graded score (Pass/Fail); the focus is on maturity.

Once the maturity level is captured, SAMM assists with developing a roadmap to a maturity level aligned with the organization's risk appetite. The output provides a visual showing of deficiencies and provides leadership with the information needed to invest resources to align with the desired maturity model. The roadmap provided by SAMM is broken into four phases. Depending on how aggressive the organization wishes to address security, each phase could align with a roadmap strategy addressing deficiencies in quarter one, two, three, and four (Q1-Q4) of a calendar year.

BSIMM

If you're new to the BSIMM concept, it will help to understand the framework's various domains. Here's a summary:

- Governance- This pertains to practices that manage, organize, and measure a software security initiative (SSI).
- Intelligence- This is about gathering information that's required to improve your organization's SSIs. Organizational threat modelling and proactive security guidance help here.
- SSDL Touchpoints- This is about the software security development lifecycle! They include practices associated with analysis and assurance of particular software development artefacts and processes.
- Deployment- Finally, let's get your secure software development into production. This domain is about practices that interface with traditional network security and software maintenance organizations.

BSIMM10 has just been recently released. Here are some of the key points from it:

- DevOps adoption has advanced far enough to affect the way we approach software security as an industry.
- Software security is being led by engineering teams. One reason cited is the technical and pragmatic demands of application development models like Agile and DevOps. Another is the need to automate what were once human-driven tasks. Then there's the combination of unpredictable impacts on delivery schedules, human-intensive processes from existing SSIs (software security initiatives), and process friction. Finally, there's the ever-increasing pace of application lifecycle processes to consider.

- They observed that organizations improve their security maturity over time, eventually focusing more on the scale, breadth, and depth of their activities rather than constantly starting new activities.

Inevitably, the maturity of the BSIMM methodology itself has improved over time. Imagine that!

As mentioned, engineering teams are leading innovation in software security. This engineering mindset is an asset to implementing BSIMM into DevOps, but it requires a cultural change! A lot of progress still needs to be made in this area. As Contrast Security's Jeff Williams said, "A lot of people take 'shift left' to mean we should take the legacy tools that are used for security and push them to developers so they can do their own security. The problem is those tools were built for experts. Developers don't have the skills to run those tools."

The nice thing about starting your DevOps processes with BSIMM from the very beginning is that once you have your security baselines, policies, procedures, and automation in place, your applications will be able to face new challenges in the evolving cyber threat landscape.

NSAMM

With the evolution of the two models at [NSC42](#) we figured why not map BSIM to SAMM and integrate the two methodologies?

We've also selected and aggregated some areas (like Security Build & Test as a single step). The models are retrofitted and compatible with the standards.



Copyright © NSC42 Ltd 2019 (all rights reserved)

[NSAMM Pillars](#)



OWASP Sann Macro Areas

What are the key pillars of NSAMM (NSC42 adaptation of SANN) and SANN?

While NSAMM focuses on broader work application (governance oversight, education and cultural shift) and metrics, SANN focuses heavily on implementation and verification. Nonetheless, NSAMM maps to SANN and BSIMM nicely.

Maturity Steps		KCI	
Mat	How much		Reporting Frequency
1	No Peer Review		
1	No Code Scanning		
1	No library update		
1	No risk management		
1	No visibility on vulnerabilities		
1	No knowledge of pods		
1	No team to pod mapping		
1	No Documentation of Fixes		
2	Peer Review		
2	Team to Pod to Stash recorded		
2	Onboarded application on code scanners		
2	Library scanning		
2	trriage of vulnerabilities (base) - Consider only high medium and low		
2	Manual Evaluation of team and Allocation of Licence to Operate		
2	No SLA		
2	No Documentaiton		
2	Adoption Dashboard		
3	Peer Review with Toolset		
3	Updated Teams and asset register		
3	Basic Triage of vulnerabilities		
3	Code Scan with Pipeline Break & Basic SLA		
3	Adoption Dashboard (advanced) Per A.C. and Per Region		
3	Risk Assessment from code scanning with record in Risk management Register		
3	Automated Licence to operate: Code Scanning, Libraries, Internal Training		
4	Fix time of vulnerabilities recorded		
4	T-Shirt Sizing of fixes and Adaptation of SLA based on fixes		
4	Visualization of pod to fix		
4	segmentation dev and prod		
4	Fix ticket in Jira & Build vs Fix Concept		
4	Fuzzing (basic with generic per app)		
4	Automated Licence to operate: Code Scanning, Libraries, Internal Training, Build Vs Fix		
5	Automated Fuzzing & Library of tests		
		KCI Mat	
		Build/Test	
		rutiv	
	Prereq ->	0 Who is working on which repository	Monthly
		1 Team On-boarded on scanners (per pod)	Monthly
		1 Code Scanning Frequency per project (min 1 per week)	Monthly
		1 Dashboard for Scanners created	Monthly
		2 Number of vulnerabilities ticket recorded	Weekly
		2 Dashboard for vulnerabilities - Onboarded Projects	Weekly
		2 Vulnerability Fixed (quarter)	Monthly/Quarterly Checks
		3 Project vulnerabilities integrated in Vulnerability management programme	Monthly
		3 Projects breaching the Build vs Fix target	Monthly/Quarter Checks
		4 Fixes per thematic in SLA	Monthly
		4 SLA for Fixes (breached/achieved)	Monthly
		4 Team Achieving Licence to operate and Out of the licence	Monthly
		5 Build vs fix	Monthly
		5 Licence to operate	Monthly

NSAMM Rearranged and simplified some of the maturity steps highlighted in OWASP Sann taking the assessment questionnaire a step further with the recommendations linked to various maturity.

Differently from OWASP SANN the NSAMM Model is on a scale from 1-5

The NSAMM Model is mapped to Measurement and Frequency of measurements

Application Security Roadmap Actions and Metrics part of NSC42 Appsec Programme

No matter which framework you adopt, ultimately the structure falls into this pattern

- Programme of work
- Governance
- Risk Management
- Implementation

- Verification
- Operation
- Education

NSC42 is currently working on a web assessment version called the [Appsec Clinic](#) that will enable to evaluate your posture quickly.

The [Appsec Clinic](#) will be have an enterprise/premium component will cover SAMM and BSIMM, ASVS other standards in a wider assessment with the recommendation of actions. Moreover, you will be able to provide security teams or auditors (internal and external) the link to the evidence of the assessment/report.

Section 1 - the What

SDLC Security

Intro

In 2004, Microsoft introduced their [Security Development Lifecycle](#), and in 2006, Gary McGraw described security [touchpoints in the development lifecycle in 2006](#). Both approaches aimed to improve the security of software, each with their own vision and approach. But one thing is certain. These approaches have had a monumental impact on the way we build secure software today.

What once started as a painstakingly slow, mostly manual effort, has rapidly involved in a largely automated process, running continuously in the background. Righteously so, since modern applications are more intertwined than ever while being continuously built and released. Today, security in the SDLC takes the form of automated scanning in the deployment pipeline, continuous monitoring of dependencies for vulnerabilities, and manual testing by skilled appsec engineers and bug bounty hunters.

TIME TO FIX VS TIME TO ATTACK

Adversaries don't need many vulnerabilities ONE is enough.

Make sure you prioritize the right vulnerabilities

According to industry statistics the average time to fix a website vulnerability after it has been reported is 150-180 days.

Every
36 minutes

A new security vulnerability is identified

It takes
150-180 days

To fix a vulnerability

N. Of Vuln per year

1400 vuln

As disclosed vulnerabilities

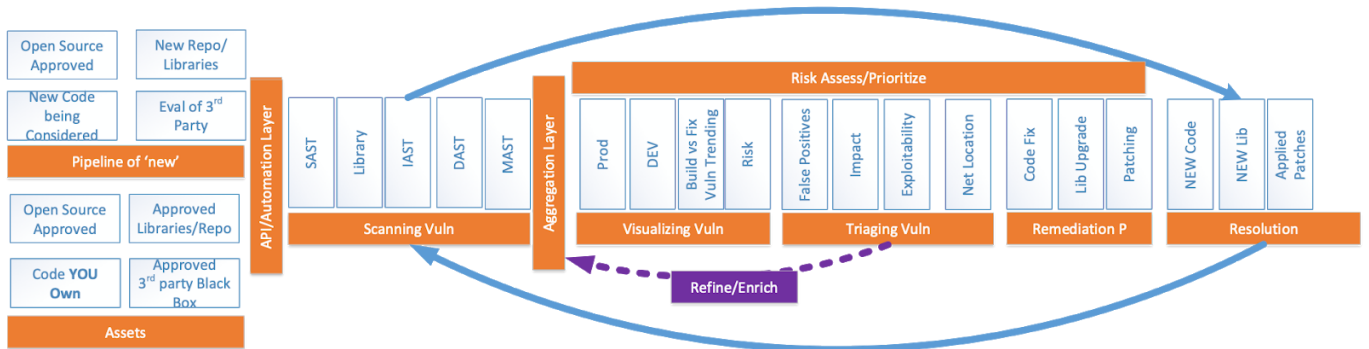
It takes
3-15 days

To exploit a vulnerability

[From Francesco's NSC42 Security Phoenix Presentation](#)

While the process today looks vastly different than it did 15 years ago, the fundamentals are still the same. Vulnerabilities need to be fixed as early in the life cycle as possible. For example, applying static analysis techniques during development will flag common security vulnerabilities in the code being developed. As much as you try, production apps will always have vulnerabilities, so you need processes to manage to report. Today, that means having automated scanners to find a vulnerability in an application's dependencies. Additionally, a dedicated security team needs to triage incoming reports from security researchers and bug bounty hunters. Regardless of when they are discovered, vulnerabilities need to be analyzed, triaged, and remediated. Many of these tasks can be automated, but for now, humans are still actively involved in this life cycle. More often than not, the application developers cooperate with the organization's appsec team to combine expertise when addressing vulnerabilities. This cooperation is crucial to develop an effective and productive security culture.

This section offers a deep dive into the various security components in the SDLC. Topics include the different flavours of Application Security Testing, Vulnerability Management, and Software Asset Management.

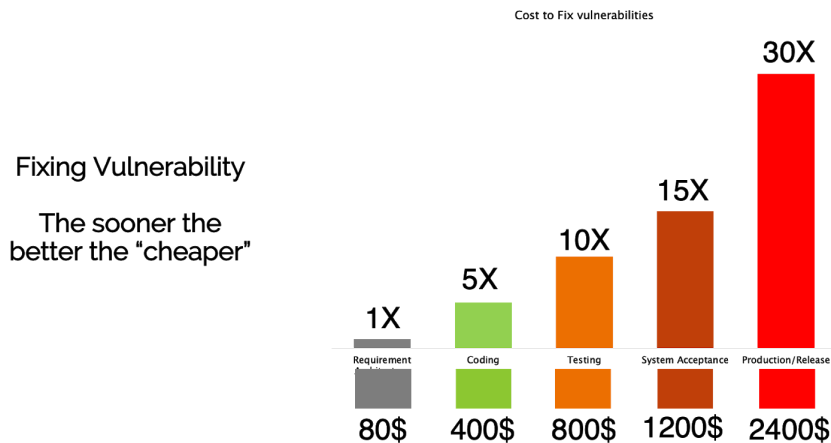


Application Security Program and Steps from NSC42

Why Secure code is important and how to achieve it

Building 100% secure applications is a utopian pipe dream. Security in any real-world application is always a moving target. Software engineers or operations engineers will always make mistakes. New threats or vulnerabilities will be discovered continuously. Even the threat model of the application may change over time. Instead of chasing this absolute state of security, we need to strive to set up proper processes to maximize the security of an application.

WHY INVEST IN APPSEC?



Fixing Bugs The earlier The better

Many organizations work hard on deploying security processes to support the quick remediation of security issues. While extremely useful, remediation is only half of the picture (or even 30%, but let's not argue about it). Complementary to remediation is prevention. Vulnerabilities that don't make it into the application do not need to be remediated.

I'm sure you are familiar with the axiom that the later in the lifecycle a vulnerability is discovered, the more it costs to fix it. Vulnerabilities that are never written induce zero additional costs to remediate since they were never there.

Of course, it's easy to say "**just prevent vulnerabilities**", but doing that in practice is entirely different. Preventing vulnerabilities requires a mature appsec program, where various activities are tuned into each other. A few example activities are:

- making sure software engineers can follow proper security training
- providing detailed secure coding guidelines to avoid common vulnerabilities
- dev-time and build-time application security testing
- pentesting and responsible disclosure programs for production applications

In this section, we will take a closer look at the importance of creating feedback loops between various security activities in the SDLC. Further sections in this chapter provide an in-depth look at implementing some of these security activities. Remember that even with the perfect processes, there will always be vulnerabilities that need to be remediated.

Create a feedback loop between activities

The image picturing various security activities throughout the SDLC, which was included earlier in this chapter, already included a feedback loop between multiple activities. This feedback loop is essential to ensure that security activities are relevant and useful.

There are no strict rules about which activities can provide feedback to each other. Interactions between activities are not static and can change over time. Instead of giving a rigid list that imposes restrictions, we will discuss a few examples instead. Use these examples to customize your appsec program.

The holy grail, or validation, of many appsec programs, is a pentest by an external company. Typically, such a pentest result in a report with findings, some critical, many important, and even more low-risk findings. Such a report offers valuable feedback to make other activities more relevant. Developers can use this feedback to fix vulnerabilities. The feedback will likely push application security testing to ensure the vulnerabilities are fixed. Ideally, the findings even drive security training and secure coding guidelines to mature the appsec program further.

Also, the KCI illustrated in the [Previous Maturity Metrics](#) chapter help identifying if the appsec programme is delivering the expected results

In turn, secure coding guidelines also drive application security activities. By following a strict set of guidelines, static analysis tools can be configured to increase their precision. Additionally, specific critically insecure code paths can be banned from the codebase using code scanning tools.

Another example occurs very early in the lifecycle. In-depth security training for developers and appsec teams often drives the development of organization-wide secure coding guidelines, which in turn drive application security testing.

As you can see from these examples, the feedback loop between activities is essential. Consequentially, it is imperative to foster a culture that supports feedback without conflict or blaming.

Gradually push appsec further

Building an effective appsec program is not a one-off activity. It's not something you set up and just works. Instead, appsec programs need to be supported and nurtured. Over time, they grow and become much more powerful. But doing so requires you to take it one step at a time.

In reality, most organizations do not start from a clean slate. They are doing some appsec activities. Regardless of their effectiveness, these activities are the perfect starting point to start growing an appsec program.

Let's discuss an example using a common scenario as a starting point: a regular pentest of an application. We're not talking about the effectiveness of pentesting here, but are more interested in the outcomes. If the pentest is the only activity in the appsec program, you will have some remarks that came out of it. Good, that is your feedback.

Now, use the most critical points from that pentest to drive activities earlier in the lifecycle. If the pentest of a React application points out that there are numerous Cross-Site Scripting (XSS) vulnerabilities, you should first remediate them. Next, you should go through the application to address other cases. Finally, you need to find a way to prevent them in the future.

Based on the activities we discussed before, this would start with training developers on the dangers of XSS and how such vulnerabilities are introduced in a React application. Next, you should implement a step in the build process that checks the codebase for dangerous coding patterns. A simple way to achieve that is by running a linter with a couple of security-specific rules. Sure, this is not going to be the perfect XSS defence, but it's a step forward.

With that done, you can start building a React security library for internal use. This library will offer safe functions to perform potentially unsafe actions, such as putting HTML in the page. This library is built by security professionals and thoroughly vetted for security. With that library, you can now draw up secure coding guidelines that dictate the use of this library for certain features.

With the mandatory use of the security library, specific insecure code paths can be eradicated from the application. Update the linting rules to ensure that the use of these code paths triggers the build to fail. Now you have assured that a developer can no longer introduce an XSS vulnerability in a React application.

But we can push it even further. We can use IDE plugins to perform code analysis at development time, allowing us to notify the developer the moment they use the insecure code path. This gives developers immediate feedback, instead of only at build time.

Reaching this point means you will have gained a ton of appsec maturity. Confidence in these coding guidelines and tools also allows you to focus future pentests on other areas of the application, making them much more valuable.

Adopt a defense-in-depth strategy

Defense-in-depth is critical in every appsec program. That holds both for activities in the SDLC and security mechanisms in an application. Let's take a look at a few examples.

Developer training, secure coding guidelines, application security testing, and pentesting all focus on avoiding vulnerabilities in an application. Having each of these steps in place builds a solid defense-in-depth strategy that will be effective at catching vulnerabilities before they reach production. Similarly, a defense-in-depth strategy should be applied for security mechanisms too. The recent incident with a [Starbucks API](#), exposing approximately 100 million records and previously seen Facebook, provides us with a teachable moment. The incident in question was a bug bounty report, but the implication of the attack is the same. Without diving too deep into the details, what happened was that a proxy server running on the perimeter of the network forwarded a couple of requests it should not have forwarded. Through that loophole, the attackers could query an internal database and extract information. This type of problem is known as a "[Confused Deputy](#)".

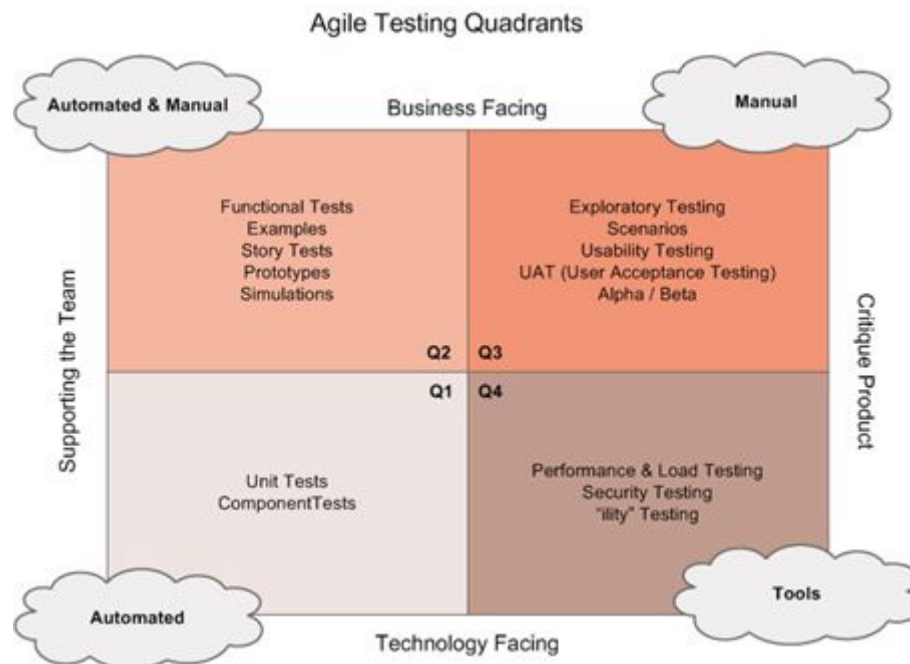
The vulnerability at the basis of this breach was the proxy server forwarding requests it should not have forwarded. However, the internal system should not have disclosed that much information without issue. Ideally, it would have rejected that type of access, but even triggering an alarm bell would have been a useful response. By shielding the system from outside access, it was mistakenly believed to be secure. A defense-in-depth strategy would have likely prevented the exploitation of this vulnerability. To conclude, there is one critical lesson to draw from the Starbucks example: it is tough to make accurate security assumptions about a system or application. The internal system was believed to be unreachable while it was not. As a consequence, you should avoid wasting time on determining the exact exploitability of a disclosed vulnerability. Discussions on whether a database without any authentication or restrictions is reachable are not very productive. Instead, recognize the severity of an issue and aim to address it.

Agile Testing Quadrants

Now moving into one of the core section that probably everyone is considering or doing let's explore the testing stages and see how all the products fits together.

Security testing, regardless of the tool, aligns with one of the four agile testing quadrants. Understanding where each tool fits in the four quadrants enables one to understand the level of effort, time, and value the tool(s) provide. Value is provided to decision-makers determining return on value and for security professionals aligning security tests with standard development tests. Aligning security tests with developer methodologies is a strategy recommended by Bell et al. (2017) and Kim et al. (2016).

Testing falls in one of four categories: support programming, business-facing, critique project, and technology facing (Humble & Farley, 2011). Each category includes automating a manual testing process. The tests are further categorized by automated or manual. The diagram below depicts the four testing quadrants.



(Gregory & Crisping, 2009)

Business-facing tests supporting development are often referred to as functional or acceptance tests and are executed in a production-like environment (Humble & Farley, 2011). Acceptance testing validates features meet defined business requirements, referred to as the happy path, and are based on user stories. While there are functional and nonfunctional acceptance tests, tests in this quadrant focus on the functionality of the feature. Successfully passing acceptance tests results in the application changes being complete, as it proves the change meets defined requirements and integrates with the entire application.

Technology-facing tests supporting development are tests written and maintained by developers. According to Humble and Farley (2011), tests include unit, component, and deployment tests. Unit tests run in isolation, never interfacing with databases, networks or other systems and focus on a specific piece of code. Shahin, et al. (2017) argued identification of flaws is possible without interfacing with actual dependencies. Unit testing should have high code coverage and be extremely fast. Due to the self-service nature of unit tests, the code coverage, and the speed, they are a critical part of the feedback loop, providing developers instant feedback on code fitness (Humble & Farley, 2011).

Component testing, sometimes referred to as integration testing, focuses on larger portions of the applications and functionality (Humble & Farley, 2011). Testing in the component phase is not isolated like unit testing. Component testing involves interfacing with components, resulting in slower test times.

Deployment tests involve deploying the applications and validating installation, configuration, and interaction with other services (Humble & Farley, 2011). The environment in deployment testing mimics production. Testing in a production-like environment ensures the feature interoperates with all components of the application, the hosting system(s), network, and potential backend databases.

Business-facing tests critiquing the project are scoped to validate the application will provide the expected value users expect (Humble & Farley, 2011). Users may find using the application does not

work as expected or breaks when performing some activity developers did not plan for in advance. The manual portion of testing involves users interacting with the application and providing insight on ways to make the application better, more user friendly, or some other way to improve the application. Humble and Farley (2011) claimed the testing performed in this phase is often performed as part of a beta program.

Technology-facing tests critiquing the project have two categories: functional and non-functional tests (Humble & Farley, 2011). Security, availability, and capacity testing fall in the nonfunctional testing category. Nonfunctional testing are parts of the application users do not normally interface with, however, are impacted by deficiencies with any of the nonfunctional qualities. Humble and Farley (2011) claimed nonfunctional tests are not viewed as important as functional testing and normally run less frequently or are performed near the end of the pipeline.

From a security perspective, DAST and abuser stories run in acceptance testing and would be considered technology-facing tests critiquing the project. DAST tests all the parts of an application, the system hosting the application and systems interfacing with the application (Bell et al., 2017). System testing provides the environment for DAST to perform runtime security tests.

Abuser stories align with user stories but provide an attacker's view of the application, occurring late in the process (Bell et al, 2017; Bird, 2015). Fully testing an attacker's perspective of the application requires a complete environment provided during the system's phase of acceptance testing. The security tests described focused on non-functional qualities of the application, and occur in later stages of the pipeline.

SAST aligns with technology-facing tests critiquing the project. Chess (2004) stated SAST does not validate functional qualities of an application, putting it in the nonfunctional category. However, unlike DAST and abuser stories which fall at the end of the pipeline, SAST occurs early in the pipeline. Testing early in the process follows the concept of shifting security left, providing an opportunity to maximize the value of the feedback loop by providing immediate feedback while the cost to remediate is low (Bell et al., 2017; Vehent, 2018).

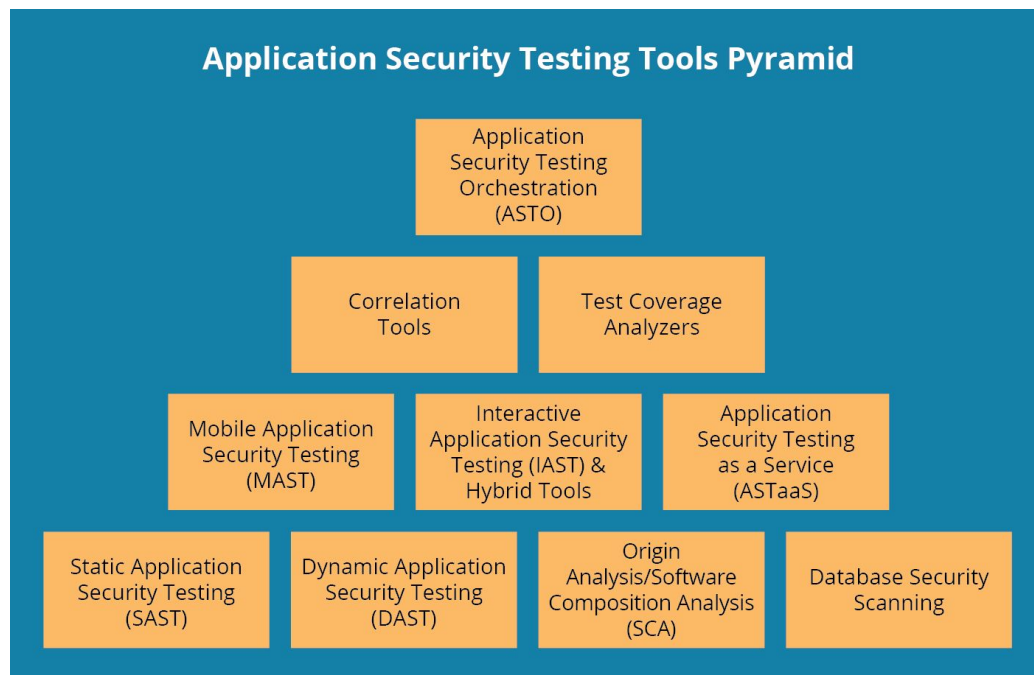
Documented Benefits of a Secure SDLC

Mainstay Partners (2014) identified companies implementing software security assurance (SSA) solutions in DevOps programs could realize potential cost savings of \$2,400,000 annually, an estimated savings of \$44,000 per application. Cost savings were attributed to faster scanning driving down overall code scanning costs, fewer vulnerabilities reducing time spent patching or mitigating vulnerabilities, improved productivity, and streamlining compliance requirements. The cost savings were realized by embedding security through the entire SDLC process.

Respondents reported implementing SSA solutions during SDLC decreased the number of vulnerabilities per application, the average time required to fix vulnerabilities, and the reoccurrence of repeated vulnerabilities. The study focused on implementing SSA during all phases of the SDLC, resulting in the study not identifying the effect of specific tools or processes (SAST, DAST, SCA, etc.). Research by Mainstay Partners provided evidence of embedding a suite of security testing in the SDLC on the security hygiene of code.

Type of Testing

Writing a section on all the possible testing could take a long time, we will summarize here some of the most used testing methodologies that we've seen implemented and effective. Below an extract of a very good article from [SEI of Carnegie Mellon University](#) on all the type of testing

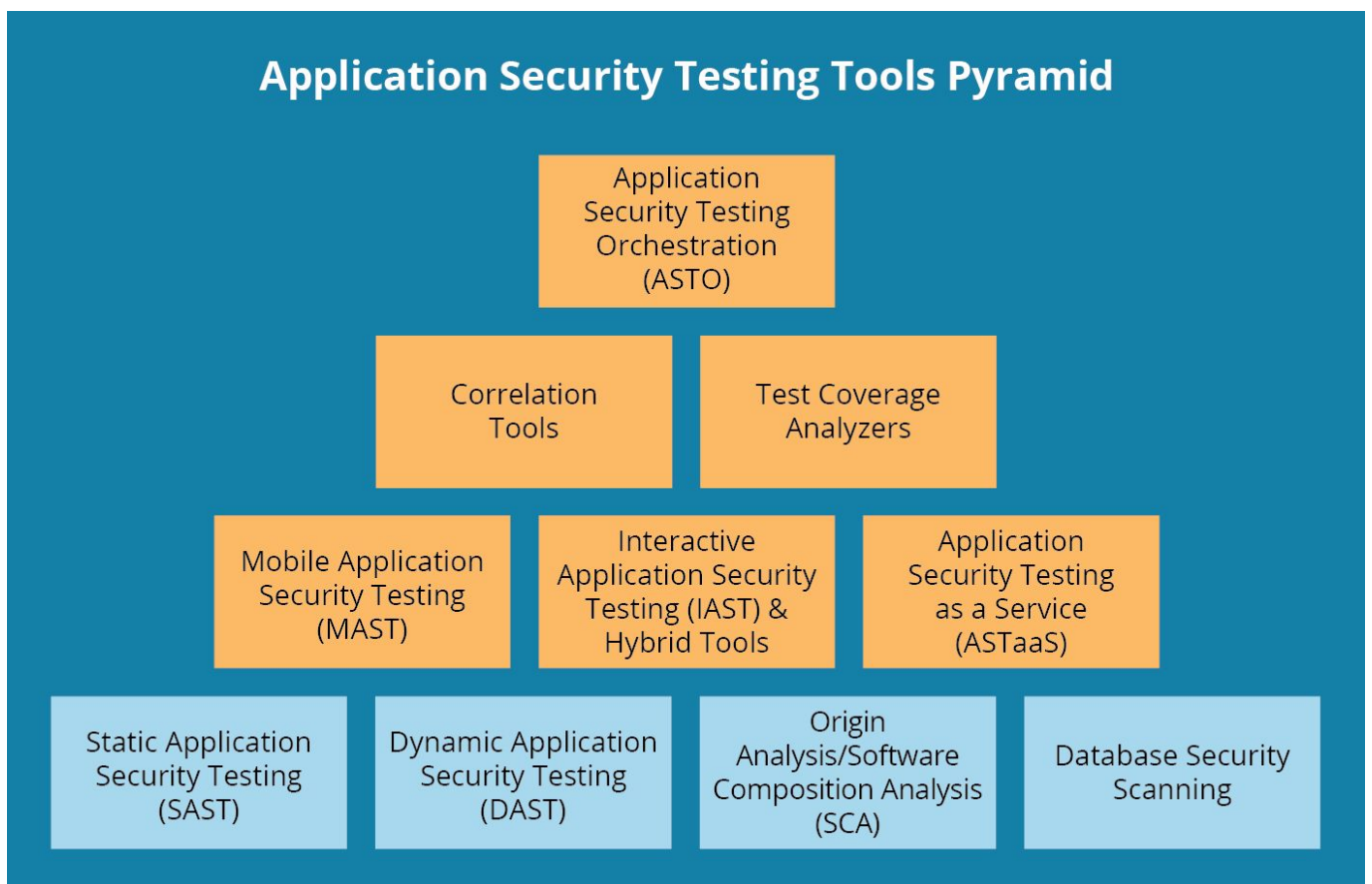


Source : [SEI of Carnegie Mellon University](#)

- **Static Application Security Testing (SAST)** - SAST tools can be thought of as [white-hat or white-box testing](#), where the tester knows information about the system
- **Dynamic Application Security Testing (DAST)** - In contrast to SAST tools, DAST tools can be thought of as [black-hat or black-box testing](#), where the tester has no prior knowledge of the system
- **Origin Analysis/Software Composition Analysis (SCA)** - [Software-governance processes that depend on manual inspection are prone to failure](#). SCA tools examine software to determine the origins of all components and libraries within the software. **[not covered here]**
- **Database Security Scanning** - The [SQL Slammer](#) worm of 2003 exploited a known vulnerability in a database management system that had a patch released more than one year before the attack. **[not covered here]**
- **Interactive Application Security Testing (IAST) and Hybrid Tools** - Hybrid approaches have been available for a long time, but more recently have been categorized and discussed using the term IAST. **[not covered here]**
- **Mobile Application Security Testing (MAST)** - MAST Tools are a blend of static, dynamic, and forensics analysis. They perform some of the same functions as traditional static and dynamic

analyzers but enable mobile code to be run through many of those analyzers as well. [**not covered here**]

- **Application Security Testing as a Service (ASTaaS)** - As the name suggests, with ASTaaS, you pay someone to perform security testing on your application. The service will usually be a combination of static and dynamic analysis, penetration testing, testing of application programming interfaces (APIs), risk assessments, and more. [**Not covered in this current version of the report, current under work at Security Phoenix**]
- **Correlation Tools** - Dealing with [false positives](#) is a big issue in application security testing. Correlation tools can help reduce some of the noise by providing a central repository for findings from other AST tools. [**not covered in this version of the report, current under work at Security Phoenix**]
- **Test-Coverage Analyzers** - Test-coverage analyzers measure how much of the total program code has been analyzed.
- **Application Security Testing Orchestration (ASTO)** - [ASTO integrates security tooling across a software development lifecycle \(SDLC\)](#). While the term ASTO is newly coined by Gartner since this is an emerging field, there are tools that have been doing ASTO already, mainly those created by correlation-tool vendors.



Static code analysis

Static application security test (SAST), also called static code analysis, analyzes source code for vulnerabilities. SAST differs from testing methodologies such as DAST, as SAST analyzes the code in a non-running state. There are two approaches to SAST: analysis of the source code and analysis of the compiled bytecode. Analysis of the source code does not analyze the code in native format; analysis begins with creating an intermediate representation of the code using an abstract syntax tree (AST). Depending on the specific tool, ASTs are further decomposed into control flow graphs and data flow graphs. Analysis of the source code is conducted on the CFGs/DFGs.

The second approach is working with compiled bytecode. A noted drawback described by Logozzo and Fahndrich (2008) was that bytecode may not be an exact replica of the source code because compilers optimize the code. Bytecode analysis is considered faster than source code analysis, which aligns well with the DevOps First Way: Fast Flow. Bytecode analysis does not require name resolution or type checking - work already completed by the compiler. While several benefits are presented for bytecode analysis, a negative identified is the lack of precision provided by SAST that analyzes the source code.

Benefits

SAST provides one of the first opportunities in the build process to analyze the fitness of source code. SAST aligns with technology-facing tests critiquing the project. Due to SAST's alignment with development testing, it occurs early in the build process. Early testing helps maximize the efficiency of tests aligned with the DevOps First and Second Way: Fast Flow and Fast feedback. Developers are provided quick feedback on the fitness of the code and potential vulnerabilities that need to be addressed.

Due to SAST being a white box test, having access to the source code, SAST has the potential to find vulnerabilities other testing methodologies may not identify. Antunes and Vieira (2009) conducted an experiment where they tested the effectiveness of penetration testing and SAST to detect SQL injection vulnerabilities in a known vulnerable application. The results showed SAST located more SQL injection vulnerabilities than penetration testing. Benefits from the study point to the effectiveness and potential return on the value of using SAST to test modular pieces of code early, rather than relying on penetration testing the application late in the development process.

Veracode (2018) conducted a study of their customer base. Results identified the median scan rate of source code was twice a year; the same median scan rate noted in their 2017 study. Veracode identified the second scan was completed within days of the first scan. Veracode (2018) hypothesized developers were addressing policy requirements, rather than addressing risk. Embedding Automated SAST in the build processes offers an opportunity to increase the scan rate.

The research identified organizations conducting 300 or more scans a year showed the vulnerability persistence ratio was 12 times shorter than applications aligned with the median scan rate (Veracode, 2018). The study was not able to identify if increased scan rates were manual or automated; however, Veracode hypothesized automated, incremental scanning identifies vulnerabilities quicker and promotes shorter remediation windows. Veracode claimed results from the study indicated increased scan rates reduced risks associated with applications. Veracode believed the increased scan rates pointed to the value of security in DevOps.

Drawbacks

Historically, SAST tools were plagued with false positives (FP) (Dimastrogiovanni & Laranjeiro, 2016). The authors further claimed the growing complexity of applications presented challenges for SAST to accurately identify vulnerable code. Bird (2017) claimed development teams spend a lot of time weeding out FPs, slowing the velocity of DevOps.

According to Sozer (2015), for every thousand lines of code SAST will generate 40 alerts. Compounding the problem, SAST alerts have shown to have a false positive rate of 30% to 90%. Sozer (2015) further claimed limited resources often resulted in developers ignoring or suppressing alerts to ensure the code is compiled.

Antunes and Vieira (2009) conducted research on three SAST tools; the SAST tools tested presented with 23%, 26% and 27% false positives. A finding of a specific false positive related to SQL injection was flagged because concatenation of the string values included one input parameter. The finding was a false positive because the code executed a function to inspect the query and throw a runtime exception if a non-integer value was detected. According to Antunes and Vieira (2009), false positives such as the one shown in the research are common of SAST implementing data flow analysis, as they do not identify functions called to validate input or understand conditional logic. Tuning SAST tools based on threat modelling helps reduce the generated false positives, making the tool more effective (Bell et al., 2017).

Researching FPs is seen as a manually intensive process distracting developers from focusing on writing code and delivering value to the organization (Dimastrogiovanni & Laranjeiro, 2016; Hall, 2017). A research study conducted by 451 Research (2018) found FPs as a barrier for SAST adoption. Forty-six per cent of respondents claimed FPs negated the benefits provided by a SAST solution.

Static Code Analysis - Why should I?

SAST has the potential to reduce the costs associated with identifying and remediating vulnerabilities in source code. Baca, Carlsson, and Lundberg (2008) tested three SAST tools against code known to be vulnerable. The study identified SAST reduced the risks associated with the applications. Additionally, implementing SAST would have provided a cost savings of 17% based on previously reported failures associated with the vulnerabilities.

SAST is one of the first tools run in a pipeline and provides opportunities for fast feedback on code fitness. Research by 451 Research (2018) revealed 51% of the companies identified SAST as a critical tool to embed in the pipeline. DAST was selected by 59% of respondents as more important than SAST, even though DAST executes later in the pipeline, which would cause longer lead times to identify vulnerabilities. The further down the pipeline security tests find vulnerabilities, the more time between code commit, identifying and remediating vulnerabilities. According to Bell et al. (2017), security testing at code commit provides an early point in the process to perform security testing, potentially avoiding time delays identifying vulnerabilities.

Bell et al. (2017) stated ensuring proper code coverage and gaining visibility of the security fitness of code can only be realized by automating the SAST process. Automated SAST (ASAST) as part of the CI process ensures code is scanned during every commit and aligns with developer testing practices. A SAST should focus on incremental scanning (Bell et al., 2017; Bird, 2015). When code is committed initiating tests in the CI environment, only the modified code should be scanned. Incremental scanning reduces overhead, increases scan speed, and allows developers to experiment without introducing delays; full SAST scans of the codebase are completed nightly or weekly, as the process is time-intensive and does not align with pipeline processes (Bell et al., 2017; Bird, 2015; Vehent, 2018).

Automated SAST, using secure frameworks and secure compiler configurations makes writing secure code easier and lowers the likelihood of coding mistakes introducing vulnerabilities in the application (Bird, 2015). The author defined the tasks as secure by default; processes designed to remove the complexity of application security by working in the framework. Continuous testing ensures the most recent version of the code is in a continuous delivery state, ready for continuous deployment (Smada et al, 2018).

Addressing FPs and low developer adoption begins with properly onboarding the application. Merely handing developers a SAST tool will not meet the risk reduction goals of the business. Plugging automated SAST into your pipeline without proper tuning will result in developers bypassing the job or suppressing findings.

Scan the code and review the findings with the developers; they know their code and can help identify false positives. Understand where the application receives input and map that to trust boundaries. Nine of the OWASP Top 10 manifest at trust boundaries. Knowing where input is accepted and what trust boundaries the data crosses will help when analyzing SAST findings. There may be an input validation finding, but the source is a trusted database that untrusted users can never write to; mark it a false positive and move on (if the application allows users to write to the database, even though the database is on a trusted network, reads from the database should be considered as untrusted and data sanitized/validated).

Does it use a control flow graph or data flow graph? When performing taint analysis, identifying sources (input) and following the flow to a sink (writes to a database, files, to a user's browser), the code may validate input and still show vulnerabilities if your SAST uses Data Flow Analysis (DFA). DFA performed on a data flow graph is great at finding sanitization methods; however, DFA does not do well with validation methods, as DFA doesn't understand conditional logic. Mark it a false positive or configure the tool to understand the validation so the findings don't show up again.

Review your logs to ensure scan quality and accuracy. Are you scanning files that should be excluded, e.g., test code, 3rd-party components, and/or code not used in production? Are you getting good code coverage? Some SQLi will show up because of odd approaches to parameterized statements. Mark them false positives and move to the next findings. Continue until all findings are reviewed. Rescan the code. Now you have actionable data to provide developers. Only after properly onboarding an application, weeding out false positives, and ensuring accurate scans is it acceptable to embed SAST in a DevOps pipeline.

The immediate goal when inserting automated SAST in a pipeline should not be to fail builds based on findings. While onboarding and tuning scans may have occurred, the initial goal should be ensuring the tool works correctly. Second, ensure scan times fall within a predefined range, such as 5-10 minutes. Having defined scan times allows SAST to be a variable one can calculate when determining predictable build times. Finally, analyze findings to validate quality findings and true positives; at this point, moving to a fail-the-build model is an option; marking builds as unstable is also suitable during the maturation phase.

When you insert SAST in a pipeline, customize the queries to the application. What languages and frameworks are used? Are you failing on highs only? If so, customize the queries to just scan the languages/frameworks used and only look for high vulnerabilities. Why look for medium/low if they will not cause a build to fail? Schedule nightly full scans to look for medium and low vulnerabilities. Write the findings to a bug tracking system.

Does the application have 8 high vulnerabilities? Configure the build to fail on 9. This will allow you to leverage the benefits of automated SAST without failing on current vulnerabilities. However, it will prevent new highs from being introduced to the application, while tracking remediation efforts of the current 8 highs.

DevOps is based on the principles of fast flow, quick feedback, and continuous learning. Spending the initial time upfront to properly onboard the application and customize pipeline scans to the application will facilitate these three principles. It will enable you to quickly perform incremental scans (only scan code/files that have changed) on every commit (fast flow), fail or pass builds quickly (quick feedback), and provide feedback for developers to learn from coding mistakes leading to vulnerabilities (continuous learning).

Web Application Testing - a form of DAST

Automated vs Pentesting - differences between pen testing and automated testing

Automated Web application testing is the practice of implementing toolset to test APIs of web application (or alike) and full website functionalities against a predefined set of

Web Application Testing - why should I

Web Applications if we include Cloud-based applications those are the front-facing surface. Companies don't open the internal network to everyone but they open web applications. With Cloud, you can't prevent applications from going public. E.g. if you are using Office365. It is available everywhere. That's why web apps become the most lucrative and most common attack surface.

Students or the beginners in AppSec always ask the question, where to start the pentesting on applications or security testing of applications.

It starts with the thought of an application designing. Security has to be part of the whole Lifecycle of the application from Designing to move to production. We can say we will start later or we can't consider the security. We have seen the biggest breaches of the decade in the past couple of year e.g. Equifax, British Airlines, and many more. This is not a blame game. We just need to be considerate of security. Security should not be considered as a side actor, it has to be a key player.

Security needs the utmost priority when we talk about Modern Application, Serverless applications or functions as a service. The needs to understand and implement security is very important. The organisations have been breached and breaches will happen. However, we can minimise the impact or might be able to eliminate to a certain extent with the efforts.

Application Security goes in multiple phases as per SDLC

- Requirements
- Design
- Implementation
- Verification
- Release
- Response

All these phases or application development lifecycle should have security.

Architects need to architect the design security, developers should be trained on secure coding, Operations should be made aware of the secure testing procedures, and applications should be tested post-production. Security has to be made a continuous cycle.

Not to Miss – Security Training shouldn't be just for security teams. It should be for all the teams at each level of security lifecycle.

The first and foremost thing before starting any application security testing, Understand the application and the attack surface. Check the application URLs, IP's, platform and other details.

Some of the questions could be asked like

- What is the language that the application is built on
- What is the platform behind the application
- The application is Internet-facing or Intranet facing
- Understand client and server communication model

An important thing to remember, don't start the testing with initiating the scans from automated scanners.

For the beginners, the best to start the testing is on vulnerable applications to get the feel of how the pentesting is being done. There are multiple vulnerable web applications which are available for testing.

- [Damn Vulnerable Node Application](#)
- [Damn Vulnerable Web Application](#)
- [Security Shepherd](#)
- [bWaPP](#)
- [Mutilldae](#)
- [Owasp Juice Shop](#)

The one amazing technic to find security bugs which are open is Google Dorking. Searching with certain queries on google search which might provide certain confidential information. Gather as much information to get to know about the application.

Use Nmap the application server for possible open ports, applications or services running. If there are applications with old versions, they might be susceptible to attack.

Setting up a lab is the next important and challenging task. While you are scrubbing the information about the application, set up the proxy (BurpSuite, Zap or any other) and start the spidering/crawling. About the proxy, Proxy acts as a middle man between the browser and the server. You can intercept the request which is being sent to the server and check the response from the server. This really helps in analysis the application and crafting attacks against the applications. Proxy runs on the local machine from where you perform the testing.

I use the proxy in Kali Linux which is a Virtual machine with all the magical security testing tools. Under the web application security tools, you can find BurpSuite, Zap and others. While configuring, Proxy and Browser should have the same configuration. You can automate the fuzzing by using proxy and perform many other useful tasks like finding the hidden directories, creating CSRF POC's, etc.

Once the proxy is set up, we can get starting with the application pentesting. We can take Owasp Top 10 as the basis to start our pentesting on the vulnerable web applications

OWASP Top 10-2017 vulnerabilities are:

- [Top 10-2017 A1-Injection](#)
- [Top 10-2017 A2-Broken Authentication](#)
- [Top 10-2017 A3-Sensitive Data Exposure](#)
- [Top 10-2017 A4-XML External Entities \(XXE\)](#)
- [Top 10-2017 A5-Broken Access Control](#)
- [Top 10-2017 A6-Security Misconfiguration](#)
- [Top 10-2017 A7-Cross-Site Scripting \(XSS\)](#)
- [Top 10-2017 A8-Insecure Deserialization](#)
- [Top 10-2017 A9-Using Components with Known Vulnerabilities](#)
- [Top 10-2017 A10-Insufficient Logging & Monitoring](#)

Wonderful references to know that can information from are:

- [OWASP Top 10 user guide](#)
- [Web application hackers handbook](#)

Open Web Application Security Project (OWASP) which is a non-profit organisation or community has contributed a lot to the web application security space.

These three areas where the Projects concentrate a lot is

- Tooling - Predominately attacker focused wherein tools will help you find bugs, slowly owasp is moving to devsecops as well.

Documentation

- [Owasp ASVS](#)
- [OWASP Security knowledge framework](#)

- [OWASP API testing](#)

You may have a project at different levels

- It's an open cultural organisation and anyone can contribute and share it to the world
- Go through them in detail, understand the concepts and then start the testing.

Along with that third is the Community part where OWASP is bringing people together to share knowledge and continuous learning. Besides community, owasp has conferences and owasp summits.

We should have a clear idea of what are the vulnerabilities which exist on the application, what is the impact or risk if the vulnerability is present on the application and how to mitigate them. If there is a vulnerability that exists on the server or an application which is not required anymore that should be decommissioned.

Applications are always easy targets, however, if we have proper security measures and processes in place, the risk impact reduces to a great extent.

Some of the best practices.

- All input data should be validated
- Implement a web application firewall (WAF)

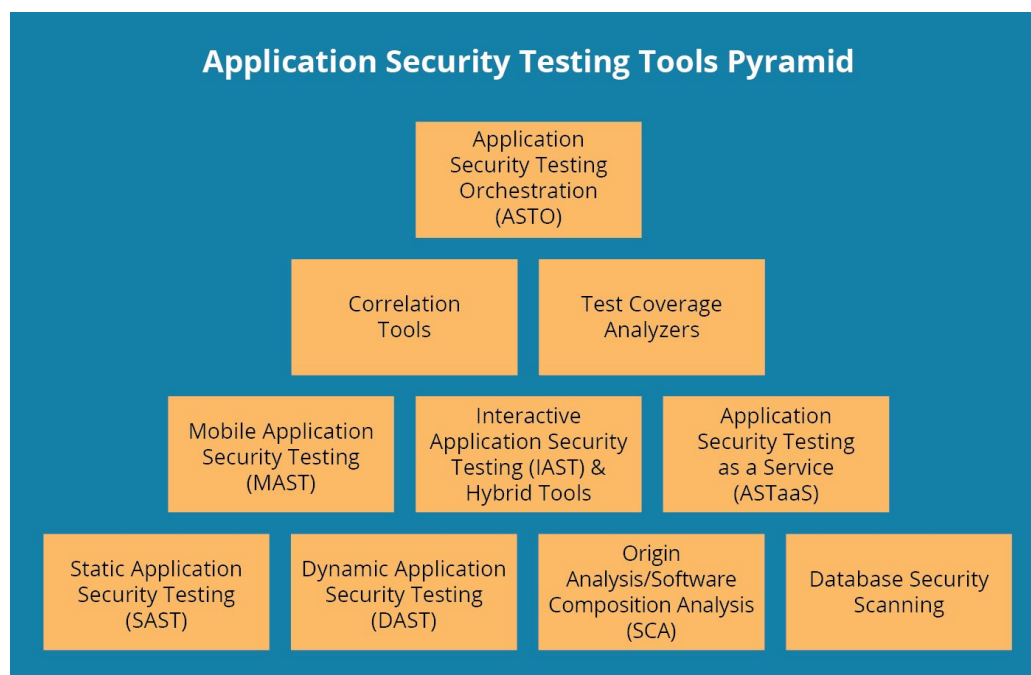
Libraries and open source vulnerability analysis

From [Synopsys report Why should I](#)

- In 9 of 17 industries, 100% of codebases contained open source. Open Source made up 70% of the audited codebases
- 75% of codebases contained vulnerabilities (up from 60% in 2018). 49% contained high-risk vulnerabilities (up from 40% in 2018)
- 33% of codebases contained unlicensed software. 67% of codebases had license conflicts
- 82% of codebases had components of more than four years out of date. 88% of the codebases had components with no development activity in the last two years
- Black Duck Audits found an open-source in nearly 99% of codebases audited in 2019
- In 2015 36% of code audited was open source. Today the figure is 70%.

Dynamic code analysis

Difference between SAST and DAST is that there is more context on the code (data is running in the application) but the test could be conducted externally (API web interfaces) or from within the application (elements of code within the application), this differentiation could be linked to [black-hat or black-box testing](#), where the tester has no prior knowledge of the system



Source : [SEI of Carnegie Mellon University](#)

Benefits,

Dynamic application security testing (DAST) is considered backbox testing because it does not have access to the application's source code. DAST sends HTTP requests and evaluates the responses to identify all the locations where an application accepts input. The process is called crawling the website, allowing the application to catalogue the URLs. Areas of an application accepting input are considered the attack surface (Mendele, Madou, & Sum, 2016). The catalogued URLs are then scanned for vulnerabilities. DAST analyzes an application by submitting various inputs against an application in an executed state to identify unexpected behaviour; unexpected behaviour and error messages may indicate a vulnerability exists in the application.

Unlike SAST, which analyzes source code, DAST tests the application, the system hosting the application and systems integrated with the application. DAST enables identifying vulnerabilities in the application, a misconfigured web server hosting the application, and vulnerabilities in a connected backend database - e.g., SQL injection. Running DAST against application tests within the context and environment where it is deployed (Halfond, Choudhary, & Orso, 2011).

Drawbacks

Drawbacks associated with DAST are it cannot analyze all execution paths of an application and ones that will never be executed (Prähofer et al., 2016). DAST is limited to specific use cases in a running state and may not test all aspects of the application. Vulnerabilities identified during DAST may be more complicated to fix, especially if fixing would break the functionality of the application. DAST executes

late in the testing phase of the SDLC, aligned with systems testing. Testing and fixing late in the SDLC is more expensive in terms of time, resources, and financial costs.

DAST may render an application unstable or nonresponsive (Mendele, Madou, & Sum, 2016). When an application is under load, the unpredictable nature of crawling and scanning an application may exhaust system resources. Additionally, large websites with numerous URLs and redirects may exhaust the scanner's resources causing the scanner to become unresponsive. This presents the problem of hanging scans and potentially distorted results as the scan fails to test all available URLs and possible vulnerabilities.

DAST Why should I?

SAST must understand the language and framework supporting the application. DAST is language agnostic and technology independent. DAST can work with applications developed using different languages and running on different platforms because it works within the HTTP protocol. Relying on HTTP requests and responses allows DAST to work with any language and web application framework (Acunetix). DAST provides an attacker's view of the application; running DAST on a live application provides a proactive approach to securing a web application.

Modern Application Security in Production

Introduction

The fast pace and iterative nature of the modern software delivery lifecycle have forced a fundamental change in the way organizations to approach application security. In the past, security teams often viewed their goal as to slow development, inhibit change, and prevent vulnerabilities from reaching production. As digital transformation accelerates, developers and DevOps teams have held very different priorities: to speed innovation and bring code to market as quickly as possible—even if this means leaving a few bugs to be addressed in the next iteration. As a result, tension has been increasing between security and development.

But agile development doesn't have to come at the expense of security. Done right, processes for continuous development and deployment can still produce reliable and secure software. But developers need more information, delivered as part of their process, as well as consistent training to do so successfully. We've already seen the rise of "shifting left," in which security becomes part of the design and development process from the beginning of the cycle rather than being addressed after the application is complete. "Shifting right" is just as important, allowing developers to gather data about the application in production and make decisions accordingly. Just as shifting left pushed security concepts and knowledge to developers, shifting right is about pushing the developers' responsibility for the product all the way to production.

In this section, we'll talk about practices, requirements, and tools for developers to play a meaningful role in production security.

Focus on bugs that matter

In an agile world, it's inevitable that some vulnerabilities will reach production. For developers to address the most critical bugs as quickly as possible, they need visibility into attacks as they occur. If they can see all of their actual attack traffic and understand what parts of their app attacks to focus on, then based on that information, they can make decisions about where and how to harden their code. The real attack is not a list of theoretical vulnerabilities. It's actual attackers attempting to breach the code developers wrote.

Enable real-time visibility

In simple terms, developers first need to know two things: How am I currently being attacked? And what vector of attack is being attempted? They need to proactively review this kind of attack data in real-time. To provide this information, security events can be pulled into DevOps tools (such as Slack, OpsGenie, VictorOps, and PagerDuty) they already use to monitor other aspects of their apps. When developers understand how their applications are being misused and attacked, it naturally facilitates the discussion between security and developers about what parts of the code should be proactively hardened -- e.g., the ones with the highest risk which have most critical bugs and are highly targeted by attackers.

Track vulnerability metrics

An informed approach to security should also include vulnerability metrics to help developers understand the urgency of remediating specific bugs, including:

- Mean Time to Attack – Having a list of attacks, the vulnerabilities they exploit, and how frequently they occur for a specific page can give the development team a sense of how much time they have before a given flaw will be attacked.
- Mean Time to Remediate – For third-party, open-source components and frameworks, developers should know how fast the project is updated when a security issue arises so they can decide whether to wait for a patch or take other mitigation measures first.

Build feedback loops

Technologies like next-generation web application firewalls (WAF) and runtime application self-protection (RASP) can be used to create feedback loops that help developers and DevOps teams protect applications. It's important to make this information available to everyone, not just the security team. By involving developers in security events as they happen, feedback loops can help move application security from a push model to a pull model—a more responsive and effective approach.

Set up usage feedback

Divergent usage patterns and anomalous behaviours can be a clear signal of a possible attack. Security and DevOps teams should get real-time notification of trends, such as unusually high volumes of logins, password changes, or new accounts being created. Web request traffic that attempts to access resources that don't exist, or result in spikes in traffic from uncommon sources, can also be red flags. Instrumenting common flows for users and tying them to application security feedback can provide further clarity—for example, correlating a spike in XSS attacks with a high number of password change requests.

Provide easy-to-implement recommendations within DevOps tools

Integrating security into the DevOps pipeline can lead to more secure code as well as lower development costs. There are many places in the cycle where information can be presented to the developer and provide feedback to fuel a virtuous loop. Policy and standards reminders, frameworks, and templates can help enforce secure coding and design patterns on development. As security checks are performed, developers should be offered proposed solutions for the problems highlighted, such as being given a secure code pattern in their development environment whenever they introduce a vulnerability into the code. This can prevent cross-site scripting flaws and database-injection vulnerabilities. Information from production can be used to inform the advice given to developers for a self-reinforcing feedback loop that builds best practices while empowering developers to make the codebase more resilient against attacks.

Deploy proactive web defense

Real-world application security means accepting that some vulnerabilities will reach production. Organizations should have a robust capability to detect potential attacks and address the underlying vulnerability before they impact the business, but defense in depth also includes being able to discover vulnerabilities even before they're exploited in an attack. There are several ways to do this, each with its own pros and cons.

Vulnerability scanners

Vulnerability scanning is a core element of defense in depth, though not enough in itself to ensure effective security. Scanning approaches include:

- **SAST (static analysis software testing)** – Code is scanned prior to deployment, with automated feedback and recommended fixes provided within the development environment.
 - **Pro:** The tester or developer has access to the underlying framework, design, and implementation of the code, making it simpler to fix.
 - **Con:** The test is performed on the code in a non-running state, not the application itself, so results are limited to coding errors and may not reflect how the code will operate and be attacked in production. SAST can also be challenging to implement at scale.
- **DAST (dynamic analysis software testing)** – This is similar to IAST, but performed at scale on multiple applications. As an application is running, the test tries to penetrate it from the outside in to identify potential vulnerabilities, including those outside the code and in third-party interfaces.
 - **Pro:** DAST provides context about how the application was exploited and how it responded to the attack. In-line advice speeds remediation.
 - **Con:** Since testing is done pre-production, it offers limited or no visibility into how the application will be attacked by real adversaries.
- **IAST (interactive application security testing)** – Typically used in a QA environment with automated tests running, IAST works inside the application.
 - **Pro:** Reporting vulnerabilities in real time, IAST does not slow down the CI/CD pipeline.
 - **Con:** IAST does not test the entire application or codebase, only whatever is exercised or probed by the functional test. Even if human QA staff devise the test, they can't foresee every attack method that could occur in production.

In general, while vulnerability scanning is an important part of testing strategy, its effectiveness is limited. As a scan of a point-in-time snapshot of the codebase, the test will not reflect any further changes made prior to deployment. And no scan can predict or provide visibility into how attackers will abuse the application once deployed to production. They also can't test the application in the real environment, so vulnerabilities in application logic or insecure configuration are not detectable.

Application testing, e.g. fuzzing in production

Designed to discover coding errors and security loopholes in software, including operating systems, fuzzing involves inputting massive amounts of random data (“fuzz”) in an attempt to make the application crash. If a vulnerability is found, a software tool called a fuzzer can be used to identify potential causes.

- **Pro:** Fuzzing can reveal serious defects in the codebase that are overlooked when originally designed, written, and debugged. Fuzzers can uncover vulnerabilities that can in turn be exploited by many OWASP Top 10 injection-style attacks.
- **Con:** To avoid crashing a business-critical application, fuzzing is performed only prior to deployment. While it can uncover serious code issues that could lead to an exploitation, it will not detect all the advanced tactics that real-world attackers would leverage in production.

As with the other testing types, fuzzing can be part of an overall defense-in-depth strategy to harden the code base prior to production, but it can’t provide a complete picture of the overall attack resilience of an application.

Live monitoring

Applications are monitored live in production by the WAF or RASP. A WAF acts as a first line of defense, applying rules to detect attacks, such DDoS and malicious bot-generated traffic. When an attack is detected, the WAF immediately blocks traffic to limit its scope while security and DevOps teams begin remediation. A RASP sits within the application and neutralizes malicious or malformed payloads and specific inputs to protect against both known and zero-day exploits.

- **Pro:** Implemented in production, live monitoring provides visibility into real-world attacks as they happen.
- **Con:** Many WAFs and RASPs require learning mode and constant signature tuning to rule out false positives. As a result of overly aggressive false positives, the aggressiveness of blocking rules gets turned down or completely turned off for fear of breaking the application. Note: not all WAFs and RASPs are created equal and newer next-gen iterations of these solutions are more flexible and effective at detection and blocking than legacy WAFs that rely on outdated regular expression pattern matching.

For live monitoring to be effective, false positives must be minimized to allow fully automated operation without unnecessarily disrupting production applications. One approach is to take a threshold approach that looks at suspicious payloads over time and with context to determine whether an actual attack is occurring, rather than simply making decisions on each request in isolation.

Bug bounty programs

As a form of crowdsourced penetration testing, a bug bounty program enables continuous, real-time testing by incentivizing independent researchers to report issues they discover. To be effective, a bug bounty program should have clear rules as to where researchers can look for bugs and how they should be reported. Internally, the organization should set baseline expectations and prepare to record key metrics such as number of bugs, the severity of each, and times to remediation and triage. The organization should also be prepared for a higher volume of attack, though actual risk will not increase.

- **Pro:** Bug bounties can validate where the internal security program is and isn’t working, and bring a broader range of expertise to bear on testing.
- **Con:** Organizations can be concerned about the cost of the rewards offered in the program, though many researchers are motivated more by recognition than money, making this less of an issue. Other reservations include the risk of inviting attacks, though in reality, web applications are under frequent attack anyway—at least a bug bounty program will yield a report on the attack.

Compared with penetration testing, a bug bounty is more likely to yield general or broad feedback. On the other hand, pentests, while less frequent, can be more directed and highly targeted on specific areas of concern. Together, the two methods are far more effective than either used in isolation.

Middle Ground

One new trend that could be adopted and appearing is the mitigation of vulnerabilities not fixed. One element to consider in all the previously mentioned technologies is the mitigation of vulnerabilities that can't be fixed immediately. A good tactic and approach are to inject the vulnerabilities from the code scanners or even better the testing environment as WAF block rules. Now in a pure DevOps way if the WAF is software side the rules can be easily adapted and tailored to a specific application. Nonetheless, network WAF can achieve the same but with either a list of rules installed or even better an API interface and the same ruleset pushed centrally to the WAF via API.

Conclusion

As development cycles accelerate, security can become an afterthought—but this doesn't have to be the case. By shifting development right, into production—with full visibility into security events and metrics—organizations can empower developments to take responsibility for the development and deployment of their applications and services. Development teams can create more secure code, gain visibility into vulnerabilities and actual attacks on their software, and use that information to harden their code. Feedback loops and inline advice can help enforce best practices and company policy across the development cycle. Complemented with proactive web defense, this makes it possible to ensure security at DevOps speed—for a fully modern approach to application security.

Conclusion

Closing Remarks

The Report is a live one and will continue changing. We would love to hear more from you and your feedback. Application security is an ever-changing landscape and currently, there is a fundamental battle between production and non-production, tools and too many tools.

Who will aggregate the two views and bring the humans (developers and management) together with the journey is going to win the battle?

if you want to hear more or you have any feedback please let us know something at

www.nsc42.co.uk/appreport

Thank You notes

A massive thank you to all the authors, contributors, editors, and reviewers of the document. This document truly embodies the power of the information security community full of amazing selfless and inspirational professionals

Contributors

Confirmed Quotes

- [] Tanya Janca

Confirmed authors:

- [] Kim C. - Methodology & Appsec
- [] Vandana V. - IBM - TBD - The art and Science of AppSec
- [] Sasha R. - GitHub - TBD - Pick a section
- [] Clint G. - Clint Gibler - Static code analysis
- [] Chris S. - Argp Grp - Static Code Analysis
- [] Jim M. - Jim Manico - Overview, Exec, State of appsec & method
- [] Philippe D. R. - SDLC code intro and common issue in code
- [] Emma H. - Report and Exec Summary
- [] Alyssa M. - Dependency check and libraries
- [] Andrew P. - Exec Summary and Prod security
- [] Jo Santiago - SDLC Section with Philippe
- [] Nicole B - TBD
- [] Jason T - TBD

Reviewers

- [] Grant O. - TBD Review

Bios

Francesco Cipollone



[Francesco Cipollone](#)

Francesco is an Executive, Public Speaker, out of the box thinker. Francesco is the Executive director of [NSC42 Ltd](#) a UK based cybersecurity consultancy. As an executive, he loves to stay close to the technology but to keep it simple. Francesco is data and result-driven Cyber Security Executive/vCISO highly regarded for planning and executing strategic infosec improvement programs that protect data and technical assets, reduce security risks, and align with long-term organisational goals. Francesco held a number of strategic positions ranging from Head of Application Security to Head of Security Architecture. Extensive experience with implementing security across multi-cloud providers (Amazon AWS, Microsoft Azure, Google Cloud). Francesco defines himself as driven to elevate the cybersecurity world one organization at a time, embracing an innovative approach to application security to protect the engineering environment. Recognized as a motivational, influential leader who guides high-performing teams to deliver projects on time and exceeding quality expectations, while instilling a culture of best practices and collaboration. Builds lasting relationships with board members and C-level executives. Delivers education and training to members at all levels of an organisation, building awareness for security initiatives while fostering a common security purpose. Internationally renowned public speaker, with multiple interviews in high-profile publications (eg. Forbes), and an author of numerous books and articles, who utilises his platform to evangelise the importance of cloud security and cutting-edge technologies on a global scale.

Dr. Chris Sellards



Chris Sellards has a Doctor of Science in Cybersecurity from Capitol Technology University. His dissertation was a quantitative study focused on DevSecOps. He has 24 years of experience in IT, over 20 years in information security, and 15 years working with application security. He has built AppSec programs in the medical, financial services, and insurance industries. He has developed the strategy driving AppSec programs aligned with business security requirements (both for in-house dev teams and outsourced) and has done the hands-on work - implementing automated SAST into multiple DevOps pipelines and analyzed findings with developers to identify false positives, tuning queries, setting up incremental scans, and integrating output with tracking tools. He has worked with software composition analysis using open source and commercial products. He has worked with QA to implement tools and processes enabling QA to perform automated testing aligned with threat models. He currently serves as Director of Security Architecture & Engineering at The Argo Group and as an Adjunct Professor at the University of Texas at San Antonio.

I am passionate about application security; even more passionate about peer-reviewed work adding value to a market saturated industry. Applications have become a critical component of the world's infrastructure and economy. The rapid culture shift driving the expansion of applications becoming critical infrastructure has done little to address security. The interconnected world provides a pathway for application vulnerabilities to escalate localized incidents to a global scale. Following Dr. Josiah Dykstra's claims of the lack of science in cybersecurity, working on this project provides an opportunity to build axioms into the SDLC.

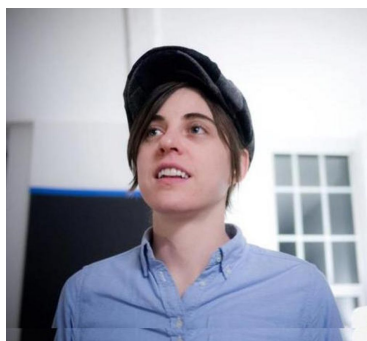
Tanya Janca



Tanya Janca, also known as ‘SheHacksPurple’, is the founder of We Hack Purple, a tech startup specializing in security training for IT professionals. Tanya has been coding since she was a teen, has worked in IT for over twenty years, has won numerous awards, and has done everything from start her own company several times, been a public servant, and worked for tech giants such as Microsoft, Adobe, and Nokia. She has been a startup founder, pentester, CISO, CEO, AppSec Engineer, Sys and Network Admin but mostly a software developer. She is an award-winning public speaker, active blogger & streamer and has delivered hundreds of talks and trainings on 6 continents. She values diversity, inclusion and kindness, which shines through in her countless projects and achievements.

Founder: We Hack Purple, WoSEC (Women of Security), OWASP DevSlop, OWASP Victoria, #CyberMentoringMonday

Nicole Becher



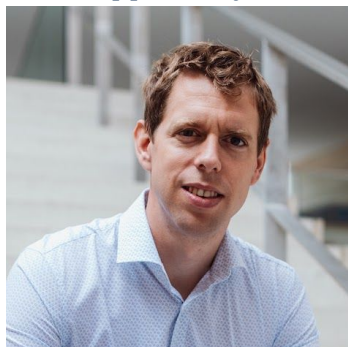
Nicole Becher is currently the Director of Information Security & Risk Management for S&P Global Platts, a leading provider of energy and commodities information and benchmark price assessments in the physical commodity markets. In this role, she works with both technology and business leadership to ensure security is built into the strategic plans of the organization, especially as new technology is deployed.

Nicole is also an Adjunct Instructor at New York University, where she teaches courses on offensive and defensive computer security, network security, web application security, and computer forensics. She is a project leader for OWASP DevSlop Project (Open Web Application Security Project).

Nicole has presented both technical talks and training, at various conferences around the world on topics related to her research interests. Since Nicole has held technical, leadership and policy roles she has a very unique perspective in the information security space. She was a Cybersecurity fellow of New

America, a Washington DC-based think-tank, and was a fellow of the Madison Policy Forum, a cybersecurity-focused policy group bridging military, government and industry.

Dr. Philippe De Ryck



Philippe De Ryck helps developers protect companies through better web security. His Ph.D. in web security from KU Leuven lies at the basis of his exceptional knowledge of the security landscape. As the founder of Pragmatic Web Security, Philippe delivers security training and security advisory services to companies worldwide. His online course platform allows anyone to learn complex security topics at their own pace. Philippe is a Google Developer Expert and an Auth0 Ambassador/Expert for his community contributions on security of web applications and APIs.

Vandana Verma Sehgal



Vandana is a seasoned security professional with experience ranging from application security to infrastructure and now dealing with Product Security. She has been Keynote speaker / Speaker / Trainer at various public events ranging from Global OWASP AppSec events to BlackHat events to regional events like BSides events in India.

She is part of the OWASP Global board of directors. She also works in various communities towards diversity initiatives InfosecGirls, WoSec and null.

She has been recipient of multiple prestigious awards like Global cybersecurity influencer among IFSEC Global's "Top Influencers in Security and Fire" Category for 2019, Cybersecurity Women of the year award by Women Cyberjutsu Society in the Category "Secure Coder". She has also been listed as one of the top women leaders in this field of technology and cybersecurity in India by Instasafe.



Andrew Peterson is the CEO and Co-Founder of Signal Sciences. Under Peterson's leadership, Signal Sciences has become the leading provider of next-gen WAF and RASP technology as well as the fastest growing application security company in the world. As CEO, Peterson is responsible for overseeing all business functions, go-to-market activities, and attainment of strategic, operational and financial goals.

Prior to founding Signal Sciences, Peterson has been building leading edge, high performing product and sales teams across five continents for over fifteen years with such companies as Etsy, Google, and the Clinton Foundation. In 2016, O'Reilly published his book [Cracking Security Misconceptions](#) to encourage non-security professionals to take part in organizational security. He graduated from Stanford University with a BA in Science, Technology, and Society.

References

- Information is beautiful - Data Breaches - <https://informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/>
- Statista Average cost of all cybersecurity breaches uk 2019 - <https://www.statista.com/statistics/586788/average-cost-of-cyber-security-breaches-for-united-kingdom-uk-businesses/>
- Uk Cybersecurity Breaches Survey - [Cyber Security Breaches Survey 2019, page 51](#)
- Verizon 2020 Data Breach Report - <https://enterprise.verizon.com/resources/reports/dbir/>
- Forrester Report - <https://reprints.forrester.com/#/assets/2/1855/RES159057/reports>
- SNYK State of Open source Report - <https://snyk.io/open-source-security-report/>
- Whitesource State of Open source Security - <https://resources.whitesourcesoftware.com/wistia-webinars/the-state-of-open-source-security-vulnerabilities-in-2020>
- Sonatype State of Application Security 2020 - <https://www.sonatype.com/forrester-state-of-appsec>
- Top 2018 appsec breaches - <https://www.immuniweb.com/blog/top-ten-application-security-databreaches-2018.html>
- Wors 2019 data breaches - <https://www.zdnet.com/article/these-are-the-worst-hacks-cyberattacks-and-data-breaches-of-2019/>
- Top 15 data breaches of the 21st century - <https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html>
- WhiteSource - SAST Basic - <https://resources.whitesourcesoftware.com/blog-whitesource/sast-static-application-security-testing>
- (TBD) - Gitlab State of the Union - <https://amazicworld.com/gitlabs-state-of-the-union-on-devsecops/>
- The state of application security testing - (Mello, 2020)
- Humble, J., & Farley, D. (2011). Continuous delivery: Reliable software releases through, build, test, and deployment automation. Boston, MA: Pearson Education, Inc.
- Bell, L., Brunton-Spall, M., & Bird, J. (2017). Agile application security: Enabling security in a continuous delivery pipeline. Sebastopol, CA: O'Reilly Media, Inc
- Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The devops handbook: How to create world-class agility, reliability, & security in technology organizations. IT Revolution: Portland, OR.
- Gregory, J., & Crispin, L. (2009) Agile testing: A practical guide for testers and agile teams. Pearson Education, Inc: Boston, MA.
- Shahin, M., Babar, M., & Zhu, L. (2017). Continuous integration, delivery, and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 5(0), 3909-3943. Retrieved from <https://ieeexplore.ieee.org/abstract/document/7884954>
- Chess, B., & McGraw, G. (2004). Static analysis for security. IEEE security & privacy, 2(6), 76-79.
- Vehent, J. (2018). Securing devops: Security in the cloud. Shelter Island, NY: Manning Publications Co.
- Mainstay Partners. (2014). Does application security pay? Measuring the business impact of software security assurance solutions. Retrieved from http://h30528.www3.hp.com/Security/Fortify_Mainstay_ROI_Study.pdf

- Logozzo, Francesco & Fähndrich, Manuel. (2008). On the Relative Completeness of Bytecode Analysis Versus Source Code Analysis. 197-212. 10.1007/978-3-540-78791-4_14.
- Antunes, N., & Vieira, M. (2009, November). Comparing the effectiveness of penetration testing and static code analysis on the detection of sql injection vulnerabilities in web services. In 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing (pp. 301-306). IEEE.
- Veracode. (2018). State of software security. State of Software Security, 2018(9).
- Bird, J. (2017). 2017 state of application security: Balancing speed and risk. SANS. Retrieved from <https://www.sans.org/reading-room/whitepapers/application/paper/38100>
- Hall, E. (2017). 5 ways to improve security in the era of continuous delivery. Gartner IT Quarterly, 2017 (Second Quarter) 14-17. Retrieved from <https://www.gartner.com/landing/home.html>
- 451 Research. (2018). Devsecops realities and opportunities. Retrieved from <https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/devsecops-realities-opportunities-451.pdf>
- Bird, J. (2015). Devopssec: Delivering secure software through continuous delivery. Sebastopol, CA: O'Reilly Media, Inc.
- Smada, D., Rotună, C., Boncea, R., & Petre, I. (2018). Automated code testing system for bug prevention in web-based user interfaces. Informatica Economica, 22(3), 23-32. doi: <http://dx.doi.org/10.12948/issn14531305/22.3.2018.03>
- Prähofner, H., Angerer, F., Ramler, R., & Grillenberger, F. (2016). Static code analysis of IEC 61131-3 programs: Comprehensive tool support and experiences from large-scale industrial application. IEEE Transactions on Industrial Informatics, 13(1), 37-47.
- Mendelev, K., Madou, M., & Sum, S. N. M. (2016). "U.S. Patent No. 9,438,617". Washington, DC: U.S. Patent and Trademark Office.
- Halfond, W., Choudhary, S., & Orso, A. (2011). Improving penetration testing through static and dynamic analysis. "Software Testing, Verification and Reliability, 21"(3), 195–214. <https://doi.org/10.1002/stvr.450>
- Li, Y., Das, P., & Dowe, D. (2014). Two decades of Web application testing—A survey of recent advances. "Information Systems, 43", 20–54. <https://doi.org/10.1016/j.is.2014.02.001>
- Acunetix <https://www.acunetix.com/blog/articles/dast-dynamic-application-security-testing/>
- Mello, J. P., Jr (2020) 'The state of application security testing: The shift is on to secure code'. TechBeacon. Available at: <https://techbeacon.com/security/state-app-sec-testing-get-your-shift-secure-code> (Accessed: 23 July 2020).