Merete Eggestad
Ingrid Høyaas
Oda Løken
Julie Stade Stålevik

Software Security Testing

Bachelor's thesis in Digital Infrastructure and Cyber Security
Supervisor: Erik Hjelmås
May 2022

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Kunnskap for en bedre verden

Merete Eggestad
Ingrid Høyaas
Oda Løken
Julie Stade Stålevik

Software Security Testing

Bachelor’s thesis in Digital Infrastructure and Cyber Security
Supervisor: Erik Hjelmås
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Abstract

Title:	Software Security Testing
Date:	20.05.2022
Participants:	Merete Eggestad Ingrid Høyaas Oda Løken Julie Stade Stålevik
Supervisor:	Erik Hjelmås, Associate Professor, Department of Information Security and Communication Technology
Employer:	Tor Birger Skogen, NDMA ICT
Keywords:	Information security, software security, automatic security testing, DAST, SAST, SCA
Pages:	92
Attachments:	10
Availability:	Open
Abstract:	Software security testing is important to detect vulnerabilities in the software before it is deployed to production. NDMA wanted suggestions on how they could improve their software security routines. The thesis group have examined different tools that can be used on software. They cover SAST, DAST, and/or SCA. Some of the tools have been tested, and the results were compared to each other. Furthermore, the tools were evaluated based on advantages and disadvantages, before the thesis group came up with recommendations on which solution that can best suit NDMA. These solutions were Veracode, or Snyk together with StackHawk.

Sammendrag

Tittel:	Software Security Testing
Dato:	20.05.2022
Deltakere:	Merete Eggestad Ingrid Høyaas Oda Løken Julie Stade Stålevik
Veileder:	Erik Hjelmås, Førsteamanuensis, Institutt for informasjonssikkerhet og kommunikasjonsteknologi
Oppdragsgiver:	Tor Birger Skogen, FMA IKT
Nøkkelord:	Informasjonssikkerhet, prorgamvaresikkerhet, automatisk sikkerhetstesting, DAST, SAST, SCA
Antall sider:	92
Antall vedlegg:	10
Tilgjengelighet:	Åpen
Sammendrag:	Sikkerhetstesting av programvare er viktig for å oppdage sårbarheter i programvaren før den distribueres til produksjon. FMA ønsket forslag til hvordan de kunne forbedre sikkerhetsrutinene sine. Prosjektgruppen har undersøkt ulike verktøy som kan brukes på programvare. De dekker SAST, DAST og/eller SCA. Noen av verktøyene er testet, og resultatene ble sammenlignet med hverandre. Videre ble verktøyene evaluert ut fra fordeler og ulemper, før prosjektgruppen kom med anbefalinger om hvilke løsninger som kan passe best for FMA. Disse løsningene er Veracode, eller Snyk sammen med StackHawk.

Preface

Many thanks to our supervisor, Erik Hjelmås, for good guidance and help throughout the whole project period.

Thanks to Tom Røise for advice on how to motivate NDMA to use the recommended tools.

Thanks to other professors at NTNU for help with issues with the testing.

Thanks to Martin Stene for letting the group borrow both an API and a Java software to test during the experiment.

Thanks to Odin Jenseg, Tor Erling Bjørstad, and Joakim von Brandis in Mnemonic for helpful information about Snyk and how they use it.

Thanks to external companies for helpful information about how they perform security checks on their software, and which tools they use.

Thanks to family and friends for reviewing the thesis and for providing feedback.

And finally, many thanks to our contact person in NDMA, Håkon Liberg, for guidance and help, for setting up interviews, and for all the information provided. Also, thanks to the developers in NDMA for partaking in the interviews, and for being helpful and sharing. We look forward to sharing the results of this collaboration with you.

Contents

Abstract . . . . **iii**
Sammendrag . . . . **v**
Preface . . . . **vii**
Contents . . . . **ix**
Figures . . . . **xv**
Tables . . . . **xix**
Acronyms . . . . **xxi**
1 Introduction **1**
1.1 Background 1
1.2 Problem Area 1
1.3 Scope and Limitations 2
1.4 Target Group 2
1.5 Goals 2
1.6 Framework 3
1.7 The Group's Academic Background 3
1.7.1 Why This Task Was Chosen 4
1.8 Methodology 4
1.8.1 Software Engineering Method 4
1.8.2 Research Methods 5
1.8.2.1 Literature Study 5
1.8.2.2 Interviews 6
1.8.2.3 Experiment 6
1.9 Thesis Structure 6
1.9.1 Chapters 7
2 Theory **9**
2.1 Introduction 9
2.2 Testing Functionality Compared to Security 9
2.3 Box Testing 10
2.3.1 White Box Testing 10
2.3.2 Black Box Testing 11
2.3.3 Gray Box Testing 11
2.4 Application Security Assessment 12
2.4.1 Shift-Left 12

2.4.2 False Positives and Negatives 12
2.5 Application Security Testing and Analysis 13
2.5.1 Static Application Security Testing (SAST) 13
2.5.2 Dynamic Application Security Testing (DAST) 14
2.5.3 Interactive Application Security Testing (IAST) 14
2.5.4 Runtime Application Security Protection (RASP) 15
2.5.5 Software Composition Analysis (SCA) 16
2.6 The Importance of Software Security Testing 16
2.7 The Odds of Finding Vulnerabilities 17
2.8 Test Environment 18
2.9 Penetration Testing 18
2.10 Fuzzing 19
2.11 OWASP Top 10 19
2.12 Vulnerability Estimators 20
2.12.1 Common Vulnerability Scoring System (CVSS) 20
2.12.2 Common Vulnerabilities and Exposures (CVE) 20
2.12.3 Common Weakness Enumeration (CWE) 21
2.12.4 OWASP Risk Rating Methodology 21
2.13 Containers 22

3 Tools for Security Testing 25
3.1 Introduction 25
3.2 Today’s Solution 25
3.3 Tools 26
3.3.1 Comparison Table 26
3.3.2 Overview of Tools 28

4 Analysis of Selected Tools 33
4.1 Introduction 33
4.2 Contrast Security 33
4.2.1 Why Contrast Security Was Chosen 33
4.2.2 Product Overview 34
4.2.2.1 Contrast Scan 34
4.2.2.2 Contrast SCA 34
4.2.2.3 Contrast Assess 34
4.2.2.4 Contrast Serverless 35
4.2.2.5 Contrast Protect 35
4.2.3 How Contrast Security Handles Data 35
4.2.4 Vulnerability Scoring 35
4.3 Snyk 36
4.3.1 Why Snyk Was Chosen 36
4.3.2 Product Overview 37
4.3.2.1 Snyk Open Source 37
4.3.2.2 Snyk Container 37
4.3.2.3 Snyk Code 38
4.3.2.4 Snyk Infrastructure as Code 38

4.3.2.5 Snyk Learn . . . 38
4.3.2.6 Snyk Vulnerability Database . . . 38
4.3.3 How Snyk Handles User Code and Repositories . . . 39
4.3.3.1 Snyk Open Source . . . 39
4.3.3.2 Snyk Container . . . 39
4.3.3.3 Snyk Code . . . 39
4.3.3.4 Snyk Infrastructure as Code . . . 40
4.3.4 Vulnerability Scoring . . . 40
4.4 StackHawk . . . 40
4.4.1 Why StackHawk Was Chosen . . . 40
4.4.2 Product Overview . . . 40
4.4.3 How StackHawk Handles Data . . . 41
4.4.4 Vulnerability Scoring . . . 41
4.5 Veracode . . . 41
4.5.1 Why Veracode Was Chosen . . . 41
4.5.2 Product Overview . . . 42
4.5.2.1 Veracode Static Analysis . . . 42
4.5.2.2 Veracode Software Composition Analysis . . . 42
4.5.2.3 Veracode Dynamic Analysis . . . 43
4.5.2.4 Veracode Discovery . . . 43
4.5.2.5 Veracode Manual Penetration Testing . . . 44
4.5.2.6 Veracode eLearning . . . 44
4.5.3 How Veracode Handles Data . . . 44
4.5.4 Vulnerability Scoring . . . 44
4.6 Selected Tools in the SDLC . . . 45
5 Experiment . . . 47
5.1 Introduction . . . 47
5.2 The Test Environments . . . 47
5.2.1 Test Data . . . 47
5.3 Tested Tools . . . 48
5.3.1 Contrast Security . . . 48
5.3.1.1 Community Edition . . . 48
5.3.2 Snyk . . . 51
5.3.2.1 GitHub . . . 51
5.3.3 StackHawk . . . 53
5.3.3.1 HawkScan . . . 53
5.4 Untested Tools . . . 58
5.4.1 Contrast Security . . . 58
5.4.1.1 Contrast Scan (SAST) . . . 58
5.4.2 Veracode . . . 61
5.4.2.1 Veracode SCA . . . 61
5.4.2.2 Veracode Dynamic Analysis (DAST) . . . 64
5.4.2.3 Veracode Static Analysis (SAST) . . . 67
5.5 Results . . . 71

6 Discussion **73**
6.1 Introduction 73
6.2 Result Interpretation 73
6.3 Advantages and Disadvantages 75
6.3.1 Contrast Security 75
6.3.2 Snyk 77
6.3.3 StackHawk 79
6.3.4 Veracode 81
6.4 Other Security Options for Testing and Analysis 82
6.5 Manual Testing Methods 82
6.6 Containers 83
6.7 Limitations 83
6.7.1 Research 83
6.7.2 Testing 83
6.7.3 Other 84
6.8 Critique of the Thesis 84
6.8.1 Documentation 84
6.8.2 Testing 85
6.8.3 New Tools 85
6.8.4 Changes 85
6.8.4.1 Invicti Security 85
6.8.4.2 Contrast Security 86
6.8.4.3 Snyk and StackHawk 86
7 Conclusion **87**
7.1 Introduction 87
7.2 Reflections 87
7.2.1 Software From NDMA 87
7.2.2 Interviews 88
7.2.3 Customer Persuasion Meeting 88
7.2.4 Getting Acquainted With New Tools 88
7.2.5 Evaluation of the Working Process 88
7.2.5.1 Halfway Assessment 88
7.2.5.2 Gantt 89
7.2.5.3 System Development Model 90
7.2.5.4 Work Distribution 90
7.2.5.5 Goals 91
7.3 Further Work 92
7.4 Conclusion 92
Bibliography **93**
A Contrast **107**
A.1 Contrast Scan (SAST) 107
B Snyk **115**
B.1 GitHub 115
B.1.1 Test With an Old Exam 115

B.1.2 Test with FastTasks 120
C Veracode 123
C.1 Veracode SCA 123
C.1.1 Upload and Scan 123
C.1.2 Agent-Based Scan 126
C.2 Veracode Dynamic Analysis 129
C.2.1 Authenticated Dynamic Scan 129
C.2.2 Unauthenticated Dynamic Scan 130
C.2.3 Dynamic Analysis for Internal Scanning 131
C.3 Veracode Static Analysis 132
C.3.1 GitHub 132
C.3.2 Visual Studio Code 136
C.3.2.1 Installation of Veracode for VS Code 136
C.3.2.2 Scanning source code with Veracode for VS Code . 139
C.3.2.3 Review findings in Veracode for VS Code 142
C.3.3 Veracode Platform 146
D Intervjuguide 149
D.1 Intervju med ekstern bedrift 149
D.2 Intervju med utviklere i FMA 151
E Oppgavebeskrivelse 157
F Samarbeidsavtale 159
G Prosjektplan 167
G.1 Mål og rammer 167
G.1.1 Bakgrunn 167
G.1.2 Prosjektmål 167
G.1.3 Rammer 168
G.2 Omfang 168
G.2.1 Problemområde 168
G.2.2 Problemstilling og avgrensninger 169
G.3 Prosjektorganisering 169
G.3.1 Ansvarsforhold og roller 169
G.3.2 Rutiner 170
G.3.3 Grupperegler 170
G.4 Planlegging, oppfølging og rapportering 170
G.4.1 Hovedinndelingen av prosjektet 170
G.5 Organisering av kvalitetsikring 172
G.5.1 Dokumentasjon og verktøy 172
G.5.2 Plan for inspeksjoner og testing 172
G.5.3 Risikoanalyse 172
G.6 Plan for gjennomføring 178
G.6.1 Aktiviteter 178
G.6.2 Milepæler 180
G.6.3 Gantt-skjema 180
H Relevant mailkommunikasjon 181

H.1 Undersøkte verktøy 181
H.1.1 StackHawk 181
H.1.2 Invicti Security 183
H.2 Eksterne bedrifter 183
H.2.1 Mnemonic 183
H.2.2 Basefarm 186
H.2.3 Finansinstitusjon 187
H.3 Faglærere på NTNU 188
H.3.1 Ivar Farup 188
H.3.2 Kiran Raja 190
H.3.3 Lars Erik Pedersen 192
H.4 Utvikler i FMA 192
I Referat 195
I.1 Referat fra møter med veileder 195
I.2 Referat fra møter med oppdragsgiver 204
I.3 Referat fra egne møter 211
I.4 Referat fra møter med utviklere 213
I.5 Referat fra møter og intervjuer med diverse eksterne 217
J Timeliste 221

Figures

1.1 Software development model 5

2.1 Application Security Testing 13
2.2 SAST placed in the SDLC 13
2.3 DAST placed in the SDLC 14
2.4 SCA placed in the SDLC 16
2.5 OWASP Top 10 in 2017 vs. OWASP Top 10 in 2021 20
2.6 The ranking of vulnerabilities in version 3.1 20
2.7 The likelihood and impact levels 21
2.8 The risk severity 22

3.1 Information about all the examined tools 27

4.1 Contrast Security products placed in the SDLC 34
4.2 The risk severity 36
4.3 Snyk products placed in the SDLC 37
4.4 Veracode products placed in the SDLC 42
4.5 The tools placed in the SDLC 45
4.6 The tools and what type of software they can be used on 45

5.1 The results from the test 49
5.2 Overview of the vulnerability found 50
5.3 Details about the vulnerability found 50
5.4 Information about how to fix the vulnerability 51
5.5 Adding projects to scan for vulnerabilities 52
5.6 Scanned project with vulnerabilities ranked by grade of severity . . 52
5.7 An example of a suggested fix 53
5.8 Issue with permissions 54
5.9 Error fix 54
5.10 First time running the command for starting a test 54
5.11 Results from test shown in PowerShell 55
5.12 Test in progress on StackHawk's website 56
5.13 Website-version of the test results 56
5.14 Details of found vulnerability 57

5.15 Details on the threat 58
5.16 Overview over scans and add new button in the right corner 59
5.17 Starting a new scan in the CLI 59
5.18 One scan done on the project named "Java Scan" 60
5.19 Overview of a vulnerability 60
5.20 The Veracode Platform 61
5.21 The results from the SCA scan 62
5.22 The commands needed to create an agent in the CLI 63
5.23 The results from the scan in the CLI as a JSON output 63
5.24 The Dynamic Analysis workflow 64
5.25 The Blocklist 65
5.26 The Allowlist 65
5.27 Different ways to authenticate 66
5.28 The results from the test 67
5.29 The GitHub repository that is going to be scanned 68
5.30 The veracode-analysis.yml file 69
5.31 Overview of all the issues found in the scan 70

6.1 Results from Snyk retesting 74

7.1 The original Gantt 90
7.2 The actual Gantt 90

A.1 Starting a new scan in the CLI 107
A.2 Get started by choosing what is going to be scanned 108
A.3 Creating a scan project 108
A.4 A new and empty project 109
A.5 Uploading the file and starting the scanning it 109
A.6 Vulnerabilities in the project 110
A.7 Details about a vulnerability 110
A.8 Information on how to fix a vulnerability 111
A.9 Notes on a vulnerability found in the project 111
A.10 Vulnerabilities can be marked manually 112
A.11 The vulnerabilities can be sorted on different statuses 112
A.12 Overview of policies used 113
A.13 The results can be shared with others i a JSON file or CSV file . . . 113

B.1 Adding projects to scan for vulnerabilities 115
B.2 Scanned project with vulnerabilities ranked by grade of severity . . 116
B.3 Found vulnerability with context and score 116
B.4 More information about a vulnerability 117
B.5 List of scanned projects 117
B.6 Fix recommendation 118
B.7 Fix recommendation 118
B.8 Fix PR 119

B.9 Failed scan due to it being a C++ project . . . 119
B.10 Documentation for .NET support . . . 120
B.11 Partly failed scan . . . 120
B.12 Test result . . . 121
B.13 Vulnerability example . . . 121
B.14 Unscanned files . . . 122

C.1 The types of scans that can be done . . . 123
C.2 Packaging requirements for the relevant supported language . . . 124
C.3 The vulnerability database . . . 124
C.4 Different ways to start a scan . . . 125
C.5 Where to add new rules . . . 125
C.6 Where to start the Veracode agent-based scan . . . 126
C.7 Where to create a workspace . . . 127
C.8 The commands in the CLI and the how to start a scan . . . 127
C.9 The URL to the results in the Veracode Platform . . . 128
C.10 List of discovered CVEs from the scan . . . 128
C.11 Information about the Dynamic Analysis . . . 129
C.12 Uploading the CSV file with URLs . . . 131
C.13 Where to select gateway and endpoint . . . 132
C.14 The Veracode documentation on how to set up API credentials . . . 133
C.15 Where to add the credentials . . . 133
C.16 The Veracode template to perform a scan . . . 134
C.17 Overview of the running workflows . . . 135
C.18 All the different steps in the scan . . . 135
C.19 Information about one specific issue . . . 136
C.20 The files in the extension folder . . . 137
C.21 Which extension that needs to be installed . . . 138
C.22 The details and how to run the scan . . . 138
C.23 The opened folder with all the files . . . 139
C.24 The first way to scan a file . . . 140
C.25 The second way to scan a file . . . 140
C.26 The third way to scan a file . . . 141
C.27 The results in the Veracode tab . . . 141
C.28 How to scan a folder . . . 142
C.29 All CWEs in the code . . . 143
C.30 The issue marked with a red underline . . . 143
C.31 More information about the issue . . . 144
C.32 The details panel with a description of the CWE . . . 144
C.33 Different severities can be filtered out . . . 145
C.34 The filter icon . . . 145
C.35 How to temporarily ignore findings . . . 146
C.36 Guidance for the specific language and platform . . . 146
C.37 When a scan is completed . . . 147

C.38 The results from the static analysis scan 147

E.1 Oppgavebeskrivelse 157

G.1 Systemutviklingsmodell 171
G.2 Gantt-skjema 180

H.1 Mail from StackHawk 182
H.2 Mail from Invicti 183
H.3 Mail til Odin i Mnemonic 183
H.4 Svar fra Odin i Mnemonic 184
H.5 Svar til Odin 184
H.6 Svar fra Odin med kopi til Tor i Mnemonic 184
H.7 Mail fra Tor i Mnemonic 184
H.8 Mail til Tor med flere spørsmål 185
H.9 Svar fra Tor med kopi til Joakim i Mnemonic 185
H.10 Svar fra Joakim 186
H.11 Mail fra Basefarm 186
H.12 Mailkommunikasjon med finansinstituasjon 187
H.13 Tips om nettside fra finansinstitusjon 187
H.14 Mailkommunikasjon med Ivar 188
H.15 Mailkommunikasjon med Ivar 188
H.16 Mailkommunikasjon med Ivar 189
H.17 Mailkommunikasjon med Ivar 189
H.18 Mailkommunikasjon med Ivar 190
H.19 Mailkommunikasjon med Kiran 190
H.20 Mailkommunikasjon med Kiran 191
H.21 Mailkommunikasjon med Kiran 191
H.22 Mailkommunikasjon med Lars Erik 192
H.23 Mail til utvikler i FMA 192
H.24 Mail fra utvikler i FMA 193

J.1 Antall timer jobbet med Bacheloroppgave - Oda 221
J.2 Antall timer jobbet med Bacheloroppgave - Merete 222
J.3 Antall timer jobbet med Bacheloroppgave - Julie 222
J.4 Antall timer jobbet med Bacheloroppgave - Ingrid 223
J.5 Antall timer jobbet per uke 223

Tables

3.1 Aqua Security/Trivy, Burp Suite and Contrast Security 29
3.2 Detectify, Invicti Security and Klocwork 30
3.3 Nessus, OWASP ZAP and Reshift 31
3.4 Snyk, StackHawk and Veracode 32

5.1 Results from the tests 71

6.1 Advantages and disadvantages discovered about Contrast Security . 75
6.2 Advantages and disadvantages discovered about Contrast Security . 76
6.3 Advantages and disadvantages discovered about Snyk 77
6.4 Advantages and disadvantages discovered about Snyk 78
6.5 Advantages and disadvantages discovered about StackHawk 79
6.6 Advantages and disadvantages discovered about StackHawk 80
6.7 Advantages and disadvantages discovered about Veracode 81

G.1 Risikoanalyse 1 173
G.2 Risikoanalyse 2 174
G.3 Risikoanalyse 3 175
G.4 Risikoanalyse 4 176
G.5 Risikoanalyse 5 177
G.6 Milepæler 180

Acronyms

AI Artificial Intelligence. 38

API Application Programming Interface. vii, 29, 32, 34, 40, 43, 48, 53, 61, 64, 67, 71, 74, 79, 123, 132, 133, 137

ARM Azure Resource Manager. 38

AWS Amazon Web Services. 32, 36, 41

BOM Bill of Materials. 16

cURL Client URL. 32, 48

CI continuous integration. 43

CI/CD Continuous Integration/Continuous Delivery. 15, 18, 32, 37, 38, 40, 42

CLI Command Line Interface. 34, 40, 59, 62, 63, 75, 107, 127

CSV Comma-Separated Values. 60, 112, 113, 130, 131

CVE Common Vulnerabilities and Exposures. 20, 21, 35, 128

CVSS Common Vulnerability Scoring System. 20, 40, 44

CWE Common Weakness Enumeration. 20, 21, 44, 142–144

DAST Dynamic Application Security Testing. iii, v, 14, 15, 40, 41, 43, 45, 53, 64, 71, 74, 77, 79, 81, 82, 84, 86

EMEA Europe, Middle East, Africa. 32

EU European Union. 3, 26

FMA Forsvarsmateriell. v

GUI Graphical User Interface. 40

HQ Headquarter. 29, 32

HTML HyperText Markup Language. 48

HTTP HyperText Transfer Protocol. 130

IaC Infrastructure as Code. 32, 37, 38, 40

IAST Interactive Application Security Testing. 14, 15, 29, 33, 34, 48, 58, 71, 73–75, 82

ICT Information and communications technology. iii

IDE Integrated Development Environment. 14, 25, 31, 42, 51, 67, 136, 137, 142

IKT Informasjons- og kommunikasjonsteknologi. v

IP Internet Protocol. 43, 129, 131

IR Internal or Interagency Reports. 21

ISM Internal Scanning Management. 43

IT Information Technology. 1

JSON JavaScript Object Notation. 60, 62, 63, 112, 113, 128

MPT Manual Penetration Testing. 44

NATO North Atlantic Treaty Organization. 3, 26

NDMA The Norwegian Defence Materiel Agency. iii, vii, 1–3, 7, 25, 26, 28, 33, 36, 40, 45, 75, 77, 79–83, 85–89, 91, 92

NIST National Institute of Standards and Technology. 10–12, 18, 20–22, 83

NTNU Norwegian University of Science and Technology. vii, 3, 47, 71, 159, 188

NVD National Vulnerability Database. 16, 20

OS Operating System. 22, 23, 26, 29, 39

OWASP Open Web Application Security Project. 13, 19–21, 31, 34, 40, 41

PC Personal computer. 47

PHP Hypertext Preprocessor. 36, 38, 48, 86

POM Project Object Model. 73, 76

PR Purchase Requisition. 31

RASP Run-time Application Security Protection. 15, 29, 33, 35, 48, 58, 75, 82

REST Representational state transfer. 48, 53, 71, 74

SaaS Software as a Service. 39

SAST Static Application Security Testing. iii, v, 13–15, 29, 33, 34, 36, 38, 41, 42, 45, 58, 67, 71, 74, 75, 77, 81, 82, 84, 86, 107

SBOM Software Bill-of-Materials. 29

SCA Software Composition Analysis. iii, v, 16, 29, 33, 34, 36, 37, 39, 41–45, 48, 58, 61, 62, 71, 73–75, 77, 81, 82, 84, 86, 123–125, 128, 146

SCAP Security Content Automation Protocol. 20

SDLC Software Development Life Cycle. 2, 12–17, 34–38, 42, 43, 45, 74, 75, 79, 82, 91, 92, 150, 152

SP Special Publication. 10–12, 18, 20, 22, 83

SQL Structured Query Language. 14

URL Uniform Resource Locator. xxi, 27, 32, 35, 48, 64, 128–132

VS Code Visual Studio Code. 51, 136, 137, 139, 141, 142

XML Extensible Markup Language. 43

XSS cross-site scripting. 14, 73

ZAP Zed Attack Proxy. 31, 40

Chapter 1

Introduction

1.1 Background

The Ministry of Defense has four underlying agencies [1]. These are the Norwegian Armed Forces, the Norwegian Defence Research Establishment, the Norwegian Defense Estates Agency, and the The Norwegian Defence Materiel Agency (NDMA). The Norwegian Armed Forces is the military part of the Norwegian government [2], and is led by the Chief of Defense and the Defense Staff. NDMA equip the Norwegian Armed Forces with material [3] in order to contribute to a higher operational strength. This includes everything from the soldiers' equipment to different vehicles, aircrafts and seagoing vessels, weapon and sensory systems, and Information Technology (IT)-systems. NDMA is responsible for procuring, managing and phasing out this material, and they have the responsibility of ownership management for the duration of the service life of the equipment. They also have professional authority of all material in the Norwegian defence industry.

Through NDMA, the Norwegian Armed Forces get a lot of different software from various vendors [4]. It is important that the software they use is as secure as possible. This is especially important when dealing with data with classified status.

1.2 Problem Area

NDMA need to test the security of both produced and procured software before they go into production. The solution today is a manual review of code, and a review of all the libraries that the code is dependent on. This is a cumbersome and time-consuming process, and vulnerabilities are easily missed. Grading the severity of the vulnerabilities appropriately can also be difficult without a frame of reference.

NDMA need a better method for performing security and vulnerability checks of software from both internal and external developers. Their desire is an option that is as automated as possible, and that allows them to find vulnerabilities earlier in the Software Development Life Cycle (SDLC).

1.3 Scope and Limitations

The thesis will be limited to software security. One problem that is emphasized by NDMA, is that there are more libraries than necessary in the source code from the software they receive, and it is difficult to get an overview of all the libraries. NDMA want to check if these contain vulnerabilities, or if they are not in use and only pose an unnecessary threat. Thus, the main focus will be to automate these checks as much as possible. If it is not possible to recommend an alternative that is fully automated, the goal will be a combination of an automated and a manual solution of high quality. Most important is that the end result makes it easier to review software than it has been before.

1.4 Target Group

There are several target groups for the thesis. One of the target groups are the leaders of NDMA, because they are ultimately the people who decide if a solution is worth investing time and money into. Another target group is the developers of NDMA and the Norwegian Armed Forces. They will potentially be using the recommended tools to make their software more secure. A third target group is the employees responsible for reviewing external software for NDMA.

1.5 Goals

Effect goals:

- **M1** Increased awareness of security during software development.
- **M2** NDMA are happy with the methods that are proposed and use these over time.
- **M3** Reduce the time it takes to check for vulnerabilities in the software.
- **M4** Reduce the number of people that need to check for vulnerabilities in the software.

Project goals:

M5 Find tools and methods to make the security checks for the NDMA more effective, and automate this if possible.
M6 NDMA start using the tools and methods the group has come up with.
M7 Create a systematic overview of potential tools that can be used for software control.

1.6 Framework

Procurement for the Norwegian Armed Forces is subject to the Public Procurement Act [5], with some exceptions. It means that choosing suppliers is often competition driven, sometimes making it impossible to rely on one supplier only. This is part of the reason why they have several different suppliers. Having different suppliers also means more differences in procured software, thus executed security checks may vary. Some of the procured software includes source code, while some does not. Source code analysis is consequently not always possible. Additionally the Norwegian Armed Forces have different levels of access to their data. A cloud solution might be possible for data with unclassified access, but not for higher levels. Such variations need to be taken into account when choosing a solution. Lastly, it is important to consider where the potential tools have located their headquarters, due to NDMA only accepting tools located in North Atlantic Treaty Organization (NATO) or European Union (EU) countries.

1.7 The Group's Academic Background

The group is on the third and last year of a Bachelor in Digital Infrastructure and Cyber Security at the Norwegian University of Science and Technology (NTNU). Through this study, the group has acquired knowledge from several subjects that are relevant for the thesis.

DCSG1001 - Infrastruktur: Grunnleggende ferdigheter and *DCSG1005 - Infrastruktur: sikre grunntjenester* were relevant for testing the chosen tools, because they covered use of command-line.

DCSG2005 - Risikostyring was relevant for understanding why it is important to test software, as well as understanding the results from tests. It was also relevant because the group got the experience of writing an academic text for a client.

IIKG2001 - Software Security was relevant for understanding the importance of secure code and defense in depth.

1.7.1 Why This Task Was Chosen

The group chose this task because of an interest in security itself, and the future of security. The group believes that security needs to be automated to some degree, in order to make businesses take this more into consideration. This is especially true for software security, where few see the value of including it when the software being developed. If the bar is low, the thesis group believes that more businesses would be willing to implement security during the earlier phases of software development. They would also be able to protect themselves from insecure software from external developers. This can be a strenuous process if done manually, and is therefore often neglected. The thesis group believes that it would be better to use an automated security testing tool to test the software, compared to not testing it at all.

This task let the thesis group be a part of the process of finding and exploring different software security testing tools. It is also an opportunity to see how far the technology has gotten, and to understand it better.

1.8 Methodology

1.8.1 Software Engineering Method

The project period is divided into four phases that generally follow a waterfall model, with elements from the agile method in parts of the process, as displayed in figure 1.1. The first phase has a focus on all the relevant theory for the thesis, and it is in this phase the literature study will be done, together with finding and choosing different tools. After this is done, the focus is on doing the necessary research to perform the tests on the chosen tools. During this phase it is also important to figure out which type of test environment is needed. The next phase is focused on performing the tests, and studying how the tools work with the test data. In the last phase, the focus is on evaluating the results from the tests.

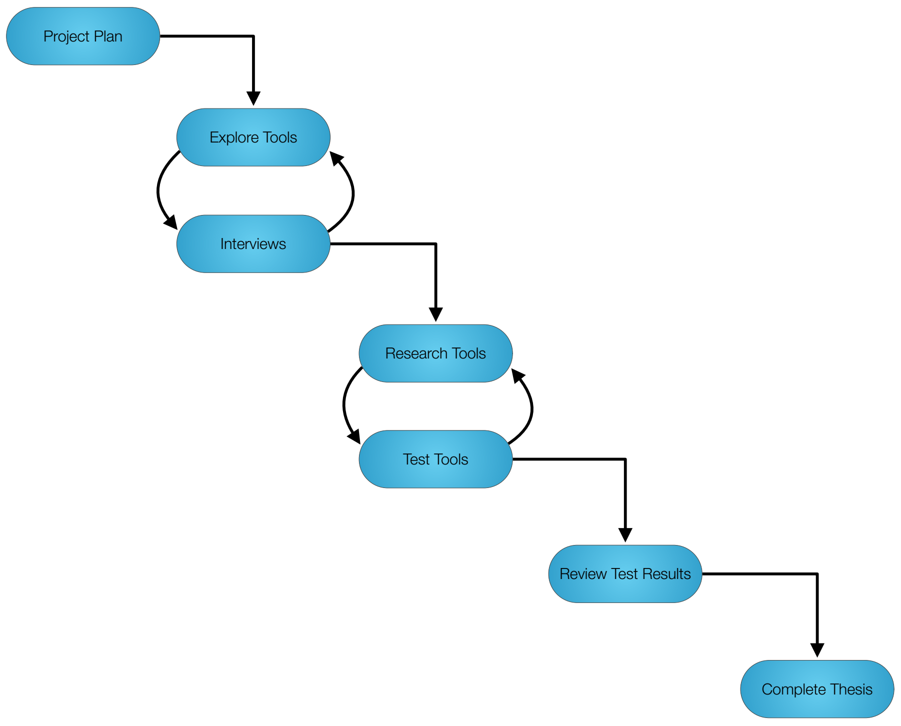


Figure 1.1: The software development model that is followed during the thesis

1.8.2 Research Methods

1.8.2.1 Literature Study

The literature study is done by looking through previous textbooks, and by researching the web for relevant information and theory that can be used in the thesis, and later to perform the tests. This will include reading documentation and other materials. One goal for the literature study is to build up the theory chapter with relevant information, as well as building up the group's knowledge on the subject. This is necessary to conduct a thesis of high quality. Another goal is to find tools and critical information about them. The tools are then chosen based on the critical information, before further research is done to get a proper overview.

1.8.2.2 Interviews

The interviews are conducted by following different strategies and guidelines from the Harvard Sociology Department, *Strategies for Qualitative Interviews* [6]. Before the interviews, an interview guide with questions will be made, together with information about the purpose of them.

1.8.2.3 Experiment

In order to perform the experiments on the selected tools, the test environment has to be made, and some test data needs to be available. When this is ready, the tests will be performed systematically. One tool will be fully tested before another tool gets tested. When the tests are done, the results will be compared based on the amount of vulnerabilities found. There will also be made a list of advantages and disadvantages for every tool, to evaluate other factors as well.

1.9 Thesis Structure

Some things to notice when reading through the thesis;

The thesis uses clickable links to chapters, sections, acronyms, figures, tables and sources.

All figures will come after they have been explained in the text, together with a reference to where they can be seen.

The language used in this thesis is English, except some attachments that are in Norwegian. This is because most of the communication through interviews, e-mails and meetings are done in Norwegian. Thus, it would be unnatural to have this in English. The project plan is also written in Norwegian, because the original plan was to write the thesis in Norwegian. After conversations with the supervisor (appendix I.1) and the employer (appendix I.2.1), the language was reconsidered and English was chosen, due to the industry using this language. Using Norwegian could lead to difficulties translating words correctly, and misunderstandings after translations could occur.

Titles are defined as chapters, sections, subsections, subsubsections and subsubsubsections; where chapter is the top level and subsubsubsection is the lowest. All titles from chapter to subsubsection are shown in the table of content.

1.9.1 Chapters

The first chapter, Introduction, is the introduction to the bachelor thesis. This section starts with the background of the project, with some general information about NDMA. Then the issue, scope and limitations are introduced, which explain the issue NDMA want to solve, as well as the limitations to solving the issue. Goals are covered as effect and project goals, before framework and the groups background is presented. The chapter ends with the methodology, as well as the thesis structure.

The second chapter, Theory, covers theory that is relevant to the rest of the thesis and it should be understood before the rest can be read.

The third chapter, Tools for Security Testing, consists of tools that have been examined as possible solutions for NDMA. It contains all the relevant information to deciding which tools to take a further look at.

The fourth chapter, Analysis of Selected Tools, consists of the chosen tools for NDMA. It covers why they were chosen as a possible solution, as well as information about what the tools are and what they do.

The fifth chapter, Experiment, covers the tests of the chosen tools. It includes an explanation of the test environment and test data, as well as explaining how the tests were performed. The chapter concludes with the results.

The sixth chapter, Discussion, covers a presentation of the group's process and findings. It also covers unpredicted limitations during the thesis, changes that were made during the project, and a discussion with critique of the thesis.

The seventh chapter, Conclusion, covers reflections and evaluations from the work, theoretically possible further work, and a conclusion to the thesis.

Attachments will have all attachments related to the thesis. Some attachments are: more detailed information about the tests, minutes of meetings, and the cooperation agreement.

Chapter 2

Theory

2.1 Introduction

The theory chapter consists of an explanation of different types of software tests, different methods used to conduct software security tests, as well as some additional information. The additional information is included because it is important to understand before reading the thesis.

2.2 Testing Functionality Compared to Security

To understand the goal of this thesis it is important to know the different types of software testing and the function of those tests. Functional testing, according to *Official (ISC) Guide to the CSSLP CK* [7], is a type of testing where the goal primarily is to test if the software functions as expected. Within this type of test, there are multiple smaller tests. These smaller tests need to be conducted successfully, in order to know that the software's functions work accordingly. One of these tests is unit testing.

Unit testing is a form of functional testing that takes place during the implementation phase. This type of testing is done by the developer, rather than a tester. Smaller sections, units, of the software are isolated and tested. The goal of this test is to see if there are any compilation errors and to validate functional logic. Unit testing can also help the developer discover vulnerabilities in the code, such as hard-coding variables or lack of input validation. Integration testing is used for testing the sum of all parts of the unit test. This type of test is the natural next step after unit testing. It helps uncover issues that might occur when the units are put together.

Logic testing is a form of testing used to make the processing logic of the software accurate. When copying code from different sources and combining those lines of code, some issues might occur. To avoid those issues in the software, it is important to validate that the implemented code is correct for the functionality and logic. This is especially important when the software allows for user input.

None of these tests are security tests, and there are more tests that can be done to a software without testing security. Performance testing, where the goal is to look for bottlenecks in the system, might make some companies rethink security measures that slows down the system. The difference between these tests and security testing, is that security tests try to break the software on purpose. The tester will try to take on a hostile mindset and work their way around the protection of the software. Security testing and testing security functionality are also different.

When testing security functionalities, the goal is to validate whether or not the protection mechanisms, such as auditing and authentication, work as intended. On the other hand, security testing will try to attack the software and validate the software's overall ability to protect itself from the attacks. Some examples of security testing methods are white box testing, black box testing and gray box testing, covered in section 2.3.

2.3 Box Testing

Box Testing is an approach to software testing, where testing is divided into white box testing, black box testing and gray box testing.

2.3.1 White Box Testing

According to NIST SP 800-137 [8], white box testing is "a test methodology that assumes explicit and substantial knowledge of the internal structure and implementation detail of the assessment object". It is also referred to as "glass box testing" [9] or "clear box testing" due to its transparency.

The upside to white box testing [10] is that it will cover the entire code, as well as finding structural problems, hidden errors and problems with specific components. It is also possible to automate, and allows for the code to continually improve. The downside to white box testing is that even though it allows for automation, it still requires a lot of effort before it can be automatic. Another downside is that it cannot test from a user's perspective, in order to find functional problems.

2.3.2 Black Box Testing

Black box testing is, according to NIST SP 800-192 [9], "a method of software testing that examines the functionality of an application without peering into its internal structures or workings". This means that it is not testing the software's internal structure [11], but rather how it reacts to interactions from an outside source.

The tester knows what the system should do, but not how it is done. This will emulate a user using the system. A user will not care about how the software is structured, only how it works. This is known as zero knowledge assessment [7]. Black box testing will test if the software works as it should for the user, and will test if there are any vulnerabilities that could be exploited.

The upsides to black box testing are that it has a low false positive rate, and that it can be done by someone with little to no technical knowledge. The downsides are that black box testing is difficult to automate, as well as being difficult to test all possible user-paths. Because of this, it is difficult to calculate test coverage.

2.3.3 Gray Box Testing

According to NIST SP 800-53A Rev. 5 [12], gray box testing is "a test methodology that assumes some knowledge of the internal structure and implementation detail of the assessment object". The goal of gray box testing [13] is to test for defects based on the software's internal structure.

With gray box testing, the tester only has insight into parts of the system. It can be used as a form of penetration test, covered in 2.9, where the tester knows the internal components, but not how they work together. By using that knowledge, it will try to act like a potential attacker. With this type of testing it is important to keep testers and developers apart, to make sure that the tester will not have any extra knowledge about the system.

The upside to gray box testing is that it will provide benefits from both black and white box testing, as well as having clear testing goals which makes it easier for both developers and testers. One downside is that it can be difficult to create the test case, which is the type of tests that will be conducted on the system. Finding the root cause of disturbances in the system after tests are done might also become difficult. This is because of the limited knowledge about the internal system, which causes gray box testing to loose some white box testing benefits.

2.4 Application Security Assessment

Application security assessment is the process of identifying, analyzing and planning mitigation of threats and vulnerabilities in an application [14]. There are many options available for automatic testing and analysis, that make the process of application security assessment more efficient.

Some things to keep in mind when choosing automatic tools are the prevalence of false positives and negatives, and how early in the Software Development Life Cycle (SDLC) they can be implemented. The SDLC is a process with a set of steps used in software development [15]. Having security testing early in the SDLC is often referred to as shift-left testing [16].

2.4.1 Shift-Left

Traditionally, testing comes late in the SDLC [17]. In more recent years, it has become more common to implement security earlier. The reason it is called shift-left, is that testing is moved to the left in the SDLC.

There are significant benefits to having security testing early in the SDLC, as it is less costly, as well as resource and schedule efficient [7].

2.4.2 False Positives and Negatives

A false positive is, according to National Institute of Standards and Technology (NIST) Special Publication (SP) 800-115 [18], "an alert that incorrectly indicates that a vulnerability is present". This often happens with automated security tests [19]. When going through the alerts, it is possible that most of them are false. When this occurs, it is easy to assume that all of them are false positives without going through them all, and then ignore the rest. By doing so, there is a good chance that the real vulnerabilities and threats to the system will go undetected.

On the opposite side, a false negative is, according to NIST SP 800-83 Rev. 1 [20], "an instance in which a security tool intended to detect a particular threat fails to do so". The reason why they are not detected can be because they are dormant [21], highly sophisticated, or the security infrastructure does not have the technological ability to detect them. This can cause a false feeling of security.

2.5 Application Security Testing and Analysis

When using a combination of application security testing tools (figure 2.1), it is possible to reduce the overall security risk of the software [22]. This causes a form of defense in depth, where the application is tested from multiple angles. When it is not possible to include all of them, some of the forms for testing should still be included if possible.

	Low False Positives	Exploitability	Code Visibility	Remediation Advice	Broad Platform Support
SAST			✓	✓	✓
DAST	✓	✓			✓
IAST	✓	✓	✓	✓	
SCA		✓	✓	✓	✓
RASP	✓	✓	✓	✓	

Figure 2.1: Application Security Testing, inspired by [22]

2.5.1 Static Application Security Testing (SAST)

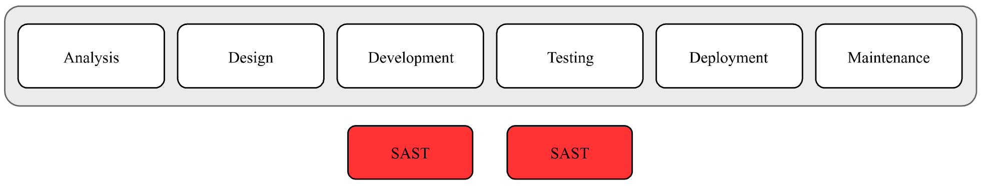


Figure 2.2: SAST placed in the SDLC

Static Application Security Testing (SAST) is a type of "white box testing", and is an integral part of shift-left methodology [23]. It can help developers find security vulnerabilities in the source code of the application earlier in the SDLC [24]. It also ensures compliance with both standards and guidelines for coding, without executing the underlying code.

SAST tools give developers feedback in real-time as they are writing the code [25]. This assists them in fixing potential issues before the next stage in the SDLC. The exact locations of the issues also get pointed out in the code. Potential issues can be vulnerabilities listed in the OWASP top 10 [26], but also other vulnerabilities. This might vary based on the tool used to conduct the test.

SAST allows developers to scan a project at the code level, which makes it easier to make the recommended changes. When the flaws are found earlier in the SDLC, it helps reduce the cost and the repercussions that result from addressing problems at the end of the process. It works well when it comes to finding errors in code, but it is not very efficient in finding data flow flaws. SAST tools are also known for the larger amount of false positives and negatives.

2.5.2 Dynamic Application Security Testing (DAST)

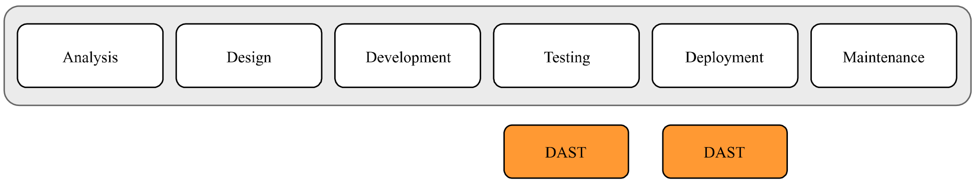


Figure 2.3: DAST placed in the SDLC

Dynamic Application Security Testing (DAST) is a type of "black box testing" [27]. It is used to find security vulnerabilities in a running application, often web applications, without having access to the source code. To do this, it employs fault injection techniques on the application [24], for example by feeding malicious data to the software, in order to identify common security vulnerabilities like SQL injection and cross-site scripting (XSS).

When using DAST in the SDLC, vulnerabilities can be detected in the application before it is deployed to the public [28]. This can be vulnerabilities that SAST is not capable of finding, because it is linked to runtime, such as authentication flaws. DAST is also better at avoiding false positives and negatives compared to SAST, as well as being less expensive and less complicated to handle.

DAST will simulate attacks on the application, detecting some vulnerabilities. The issue is that DAST does not have any insight about the application, and can thus not simulate attacks from an agent with some inside knowledge.

2.5.3 Interactive Application Security Testing (IAST)

Interactive Application Security Testing (IAST) is seen as a type of "gray box testing" [29], and is an application security testing method that tests the application for possible vulnerabilities in execution, while the application is in use. This type of testing happens during the testing and deployment phases of the SDLC. Some IAST tools come with Integrated Development Environment (IDE) integrations, which makes it possible to run the security analysis while developing the application.

IAST has an "agent-like" approach [28], which means that agents and sensors are run to continually analyze the application workings during automated testing, manual testing, or a combination of the two.

The difference between IAST compared to SAST and DAST [30], is that IAST is located inside the application. IAST can be implemented in different ways, one being a sensor placed in the back-end of the application. This sensor will then be triggered by another test being conducted, or other forms of interactions with the application.

When choosing to use IAST, potential issues or vulnerabilities can be caught earlier, which can lead to costs and delays being minimized. This is due to the application of a shift-left approach. Because of the range of information IAST has access to, it can accurately identify the source of vulnerabilities. IAST can also easily be integrated into the Continuous Integration/Continuous Delivery (CI/CD) pipeline, which is a series of steps that need to be performed in order to deliver a new version of software [31].

On the other hand, IAST tools can slow down the operation of the application, because the agents essentially serve as added instrumentation, which can lead to the code not performing as well. Furthermore, since it is a relatively new technology, there might be some undiscovered issues.

2.5.4 Run-time Application Security Protection (RASP)

Run-time Application Security Protection (RASP) is integrated into applications to analyze the traffic coming in and out of the application [32]. It will also analyze the user patterns to protect from security attacks. Users might have suspicious behavior that this method can detect and protect against. This is used in the maintenance phase of the SDLC, and is not a testing tool.

RASP is deployed to an application or web server. Here it is located next to the main application while the application is running to monitor and analyze the traffic behavior. There is no human intervention required to do this. When an issue is found, RASP will send out alerts, and the ones trying to gain access to the application will immediately be blocked.

When RASP is deployed, it will not wait and try to rely on the specific signatures of known vulnerabilities. Instead, it will secure the whole application against different attacks.

2.5.5 Software Composition Analysis (SCA)

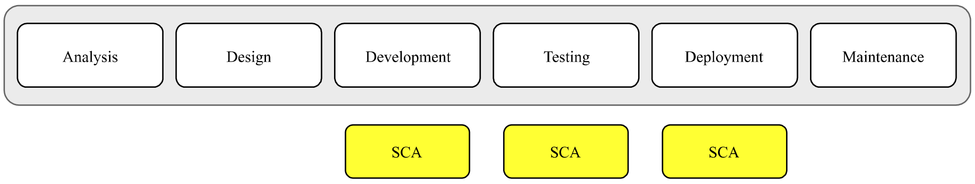


Figure 2.4: SCA placed in the SDLC

Software Composition Analysis (SCA) is a tool that performs automated scans on an application's code base to provide visibility into the usage of open source software [22]. This includes identifying all the open source components, their license compliance data, and the security vulnerabilities. In addition to this, SCA prioritizes vulnerabilities in the open source and provide insights and auto remediation to resolve security threats.

SCA tools inspect package managers and manifest files, source code, binary files and container images [33]. The open source that is identified is compiled into a Bill of Materials (BOM). This is then compared against a variety of different databases, including the National Vulnerability Database (NVD), which is a U.S. Government repository of vulnerabilities. These databases contain information about known and common vulnerabilities.

To discover licenses associated with the code and analyze overall code quality, SCA tools can compare BOMs against other databases. By doing this, security teams are able to identify critical security and legal vulnerabilities.

2.6 The Importance of Software Security Testing

It can be difficult to see why security testing is necessary. According to the *Official (ISC) Guide to the CSSLP CK* [7], it is usually impossible to know which security breaches were avoided by doing security testing and fixing the vulnerabilities found. This makes many businesses reluctant to dedicate resources to securing software. Despite this, it is important to talk about the consequences of not security testing software properly.

The earlier in the SDLC software security is considered, the less expensive it becomes. The longer it is ignored, the more expensive it becomes to deal with the consequences. The average data breach will cost 3.8 million dollars to fix, while the vulnerability itself would cost less than 500 dollars, on average, to fix during the design phase [34]. This is because a lot of resources are required to fix something that has already gone wrong. A software can contain security holes, which can be exploited by an agent in order to collect sensitive data. If this happens, data will no longer be private, and it is almost impossible to make this information completely private again. Implementing security testing tools during the development and testing process can reduce this possibility considerably.

If software is made in-house, it should be tested during development. It is important to test both the libraries being used, and the code written by the developers. A reason why the libraries need to be tested, is that they may be open source. If that is the case, malicious agents have the opportunity to look through the libraries and find vulnerabilities to exploit. Another reason is that it is difficult for humans to have control over every way it is possible to exploit code, and because human errors happen frequently.

A business will most likely buy software from an external source as well, and might not have the source code available. When this situation occurs, it is still important to take security into account. It should be made clear if the external developer used some sort of security testing, and to what degree security was taken into account. It is also important to remember that just because security functions exist in the software, it is not necessarily implemented correctly to be used in the specific environment to the business. Hence, it is important to security test.

2.7 The Odds of Finding Vulnerabilities

To understand results from software security testing, it is important to know what the chances of finding vulnerabilities are. An example on how probable it is to find vulnerabilities, is the study ‘Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages’ [35]. The study found a total of 339 new malicious packages after looking through over one million packages. It is plausible that more vulnerabilities exist in these packages, but they were left unnoticed. The result from the study indicates that it is unlikely to find copious amounts of vulnerabilities, especially when the tests are limited.

2.8 Test Environment

Testing potentially vulnerable or malicious software should be done in a safe environment, in order to avoid crashing or compromising a production environment. It can also prevent the release of vulnerable data [36]. For this reason, it is smart to create a test environment to run the tests, before the software goes into production.

A test environment should provide accurate feedback about the quality and the behaviour of the application that is being tested, and provide the necessary setup to perform the tests. Further, it provides a dedicated environment enabling the possibility to isolate the code, and verify the behaviour of the application. This ensures that the output of the tests is not influenced by other activities. A test environment can also act as an exact copy of the production environment, which is crucial for the test results to be accurate.

2.9 Penetration Testing

Scanning tools in the CI/CD pipeline are valuable for identifying issues early. To ensure defense in depth, it is important to have different methods for security testing. Manual penetration testing [7] is thus helpful to find vulnerabilities that automated assessments cannot. Penetration testing is, according to NIST SP 800-95 [37], "a method of testing where testers target individual binary components or the application as a whole to determine whether intra or intercomponent vulnerabilities can be exploited to compromise the application, its data, or its environment resources".

Manual penetration testing is a form of testing where the tester tries to act like a hacker on the system. The test should be performed by an expert in the field. The expert will write a report on everything that is done during the test, and the vulnerabilities that might have been discovered. This is done with the permission of the owner of the system, and is used in order to look for holes in the security. If a penetration tester were to find a way to hack the system, the owner will be informed. It is then up to the owner to evaluate what to do with this information.

It is preferred to perform manual tests when possible [38], especially on more critical systems. If manual tests are not possible to conduct, automated penetration tools can be used.

2.10 Fuzzing

According to *Official (ISC) Guide to the CSSLP CK* [7], fuzzing, or fault injection testing, is a form of testing where data is injected into the software, and the reaction of the software is being considered. The data would be a type of random or semi-random data that should not be injected, called fuzz data. The goal with this type of test is to find out if the input validation is effective.

Fuzzing can be used in both black and white box testing. It is used a lot in black box testing, where the fuzz data is sent to the software without any insight of internal workings. With this type of fuzzing, there is no guarantee that all code paths were covered, as opposed to with white box fuzzing. White box fuzzing lets the internal workings be known, which then guarantees better coverage. White box fuzzing would be the best option in this case, but if the internal workings are unknown, black box fuzzing should be performed.

Generation-Based Fuzzing, also called "Smart fuzzing", is a type of fuzzing where the data format or protocol is known beforehand. The fuzz data is then made by generating data based on the known format. This can be a time consuming process, due to testers having to import known data formats before generating variations. The main downside to this type of fuzzing is the lack of coverage for new or proprietary protocols.

Mutation-Based Fuzzing, also called "Dumb fuzzing", is a type of fuzzing where the data format is unknown. The fuzz data is then made by mutating existing data samples, blindly and randomly. This type of fuzzing should only be done in a simulated environment, due to its destructive potential. It can lead to Denial of Service, destruction, and complete disruption of the software's operations.

2.11 OWASP Top 10

Open Web Application Security Project (OWASP) Top 10 [39] is a document that contains ranking of and remediation about the ten most critical web application security risks. It is based on an agreement among developers around the world. All the risks are ranked and based on the frequency of discovered security defects, the severeness of the vulnerabilities, and information about their potential impact. The purpose of the document is to offer insight into the most prevalent security risks that might happen. With this information, developers and web applications security professionals can incorporate the information from the document into their security practices, and thereby minimize the presence of these risks in their applications. The most recent changes were done in 2021 (figure 2.5), which indicates that the table is updated regularly.

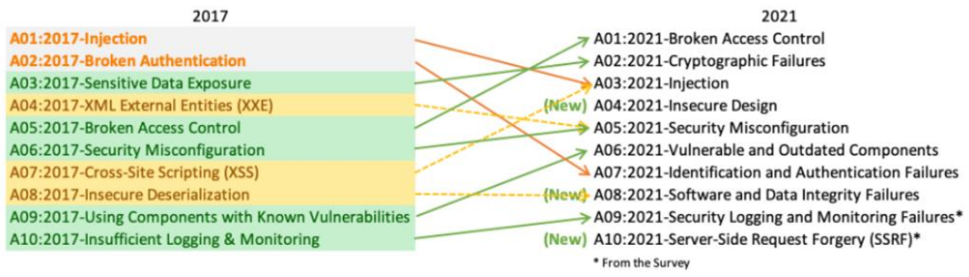

2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

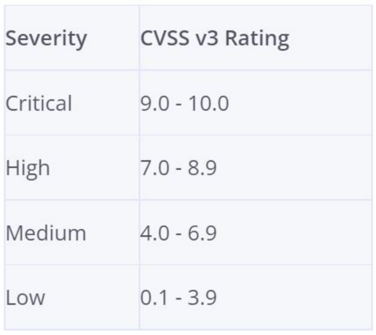
Figure 2.5: OWASP Top 10 in 2017 vs. OWASP Top 10 in 2021 [39]

2.12 Vulnerability Estimators

Vulnerabilities can be classified in a standardized way, by using CVSS, CVE, CWE, and OWASP Risk rating Methodology.

2.12.1 Common Vulnerability Scoring System (CVSS)

Common Vulnerability Scoring System (CVSS) is, according to NIST SP 800-128 [40], "an Security Content Automation Protocol (SCAP) specification for communicating the characteristics of vulnerabilities and measuring their relative severity". The CVSS offers a way to capture vulnerability characteristics, and generate a numerical score that reflects the severity (figure 2.6). This severity score can be translated into qualitative representation, such as low, medium, high and critical. The current version is CVSS v3.1 [41].



Severity	CVSS v3 Rating
Critical	9.0 - 10.0
High	7.0 - 8.9
Medium	4.0 - 6.9
Low	0.1 - 3.9

Figure 2.6: The ranking of vulnerabilities in version 3.1 [42]

2.12.2 Common Vulnerabilities and Exposures (CVE)

According to NIST SP 800-126 Rev. 3 [43], Common Vulnerabilities and Exposures (CVE) is "a dictionary of common names for publicly known information system vulnerabilities". It has to do with a specific instance within a system or a product [44], and not the underlying flaw. Each vulnerability has an identification number and they are published in the NVD [45].

2.12.3 Common Weakness Enumeration (CWE)

Common Weakness Enumeration (CWE) is, according to NISTIR 8011 Vol. 4 [46] "a taxonomy for identifying the common sources of software flaws". It is a community-developed list of different types of weaknesses in software and hardware [47]. It serves as a measurement for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts. CWE has to do with the specific vulnerability [44], and not the instance within a product or a system like with CVE.

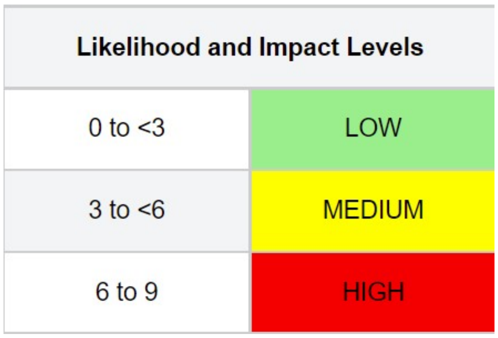
2.12.4 OWASP Risk Rating Methodology

OWASP Risk Rating Methodology [48] is a rating system where $risk = impact * likelihood$. It also gives guidelines on how to estimate the impact and likelihood levels.

The factors to estimate likelihood are divided into groups that are related to the threat agent involved, where the goal is to estimate the likelihood of the threat agent to perform an attack. The four factors are skill level, motive, opportunity, and size. Each factor is given a score between one and nine.

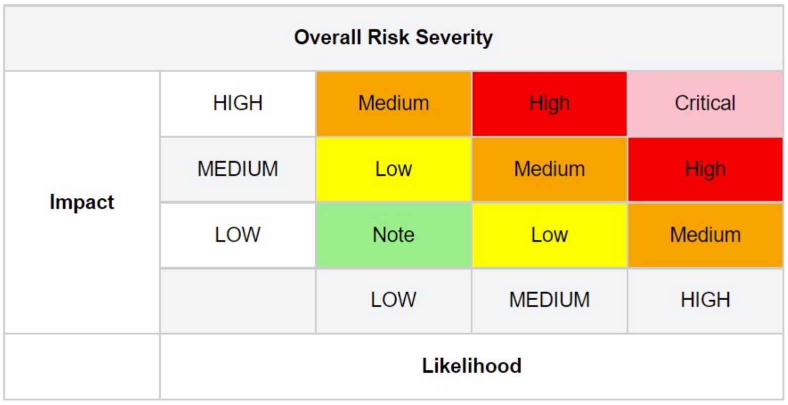
The factors to evaluate impact can be divided into technical and business impact factors. The technical impact factors can be further divided into confidentiality, integrity, availability, and accountability. Each of these factors are also given a score between one and nine to estimate the impact on the system if a vulnerability is exploited. The same applies to the business impact factors, except that the factors are financial damage, reputation damage, non-compliance, and privacy violation.

To find the severity of the risk, the likelihood estimate and the impact estimate factors are put together to get an overall severity of the risk, which can be low, medium, or high (figure 2.7). They can then be combined to get a final severity risk (figure 2.8).



Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

Figure 2.7: The likelihood and impact levels [48]



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Figure 2.8: The risk severity [48]

2.13 Containers

According to NIST SP 1800-190 [49], a container is "a method for packaging and securely running an application within an application virtualization environment". It is used to move applications from one environment to another, without having to make specific changes to the application in order for it work in the new environment. It is inspired by the containers used in the shipping industry [50], with the standard containers that make it easier to ship objects. With digital containers, a developer can work on the Operating System (OS) they want to, and still have the application work in the production environment.

A container has multiple security benefits. It isolates the application, and is constantly replaced with new updated versions, which makes fixing software vulnerabilities quicker [51]. Containers are possible to replace within minutes, while other methods can take weeks. This makes containers great for fixing vulnerabilities, but there are also vulnerabilities that come with using containers.

A common threat to containerized environments [49] are application-level vulnerabilities which are found in the software within containers. The image might be built based on a common web application, which might have vulnerabilities. If that is the case, the attacker can subvert the application within the container to try to map the system, elevate privileges within the compromised container, or abuse the container in order to attack other systems.

This is one of many threats that might happen when using containers. This does not mean that containers should not be used, but it requires a lot of changes in an environment in order to make it safe. An example of changes that should be made in order to make containers safe, are to use container-specific host OSs. This means that instead of using a standard OS, the host uses a minimalistic OS made to run containers only. Another example would be to avoid running apps with different sensitivity levels together on the same host.

Even though a system is using containers, this does not mean that security checks should not be done. It is recommended that both image software vulnerabilities and configuration settings are taken into account. There should also be container-aware run-time defense tools in use. Traditional security solutions might not be able to protect containers properly, and it will be necessary to utilize a container-native security solution.

There are multiple container solutions today, with different advantages and disadvantages. One of the most popular solutions [52] being Docker, which provides products to help developers pack applications into containers. Docker Engine [53] is the software that hosts all the containers, while the Docker daemon manages all the containers by listening for requests from the Docker Engine. Docker supports multiple OSs, making it available for most people.

Chapter 3

Tools for Security Testing

3.1 Introduction

The tools for security testing chapter consists of descriptions of different tools for security testing. It explains the current solution and all the tools that were chosen for further investigation.

3.2 Today's Solution

Today, NDMA's developers use different techniques in order to conduct security checks. Some use tools, while others manually look through the code.

One of the tools used by NDMA is Trivy (figure H.24). Trivy is used as a security check for third party dependencies. Vulnerabilities that are ranked to be low or medium are accepted. They try to avoid those that are considered to be high. If it is difficult to avoid those, it is important to specify what the vulnerability is, why they have it, and what they need to do to get rid of it. The vulnerabilities that are considered critical are not allowed. If some of these are detected, they need to rewrite the code to avoid it. Another control mechanism used, is integrated in the IDE that is used to write the code. If the code that is written can lead to any vulnerabilities, they get a notification on this when they write it, or when they compile the code. This is used by some of the developers, but not all of them.

Several developers (appendix I.4.2) read manually through their code to look for vulnerabilities, and decide for themselves if something is an issue or not. Some developers only think about security before they start writing code, while others try to keep security in mind while coding. These developers do not feel like they have time to look through the code for vulnerabilities.

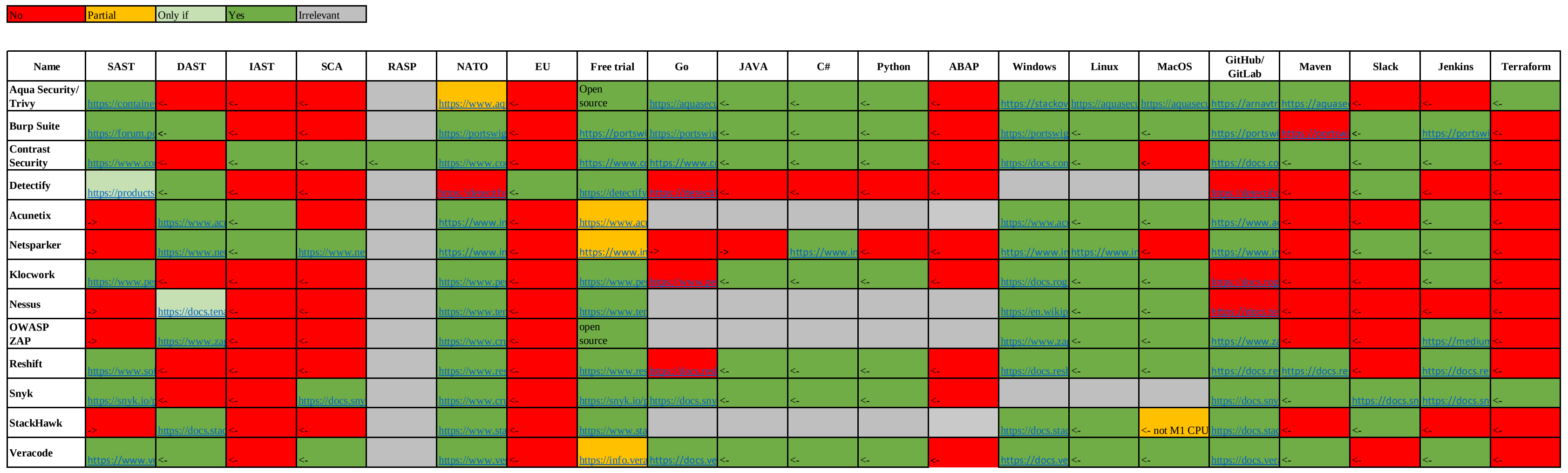
When not all developers use the same techniques, the quality of the code may vary depending on the tools that are used and the knowledge each of the developers have. Some of the developers may have more experience with implementing security, while others can lack an understanding and value of the topic. As a result, there can be an overall risk that they have vulnerabilities that they do not know of, and are more vulnerable to potential attacks.

3.3 Tools

The figure 3.1, and table 3.1-3.4 show a comparison of all the various tools that have been examined. It shows the ones NDMA use, as well as popular tools from NATO and EU countries. NDMA want tools based in NATO or EU for security reasons, which made others irrelevant. They were also chosen based on which tools are more popular and trusted by different companies around the world. Others, like for example Checkmarx were also looked into, but since it has headquarters in Israel and China [54], it was not chosen to examine further.

3.3.1 Comparison Table

The table displayed in figure 3.1 shows information about all the various tools that have been examined. The tools are compared based on what kind of security testing method they support, in order for NDMA to choose a tool that will fit their needs. The table also covers if they were founded in NATO or EU countries, and are thus accepted tools for NDMA to use. It includes whether they offer a free trial or not. This will show if it is possible for the thesis group to test the tools, before recommending them to NDMA. Lastly, the table shows supported programming languages, Operating Systems and integrations, in order to make sure that the tools fit NDMA's requirements.



🟥 No	🟧 Partial	▨ Only if	🟩 Yes	⬜ Irrelevant

Name	SAST	DAST	IAST	SCA	RASP	NATO	EU	Free trial	Go	JAVA	C#	Python	ABAP	Windows	Linux	MacOS	GitHub/ GitLab	Maven	Slack	Jenkins	Terraform
Aqua Security/ Trivy	🟩 https://containe	🟥 <-	🟥 <-	🟥 <-	⬜	🟧 https://www.aq	🟥 <-	🟩 Open source	🟩 https://aquasecu	🟩 <-	🟩 <-	🟩 <-	🟥 <-	🟩 https://stackov	🟩 https://aquasecu	🟩 https://aquasecu	🟩 https://arnavtr	🟩 https://aquasec	🟥 <-	🟥 <-	🟩 <-
Burp Suite	🟩 https://forum.po	🟩 <-	🟥 <-	🟥 <-	⬜	🟩 https://portswig	🟥 <-	🟩 https://portswi	🟩 https://portswig	🟩 <-	🟩 <-	🟩 <-	🟥 <-	🟩 https://portswig	🟩 <-	🟩 <-	🟩 https://portswi	🟥 https://portswi	🟩 <-	🟩 https://portswi	🟥 <-
Contrast Security	🟩 https://www.co	🟥 <-	🟩 <-	🟩 <-	🟩 <-	🟩 https://www.co	🟥 <-	🟩 https://www.c	🟩 https://www.c	🟩 <-	🟩 <-	🟩 <-	🟥 <-	🟩 https://docs.con	🟩 <-	🟥 <-	🟩 https://docs.co	🟩 <-	🟩 <-	🟩 <-	🟥 <-
Detectify	▨ https://products	🟩 <-	🟥 <-	🟥 <-	⬜	🟥 https://detectify	🟩 <-	🟩 https://detectify	🟥 https://detecti	🟥 <-	🟥 <-	🟥 <-	🟥 <-	⬜	⬜	⬜	🟥 https://detectify	🟥 <-	🟩 <-	🟥 <-	🟥 <-
Acunetix	🟥 ->	🟩 https://www.acu	🟩 <-	🟥	⬜	🟩 https://www.in	🟥 <-	🟧 https://www.acu	⬜	⬜	⬜	⬜	⬜	🟩 https://www.acu	🟩 <-	🟩 <-	🟩 https://www.ac	🟥 <-	🟥 <-	🟩 <-	🟥 <-
Netsparker	🟥 ->	🟩 https://www.ne	🟩 <-	🟩 https://www.ne	⬜	🟩 https://www.in	🟥 <-	🟧 https://www.in	🟥 ->	🟥 ->	🟩 https://www.in	🟥 <-	🟥 <-	🟩 https://www.in	🟩 https://www.in	🟥 <-	🟩 https://www.in	🟥 <-	🟩 <-	🟩 <-	🟥 <-
Klocwork	🟩 https://www.pe	🟥 <-	🟥 <-	🟥 <-	⬜	🟩 https://www.pe	🟥 <-	🟩 https://www.pe	🟥 https://www.pe	🟩 <-	🟩 <-	🟩 <-	🟥 <-	🟩 https://docs.rog	🟩 <-	🟩 <-	🟥 https://docs.rog	🟥 <-	🟥 <-	🟩 <-	🟥 <-
Nessus	🟥 ->	▨ https://docs.tena	🟥 <-	🟥 <-	⬜	🟩 https://www.ten	🟥 <-	🟩 https://www.ten	⬜	⬜	⬜	⬜	⬜	🟩 https://en.wikip	🟩 <-	🟩 <-	🟥 https://docs.te	🟥 <-	🟥 <-	🟥 <-	🟥 <-
OWASP ZAP	🟥 ->	🟩 https://www.zap	🟥 <-	🟥 <-	⬜	🟩 https://www.cru	🟥 <-	🟩 open source	⬜	⬜	⬜	⬜	⬜	🟩 https://www.zap	🟩 <-	🟩 <-	🟩 https://www.za	🟥 <-	🟥 <-	🟩 https://medium	🟥 <-
Reshift	🟩 https://www.so	🟥 <-	🟥 <-	🟥 <-	⬜	🟩 https://www.res	🟥 <-	🟩 https://www.res	🟥 https://docs.resh	🟩 <-	🟩 <-	🟩 <-	🟥 <-	🟩 https://docs.resh	🟩 <-	🟩 <-	🟩 https://docs.re	🟩 https://docs.re	🟥 <-	🟩 https://docs.re	🟥 <-
Snyk	🟩 https://snyk.io/p	🟥 <-	🟥 <-	🟩 https://docs.sny	⬜	🟩 https://www.cru	🟥 <-	🟩 https://snyk.io/p	🟩 https://docs.sny	🟩 <-	🟩 <-	🟩 <-	🟥 <-	⬜	⬜	⬜	🟩 https://docs.sny	🟩 <-	🟩 https://docs.sn	🟩 https://docs.sn	🟩 <-
StackHawk	🟥 ->	🟩 https://docs.stac	🟥 <-	🟥 <-	⬜	🟩 https://www.sta	🟥 <-	🟩 https://www.sta	⬜	⬜	⬜	⬜	⬜	🟩 https://docs.stac	🟩 <-	🟧 <- not M1 CPU	🟩 https://docs.stac	🟥 <-	🟩 <-	🟥 <-	🟥 <-
Veracode	🟩 https://www.ve	🟩 <-	🟥 <-	🟩 <-	⬜	🟩 https://www.ve	🟥 <-	🟧 https://info.vera	🟩 https://docs.ve	🟩 <-	🟩 <-	🟩 <-	🟥 <-	🟩 https://docs.ve	🟩 <-	🟩 <-	🟩 https://docs.vera	🟩 <-	🟥 <-	🟩 <-	🟥 <-

Figure 3.1: Information about all the examined tools
Note: The boxes contain clickable URLs to where the information has been found. Since several boxes have the same URL, there are arrows pointing to the URL where the information has been found.

3.3.2 Overview of Tools

The tables 3.1 - 3.4 contain information about what the tools are and what they do, where and when they were founded, as well as how much each of them cost.

What the tools are, is covered because it is important to know something about the tools and how they are different from each other. Where the tools are from is covered because some tools might be owned by countries NDMA might not approve of. When they were founded can play a role into how easy it is to find information about them when troubleshooting. New tools might have bugs, and a lack of documentation. Who uses the tools are covered because it shows businesses that believe in the product. Pricing is covered because it shows the budget needed in order to use the tools. The tools might have multiple different prices, which are dependent on how many features are included, and/or how many users that will use that specific tool.

Overall, this information together with figure 3.1 gives a short overview of the different tools.

Table 3.1: Aqua Security/Trivy, Burp Suite and Contrast Security
Note: "—" is placed in the table where the information could not be found

Tool	Aqua Security/Trivy	Burp Suite	Contrast Security
What	Trivy is a simple and comprehensive scanner for vulnerabilities in different container images, file systems and Git repositories. It also detects vulnerabilities in OS packages and other language-specific packages [55].	Burp Suite is a part of PortSwigger and is an application security testing software [56]. It consist of Burp Suite Enterprise Edition and Burp Suite Professional Edition, and they are used to secure the web and speed up software delivery [57].	Contrast Security specializes in application security. They add attack prevention and code analysis into the software [58].
Where	International HQ: Israel US HQ: Burlington [61]	UK [59]	USA [60]
When	2015 [62]	2003 [63]	2014 [60]
Who	—	Thrifty, P&G, Bendigo Bank [64], amazon.com, Walmart, Google [65]	BMW, Intuit, CreditSuisse, DocuSign, GreenSky, Unit4, CM [66]
Pricing	Free	**Burp Suite Enterprise Edition:** **Starter** - $6995 per year - 5 scanning agents **Grow** - $14480 per year - 20 scanning agents **Accellerate** - $29450 per year - 50+ scanning agents All of them are fully-featured with unlimited users and no application limits [67] **Burp Suite Professional Edition:** From $399 per year, depending on how many users [68]	**Community** - Free - Individual developers or small teams, SCA, IAST, RASP **Assess** - $2800 per developer per year - SCA, API Security Testing, SBOM, IAST **AST** - $3000 per developer per year - SCA, API Security Testing, SBOM, IAST, SAST, Serverless Security **Enterprise** - $3500 per developer per year - SCA, API Security Testing, IAST, SAST, Serverless Security, RASP [69]

Table 3.2: Detectify, Invicti Security and Klocwork
Note: "—" is placed in the table where the information could not be found

Tool	Detectify	Invicti Security	Klocwork
What	Detectify offer surface monitoring and application scanning. Surface monitoring makes it possible to monitor large amounts of Internet-facing assets and scan what is hosted. Application scanning checks web applications for vulnerabilities and sends alerts and guidance on how to fix them when they are detected [70].	Invicti security is a combination of Acunetix and Netsparker, which are two brands that prevent data breaches and other security incidents [71].	Klocwork is a static code analysis tool for different programming languages, such as C, C++ and Java, and it keeps development velocity high while enforcing continuous compliance for security and quality [72].
Where	Sweden [73]	USA [74] Acunetix : Malta [74] Netsparker : USA [76]	USA [75]
When	2013 [73]	2015 Acunetix: 2005 [74] Netsparker: 2006 [76]	1995 [77]
Who	Spotify, Grammarly, Trustly, King [70]	VISA, U.S. Air Force, Starbucks, NASA, Cisco, Ford, Verizon [74]	Raytheon, Spirent Communications, Motorola, Johns Hopkins, Elektrobit, AVM, ACI Worldwide, ACCESS, ABB [78]
Pricing	**Surface Monitoring** - €49 per month **Application Scanning** - €70 per month [79]	—	—

Table 3.3: Nessus, OWASP ZAP and Reshift *Note:* "—" *is placed in the table where the information could not be found*

Tool	Nessus	OWASP ZAP	Reshift
What	Nessus is a remote security scanning tool. It scans the system and raises an alert if there are any vulnerabilities that can be exploited by hackers [80].	Zed Attack Proxy (ZAP) is an open source web application scanner [81].	Reshift is a developer first code security tool that helps developers to find and fix vulnerabilities early in their IDE [82].
Where	USA [83]	USA [84]	Canada [85]
When	2002 [86]	2010 [87]	2019 [85]
Who	American Eagle Outfitters, World Wide Technology, Virtustream [88]	Mozilla, StackHawk, we45 and AppSecEngineer, Orange Business Services, Banzai Cloud, Motorola Solutions [89]	—
Pricing	1 year - NOK 36 103,55 - 2 years - NOK 70 395,88 3 years - NOK 102 876,99 Unlimited assessments, use anywhere, configuration assessment, live results, configurable reports, community support, advanced support, on-demand training available [91]	Free	**Free** - 1 user, Unlimited public repositories, 0 private repositories, 1 concurrent scan, PR workflow [90] **Pro** - $99 per month - 2 users, unlimited public repositories, unlimited private repositories, 2 concurrent scans, PR workflow, 2x scanning speed, support [90] **Team** - $299 per month - 10 users, unlimited public repositories, unlimited private repositories, 10 concurrent scans, PR workflow, 2x scanning speed, support [90] **Contact** - price based on needs - 100+ users, unlimited public repositories, unlimited private repositories, 50 concurrent scans, PR workflow, 2x scanning speed, support [90]

Table 3.4: Snyk, StackHawk and Veracode *Note: "—" is placed in the table where the information could not be found*

Tool	Snyk	StackHawk	Veracode
What	Snyk is a developer security platform. It is used to secure code, dependencies, containers, and Infrastructure as Code [92].	StackHawk is a dynamic application and API security testing tool [93]. It runs a automated scan on every pull request to find newly introduced vulnerabilities [94].	Veracode helps organizations to confidently develop software by reducing the risk of security breaches through analysis, developer enablement, and application security governance [95].
Where	USA [96]	USA [97]	Corporate HQ: USA EMEA HQ: UK [98]
When	2015 [96]	2019 [97]	2006 [99]
Who	Google, Revolut, AWS, Datadog [100]	Cloudbees, DataRobot, .planetly, Firebolt [101]	Garmin, Santander, Adidas, ebay [102]
Pricing	**Free** - Limited tests, unlimited developers, Snyk Open Source, Snyk Container, Snyk IaC, Snyk Code **Team** - $46 per month - Unlimited tests, 5-25 developers + all features in Free **Business** - $65 per month - 10-75 developers + all features in Team [103]	**Free** - Unlimited scans and environment, one application, Docker based application security scanner, CI/CD Automation, Historical scan data, cURL based reproduction criteria, Finding triage, Slack Integration, e-mail Based Support **Pro** - $35 per month - Minimum 5 developers, Unlimited applications, Jira Integration Applications Dashboard, e-mail and Slack Based Support + all features in Free **Enterprise** - $49 per month - Single Sign-On, Microsoft Teams, Generic, Webhooks Integration, Role based permissions, Activity History & Audit Log, Dedicated Slack Based Support, Premier Phone Support + all features in Pro [104]	—

Chapter 4

Analysis of Selected Tools

4.1 Introduction

The analysis of selected tools chapter consists of the tools that are chosen as possible solutions for NDMA. It contains reasons for why the various tools were chosen, how NDMA can use them, as well as how they work. It also contains information about how the tools handle uploaded data, and what kind of scoring system they use on vulnerabilities.

4.2 Contrast Security

4.2.1 Why Contrast Security Was Chosen

Contrast Security was chosen because it covers SAST, IAST, SCA and RASP. Since it uses both SAST and SCA, it works well on the internal software that NDMA develops. It checks the code for insecure libraries, as well as vulnerabilities while writing the code. Contrast Security also offers a variety of solutions with their platform, allowing NDMA to use the solution that best suits their needs. Each solution is well documented, and as a result, they can be set up according to preferred programming language. Contrast Security supports Java, .NET, .NET Core, Go, NodeJS, Python and Ruby [105], depending on which product is used. It also supports different integrations, such as Jenkins, Maven, GitHub, Slack, Microsoft Teams, IntelliJ, and more [106].

4.2.2 Product Overview

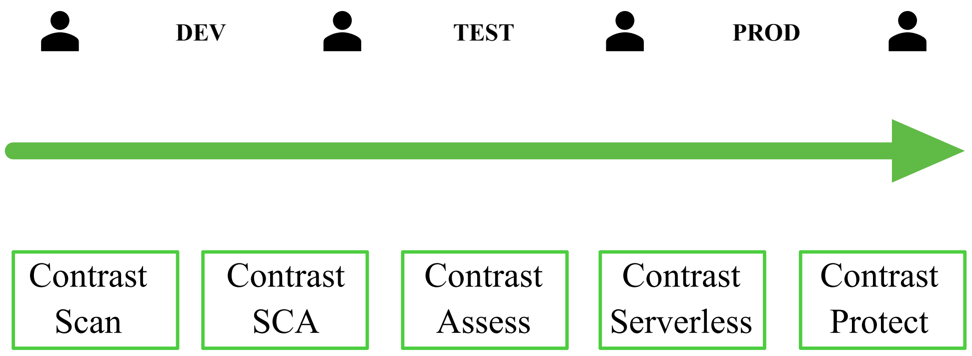


Figure 4.1: Contrast Security products placed in the SDLC (Inspired by [58])

Contrast Security [107] is a tool that instruments applications with sensors to detect vulnerabilities in the code and protect applications against potential security attacks. It has five platform solutions [108]; Contrast Scan, Contrast SCA, Contrast Assess, Contrast Serverless, and Contrast Protect (figure 4.1).

4.2.2.1 Contrast Scan

Contrast Scan [109] is a SAST tool that scans code for vulnerabilities. The scans can be initiated through Command Line Interface (CLI), build automation (Maven, Gradle, GitHub Action), simple Application Programming Interface (API) call or secure code upload. The algorithm is exploitability-focused, and the analysis is based on OWASP benchmark scores.

4.2.2.2 Contrast SCA

Contrast SCA [110] is a tool that tests and protects third party open source code. This testing can be done through the entire SDLC, and is a shift-left effort. As a shared service across the Contrast Application Security Platform, Contrast SCA also provides third-party software visibility without the need to deploy any additional tools.

4.2.2.3 Contrast Assess

Contrast Assess [111] is an IAST tool that continuously detects and prioritizes vulnerabilities in real-time. With this tool, organizations can visualize the application architecture, code trees, and message flow information. The application's major architectural components are illustrated in automatically generated diagrams, which helps the developer to quickly identify the meaning of a vulnerability that Contrast Assess pinpoints, and can then form a starting point for threat modeling remediation.

When a vulnerability is found, the Contrast Assess platform explains the vulnerability, and pinpoints the place where it is. This enables developers to fix the vulnerabilities without the need of security expertise.

Contrast Assess provides a mapping of the URLs and routes of the software that are executed during the testing phase of the SDLC. This helps increase confidence in the coverage of the Assess solution, as well as identify the effectiveness of the overall testing practice.

4.2.2.4 Contrast Serverless

Contrast Serverless [112] is a tool that can find and fix security issues across cloud-native environments. It helps find vulnerabilities in custom code and open source. Since it has near real-time monitoring and testing of every change that has been deployed in serverless environments, it provides vulnerability context of code, configuration, relationships, and flows, both to the developers and the application security teams.

4.2.2.5 Contrast Protect

Contrast Protect [113] is a RASP tool that detects and blocks run time attacks on known and unknown code vulnerabilities. When exploits are detected, it will notify about whether the attack reached its target or not, and give a detailed report. The report will include code lines, queries executed, files accessed, and more.

4.2.3 How Contrast Security Handles Data

How Contrast Security handles data [114] is largely dependent on how it is configured. Contrast Security offers data masking, which protects sensitive data in the application by redacting it in vulnerability and attack report. This is further sent to Contrast, syslog or security log. Data masking censors sensitive data, or data types, into keywords. This type of masking will happen in query parameters, request headers, cookies, and body.

Contrast Security agents will try to avoid sensitive data being logged in the log statements. However, it might be included if the level is set to DEBUG or lower.

4.2.4 Vulnerability Scoring

Contrast Security scoring system [115] involves comparing impact and likelihood, and merging this into one score (figure 4.2). This score is also compared to CVE confidence.

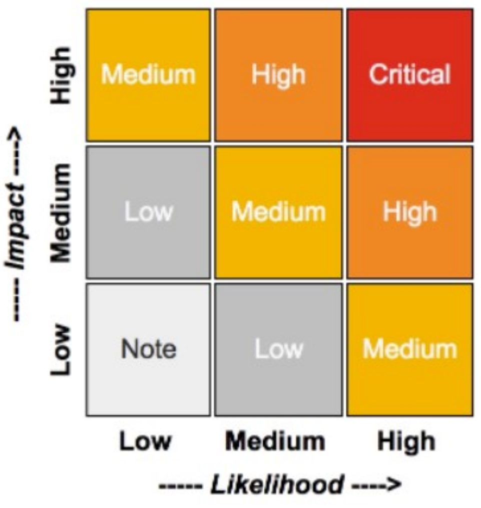


Figure 4.2: The risk severity [115]

Contrast Security also has its own scoring system for the whole application [116]. This scoring system goes from A to F, where A is the best and F is the worst. The score is calculated by starting with 100 points (maximum), then the number of critical vulnerabilities is multiplied with 20 and subtracted from the 100 points that was started with. The same is done with the high, medium and low vulnerabilities, except that for high, the number of vulnerabilities is multiplied by ten, for medium it is multiplied by five, and for low it is multiplied by one.

4.3 Snyk

4.3.1 Why Snyk Was Chosen

Snyk was chosen because it covers SCA and SAST. SCA will help NDMA with finding insecure libraries in the code they are writing, while SAST will help finding insecurities while the code is written. This will help NDMA with finding vulnerabilities early in the SDLC, hence making it relevant for the software they develop. Being in business since 2015, it is plausible that Snyk have built up comprehensive documentation, resulting in easier troubleshooting. Snyk works with containers and supports many different programming languages, such as .Net, C/C++, Go, Java, JavaScript, PHP, Python and others [117]. It also supports different integrations, such as GitHub, Jenkins, Maven, AWS, IntelliJ, and more [118]. Google, Amazon Web Services (AWS) and Mnemonic are trusted companies using Snyk.

4.3.2 Product Overview

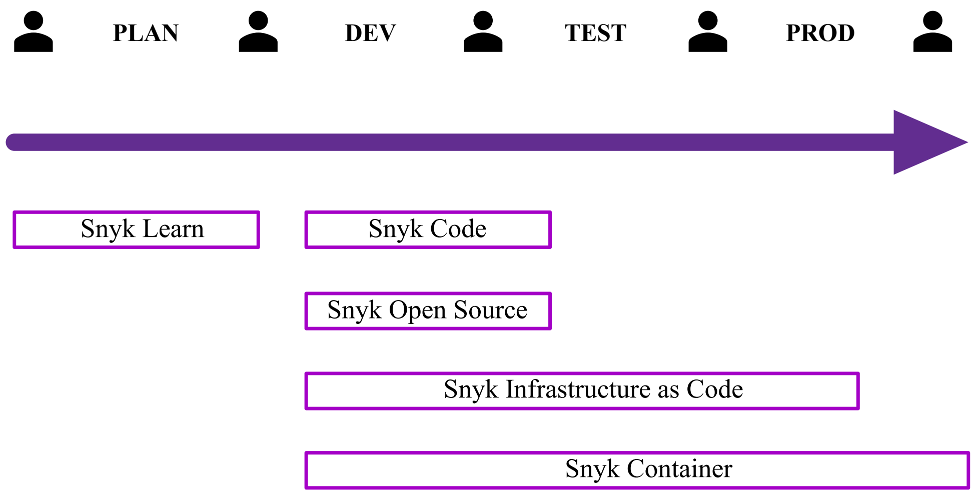


Figure 4.3: Snyk products placed in the SDLC

Snyk [92] is a developer security platform. It is used to secure code, dependencies, containers, and Infrastructure as Code (IaC). Snyk consists of four products [100], Snyk Open Source, Snyk Code, Snyk Container, and Snyk Infrastructure as Code (figure 4.3). It also provides Snyk Learn [119], and Snyk Vulnerability Database [120].

4.3.2.1 Snyk Open Source

Snyk Open Source [121] is a SCA tool that helps developers detect vulnerabilities in the open source dependencies early in the SDLC. It does this by scanning the pull requests before merging. It also tests the projects directly from the repositories regularly to see if there are any new vulnerabilities introduced after the first scan. A Snyk test is added to the CI/CD to prevent new vulnerabilities from passing through the build process as well [122].

4.3.2.2 Snyk Container

Snyk Container [123] solution has two functions. It provides tools to find and fix vulnerabilities that may exist in a container, and it helps to build security into images from the start.

4.3.2.3 Snyk Code

Snyk Code [124] is designed to be developer-first. It is a SAST tool that begins scanning the code from the start and not after compilation. To avoid false positives, Snyk Code uses Artificial Intelligence (AI). The AI uses open source code to learn, and pairs this information with Snyk's Security Intelligence database. Another feature of Snyk Code [125], is that it also gives example solutions and education about vulnerabilities.

4.3.2.4 Snyk Infrastructure as Code

Snyk Infrastructure as Code (IaC) [126] is used for helping developers write secure configurations for applications. It provides fix advice so it is possible to make changes directly into the code before the application goes to production. It is used in configurations for AWS CloudFormation, Azure Resource Manager (ARM), Kubernetes, and HashiCorp Terraform.

Snyk IaC integrates security checks into the SDLC by providing immediate local feedback when writing configurations. In this way issues can be found and fixed before they are committed. To automate security checks, it is possible to integrate Snyk directly into the CI/CD processes, and for ongoing monitoring and analysis, it is possible to import source repositories.

4.3.2.5 Snyk Learn

Snyk Learn [119] is designed to help developers learn how to exploit and mitigate vulnerabilities. This is a free web page where developers can choose the type of ecosystem they use when developing. Snyk Learn provides lessons for JavaScript, Java, C#, GO, PHP, Kubernetes and Python. In the lessons, developers get a briefing on the basics of the vulnerabilities and get hands on experience with how to exploit them [127]. It also provides information about how the vulnerabilities work, the impact of an exploitation, and how to mitigate them.

4.3.2.6 Snyk Vulnerability Database

Snyk Vulnerability Database [120] is a database with different open source vulnerabilities. It contains information about what they are, what they do, and how to prevent them. It is also possible to report threats or new vulnerabilities directly to Snyk. They can then look at them and potentially add them to the database.

4.3.3 How Snyk Handles User Code and Repositories

Snyk is a Software as a Service (SaaS) application, and they fully manage everything [128]. There is an option to use Snyk as a SaaS with Broker. This is a service where Snyk is installed in a private infrastructure, where it acts like a proxy between Snyk and the private codebase. The Broker will control the connections, encrypt data when it is being transmitted, and all the user's sensitive information will stay behind their own firewall.

4.3.3.1 Snyk Open Source

Snyk Open Source [128] has access to the build configuration files and to the manifest files to identify the open source dependencies in the code. The SCA scan does not need to access any of the source code. It only needs to access the name and version number of the dependencies, which it also stores. For the optional ADD-ONS Reachable Vulnerabilities computation and Lambda integration only, Snyk stores the code until the analysis is done. Then the code is removed.

4.3.3.2 Snyk Container

Snyk Container [128] will access and store package versions, executable hashes, what type of OS is used, metadata of the container image, and information about the overall image. From the Dockerfile, it will also store the RUN instructions. For the Kubernetes configurations, Snyk needs to access the workload security settings, and for container registry integrations, Snyk stores a temporary copy of the container image until the analysis is done. Snyk does not store the container image if a Broker is used.

4.3.3.3 Snyk Code

Snyk Code [128] stores code for a period up to 24 hours, before removing it for the network and logs. After this, Snyk only stores the position of the vulnerabilities found. The results of the scan will be stored in a database for analytic and monitoring purposes, but the results will not be used for engine training purposes, or to make examples for fixes in the code. Snyk Code only uses pointers to positions and identification meta-information when showing the results, and does not contain original source code.

4.3.3.4 Snyk Infrastructure as Code

When Snyk IaC [128] performs a CLI scan, the scan is only performed locally. For the Git-based scans, IaC files are stored temporarily and completely deleted afterwards.

4.3.4 Vulnerability Scoring

Snyk uses CVSS framework version 3.1 [42] for the vulnerability scoring.

4.4 StackHawk

4.4.1 Why StackHawk Was Chosen

StackHawk was chosen because it covers DAST, and is thus mainly relevant for the software that NDMA get from external providers. DAST does not cover the need for testing libraries, but due to defense in depth it is still considered important in order to ensure a more secure software. StackHawk can be used on the internal software as well, as a last security check before deployment. StackHawk has a collaboration with Snyk, so even though StackHawk is a relatively new security testing tool, it is supported by someone with more experience and recognition [129]. The StackHawk application is inspired by OWASP ZAP [130], which is a known security testing tool [131], and is often used for preventing OWASP top 10 vulnerabilities (figure H.1). It also supports different integrations, such as GitHub, Slack, Microsoft Teams, and more [132].

4.4.2 Product Overview

StackHawk will look for vulnerabilities in the applications, as well as services and APIs, both during and after the development phase. It is a DAST tool made with developers in mind, where it is possible to scan in the CI/CD pipeline, before hitting production. It will look for both security vulnerabilities that the team has introduced, and exploitable open source bugs [133].

StackHawk has two parts called the HawkScan Scanner and the StackHawk Platform [93]. The HawkScan Scanner is a security bug scanner, which is command-line based. It is supported by OWASP ZAP. The StackHawk Platform [134] is a Graphical User Interface (GUI) to start scans and view the results.

Results from scans are presented in the terminal and also collected on the StackHawk Platform. These results are split into two categories; a result summary and result payload. The summary shows the total number of vulnerabilities, and the payload gives specific details about each vulnerability.

4.4.3 How StackHawk Handles Data

StackHawk has a list of what is considered personal information that they collect [135]. This covers both information about the business itself and employees, as well as software frameworks and the scanning results. The information is used by StackHawk when communicating with customers, and in order to improve their service and personalize the experience for customers.

StackHawk might share personal information if affiliates, service providers, or professional advisors ask for it. StackHawk is also willing to share personal information if it is needed for compliance, fraud prevention, and safety. StackHawk might also sell personal information in special cases. These cases are listed on their site in the Privacy Policy. The Privacy Policy is often updated, so it is important to pay attention to changes.

4.4.4 Vulnerability Scoring

StackHawk risk scoring system [136] is based on the OWASP Risk Rating Methodology.

4.5 Veracode

4.5.1 Why Veracode Was Chosen

Veracode was chosen because it covers SAST, DAST and SCA, and thus can be used on both internal and external software. Having all these options available from one supplier might make security testing easier to handle. Veracode also supports many different programming languages, such as Java, Python, Go, C/C++, and others [137]. GitHub, Jenkins, Maven, and AWS are some of the integrations Veracode supports [138].

4.5.2 Product Overview

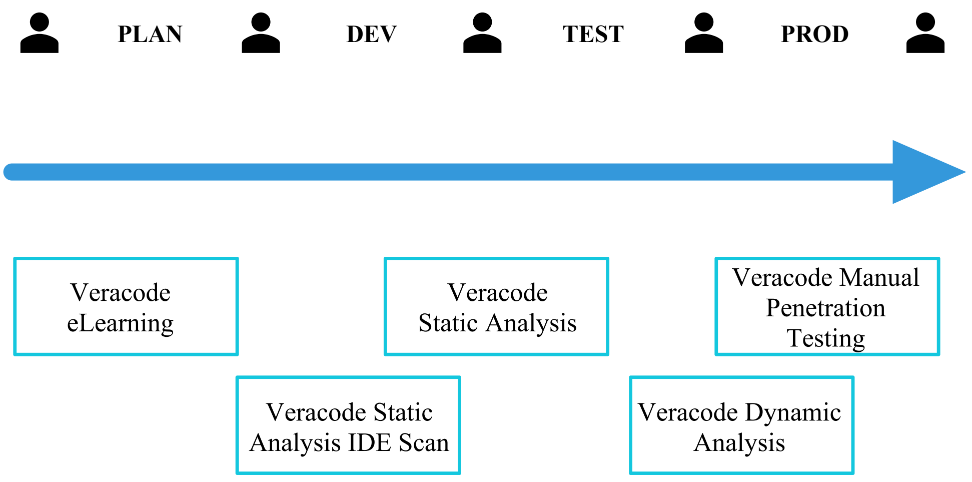

PLAN
DEV
TEST
PROD
Veracode eLearning
Veracode Static Analysis
Veracode Manual Penetration Testing
Veracode Static Analysis IDE Scan
Veracode Dynamic Analysis

Figure 4.4: Veracode products placed in the SDLC [139]

Veracode has a lot of different products to choose from (figure 4.4); Veracode Static Analysis, Veracode Software Composition Analysis, Veracode Dynamic Analysis, Veracode Discovery, Veracode Manual Penetration Testing, and Veracode eLearning. It is possible to only choose some of the products if some functions are not needed or covered by other products already owned.

4.5.2.1 Veracode Static Analysis

Veracode Static Analysis [140] is a SAST solution that makes it possible to quickly identify and remediate application security findings. It can analyze frameworks and languages without needing the source code. Code that is written, bought, or downloaded can be assessed, and the progress can be measured. It gives feedback to the developers in the IDE, and in the CI/CD pipeline. Before deployment [141], Veracode Static Analysis conducts a full policy scan, that offers guidance on how to find, prioritize, and fix issues in the code, and it does not need tuning.

4.5.2.2 Veracode Software Composition Analysis

Veracode Software Composition Analysis (SCA) [142] helps building an inventory of all third party components to identify potential vulnerabilities, including both open source and commercial code. It scans a list of libraries, and identifies the vulnerabilities in each of them. A notification is sent out for any new vulnerabilities discovered that may impact the application, without the need to manually perform a new scan. There are two types of possible scans, which can be run at different stages in the SDLC. These are the upload and scan method and the agent-based scan.

The upload and scan method scans the application after it is compiled and the application binaries are uploaded to the Veracode platform. The binaries can be uploaded through the Veracode platform user interface, or by using the Veracode Extensible Markup Language (XML) APIs. This method makes it possible to perform a SCA scan together with a Veracode Static Analysis, or separately.

The agent-based scan method scans the code early and frequently in the SDLC. This method makes it possible to scan repositories or projects cloned locally on the computer. Agent-based scanning can also be integrated into the continuous integration (CI) pipelines. C/C++ scanning, Docker container scanning, and other additional insights, for example vulnerable methods and dependency graphs, can only be done through agent-based scanning.

4.5.2.3 Veracode Dynamic Analysis

Veracode Dynamic Analysis [143] is a DAST solution that delivers a scalable and automated dynamic scanning capability. It can be used to scan both web applications and API specifications.

Veracode Dynamic Analysis can be used to run security tests against live web applications in the last stages of the SDLC, for example during the testing or production stage. It can also be used for API scanning, to test the security of the endpoints in the API specifications. The different scanning methods with this solution are an authenticated and unauthenticated dynamic analysis of a web application, and a dynamic analysis for an internal web application.

Veracode Dynamic Analysis integrates with Veracode Discovery and it provides Veracode Internal Scanning Management (ISM) to access applications and API specifications behind a firewall.

4.5.2.4 Veracode Discovery

Veracode Discovery [144] makes it possible to manage the elusive web attack surface by discovering and inventorying all public-facing applications, both inside and outside of the IP range. This provides an easy workflow to scan different sites for vulnerabilities. It can be used alone or together with Veracode Dynamic Analysis to discover potential flaws in assets that have been identified.

4.5.2.5 Veracode Manual Penetration Testing

Veracode Manual Penetration Testing (MPT) [145] involves different penetration testers who simulate real-life attacks and perform tests. The goal with this is to determine if attackers have the potential to successfully access and perform malicious activities, by exploiting previously known or unknown vulnerabilities in the software. This is done by using a combination of manual testing and automatic penetration testing. It is recommended that MPT is used in conjunction with other automated security assessments. By using this approach, the penetration testers can focus on complex attack schemes and business logic flaws that are not as easy to automate.

4.5.2.6 Veracode eLearning

Veracode eLearning [146] consists of course-based training that will help developers get the necessary knowledge to identify and address potential vulnerabilities when developing. This includes online courses to improve security knowledge and a Knowledge Base on secure software development. The Knowledge Base is a collection of information about different vulnerabilities, together with techniques that can be used to prevent them.

4.5.3 How Veracode Handles Data

Veracode recognizes uploaded application files as binaries [147]. To allow for re-scanning applications without re-uploading modules with issues or errors, and performing results-quality investigations upon request, Veracode retains uploaded binaries for a number of days. Binaries for submitted scans are retained for 30 days, whereas binaries for scans that did not complete are retained for 90 days.

4.5.4 Vulnerability Scoring

Veracode vulnerability scoring [148] is based on both CWE and CVSS. CVSS is only used on the SCA and the MPT products, where the user can choose to use either version two or three [149], while CWE is used on all of Veracode's products.

4.6 Selected Tools in the SDLC

Figure 4.5 shows the four examined tools with their products, and where they are placed in the SDLC.

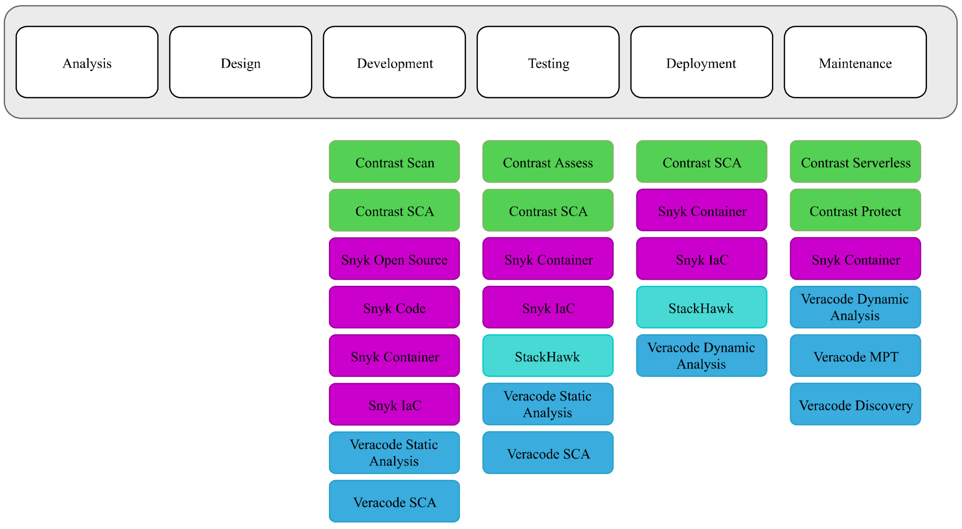


Figure 4.5: The tools placed in the SDLC

Figure 4.6 shows in which context the different tools can be used. NDMA has both internal and external software that is either compiled or source code. These types of software can be security tested through DAST, SCA, or SAST. The dotted lines show that it is partly connected. For some tools, SCA works with compiled software, while for others it does not.

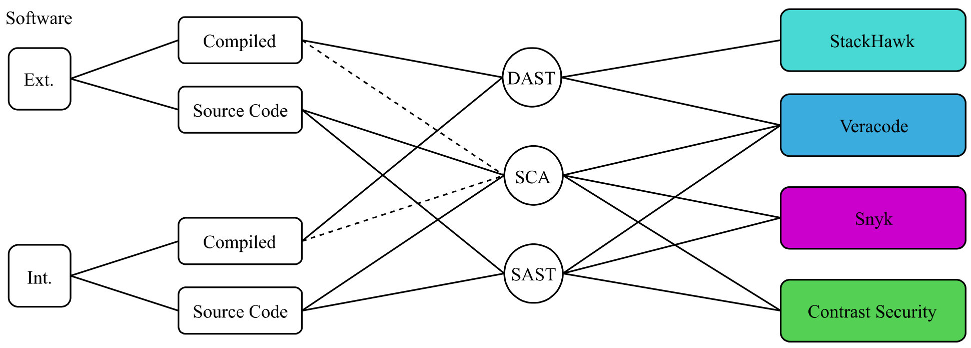


Figure 4.6: The tools and what type of software they can be used on

Chapter 5

Experiment

5.1 Introduction

The experiment chapter starts with a description of the test environment and test data. Thereafter, explanations of how the tests were conducted together with their corresponding results are presented. The tests are divided into two sections: tools that were tested by the group, and tools that could not be tested by the group.

5.2 The Test Environments

There were different testing environments used. The environment used for testing Contrast Security and StackHawk was a PC with Windows 10 pro. A browser, Microsoft Edge, was used to see results on the tools website. PowerShell was used for the experiment itself, while Docker was only used for StackHawk tests. The environment used to test Snyk was a Microsoft Edge browser with a user logged into GitHub through the Snyk website.

5.2.1 Test Data

There were three sets of test data used to conduct the tests, two from a student at NTNU, and one from a former exam of a group member. These were used because they met the requirements needed in order to conduct the tests.

The first test data was a "To do list" application called "FastTasks", which let the user add different activities, prioritize them, and set a due date. This application was written in Java, and the file was a .jar file. The application was not connected to the Internet. This test data was tested with Contrast Security and Snyk.

The second test data was used to test StackHawk. This was a Representational state transfer (REST) API, which let the user write a country in the URL, in order to find some information about it, and universities located in that country. This API was written in GO, and the information was fetched from two other APIs.

Lastly, Snyk was also tested with a prior exam called "prog2053-exam2020", to show the automatic fix function. It consists mostly of PHP code, with some HTML and JavaScript.

5.3 Tested Tools

Some tools could be tested by using a free trial. Not all of the free trials offer the same functions as the purchased product, which makes some of the tests limited. These tests are meant to give some insight on how the tools work, how difficult they are to set up, and how they perform in terms of functionality.

5.3.1 Contrast Security

There are different ways to test Contrast Security, but there was only one way that was possible to test with the free trial. This was with the community edition, which covers SCA, IAST, and RASP.

5.3.1.1 Community Edition

The Java Quick Starter Guide [150] was used in order to do the test. This guide was made for use with Linux, and since this was done on Windows, a few changes had to be made.

The first command, shown in code listing 5.1, did not work because the cURL command is not the same in PowerShell as in Linux. The PowerShell command replacing `curl` is `curl.exe`, and this was thus used instead.

Code listing 5.1: Linux command to download the JAR file agent

```
curl -L 'https://repository.sonatype.org/service/local/artifact/maven/redirect?r=
central-proxy&g=com.contrastsecurity&a=contrast-agent&v=LATEST' -o contrast.jar
```

Then the YAML file had to be changed according to the guide.

The last command needed to do the test, was 5.2, but this would not work in PowerShell. This was because it did not read the command correctly. To make it work, a suggestion from Stack Overflow [151] was followed, and it was changed to the command shown in 5.3.

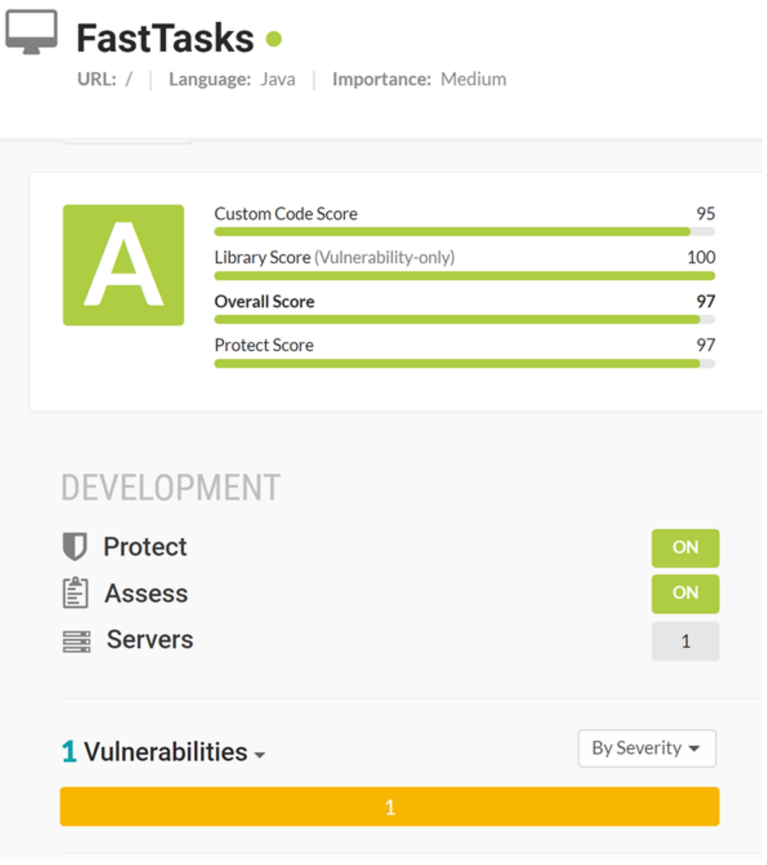
Code listing 5.2: Linux command to tell the agent where to find the YAML file

```
java -javaagent:./contrast.jar -Dcontrast.config.path=contrast_security.yaml -jar
<ApplicationJarPath>
```

Code listing 5.3: PowerShell command to tell the agent where to find the YAML file, with the application path for this test

```
java -javaagent:'./contrast.jar' '-Dcontrast.config.path=contrast_security.yaml'
-jar .\FastTasks.jar
```

Then the application being tested started automatically. When the application was interacted with, Contrast Security was able to find vulnerabilities. After reloading the Contrast Security site, the results appeared (figure 5.1 - 5.4):



FastTasks

URL: / | Language: Java | Importance: Medium

Custom Code Score 95

Library Score (Vulnerability-only) 100

Overall Score 97

Protect Score 97

DEVELOPMENT

Protect ON

Assess ON

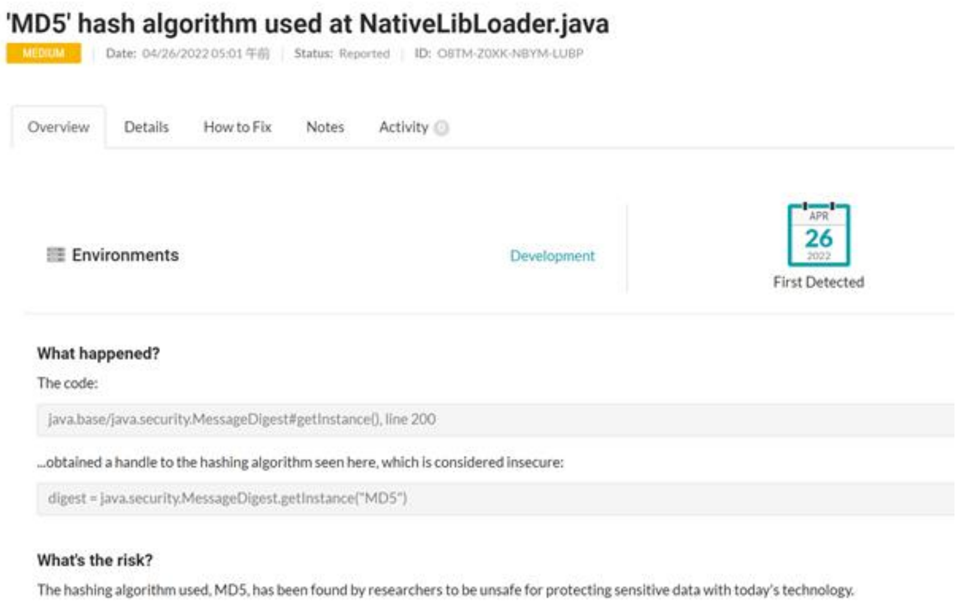
Servers 1

1 Vulnerabilities

By Severity

1

Figure 5.1: The results from the test


'MD5' hash algorithm used at NativeLibLoader.java

MEDIUM | Date: 04/26/2022 05:01 午前 | Status: Reported | ID: O8TM-Z0XK-NBYM-LUBP

Environments: Development

APR 26 2022 First Detected

What happened?

The code:

```
java.base/java.security.MessageDigest#getInstance(), line 200
```

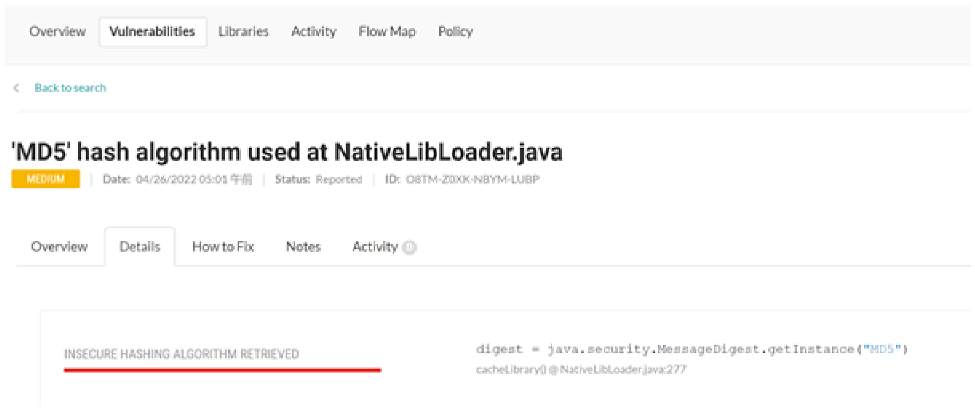
...obtained a handle to the hashing algorithm seen here, which is considered insecure:

```
digest = java.security.MessageDigest.getInstance("MD5")
```

What's the risk?

The hashing algorithm used, MD5, has been found by researchers to be unsafe for protecting sensitive data with today's technology.

Figure 5.2: Overview of the vulnerability found



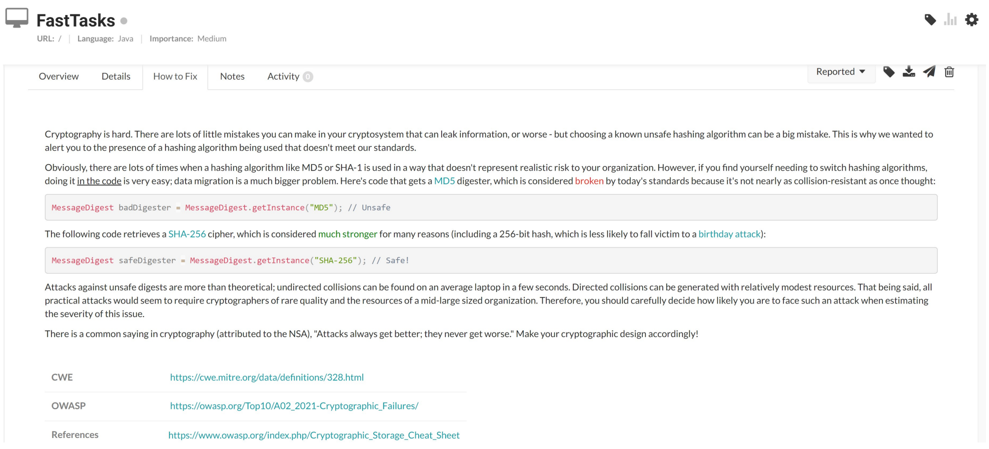
'MD5' hash algorithm used at NativeLibLoader.java

MEDIUM | Date: 04/26/2022 05:01 午前 | Status: Reported | ID: O8TM-Z0XK-NBYM-LUBP

INSECURE HASHING ALGORITHM RETRIEVED

```
digest = java.security.MessageDigest.getInstance("MD5")
cacheLibrary() @ NativeLibLoader.java:277
```

Figure 5.3: Details about the vulnerability found



Cryptography is hard. There are lots of little mistakes you can make in your cryptosystem that can leak information, or worse - but choosing a known unsafe hashing algorithm can be a big mistake. This is why we wanted to alert you to the presence of a hashing algorithm being used that doesn't meet our standards.

Obviously, there are lots of times when a hashing algorithm like MD5 or SHA-1 is used in a way that doesn't represent realistic risk to your organization. However, if you find yourself needing to switch hashing algorithms, doing it in the code is very easy; data migration is a much bigger problem. Here's code that gets a MD5 digester, which is considered broken by today's standards because it's not nearly as collision-resistant as once thought:

```
MessageDigest badDigester = MessageDigest.getInstance("MD5"); // Unsafe
```

The following code retrieves a SHA-256 cipher, which is considered much stronger for many reasons (including a 256-bit hash, which is less likely to fall victim to a birthday attack):

```
MessageDigest safeDigester = MessageDigest.getInstance("SHA-256"); // Safe!
```

Attacks against unsafe digests are more than theoretical; undirected collisions can be found on an average laptop in a few seconds. Directed collisions can be generated with relatively modest resources. That being said, all practical attacks would seem to require cryptographers of rare quality and the resources of a mid-large sized organization. Therefore, you should carefully decide how likely you are to face such an attack when estimating the severity of this issue.

There is a common saying in cryptography (attributed to the NSA), "Attacks always get better; they never get worse." Make your cryptographic design accordingly!

CWE	https://cwe.mitre.org/data/definitions/328.html
OWASP	https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
References	https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

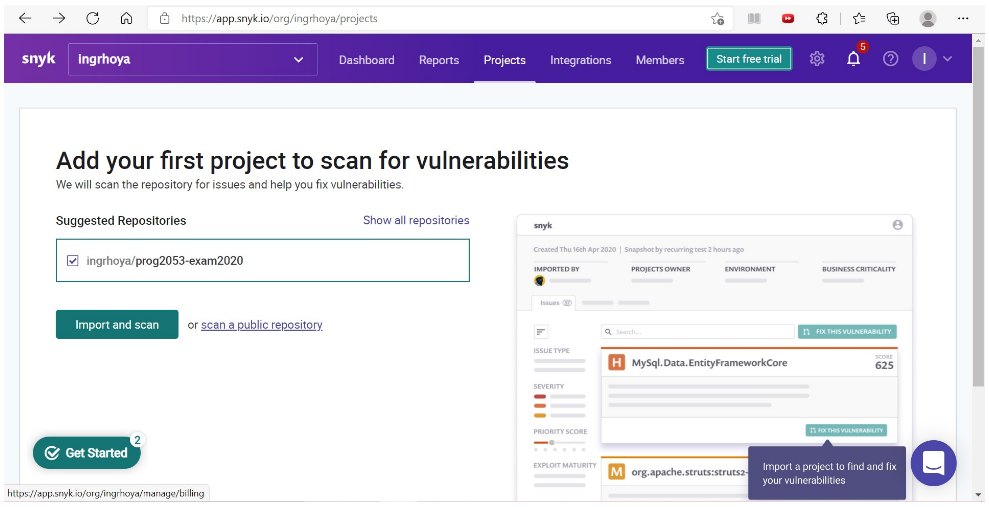
Figure 5.4: Information about how to fix the vulnerability

5.3.2 Snyk

Snyk has different ways to test Snyk Code and Snyk Open Source. They can for example be tested with the GitHub integration, or in a IDE integration like the VS Code plugin. Here it is tested with the GitHub integration, with the "prog2053-exam2020" application. A more detailed description of the procedure and how it works, can be found in appendix B.1. The GitHub test with the "FastTasks" application can be seen in appendix B.1.2. This test did not show examples on automatic fixes for the vulnerabilities, but can be used for comparison with the other tools. The test with the IDE integration should give the same results as with the GitHub integration.

5.3.2.1 GitHub

A free Snyk account was made when a GitHub account was used to sign in on Snyk's website. The first page that was opened (figure 5.5) showed a list of suggested repositories to scan from the GitHub profile. In this example only one repository showed up in the suggestions, but other repositories could be seen by clicking the "show all repositories" button. The wanted repository was the one suggested, so it was imported and scanned.


https://app.snyk.io/org/ingrhoya/projects
snyk
ingrhoya
Dashboard
Reports
Projects
Integrations
Members
Start free trial
Suggested Repositories
Show all repositories
ingrhoya/prog2053-exam2020
Import and scan
or scan a public repository
Get Started
MySql.Data.EntityFrameworkCore
org.apache.struts:struts2-
https://app.snyk.io/org/ingrhoya/manage/billing

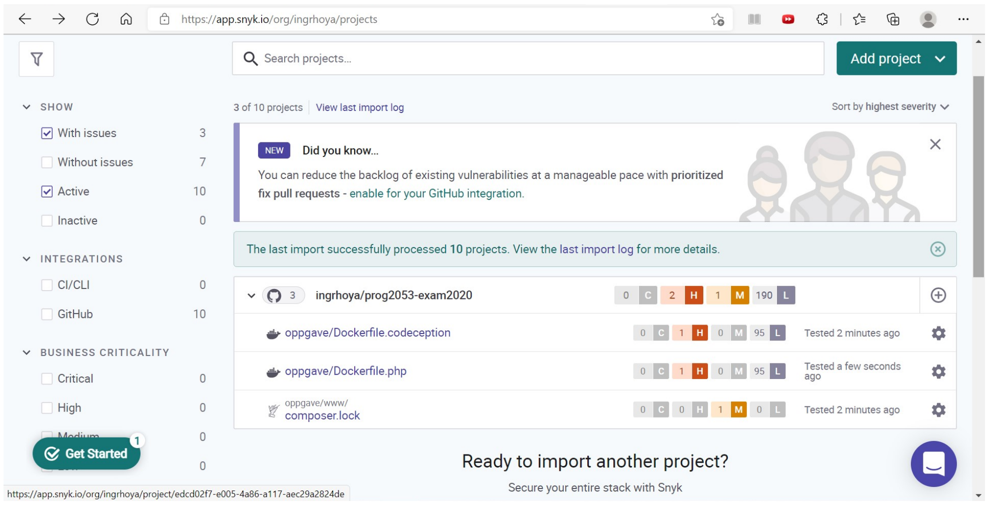
Add your first project to scan for vulnerabilities

We will scan the repository for issues and help you fix vulnerabilities.

Import a project to find and fix your vulnerabilities

Figure 5.5: Adding projects to scan for vulnerabilities

When the repository was scanned, Snyk showed an overview of the vulnerabilities found, ranked by grade of severity (figure 5.6).



You can reduce the backlog of existing vulnerabilities at a manageable pace with prioritized fix pull requests - enable for your GitHub integration.

The last import successfully processed 10 projects. View the last import log for more details.

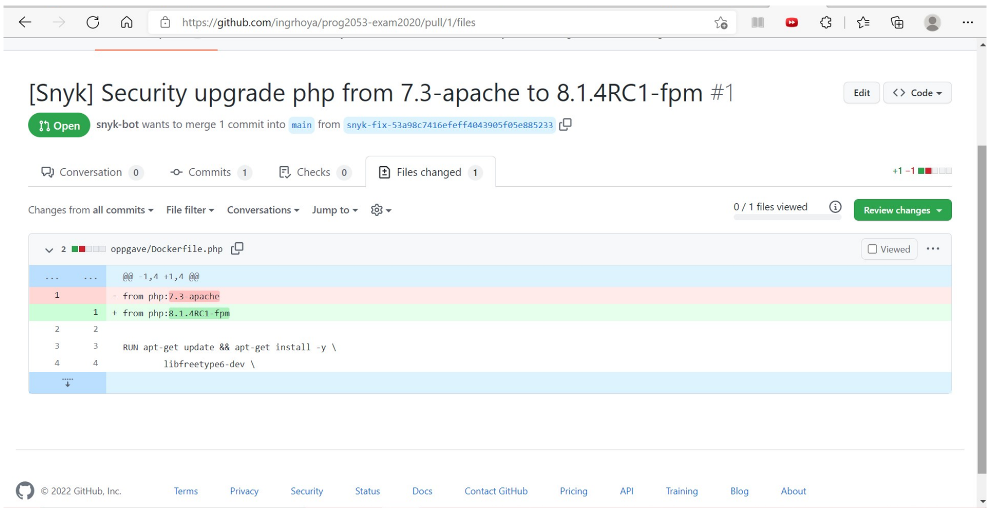
Project	C	H	M	L	Tested
ingrhoya/prog2053-exam2020	0	2	1	190	
oppgave/Dockerfile.codeception	0	1	0	95	Tested 2 minutes ago
oppgave/Dockerfile.php	0	1	0	95	Tested a few seconds ago
oppgave/www/composer.lock	0	0	1	0	Tested 2 minutes ago

Ready to import another project?

Secure your entire stack with Snyk

Figure 5.6: Scanned project with vulnerabilities ranked by grade of severity

It was also possible to see more information about the found vulnerabilities and learn how to fix them. Snyk can generate automatic pull requests for the tested GitHub repository. For this test, Snyk gave suggestion for a fix. Under, in figure 5.7, is an example of a fix, where Snyk wanted to replace the library php:7.3-apache to php:8.1.4RC1-fpm.



https://github.com/ingrhoya/prog2053-exam2020/pull/1/files

[Snyk] Security upgrade php from 7.3-apache to 8.1.4RC1-fpm #1

Open snyk-bot wants to merge 1 commit into `main` from `snyk-fix-53a98c7416efeff4043905f05e885233`

2 oppgave/Dockerfile.php

```
...  ...   @@ -1,4 +1,4 @@
1         - from php:7.3-apache
     1    + from php:8.1.4RC1-fpm
2    2
3    3      RUN apt-get update && apt-get install -y \
4    4            libfreetype6-dev \
```

Figure 5.7: An example of a suggested fix

5.3.3 StackHawk

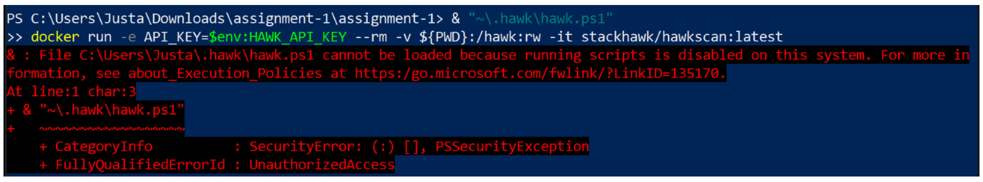
There are different ways to test StackHawk, for example with HawkScan and the GitHub integration. In this test, it is tested with HawkScan. The GitHub scan would have given the same results, due to them both testing with DAST.

5.3.3.1 HawkScan

When testing StackHawk, a user was made in order to test it for a limited period of time. StackHawk gave a clear guide on how to set up a web-application security test, when first logging into the new user. This guide was easy to follow, but it came with some issues explained in this experiment.

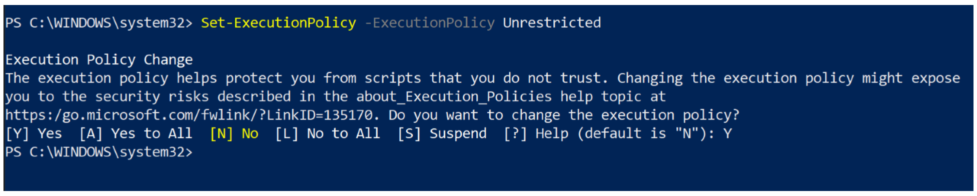
When first logging into StackHawk, an API-key was made by StackHawk, and the user is asked to save this key for later use. To run the test, some information about the application being tested had to be given. In this case the information given was that the application is a REST API on localhost:8080, and information on the environment used to conduct the tests. StackHawk gave commands to follow in order to save important information, as well as downloading everything that is needed in order to conduct the tests, both with PowerShell and Bash.

The first issue that occurred, shown in figure 5.8, happened on the last command before the test would start. This issue only happened because of the default permission in the testing environment. To fix this, the command shown in figure 5.9 was used. This command will change the policy for execution.



```
PS C:\Users\Justa\Downloads\assignment-1\assignment-1> & "~\.hawk\hawk.ps1"
>> docker run -e API_KEY=$env:HAWK_API_KEY --rm -v ${PWD}:/hawk:rw -it stackhawk/hawkscan:latest
& : File C:\Users\Justa\.hawk\hawk.ps1 cannot be loaded because running scripts is disabled on this system. For more in
formation, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:3
+ & "~\.hawk\hawk.ps1"
+   ~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

Figure 5.8: Issue with permissions

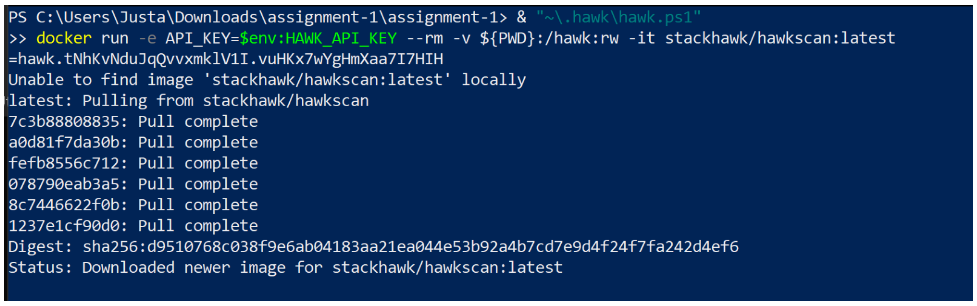


```
PS C:\WINDOWS\system32> Set-ExecutionPolicy -ExecutionPolicy Unrestricted

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): Y
PS C:\WINDOWS\system32>
```

Figure 5.9: Fix the error in 5.8

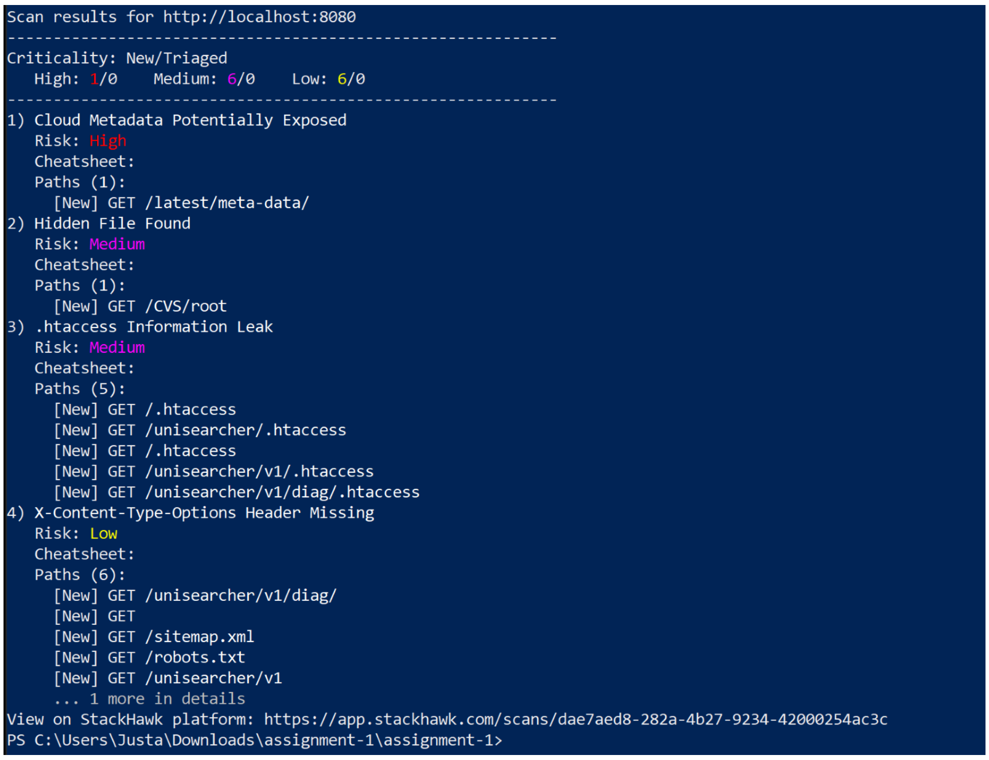
The first time running this test, it pulled a Docker image from StackHawk before starting the test, shown in figure 5.10.



```
PS C:\Users\Justa\Downloads\assignment-1\assignment-1> & "~\.hawk\hawk.ps1"
>> docker run -e API_KEY=$env:HAWK_API_KEY --rm -v ${PWD}:/hawk:rw -it stackhawk/hawkscan:latest
=hawk.tNhKvNduJqQvvxmklV1I.vuHKx7wYgHmXaa7I7HIH
Unable to find image 'stackhawk/hawkscan:latest' locally
latest: Pulling from stackhawk/hawkscan
7c3b88808835: Pull complete
a0d81f7da30b: Pull complete
fefb8556c712: Pull complete
078790eab3a5: Pull complete
8c7446622f0b: Pull complete
1237e1cf90d0: Pull complete
Digest: sha256:d9510768c038f9e6ab04183aa21ea044e53b92a4b7cd7e9d4f24f7fa242d4ef6
Status: Downloaded newer image for stackhawk/hawkscan:latest
```

Figure 5.10: First time running the command for starting a test

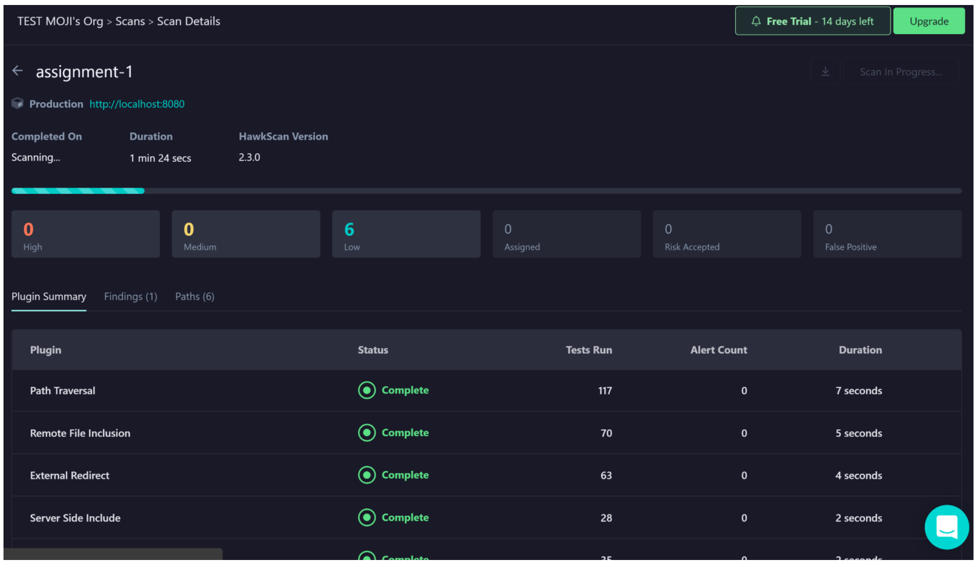
After running the test, the results showed up in the terminal, shown in figure 5.11. It showed which vulnerabilities were found, the paths they were found on, and how high of a risk that vulnerability has.



```
Scan results for http://localhost:8080
------------------------------------------------------------
Criticality: New/Triaged
   High: 1/0    Medium: 6/0    Low: 6/0
------------------------------------------------------------
1) Cloud Metadata Potentially Exposed
   Risk: High
   Cheatsheet:
   Paths (1):
     [New] GET /latest/meta-data/
2) Hidden File Found
   Risk: Medium
   Cheatsheet:
   Paths (1):
     [New] GET /CVS/root
3) .htaccess Information Leak
   Risk: Medium
   Cheatsheet:
   Paths (5):
     [New] GET /.htaccess
     [New] GET /unisearcher/.htaccess
     [New] GET /.htaccess
     [New] GET /unisearcher/v1/.htaccess
     [New] GET /unisearcher/v1/diag/.htaccess
4) X-Content-Type-Options Header Missing
   Risk: Low
   Cheatsheet:
   Paths (6):
     [New] GET /unisearcher/v1/diag/
     [New] GET
     [New] GET /sitemap.xml
     [New] GET /robots.txt
     [New] GET /unisearcher/v1
     ... 1 more in details
View on StackHawk platform: https://app.stackhawk.com/scans/dae7aed8-282a-4b27-9234-42000254ac3c
PS C:\Users\Justa\Downloads\assignment-1\assignment-1>
```

Figure 5.11: Results from test shown in PowerShell

StackHawk does not only show tests and results in the terminal, it can be seen on their platform as well. Figure 5.12 shows how the test looks on the website before finishing. Figure 5.13 shows the results after the test is done.


TEST MOJI's Org > Scans > Scan Details

assignment-1

Production http://localhost:8080

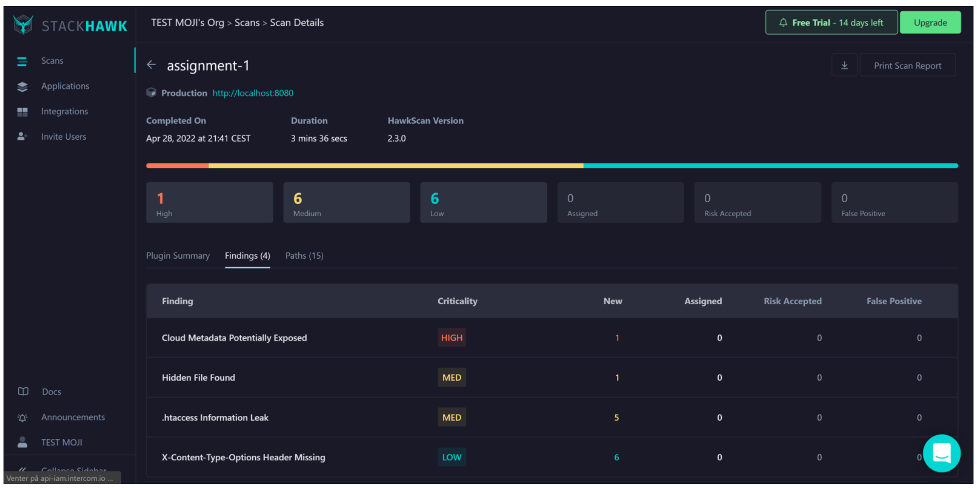
Completed On	Duration	HawkScan Version
Scanning...	1 min 24 secs	2.3.0

0 High · 0 Medium · 6 Low · 0 Assigned · 0 Risk Accepted · 0 False Positive

Plugin Summary · Findings (1) · Paths (6)

Plugin	Status	Tests Run	Alert Count	Duration
Path Traversal	Complete	117	0	7 seconds
Remote File Inclusion	Complete	70	0	5 seconds
External Redirect	Complete	63	0	4 seconds
Server Side Include	Complete	28	0	2 seconds

Figure 5.12: Test in progress on StackHawk's website


TEST MOJI's Org > Scans > Scan Details

assignment-1

Production http://localhost:8080

Completed On	Duration	HawkScan Version
Apr 28, 2022 at 21:41 CEST	3 mins 36 secs	2.3.0

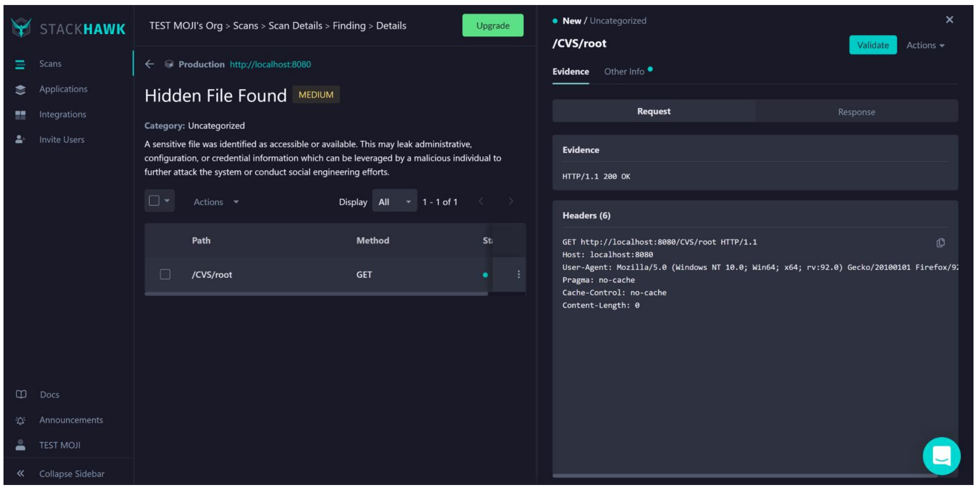
1 High · 6 Medium · 6 Low · 0 Assigned · 0 Risk Accepted · 0 False Positive

Plugin Summary · Findings (4) · Paths (15)

Finding	Criticality	New	Assigned	Risk Accepted	False Positive
Cloud Metadata Potentially Exposed	HIGH	1	0	0	0
Hidden File Found	MED	1	0	0	0
.htaccess Information Leak	MED	5	0	0	0
X-Content-Type-Options Header Missing	LOW	6	0	0	0

Figure 5.13: Website-version of the test results

By clicking on vulnerabilities on the website, StackHawk showed a new site with more information about them (figure 5.14). This site showed details about the threats found to the left, shown in figure 5.15, as well as the evidence of the vulnerability to the right.



Production http://localhost:8080

Hidden File Found MEDIUM

Category: Uncategorized

A sensitive file was identified as accessible or available. This may leak administrative, configuration, or credential information which can be leveraged by a malicious individual to further attack the system or conduct social engineering efforts.

Path	Method	St
/CVS/root	GET	●

/CVS/root

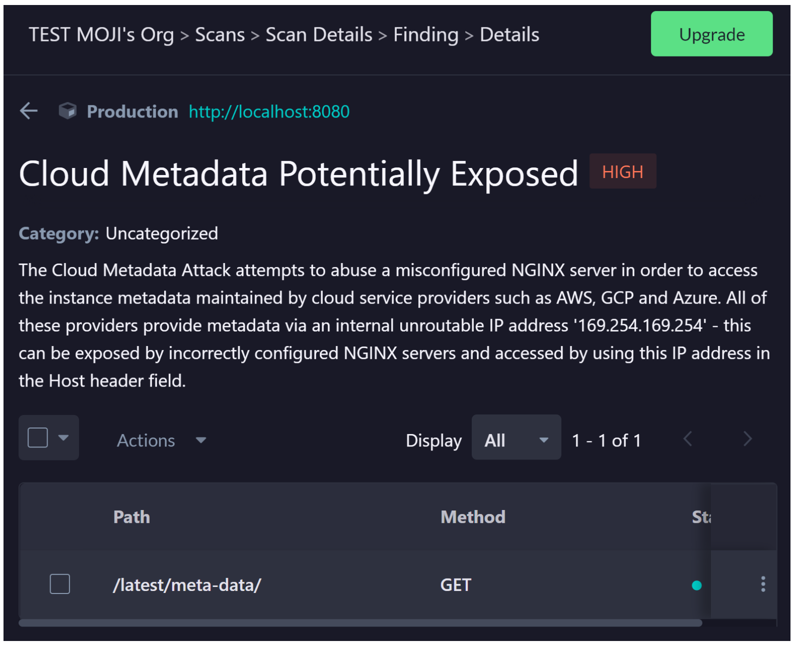
Evidence

```
HTTP/1.1 200 OK
```

Headers (6)

```
GET http://localhost:8080/CVS/root HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:92.0) Gecko/20100101 Firefox/92
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
```

Figure 5.14: Details of found vulnerability


TEST MOJI's Org > Scans > Scan Details > Finding > Details

Upgrade

Production http://localhost:8080

Cloud Metadata Potentially Exposed HIGH

Category: Uncategorized

The Cloud Metadata Attack attempts to abuse a misconfigured NGINX server in order to access the instance metadata maintained by cloud service providers such as AWS, GCP and Azure. All of these providers provide metadata via an internal unroutable IP address '169.254.169.254' - this can be exposed by incorrectly configured NGINX servers and accessed by using this IP address in the Host header field.

Actions Display All 1 - 1 of 1

Path	Method	St
/latest/meta-data/	GET	●

Figure 5.15: Details on the threat

5.4 Untested Tools

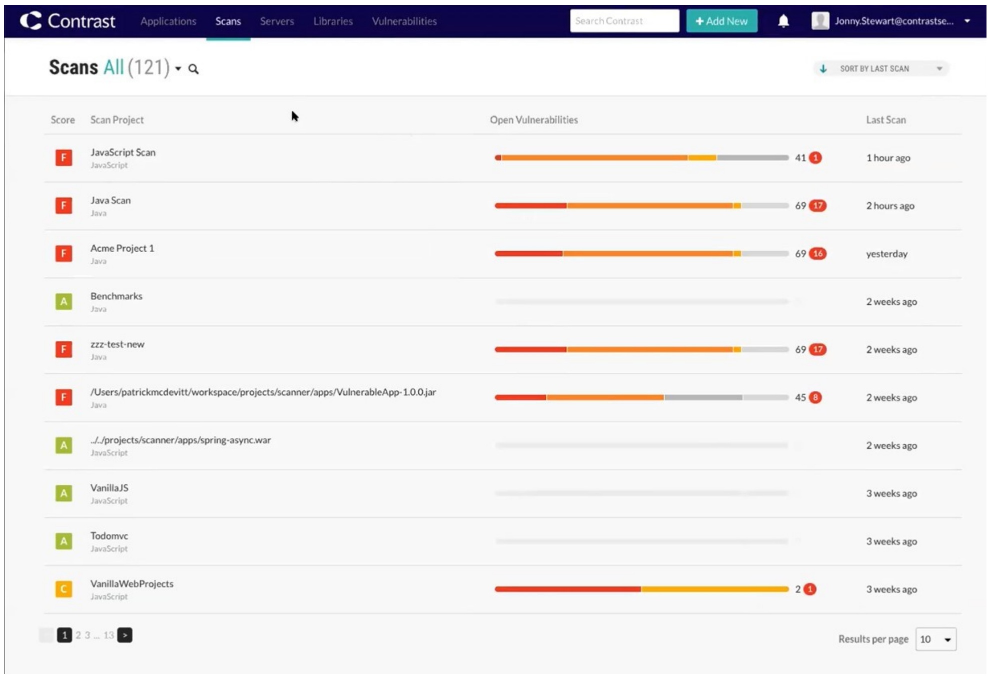
Some of the tools could not be tested. This was because the tools did not offer a free trial. The tests are hence based on documentation from the tools' websites and videos where they are tested by employees of the companies.

5.4.1 Contrast Security

Since the free version of Contrast Security only covers SCA, IAST, and RASP, it was not possible to test it with SAST [69]. To see how it works with SAST, it was thus necessary to watch videos and read documentation. A more detailed description of the procedure and how it works, can be found in appendix A.1.

5.4.1.1 Contrast Scan (SAST)

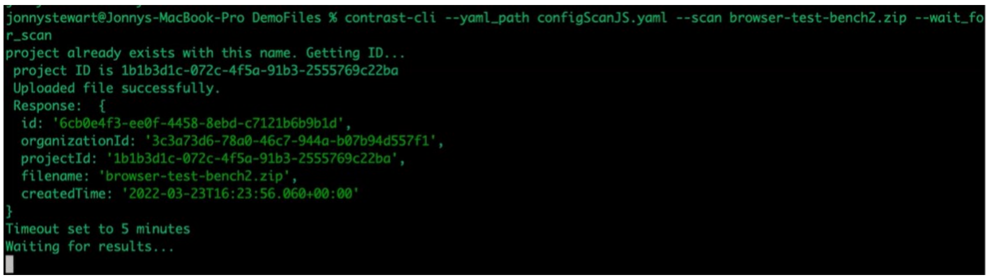
Contrast Scan (SAST) can be found in the Contrast Security editions called Detect, Detect & Protect and Enterprise [69]. A user who has access to Contrast Scan [152] can get this page in their account (figure 5.16). This is a list of all the scanned projects linked with this profile. In this page the user can also add new scans.



Score	Scan Project	Open Vulnerabilities	Last Scan
F	JavaScript Scan JavaScript	41 1	1 hour ago
F	Java Scan Java	69 17	2 hours ago
F	Acme Project 1 Java	69 16	yesterday
A	Benchmarks Java		2 weeks ago
F	zzz-test-new Java	69 17	2 weeks ago
F	/Users/patrickmcdevitt/workspace/projects/scanner/apps/VulnerableApp-1.0.0.jar Java	45 8	2 weeks ago
A	../../projects/scanner/apps/spring-async.war JavaScript		2 weeks ago
A	VanillaJS JavaScript		3 weeks ago
A	Todomvc JavaScript		3 weeks ago
C	VanillaWebProjects JavaScript	2 1	3 weeks ago

Figure 5.16: Overview over scans and add new button in the right corner [152]

There are two ways to start a new scan. The first way is to use CLI (figure 5.17) and the second is to add the files manually in the Contrast interface.

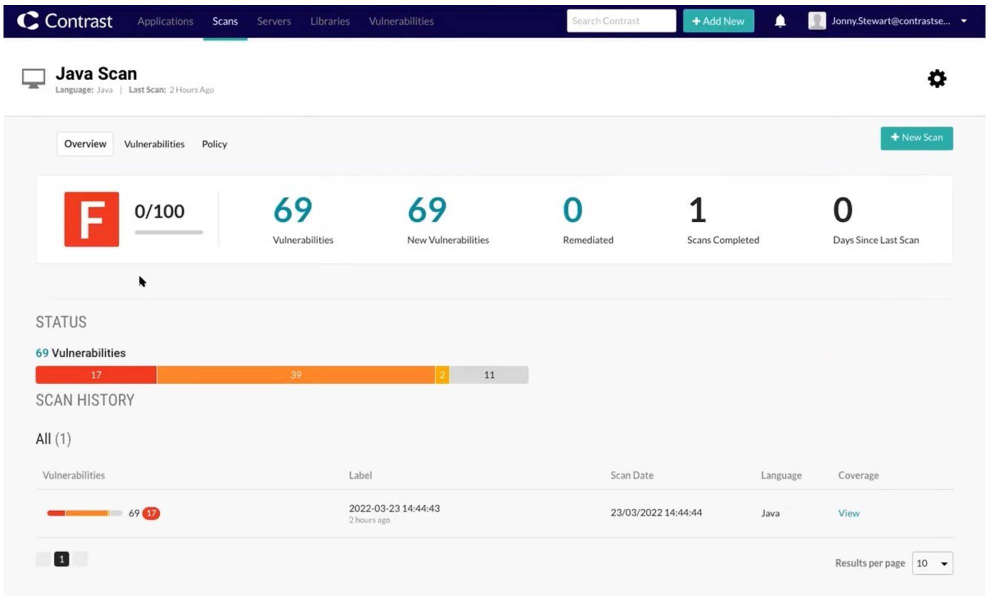


```
jonnystewart@Jonnys-MacBook-Pro DemoFiles % contrast-cli --yaml_path configScanJS.yaml --scan browser-test-bench2.zip --wait_fo
r_scan
project already exists with this name. Getting ID...
 project ID is 1b1b3d1c-072c-4f5a-91b3-2555769c22ba
 Uploaded file successfully.
 Response:  {
  id: '6cb0e4f3-ee0f-4458-8ebd-c7121b6b9b1d',
  organizationId: '3c3a73d6-78a0-46c7-944a-b07b94d557f1',
  projectId: '1b1b3d1c-072c-4f5a-91b3-2555769c22ba',
  filename: 'browser-test-bench2.zip',
  createdTime: '2022-03-23T16:23:56.060+00:00'
}
Timeout set to 5 minutes
Waiting for results...
```

Figure 5.17: Starting a new scan in the CLI [152]

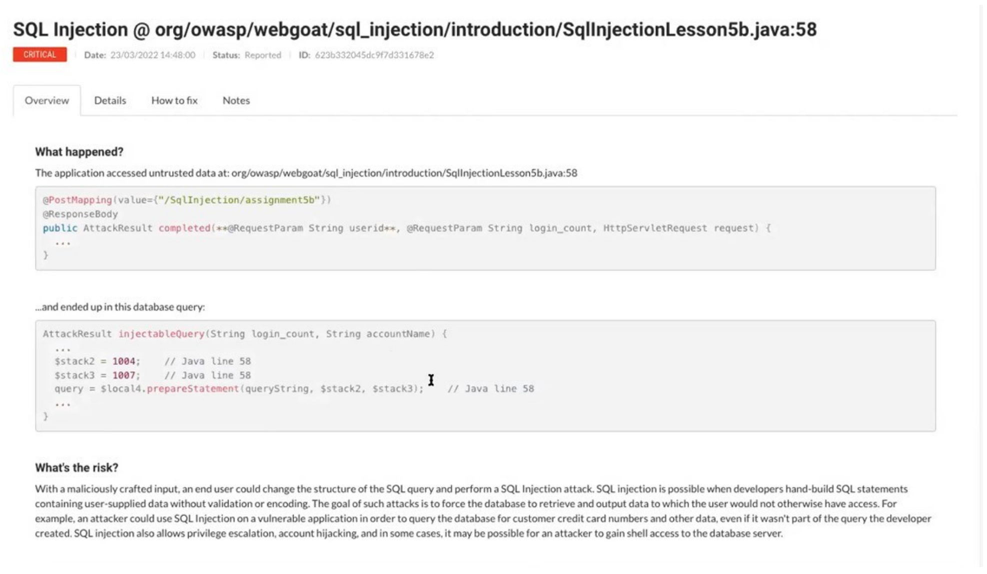
When adding a new project manually, the user can choose if Contrast Security should scan a live application or a local file. In this example, a local file will be used. When the project is created, it is possible to create a scan of the chosen file.

When the scan is complete, the Contrast Security interface will show the results from the scan (figure 5.18). This contains information about how many vulnerabilities there were, how many scans, together with a list of all of them. Contrast Security will also show more details about each vulnerability (figure 5.19), as well as solutions for how to avoid them.



Vulnerabilities	Label	Scan Date	Language	Coverage
69 17	2022-03-23 14:44:43 2 hours ago	23/03/2022 14:44:44	Java	View

Figure 5.18: One scan done on the project named "Java Scan" [152]



SQL Injection @ org/owasp/webgoat/sql_injection/introduction/SqlInjectionLesson5b.java:58

CRITICAL | Date: 23/03/2022 14:48:00 | Status: Reported | ID: 623b332045dc9f7d331678e2

What happened?

The application accessed untrusted data at: org/owasp/webgoat/sql_injection/introduction/SqlInjectionLesson5b.java:58

```
@PostMapping(value={"/SqlInjection/assignment5b"})
@ResponseBody
public AttackResult completed(**@RequestParam String userid**, @RequestParam String login_count, HttpServletRequest request) {
  ...
}
```

...and ended up in this database query:

```
AttackResult injectableQuery(String login_count, String accountName) {
  ...
  $stack2 = 1004;    // Java line 58
  $stack3 = 1007;    // Java line 58
  query = $local4.prepareStatement(queryString, $stack2, $stack3);    // Java line 58
  ...
}
```

What's the risk?

With a maliciously crafted input, an end user could change the structure of the SQL query and perform a SQL Injection attack. SQL injection is possible when developers hand-build SQL statements containing user-supplied data without validation or encoding. The goal of such attacks is to force the database to retrieve and output data to which the user would not otherwise have access. For example, an attacker could use SQL Injection on a vulnerable application in order to query the database for customer credit card numbers and other data, even if it wasn't part of the query the developer created. SQL injection also allows privilege escalation, account hijacking, and in some cases, it may be possible for an attacker to gain shell access to the database server.

Figure 5.19: Overview of a vulnerability [152]

To share the findings with other developers or interested parties, it is possible to download a Comma-Separated Values (CSV) file with all vulnerabilities, or a JavaScript Object Notation (JSON) file with the results shown in the overview of the project.

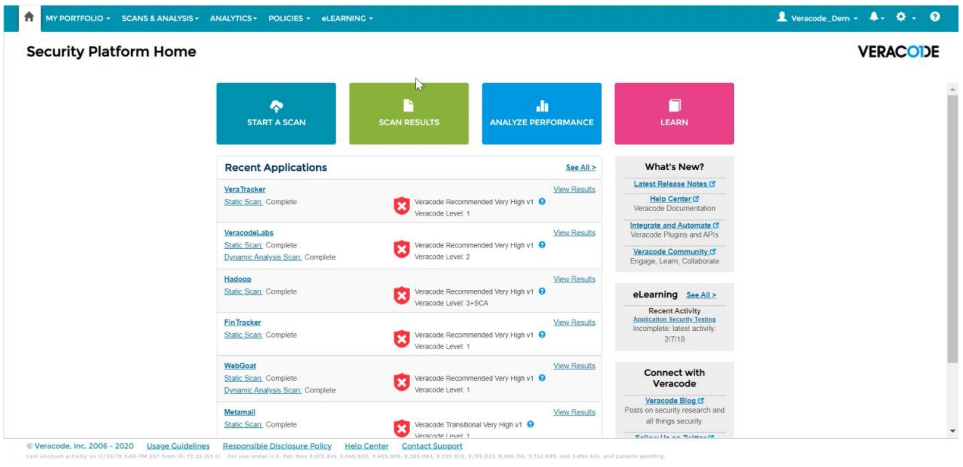
5.4.2 Veracode

5.4.2.1 Veracode SCA

Veracode SCA has two different methods to perform a scan [153]: the upload and scan method, and the agent-based scan method. A more detailed description of the procedures and how they work, can be found in appendix C.1.

Upload and Scan

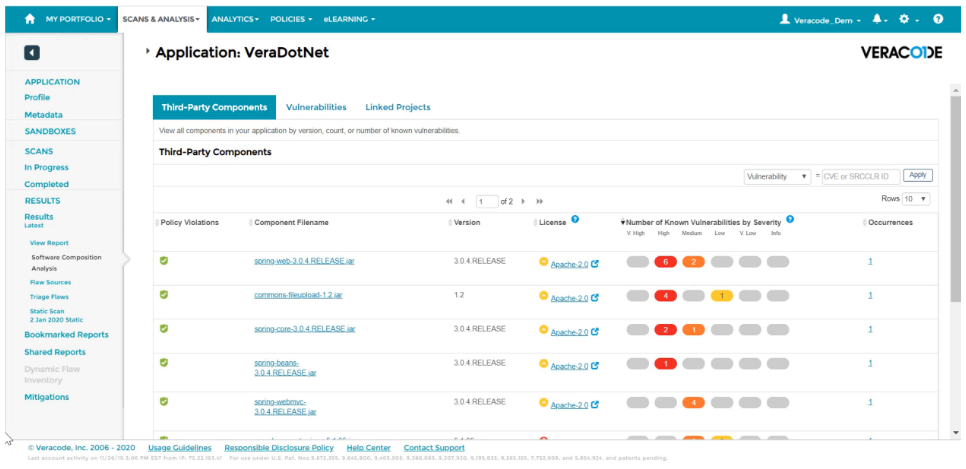
The upload and scan method [153] scans the application after it is built and uploaded to the Veracode platform (figure 5.20). It could be binaries, or a packaged application with a dependencies file. It is possible to upload the application through the Veracode platform user interface, or by using the Veracode API. The upload and scan method makes it possible to perform a SCA scan together with Veracode Static Analysis.



VeraTracker Static Scan: Complete	Veracode Recommended Very High v1 Veracode Level: 1	View Results
VeracodeLabs Static Scan: Complete Dynamic Analysis Scan: Complete	Veracode Recommended Very High v1 Veracode Level: 2	View Results
Hadoop Static Scan: Complete	Veracode Recommended Very High v1 Veracode Level: 3+SCA	View Results
FinTracker Static Scan: Complete	Veracode Recommended Very High v1 Veracode Level: 1	View Results
WebGoat Static Scan: Complete Dynamic Analysis Scan: Complete	Veracode Recommended Very High v1 Veracode Level: 1	View Results
Metamail Static Scan: Complete	Veracode Transitional Very High v1 Veracode Level: 1	View Results

Figure 5.20: The Veracode Platform [153]

The scan can be done by creating a new application or using an existing one. After all the files are added and the scan is complete, all users and teams associated with the application will receive an e-mail notification. The results from the scan can be reviewed when they are available, either from the application profile or the SCA screen (figure 5.21).



Application: VeraDotNet

View all components in your application by version, count, or number of known vulnerabilities.

Third-Party Components

Policy Violations	Component Filename	Version	License	V. High	High	Medium	Low	V. Low	Info	Occurrences
✓	spring-web-3.0.4.RELEASE.jar	3.0.4.RELEASE	Apache-2.0		6	2				1
✓	commons-fileupload-1.2.jar	1.2	Apache-2.0		4		1			1
✓	spring-core-3.0.4.RELEASE.jar	3.0.4.RELEASE	Apache-2.0		2	1				1
✓	spring-beans-3.0.4.RELEASE.jar	3.0.4.RELEASE	Apache-2.0		1					1
✓	spring-webmvc-3.0.4.RELEASE.jar	3.0.4.RELEASE	Apache-2.0			4				1

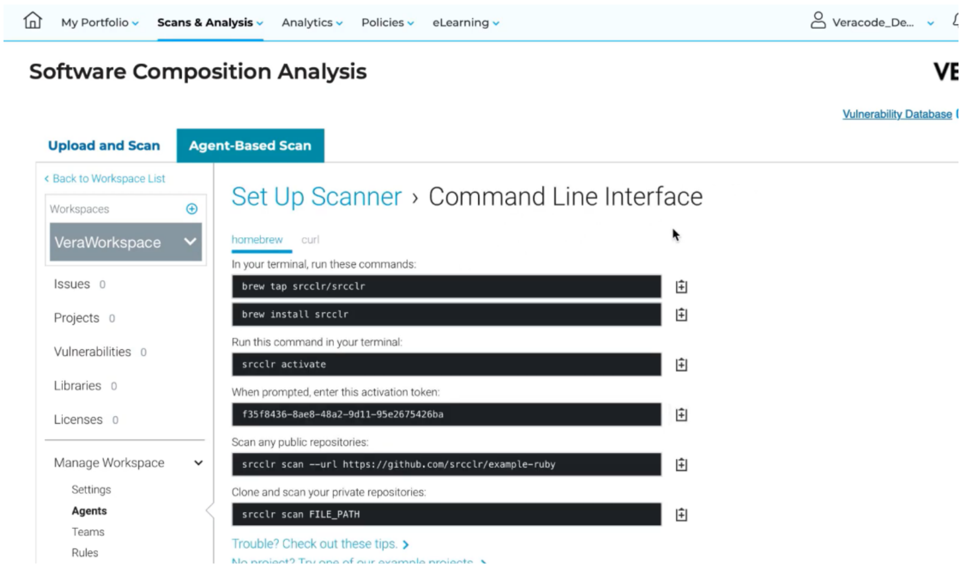
Last account activity on 11/26/19 3:06 PM EST from IP: 72.22.163.41 For use under U.S. Pat. Nos 9,672,355, 9,645,800, 9,405,906, 9,286,063, 9,207,920, 9,195,833, 8,365,155, 7,752,609, and 5,854,924, and patents pending.

Figure 5.21: The results from the SCA scan [153]

Agent-Based Scan Method

An agent-based scan can be done through the Veracode Platform (figure 5.20). The Veracode agent-based scan [154] is a program that builds and scans code to find third party libraries and the vulnerabilities in those libraries. To perform an agent-based scan, it is necessary to create a workspace with an agent to scan projects. The agent make it possible to scan projects and put the results in a specific workspace.

The agent is created with commands in the CLI as an administrator, together with a token (figure 5.22). After a scan is done the results will appear in in the command line as a JSON output (figure 5.23). They can also be seen in the Veracode Platform.



In your terminal, run these commands:

```
brew tap srcclr/srcclr
```

```
brew install srcclr
```

Run this command in your terminal:

```
srcclr activate
```

When prompted, enter this activation token:

```
f35f8436-8ae8-48a2-9d11-95e2675426ba
```

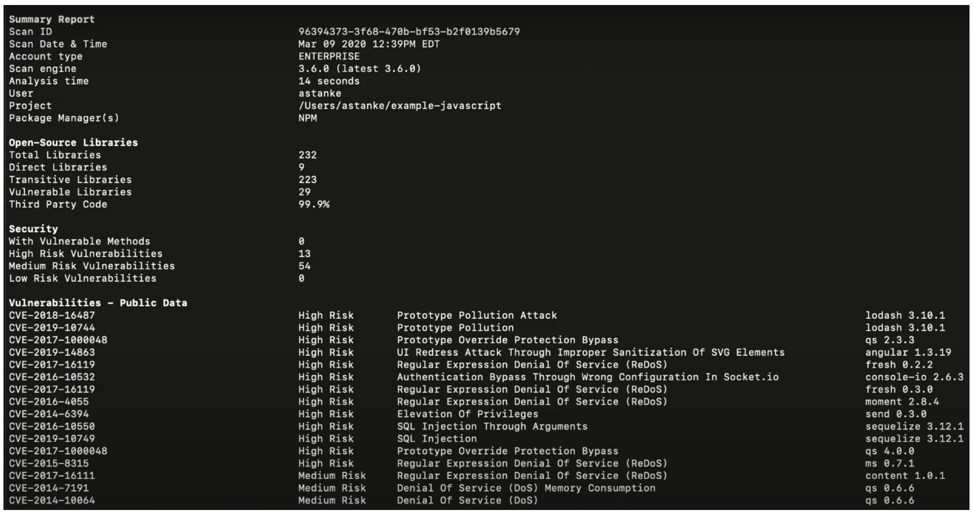
Scan any public repositories:

```
srcclr scan --url https://github.com/srcclr/example-ruby
```

Clone and scan your private repositories:

```
srcclr scan FILE_PATH
```

Figure 5.22: The commands needed to create an agent in the CLI [154]



```
Summary Report
Scan ID                                               96394373-3f68-470b-bf53-b2f0139b5679
Scan Date & Time                                      Mar 09 2020 12:39PM EDT
Account type                                          ENTERPRISE
Scan engine                                           3.6.0 (latest 3.6.0)
Analysis time                                         14 seconds
User                                                  astanke
Project                                               /Users/astanke/example-javascript
Package Manager(s)                                    NPM

Open-Source Libraries
Total Libraries                                       232
Direct Libraries                                      9
Transitive Libraries                                  223
Vulnerable Libraries                                  29
Third Party Code                                      99.9%

Security
With Vulnerable Methods                               0
High Risk Vulnerabilities                             13
Medium Risk Vulnerabilities                           54
Low Risk Vulnerabilities                              0

Vulnerabilities - Public Data
CVE-2018-16487                                        High Risk      Prototype Pollution Attack                                                 lodash 3.10.1
CVE-2019-10744                                        High Risk      Prototype Pollution                                                        lodash 3.10.1
CVE-2017-1000048                                      High Risk      Prototype Override Protection Bypass                                       qs 2.3.3
CVE-2019-14863                                        High Risk      UI Redress Attack Through Improper Sanitization Of SVG Elements            angular 1.3.19
CVE-2017-16119                                        High Risk      Regular Expression Denial Of Service (ReDoS)                               fresh 0.2.2
CVE-2016-10532                                        High Risk      Authentication Bypass Through Wrong Configuration In Socket.io             console-io 2.6.3
CVE-2017-16119                                        High Risk      Regular Expression Denial Of Service (ReDoS)                               fresh 0.3.0
CVE-2016-4055                                         High Risk      Regular Expression Denial Of Service (ReDoS)                               moment 2.8.4
CVE-2014-6394                                         High Risk      Elevation Of Privileges                                                    send 0.3.0
CVE-2016-10550                                        High Risk      SQL Injection Through Arguments                                            sequelize 3.12.1
CVE-2019-10749                                        High Risk      SQL Injection                                                              sequelize 3.12.1
CVE-2017-1000048                                      High Risk      Prototype Override Protection Bypass                                       qs 4.0.0
CVE-2015-8315                                         High Risk      Regular Expression Denial Of Service (ReDoS)                               ms 0.7.1
CVE-2017-16111                                        Medium Risk    Regular Expression Denial Of Service (ReDoS)                               content 1.0.1
CVE-2014-7191                                         Medium Risk    Denial Of Service (DoS) Memory Consumption                                 qs 0.6.6
CVE-2014-10064                                        Medium Risk    Denial Of Service (DoS)                                                    qs 0.6.6
```

Figure 5.23: The results from the scan in the CLI as a JSON output [154]

5.4.2.2 Veracode Dynamic Analysis (DAST)

The Dynamic Analysis workflow [155] for scanning web applications or API specifications consists of different steps to configure, execute, and view the results of the scan (figure 5.24).

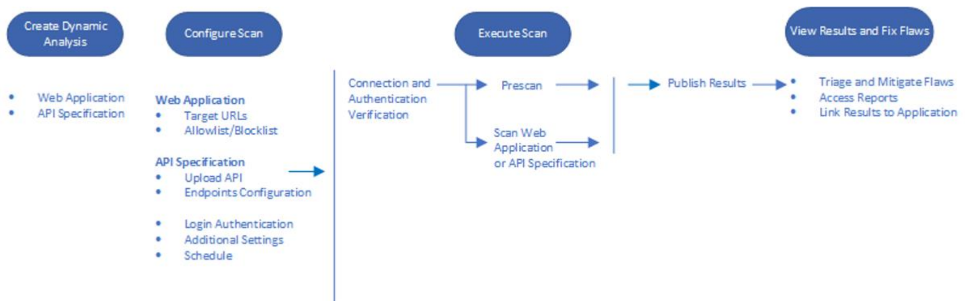


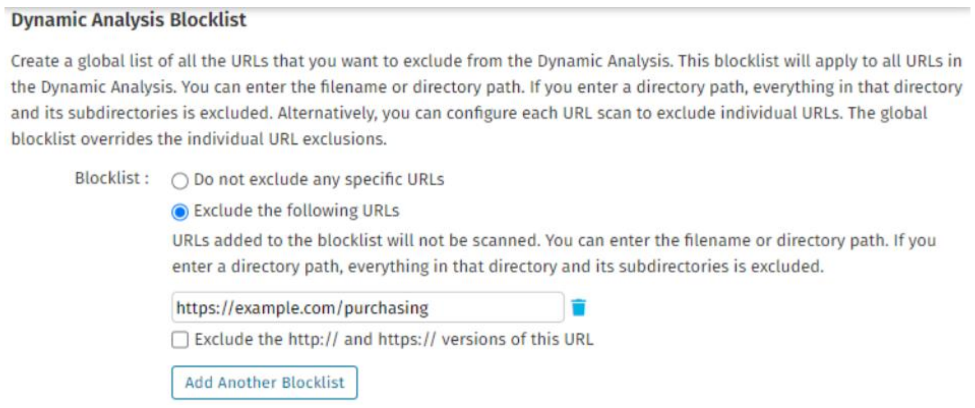
Figure 5.24: The Dynamic Analysis workflow [155]

Veracode Dynamic Analysis has several use cases. It is possible to use it to run an authenticated and unauthenticated dynamic analysis on a web application, as well as configuring a dynamic analysis of a web application for internal scanning. A more detailed description of the procedures in an authenticated dynamic analysis, and how the other use cases work, can be found in appendix C.2.

5.4.2.2.1 Authenticated Dynamic Analysis

To perform an authenticated dynamic analysis [156], login credentials are needed. After starting a scan of a web application, the URL of the website must be added together with the name of the dynamic analysis. Next, who is going to have access to the results need to be selected.

It is possible to create a Blocklist (figure 5.25) and an Allowlist (figure 5.26) with different URLs. URLs in the Blocklist will be excluded from the scan, while the ones in the Allowlist will be included to make sure that Veracode can scan the entire application.



Dynamic Analysis Blocklist

Create a global list of all the URLs that you want to exclude from the Dynamic Analysis. This blocklist will apply to all URLs in the Dynamic Analysis. You can enter the filename or directory path. If you enter a directory path, everything in that directory and its subdirectories is excluded. Alternatively, you can configure each URL scan to exclude individual URLs. The global blocklist overrides the individual URL exclusions.

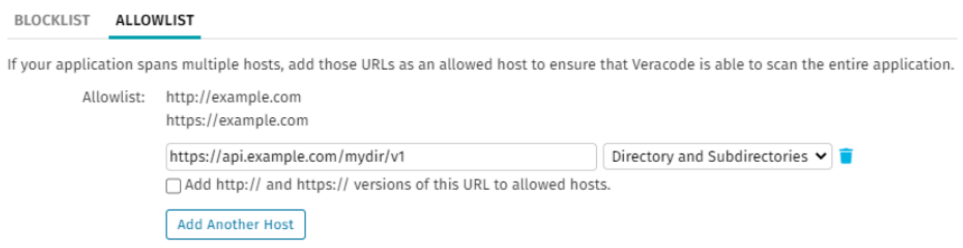
Blocklist : Do not exclude any specific URLs [✓ Exclude the following URLs]

URLs added to the blocklist will not be scanned. You can enter the filename or directory path. If you enter a directory path, everything in that directory and its subdirectories is excluded.

https://example.com/purchasing

☐ Exclude the http:// and https:// versions of this URL

Figure 5.25: The Blocklist [156]



If your application spans multiple hosts, add those URLs as an allowed host to ensure that Veracode is able to scan the entire application.

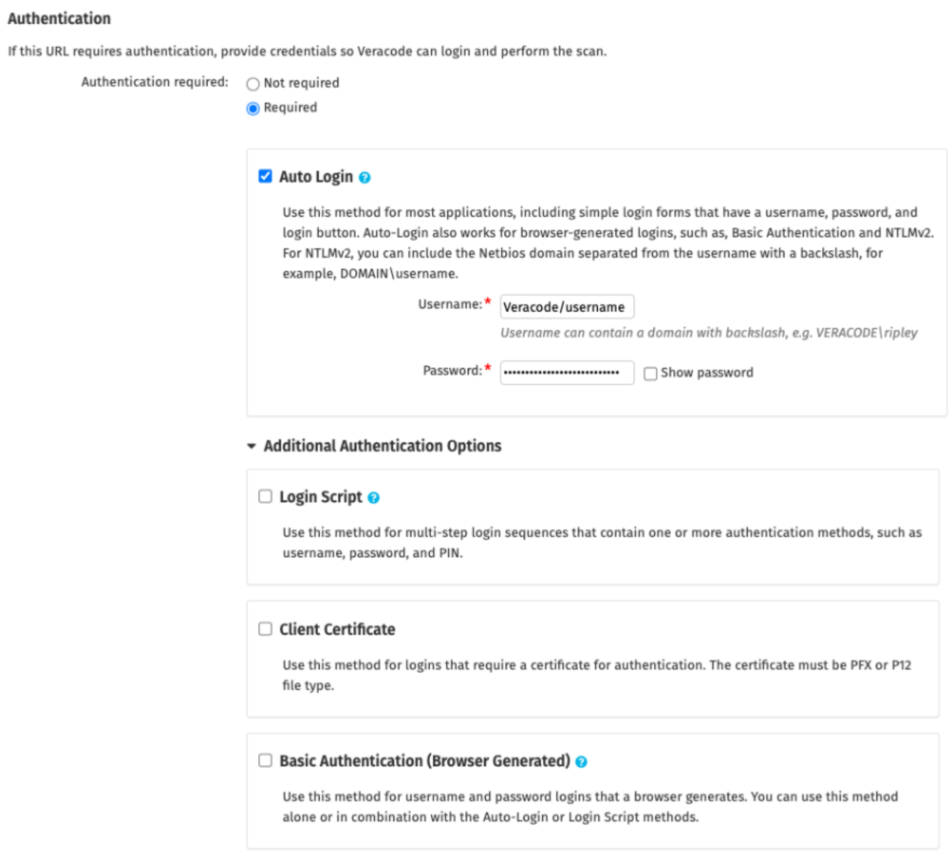
Allowlist: http://example.com
https://example.com

https://api.example.com/mydir/v1 | Directory and Subdirectories

☐ Add http:// and https:// versions of this URL to allowed hosts.

Figure 5.26: The Allowlist [156]

In the Authentication section, authentication credentials must be provided, which can be done in several ways (figure 5.27). Auto-login is selected by default.


Authentication

If this URL requires authentication, provide credentials so Veracode can login and perform the scan.

Authentication required: ○ Not required [✓ ● Required]

[✓] **Auto Login**

Use this method for most applications, including simple login forms that have a username, password, and login button. Auto-Login also works for browser-generated logins, such as, Basic Authentication and NTLMv2. For NTLMv2, you can include the Netbios domain separated from the username with a backslash, for example, DOMAIN\username.

Username:* Veracode/username

Username can contain a domain with backslash, e.g. VERACODE\ripley

Password:* •••••••••••••••••••••••••• ☐ Show password

Additional Authentication Options

☐ **Login Script**

Use this method for multi-step login sequences that contain one or more authentication methods, such as username, password, and PIN.

☐ **Client Certificate**

Use this method for logins that require a certificate for authentication. The certificate must be PFX or P12 file type.

☐ **Basic Authentication (Browser Generated)**

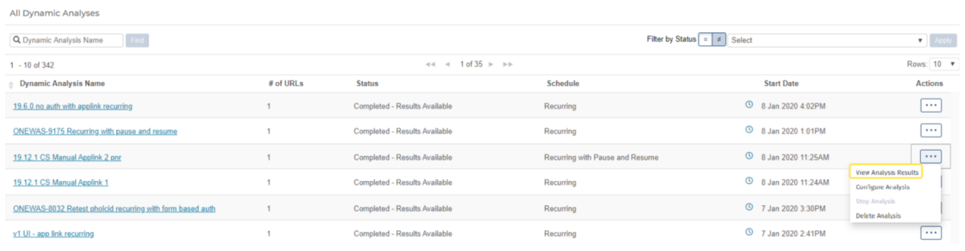
Use this method for username and password logins that a browser generates. You can use this method alone or in combination with the Auto-Login or Login Script methods.

Figure 5.27: Different ways to authenticate [156]

When it is ready to be run, the Dynamic Analysis can be run immediately, or it can be scheduled to be run at a date up to 90 days in the future. Dynamic Analysis is not scheduled by default.

The results from the scans and their status can be seen in the All Dynamic Analysis page, given that the necessary roles has been set, and that the user has permission to see them.

All the results will open in the Triage Flaws view of the selected Dynamic Analysis. They are listed with all the vulnerabilities found, together with details about each of them (figure 5.28).



Dynamic Analysis Name	# of URLs	Status	Schedule	Start Date	Actions
19.6.0 no auth with applink recurring	1	Completed - Results Available	Recurring	8 Jan 2020 4:02PM	···
ONEWAS-9175 Recurring with pause and resume	1	Completed - Results Available	Recurring	8 Jan 2020 1:01PM	···
19.12.1 CS Manual Applink 2 pnr	1	Completed - Results Available	Recurring with Pause and Resume	8 Jan 2020 11:25AM	···
19.12.1 CS Manual Applink 1	1	Completed - Results Available	Recurring	8 Jan 2020 11:24AM	
ONEWAS-8032 Retest pholcid recurring with form based auth	1	Completed - Results Available	Recurring	7 Jan 2020 3:30PM	
v1 UI - app link recurring	1	Completed - Results Available	Recurring	7 Jan 2020 2:41PM	···

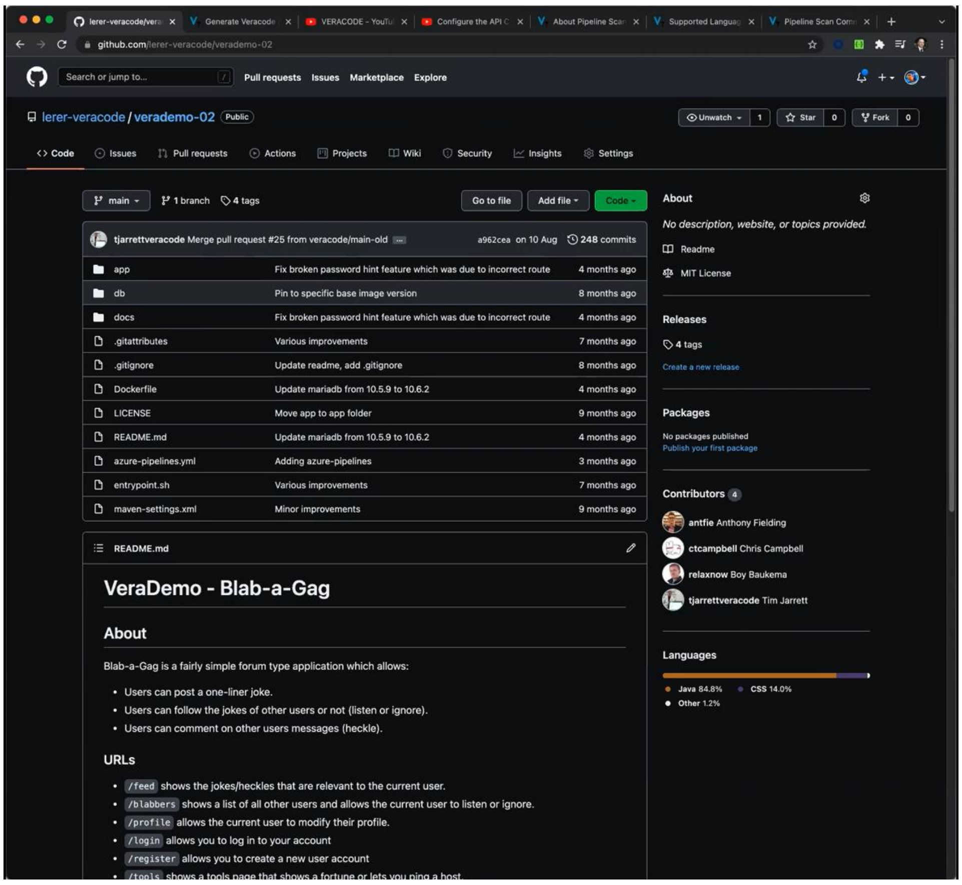
Figure 5.28: The results from the test [156]

5.4.2.3 Veracode Static Analysis (SAST)

Veracode Static Analysis can be used in several ways. It can be integrated with GitHub, where the code can be scanned directly in the repository, it can be used together with an IDE, where code can be scanned at same time as it is being written, and it can be used from the Veracode Platform where all the other scans can be seen. All these solutions should show the same results. A more detailed description of the procedures with the GitHub integration and how the other methods work, can be found in appendix C.3.

5.4.2.3.1 GitHub

Veracode pipeline scan can be integrated into GitHub as a workflow [157]. The Veracode pipeline scan directly embeds into team development pipelines and provides feedback on flaws introduced after new commits to the repository (figure 5.29). To perform a scan, an API key and a secret key is needed to submit the code for scanning. Veracode provides documentation on how to set this up [158]. After these keys are generated, they can be added to the secrets in the GitHub repository.



tjarrettveracode Merge pull request #25 from veracode/main-old ... a962cea on 10 Aug 248 commits

app	Fix broken password hint feature which was due to incorrect route	4 months ago
db	Pin to specific base image version	8 months ago
docs	Fix broken password hint feature which was due to incorrect route	4 months ago
.gitattributes	Various improvements	7 months ago
.gitignore	Update readme, add .gitignore	8 months ago
Dockerfile	Update mariadb from 10.5.9 to 10.6.2	4 months ago
LICENSE	Move app to app folder	9 months ago
README.md	Update mariadb from 10.5.9 to 10.6.2	4 months ago
azure-pipelines.yml	Adding azure-pipelines	3 months ago
entrypoint.sh	Various improvements	7 months ago
maven-settings.xml	Minor improvements	9 months ago

README.md

VeraDemo - Blab-a-Gag

About

Blab-a-Gag is a fairly simple forum type application which allows:

- Users can post a one-liner joke.
- Users can follow the jokes of other users or not (listen or ignore).
- Users can comment on other users messages (heckle).

URLs

- `/feed` shows the jokes/heckles that are relevant to the current user.
- `/blabbers` shows a list of all other users and allows the current user to listen or ignore.
- `/profile` allows the current user to modify their profile.
- `/login` allows you to log in to your account
- `/register` allows you to create a new user account
- `/tools` shows a tools page that shows a fortune or lets you ping a host.

About

No description, website, or topics provided.

Readme

MIT License

Releases

4 tags

Create a new release

Packages

No packages published
Publish your first package

Contributors 4

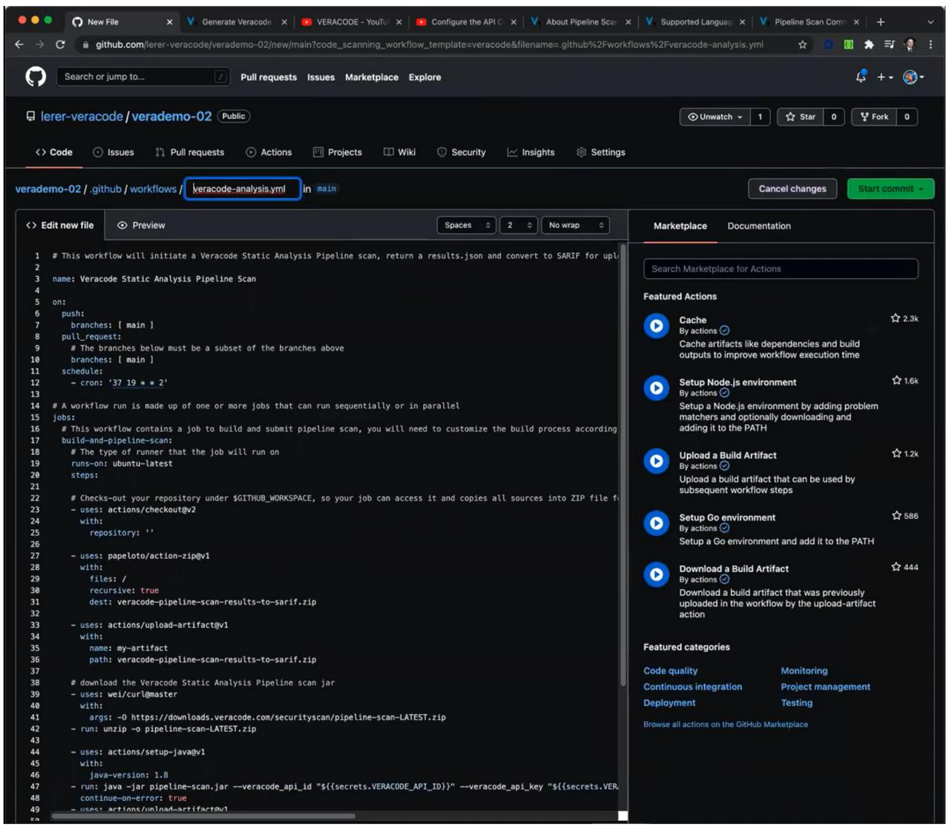
antfie Anthony Fielding
ctcampbell Chris Campbell
relaxnow Boy Baukema
tjarrettveracode Tim Jarrett

Languages

● Java 84.8% ● CSS 14.0%
● Other 1.2%

Figure 5.29: The GitHub repository that is going to be scanned [157]

To set everything up properly, Veracode provides a template. When this is added to the repository, it creates a new file called “veracode-analysis.yml” (figure 5.30).


veracode-demo-02 / .github / workflows / veracode-analysis.yml in main

```yaml
# This workflow will initiate a Veracode Static Analysis Pipeline scan, return a results.json and convert to SARIF for upl

name: Veracode Static Analysis Pipeline Scan

on:
  push:
    branches: [ main ]
  pull_request:
    # The branches below must be a subset of the branches above
    branches: [ main ]
  schedule:
    - cron: '37 19 * * 2'

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a job to build and submit pipeline scan, you will need to customize the build process according
  build-and-pipeline-scan:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest
    steps:

    # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it and copies all sources into ZIP file f
    - uses: actions/checkout@v2
      with:
        repository: ''

    - uses: papeloto/action-zip@v1
      with:
        files: /
        recursive: true
        dest: veracode-pipeline-scan-results-to-sarif.zip

    - uses: actions/upload-artifact@v1
      with:
        name: my-artifact
        path: veracode-pipeline-scan-results-to-sarif.zip

    # download the Veracode Static Analysis Pipeline scan jar
    - uses: wei/curl@master
      with:
        args: -O https://downloads.veracode.com/securityscan/pipeline-scan-LATEST.zip
    - run: unzip -o pipeline-scan-LATEST.zip

    - uses: actions/setup-java@v1
      with:
        java-version: 1.8
    - run: java -jar pipeline-scan.jar --veracode_api_id "${{secrets.VERACODE_API_ID}}" --veracode_api_key "${{secrets.VER
      continue-on-error: true
    - uses: actions/upload-artifact@v1
```

Featured Actions

Cache — By actions — Cache artifacts like dependencies and build outputs to improve workflow execution time

Setup Node.js environment — By actions — Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH

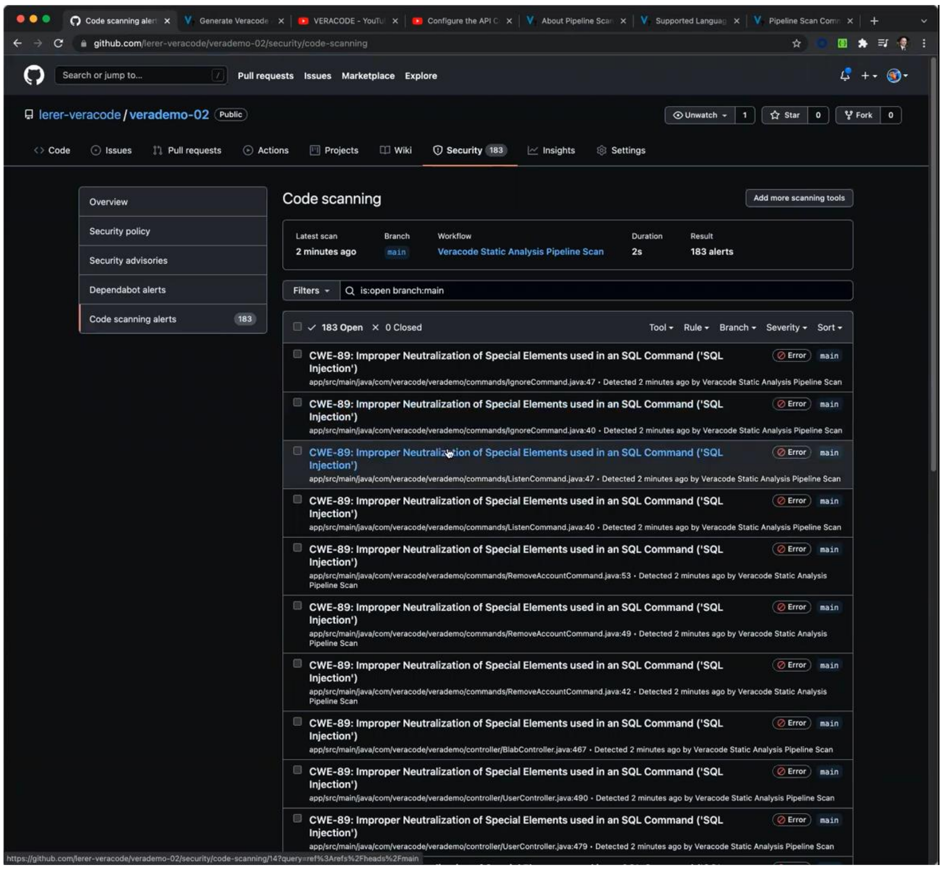
Upload a Build Artifact — By actions — Upload a build artifact that can be used by subsequent workflow steps

Setup Go environment — By actions — Setup a Go environment and add it to the PATH

Download a Build Artifact — By actions — Download a build artifact that was previously uploaded in the workflow by the upload-artifact action

Figure 5.30: The veracode-analysis.yml file [157]

When the scan is complete, it is possible to see all the found issues from the Veracode pipeline scan (figure 5.31). These issues contain information about the severity of the issue and where in the code it can be found. It also contain information about what to do to avoid it.



Code scanning

Latest scan: 2 minutes ago · Branch: main · Workflow: Veracode Static Analysis Pipeline Scan · Duration: 2s · Result: 183 alerts

is:open branch:main

183 Open · 0 Closed

```
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/commands/IgnoreCommand.java:47 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/commands/IgnoreCommand.java:40 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/commands/ListenCommand.java:47 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/commands/ListenCommand.java:40 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/commands/RemoveAccountCommand.java:53 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/commands/RemoveAccountCommand.java:49 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/commands/RemoveAccountCommand.java:42 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/controller/BlabController.java:467 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/controller/UserController.java:490 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')   Error  main
app/src/main/java/com/veracode/verademo/controller/UserController.java:479 • Detected 2 minutes ago by Veracode Static Analysis Pipeline Scan
```

https://github.com/lerer-veracode/verademo-02/security/code-scanning/14?query=ref%3Arefs%2Fheads%2Fmain

Figure 5.31: Overview of all the issues found in the scan [157]

5.5 Results

The table 5.1 shows an overview of the different results from the experiment. As shown, Contrast Security and Snyk got a big difference in found vulnerabilities when testing "FastTasks". A reason for this can be that they used different methods. Since "FastTasks" and the REST API are applications borrowed from another student at NTNU, the owner has become aware that there are vulnerabilities in them, and has given consent for it to be published (appendix I.5.3). The results found in the table are retrieved from figure 5.2, B.12, 5.6 and 5.11.

Table 5.1: Results from the tests

	Application	Method	High	Medium	Low
Contrast Security	"FastTasks"	IAST		1	
Snyk	"FastTasks"	SAST, SCA	14		
Snyk	prog2053-exam2020	SAST, SCA	2	1	190
StackHawk	REST API	DAST	1	6	6

Chapter 6

Discussion

6.1 Introduction

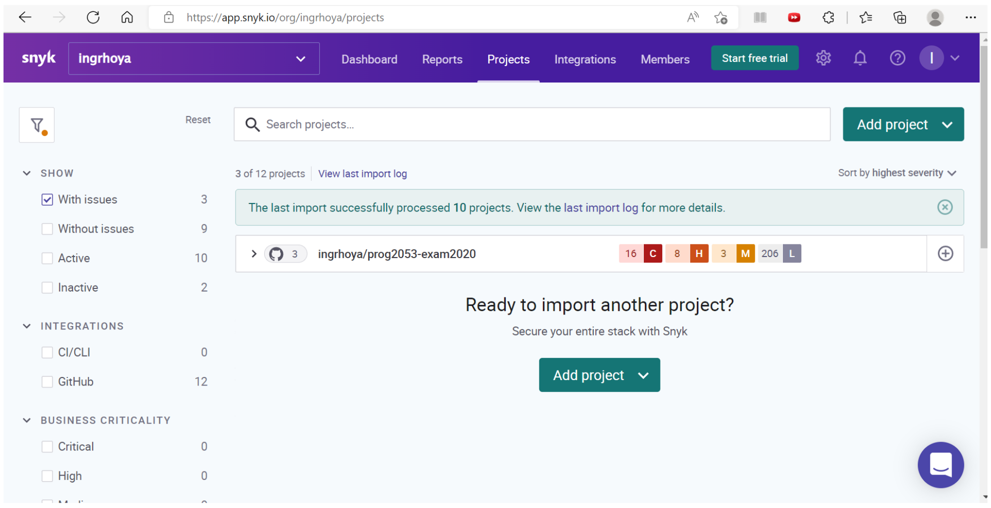
The discussion chapter consists of a detailed presentation of the group's process and findings. It also covers limitations during the thesis, changes that were made during the project, and a critique of the thesis.

6.2 Result Interpretation

Contrast Security found one medium vulnerability in "FastTasks", which was an insecure hashing algorithm. In the test, Contrast Security was supposed to test both SCA and IAST, but this did not happen. The test was done by following the instructions in the guide they gave, but when the test was finished, no libraries were found. This means that SCA was not tested. Contrast Security did not say it was needed, but after e-mail communication with Ivar Farup (H.18), it was discovered that Contrast Security most likely needs a specific file, a Project Object Model (POM) file, in order to find the libraries. Due to the lack of the POM file, the test became irrelevant, because it was the SCA test that was most interesting for this thesis, and not the IAST test. The reason why is mentioned in section 6.4.

Snyk found 14 vulnerabilities in "FastTasks" of high severity. They were ranked with a severity score of 922, and all of them were XSS vulnerabilities. This seems fair, as it potentially is a very damaging vulnerability, and there are no preventative measures in place. As can be seen in B.14, there were also some files that did not get tested by Snyk. This could be because of a file type or content that is not supported, or because the files are too small or too big [159]. Snyk states in its documentation that minified JavaScript files with three lines or less is too small for the scan, and single files larger than 1MB will be ignored.

Snyk was also tested on the "prog2053-exam2020" repository. In this test, two high, one medium and 190 low vulnerabilities were found. The results were varied, and some examples of vulnerabilities found were cross site scripting, code injection and server-side request forgery. This was expected, as part of the code for the exam has been used for many years, and it is not made to be secure. A few weeks after the first test was complete, Snyk ran a test on this repository over again. More vulnerabilities were found, the most surprising being 16 critical (figure 6.1). This shows how important continuous testing is; new vulnerabilities are bound to be discovered over time.



The last import successfully processed 10 projects. View the last import log for more details.

Ready to import another project?

Secure your entire stack with Snyk

Figure 6.1: The results from Snyk retesting the exam software after several weeks

StackHawk found six low, six medium and one high when testing with the REST API. The API used in the test was not made with security in mind, so it is likely that it has this many vulnerabilities, but it might also be possible that some of these alerts were false positives.

These differences in yielded results between the tools demonstrates the benefit to testing with several tools that have a different focus. The Snyk scan covered SAST and SCA, the Contrast scan covered IAST, and the StackHawk scan covered DAST. The test results show how including tests through all stages of the SDLC can reveal vulnerabilities that might not have been discovered after only testing in one SDLC phase.

6.3 Advantages and Disadvantages

To choose which tools to recommend for NDMA, the advantages and disadvantages of each tool are listed in table 6.1-6.7. The contents of the tables are arranged in prioritized order. This is based on what the thesis group consider to be most important for NDMA, such as ease of use and how much of the SDLC the tools cover.

Ease of use is prioritized due to developers feeling they do not have time to security test their software (I.4.2). If the tools are easy to use, it is more likely that the developers will be satisfied with the solutions, and start including security testing in their workflow.

NDMA emphasized early in the process that library testing was necessary to include (appendix I.2.1). This made SCA an important feature to include in the suggested solutions, even though this only covers a small part of the SDLC. Although SCA is important, the group realized that only testing libraries was not enough. To achieve defense in depth, the rest of the SDLC and how much of it the tools cover needs to be prioritized as well.

6.3.1 Contrast Security

Table 6.1: Advantages and disadvantages discovered about Contrast Security throughout the thesis period

Advantages	Disadvantages
Can be used with a wide variety of languages and environments.	To test SCA, the application needs to be compiled and run.
Their SCA product can be used on compiled code, letting it be used on both external and internal code.	To test SAST, the files in the software need to be uploaded manually each time the software is tested, or started manually in the CLI.
The SAST product can be used on internal code.	Has a lot of requirements for the application to be able to scan it properly.
Contrast Security can potentially cover IAST and RASP. In this way it will cover more of the SDLC.	Most of the programming languages supported only work with some of the products.

Table 6.2: Advantages and disadvantages discovered about Contrast Security throughout the thesis period

Advantages	Disadvantages
Contrast Security goes in depth on the used libraries and shows a lot of useful information, such as how many times the libraries are used in the code.	Lack of some important documentation. An example is the lack of information and the need for a POM file.
Uses data masking to avoid storing sensitive data.	Intended for use with Linux, works for Windows, but more cumbersome
Can get customized notifications about the scans sent to the users e-mail.	Hard to get in contact by e-mail when asking questions about the product.
	If the scan level is set to DEBUG or lower Contrast Security may store some sensitive data.
	Does not have similar products for learning such as Snyk and Veracode.
	Contrast Security shows different product plans on the website, but requires an inquiry for pricing.

6.3.2 Snyk

Table 6.3: Advantages and disadvantages discovered about Snyk throughout the thesis period

Advantages	Disadvantages
Support for many programming languages and can be integrated with many environments.	When using GitHub, it can be difficult to add new projects after the initial project.
Gives the user an automatic fix to some of the vulnerabilities it finds.	Snyk does not provide DAST.
Monitors the applications and projects uploaded in the platform. Hence, it is not necessary to manually start a scan each time. How often it scans can be adjusted in the settings.	Some languages are not supported for certain integrations, such as C/C++ do not work with the GitHub integration.
The rulset Snyk uses for scans was discovered through the tests to be easy to change/manipulate. This can make it easier for NDMA to test for what they want.	
Easy to get in contact with Snyk and they are eager to answer any questions about their product and how to set it up.	
It provides SAST and SCA and thus help NDMA to security test internal or external code if it provides the source code.	
Snyk stores some data in order to conduct tests, but the data is encrypted before being transmitted, and code is removed from the network and logs within 24 hours.	

Table 6.4: Advantages and disadvantages discovered about Snyk throughout the thesis period

Advantages	Disadvantages
Gives notifications through e-mail or Slack when new vulnerabilities are discovered. Can scan a private repository in GitHub without paying for GitHub Advanced Security. Snyk offers training and help with technical setup with the Enterprise plan. It has Snyk Learn which can help developers get a better understanding of security and learn more about it. Offers a free trial with most of the functionality, so it can be tested without commitment. Descriptions and comparison of plans and pricing can be found easily through the website.	

6.3.3 StackHawk

Table 6.5: Advantages and disadvantages discovered about StackHawk throughout the thesis period

Advantages	Disadvantages
A security scan is automatically run on every pull request, so there is no need to start the scan each time.	StackHawk is only a DAST tool. This means that it only covers a small part late in the SDLC.
It is well documented and the documentation is clear and easy to navigate through. This makes it easier to find information about the product and how to set it up.	StackHawk only works for web or API applications.
Easy to get in contact with StackHawk and they are eager to answer any questions about their product and how to set it up.	To use StackHawk with the GitHub integration, the repository needs to be public, or GitHub Advanced Security needs to be paid for.
Adapted guide for conducting tests with PowerShell and bash.	StackHawk say they store data about the results form the tests, but they do not say what type of information or how much about the results they store.
StacHawk provides DAST and can help NDMA security test their compiled external or internal code.	Their privacy policy gives them a lot of freedom on what they are allowed to do with personal data.
Can easily get notifications about scans through Microsoft Teams or Slack.	Do not have similar products for learning such as Snyk and Veracode.

Table 6.6: Advantages and disadvantages discovered about StackHawk throughout the thesis period

Advantages	Disadvantages
It is a relatively new tool, but it is partnered with Snyk which has more experience and recognition. Gives the full version of StackHawk in a free 2 week trial. In this way NDMA can test the product in its entirety without having to pay for it. Pricing is easily available through the website.	

6.3.4 Veracode

Table 6.7: Advantages and disadvantages discovered about Veracode throughout the thesis period

Advantages	Disadvantages
Each Veracode product has different ways it can be used, so NDMA can choose the way that suits them best.	Hard to get in contact with them or receive follow-up to inquiries.
Veracode supports many different programming languages.	To use Veracode with the GitHub integration, the repository needs to be public, or GitHub Advanced Security needs to be paid for.
Veracode covers SAST, DAST and SCA, thus including the prioritized methods. When having all these options available from one supplier, it might be easier to handle.	It must be purchased to be able to use it.
Their SAST and SCA products work with both compiled and source code.	Pricing is not listed on the website.
It is a well established tool that has been around for a long time.	
Can get customized e-mail notifications about the scans Veracode has done.	
It has Veracode eLearning which will help developers get a better understanding of security and learn more about it.	

6.4 Other Security Options for Testing and Analysis

The focus of this thesis was on SCA, SAST and DAST, although RASP and IAST also are possible methods for securing software.

RASP was not included throughout the thesis because it did not fit the criteria of NDMA, and thus seemed irrelevant. RASP is not a tool to test the software for security issues during development or before reaching production, but rather a tool for protection against incoming attacks [32]. This does not cover NDMA's goal of finding a tool to test the security of a software.

IAST is more relevant than RASP, considering NDMA's goals, but it is new technology [28] which makes it difficult to recommend. New technology might have unknown vulnerabilities or issues which are not yet found, and might cause problems in the future for NDMA. Although this technology is not recommended in this thesis, it might be a tool to consider in the future.

6.5 Manual Testing Methods

Even though there are tools for automating software security testing, there should still be some sort of manual testing in the SDLC as well [7]. Automated testing is still limited, and it cannot always simulate the acts and thoughts of a real human being. Another reason why manual testing should be implemented is because of defense in depth.

Penetration testing is one type of manual testing that should preferably not be replaced by automatic testing. Even though there are tools for simulating attacks on software, these mainly simulate known attacks, and might not consider the architecture of the software, like a manual test would. A machine can carry out attacks without issues, but it cannot customize attacks for a specific software. Although this type of testing has not reached its peak yet, it can still be used for smaller projects and for defense in depth. With smaller projects it might perform just as well as a manual test [38]. On bigger projects it might not perform as well as humans, but it can still find gaps that were not detected during penetration testing.

Another form for testing that should preferably not be replaced by automatic testing, is fuzzing. This type of testing is more automated, and might be easier to implement than penetration testing [7]. Not testing for input validation in an application could lead to critical security breaches. With fuzzing it is possible to avoid multiple zero-day attacks, which might be difficult to find by using other testing techniques.

Since NDMA wanted solutions that were as automated as possible (appendix I.2.1), this was a main priority, making manual testing less prioritized. The thesis group realized that recommending manual testing could become difficult, considering results being dependent on the person conducting the manual tests. Manual tests are often done by experts, and it would be difficult for the thesis group to reach this level of quality. After looking more into the automated tools available, the thesis group concluded that the scope would become too overwhelming if it also included manual testing. As a result, the recommended solutions do not cover methods on how to perform manual tests together with the automated methods. If manual testing was taken more into consideration, the recommendations for possible solutions could have been different.

6.6 Containers

Using containers was suggested as an example of a possible solution, but the group decided to not recommend this to NDMA after looking into the process of making containers safe. Containers are great for providing portable, reusable, and automatable ways to package and run applications, and are often considered a safe tool to use in order to keep elements of the system safe. According to NIST SP 1800-190 [49], this is incorrect. A container solution can be used to make the environment safer, but it can also make it less secure. A lot of resources are needed in order to mitigate these risks. The thesis group found that this was not the solution to the issue at hand, considering that NDMA was looking for a way to make security more efficient.

6.7 Limitations

6.7.1 Research

An issue with the research was limited information from other people and business' experience with the tools. Since a lot of tools have not been on the market for long, most of the information was from the tool's own websites, which may contain biased information. The group tried to look at the tools with objectively, not based on who is better at marketing, but rather which tools seem like the best fit for NDMA.

6.7.2 Testing

During the thesis period, it was important to conduct as much research as possible on the different methods and tools. A way to find out more about the tools was to try and test them on applications. The issue with this process was that the group did not have access to the full versions of the tools, only free trials which were limited, if even possible.

To see how the tools worked without free trials, the group had to watch videos of other people testing them. This makes the information less reliable, as most videos were demonstrations made by the creators of the tools. It also made it difficult for the group to get an overview on how complicated implementation can be, and which obstacles might come up in the process.

The tools that did have free trials did not have all functions available, and it was difficult to find fitting test data to use. Different tools needed different environments and applications in order to conduct the tests. This caused the testing phase to take longer than first anticipated, and there was not as much time left to find suitable applications. Since the tools had different requirements, the comparison of the results was limited.

The tests used different kinds of software security testing methods on the applications that were tested, such as SAST, DAST, and SCA. The tools also use different scoring systems to evaluate the software. Thus, it is hard to accurately compare them and say something about which tools gave better results. This especially applies to comparison with Veracode, since it could not be tested by the thesis group, or with the software that was used to test the other tools.

6.7.3 Other

Towards the end of the period, the group had some issues with illness. This meant that there were some delays from the group's own deadlines for when things should be finished. It also affected the results, since multiple members of the group were sick during the testing period, the tests were not done as effectively as they could have been. This lead to less tests being done, and with a lower quality than wished for.

6.8 Critique of the Thesis

6.8.1 Documentation

The websites of the examined tools have different structures for their documentation, and contain different information. The documentation for the tools varied a lot in detail and thoroughness. An example where this is done differently is where the tools write about how they handle system and project data. Snyk covers this in detail for each product they have, which can be seen in section 4.3.3. On the other side, Veracode had their main focus on how the personal data of the users were handled. When requesting more details about how they handle data, the thesis group did not get an answer.

6.8.2 Testing

There were not a lot of tools being tested in this thesis. If more tools were tested, it would be easier to get an overview of which tools were worth recommending. The reason why there was a lack of tested tools was because few tools offered free trials.

Knowing about vulnerabilities that existed in the applications before testing them, would have made it easier to check the tests for false positives and negatives. With the results from the tests in this thesis, it might be difficult to evaluate the value of the alerts from the tools. Some tools might deliver a lot of false positives without the group knowing. A table of already known vulnerabilities compared to found vulnerabilities could have been made, which could help indicate which tool works best in numbers.

6.8.3 New Tools

Even though the group spent a lot of time in the literature study phase to find different tools to look further into, there were discovered new tools later in the project period. Some of these tools were looked into, but they did not offer free trials for educational or research purposes, or the tools already examined seemed like a better fit for NDMA. It was thus better to focus on the tools chosen from the beginning.

6.8.4 Changes

Towards the end of the project period, there were discovered different changes in the selected tools. Some changes resulted in the group choosing not to have them as a suggested solution anymore, while other changes were not significant enough to cause that reaction. Some changes were also improvements, which caused the group to be more likely to want to recommend those tools.

Since there are changes being made often, it is important to take into consideration that the information about the other tools listed in section 3.3.1 may be outdated.

6.8.4.1 Invicti Security

Originally, the group had Invicti Security as one of the suggested solutions for NDMA to use. The main reason why it was not chosen for further evaluation, was because they did not offer a free trial for research purposes (H.2), which means it would not be possible for the group to test the tool. Even though the tool had good documentation and was supported by many well known companies, it was better to focus on tools that could be tested.

Another reason was that after looking further into this tool, the group found out that they had done some changes halfway into the project period [160]. When the group discovered the tool at the beginning, Invicti Security originally consisted of both Acunetix and Netsparker. Later Netsparker was named Invicti, while Acunetix stayed the same. This indicates that Invicti might be changing in an organizational manner, making it difficult to have an overview of the tools.

6.8.4.2 Contrast Security

When Contrast Security was further examined at the beginning of the project period, it did not support many programming languages for the relevant products it offered, only Java and .NET. These products were Contrast Scan and Contrast SCA. At the end of the period, in the middle of April, they did some improvements and added more supported programming languages for the products. Now it is possible to use JavaScript, GO, Scala, PHP, and Kotlin for these products as well.

6.8.4.3 Snyk and StackHawk

The group chose to look at Snyk and StackHawk as two different potential methods for NDMA, since Snyk covers SAST and could be used on internal software, while StackHawk covers DAST that mainly could be used on external software. Towards the end of the project period, StackHawk released that it now can be integrated with Snyk [161]. This integration will correlate findings from StackHawk's DAST tool with Snyk's SAST tool. With this, it will be easier for developers to work in a more efficient way to find vulnerabilities in their code and application, because they will get an understanding of what security issues exist. Instead of having two different user interfaces to work with, everything can be in one place.

Chapter 7

Conclusion

7.1 Introduction

The conclusion chapter consists of reflections around what has been done, evaluation of the group's working process, what can theoretically be done further, as well as a conclusion of the thesis.

7.2 Reflections

7.2.1 Software From NDMA

After researching the tools, it turned out that testing NDMA's software would be problematic. Snyk needed the source code in order to test the software, as well as there being a limit on how many files could be tested with the free trial. Contrast Security could be tested with Java projects if the project was in a .jar file format. Since the software from NDMA had a .exe format, it was not possible to test it with this tool. The software was tested with StackHawk, but it did not find any vulnerabilities, which made it irrelevant to include in the thesis. When starting a scan, the type of application has to be chosen. Since the thesis group do not know enough about the application, the "Other" option had to be chosen. Thus it was not possible to specify how StackHawk should move through the application, and it was better to use the software provided by another student.

7.2.2 Interviews

When conducting the interviews with the developers in NDMA, not all the questions were asked. The questions the group prepared were at times very specific. This was a conscious choice to ensure that the group got all answers needed. Since the questions were specific, there were many questions. When the developers were good at answering in a complementary way, many of the questions became irrelevant and the group did not have time to ask them all.

Something the group could have done to get through all the questions was to interrupt them, but there was no need to do that when they talked about relevant information. The thesis group chose instead to let them talk, and rather discard some of the less relevant questions. The questions that were not asked, are not included in the interview guide (appendix D), due to their lack of relevance.

7.2.3 Customer Persuasion Meeting

In a meeting with Tom Røise (appendix I.5.2), it was discussed how to convince leaders and developers in NDMA to invest in and start using one or more of the recommended tools. It was emphasized that it is important that they see the benefit of using the tools in connection with how they work. Instead of having this as a separate section, the group decided to add this information in section 6.3, because the advantages and disadvantages of each tool can help decide if the tool is worth spending money and time on. It is natural that this part will be emphasized, and it is thus important to see this in context with how it will benefit NDMA, and how the developers work.

7.2.4 Getting Acquainted With New Tools

The group experienced that it was difficult to get acquainted with unfamiliar tools, and how to find a way to use the results. After e-mail communication and interviews with external companies (appendix H.9, H.10, I.5.1), it turned out that they also had this experience. It is thus necessary to set aside some time to learn how the tools work properly, to get the most out of them.

7.2.5 Evaluation of the Working Process

7.2.5.1 Halfway Assessment

The group finished the project plan early, but because of some changes and specifications in the scope of the project, it was delivered later than first expected, but still within the deadline.

Writing the thesis was planned to be done throughout the entire project period, and the progress was good at the halfway point. There were some discussions about thesis structure, but the group ended up in agreement.

The group started on the literature study according to the Gantt. It was supposed to end after week 9, but it continued further throughout the project, due to new research being necessary later on in the project. Together with the literature study, the group started to explore various tools for NDMA to use. This was done within the time frame that was set in the Gantt. The group reached the milestone about choosing which tools to look further into. This was done before starting to explore each of them in week 8. The group started testing the selected tools around the planned starting time. The first test on Snyk was successful, and was finished swiftly. Other tests were somewhat delayed, since troubleshooting became necessary.

The group had multiple interviews during the project, which were done within the time frame planned out. Multiple requests for interviews were sent out, but not everyone responded. This lead to fewer interviews than expected. During the interviews, the group did not always get to ask all the prepared questions, due to lack of time and long discussions about the different topics. Even then, a lot of valuable information was collected to use further during the project.

7.2.5.2 Gantt

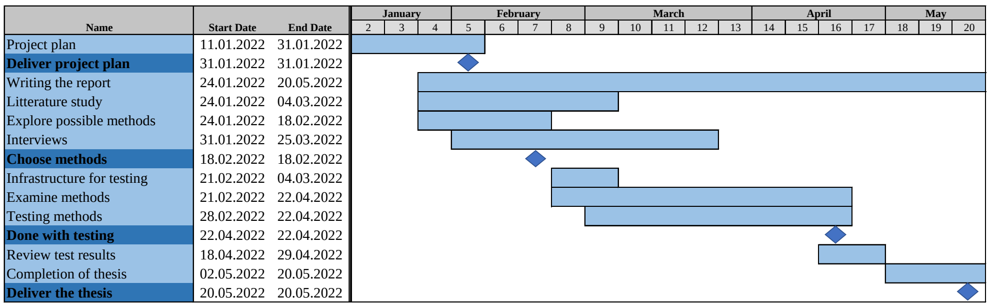
The group was not able to follow the original Gantt, and there were some changes during the project period.

The literature study ended up covering almost the whole project period. This happened because the group found out that it was necessary to do research throughout the whole project, in order to cover all relevant theory. After examining tools, more technologies were found to be relevant, and some of these would be preferred to be covered in the theory.

Another change in the Gantt was that the testing period ended up being longer than planned, due to multiple circumstances. The group found out that more information on the specific tools, and how to use them, were necessary in order to conduct the tests. Another issue was that the test data NDMA wanted the group to use, came late and was not possible to use in the tests. This caused the testing to be paused a few weeks before it could be continued. The group had a slightly too optimistic view on how much testing that could be done, and did not take into account that each tool had different requirements for them to be tested.

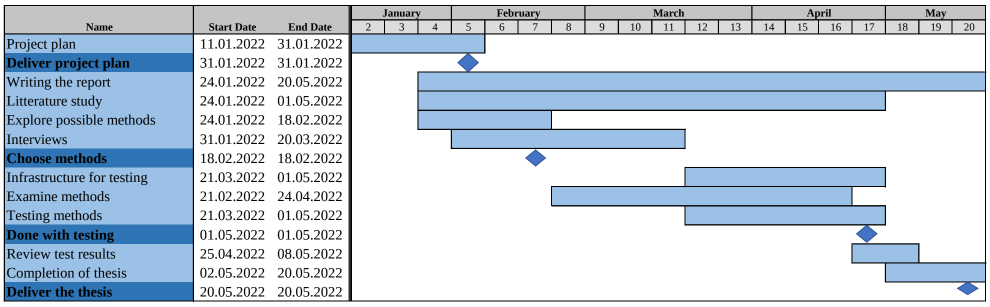
Setting up the infrastructure for testing was also pushed back. The reason why was that more information about each tool was needed in order to know which environment the group would have to set up. This ended up being an activity which were done at the same time as the testing, not something done beforehand like originally planed. With setting up the environment while testing, the group only set up relevant environments, which made the process more efficient.

With the test period being pushed back, it became difficult to reach the milestone of finishing testing by week 16, so this milestone was changed to be in week 17.



Name	Start Date	End Date
Project plan	11.01.2022	31.01.2022
Deliver project plan	31.01.2022	31.01.2022
Writing the report	24.01.2022	20.05.2022
Litterature study	24.01.2022	04.03.2022
Explore possible methods	24.01.2022	18.02.2022
Interviews	31.01.2022	25.03.2022
Choose methods	18.02.2022	18.02.2022
Infrastructure for testing	21.02.2022	04.03.2022
Examine methods	21.02.2022	22.04.2022
Testing methods	28.02.2022	22.04.2022
Done with testing	22.04.2022	22.04.2022
Review test results	18.04.2022	29.04.2022
Completion of thesis	02.05.2022	20.05.2022
Deliver the thesis	20.05.2022	20.05.2022

Figure 7.1: The original Gantt



Name	Start Date	End Date
Project plan	11.01.2022	31.01.2022
Deliver project plan	31.01.2022	31.01.2022
Writing the report	24.01.2022	20.05.2022
Litterature study	24.01.2022	01.05.2022
Explore possible methods	24.01.2022	18.02.2022
Interviews	31.01.2022	20.03.2022
Choose methods	18.02.2022	18.02.2022
Infrastructure for testing	21.03.2022	01.05.2022
Examine methods	21.02.2022	24.04.2022
Testing methods	21.03.2022	01.05.2022
Done with testing	01.05.2022	01.05.2022
Review test results	25.04.2022	08.05.2022
Completion of thesis	02.05.2022	20.05.2022
Deliver the thesis	20.05.2022	20.05.2022

Figure 7.2: The actual Gantt

7.2.5.3 System Development Model

The group mostly followed the system development model that was originally planned. The model shows that the interviews were to be done at the same time as the group worked on doing research on different tools (figure 1.1). In this way, the group would have become acquainted with more possible tools earlier, and it would have been time to take a closer look at them. Instead, some of the interviews were conducted after the group had decided on which tools to examine further, due to the fact that it was difficult to find a time that suited everyone. The group also had an extra subject in addition to the bachelor thesis that sometimes had to be prioritized. It was therefore difficult in terms of time to examine more tools.

7.2.5.4 Work Distribution

The work was evenly distributed among the group members. Everyone has contributed to what should be included in the theory, everyone has worked on examining all the recommended tools, and everyone has tried to test the tools. The group worked together to finish writing the report.

7.2.5.5 Goals

Goal M1, *increased awareness of security during software development,* might be achieved. If they read the thesis, they will hopefully think more about security and the consequences of what may happen if they do not have it in mind.

Goal M2, *NDMA are happy with the methods that are proposed and use these over time*, might be achieved. It is uncertain whether the leaders of NDMA see the benefit of the tools, and if they are willing to invest in them. Granted they do, it is furthermore dependent on the developers having a positive experience with the tools, and wanting to continue to use them.

Goal M3, *reduce the time it takes to check for vulnerabilities in the software*, is a goal that is partly achieved if NDMA decides to use some of the recommended tools. In the beginning, the developers might use more time than usual because they will have to learn how to use the tools. They will most likely also use more time on going through and remediating the test results. However, after a while when the developers have learned how to use the tools properly, they might spend less time conducting security checks. If they start using the tools, they are likely to become less vulnerable to potential security breaches and attacks.

Goal M4, *reduce the number of people that need to check for vulnerabilities in the software*, was not relevant. The group fixated on the fact that NDMA talked about being three people looking through the code together as a way of security testing software, and wanted to be able to say that the tests would need less resources like people. In reality these automated security tests will make all developers test software, but in a more efficient way.

Goal M5, *find tools and methods to make the security checks for the NDMA more effective, and automate this if possible*, was achieved. The group was able to find tools that could help NDMA improve the software security testing methods. This includes tools that can be used on both internal and external software.

Goal M6, *NDMA start using the tools and methods the group has come up with*, might be achieved. It is uncertain if NDMA will use the recommended tools.

Goal M7, *create a systematic overview of potential tools that can be used for software control*, was achieved. All tools contain information about why they were chosen as a recommended tool for NDMA, together with information about what the tools are, how they work, which products are included in each tool, and where in the SDLC they can be used (figure 4.5). Since advantages and disadvantages with each tool are also covered, it is easier for NDMA to see which tools that may best suit their needs.

In retrospect the group saw that it was not possible to evaluate some of the effect goals. The conclusion has to be delivered to NDMA before it can be known if these goals were met. The rest of the goals were either irrelevant or achieved.

7.3 Further Work

In further work, it would be a good idea to test more tools found during the thesis (appendix I.5.1), as well as further testing the tools in their full version. Another idea could be to use one single application, where the vulnerabilities are already known, on all the tools. By doing this, checking for false positives/negatives, as well as the number of real vulnerabilities, could be possible.

To further evaluate if any of the tools were good recommendations, it would be a good idea to compare manual testing to automated testing. The theory states that manual tests are better, but it would be interesting to see how the tools compare to manual tests and if any of them can do better.

7.4 Conclusion

Based on the findings, Veracode, or Snyk with StackHawk could be excellent options for NDMA. They cover a broad range of the SDLC and are easy to use, as shown in table 6.3, 6.5, and 6.7. Contrast Security is also a possible solution, but the group's experience was that it was harder to set up and use, shown in 6.1. Regardless, the thesis group hope the results of the thesis will be of some value to NDMA.

Bibliography

[1] J. Børresen. 'Forsvaret.' (11th Jan. 2022), [Online]. Available: https://snl.no/Forsvaret (visited on 18/01/2022).

[2] Forsvarsmateriell. 'Organisasjon og ledelse.' (), [Online]. Available: https://www.fma.no/om-oss/organisasjon-og-ledelse (visited on 21/01/2022).

[3] Forsvaret. 'Organisasjon.' (), [Online]. Available: https://www.forsvaret.no/om-forsvaret/organisasjon (visited on 18/01/2022).

[4] Forsvarsmateriell. 'Eierskapsforvaltning.' (), [Online]. Available: https://www.fma.no/ (visited on 19/01/2022).

[5] Forsvarsmateriell. 'Anskaffelser og investeringer.' (), [Online]. Available: https://www.fma.no/anskaffelser/anskaffe-materiell (visited on 20/01/2022).

[6] H. S. Department. 'Strategies for qualitative interviews.' (), [Online]. Available: https://sociology.fas.harvard.edu/files/sociology/files/interview_strategies.pdf (visited on 25/04/2022).

[7] M. Paul, *Official (ISC) Guide to the CSSLP CK*. Taylor & Francis Group, 2014.

[8] K. Dempsey, N. S. Chawla, A. Johnson, R. Johnston, A. C. Jones, A. Orebaugh, M. Scholl and K. Stine. 'Information security.' (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-137.pdf (visited on 11/05/2022).

[9] V. C. Hu, R. Kuhn and D. Yaga. 'Verification and test methods for access control policies/models.' (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-192.pdf (visited on 11/05/2022).

[10] imperva. 'White box testing.' (), [Online]. Available: https://www.imperva.com/learn/application-security/white-box-testing/ (visited on 15/03/2022).

[11] imperva. 'Black box testing.' (), [Online]. Available: https://www.imperva.com/learn/application-security/black-box-testing/ (visited on 15/03/2022).

[12] J. T. Force. ‘Assessing security and privacy controls in information systems and organizations.’ (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53Ar5.pdf (visited on 11/05/2022).

[13] imperva. ‘Gray box testing.’ (), [Online]. Available: https://www.imperva.com/learn/application-security/gray-box-testing/ (visited on 15/03/2022).

[14] Snyk. ‘5 application security assessment steps.’ (), [Online]. Available: https://snyk.io/learn/application-security/assessment/ (visited on 16/02/2022).

[15] tutorialspoint. ‘Sdlc - overview.’ (), [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm (visited on 18/02/2022).

[16] Veracode. ‘Application security assessment.’ (), [Online]. Available: https://www.veracode.com/security/application-security-assessment (visited on 16/02/2022).

[17] Devopedia. ‘Shift left.’ (15th Feb. 2022), [Online]. Available: https://devopedia.org/shift-left (visited on 18/02/2022).

[18] K. Scarfone, M. Souppaya, A. Cody and A. Orebaugh. ‘Technical guide to information security testing and assessment.’ (Sep. 2008), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf (visited on 11/05/2022).

[19] NTT. ‘False positive.’ (), [Online]. Available: https://www.whitehatsec.com/glossary/content/false-positive (visited on 04/03/2022).

[20] M. Souppaya and K. Scarfone. ‘Guide to malware incident prevention and handling for desktops and laptops.’ (Jul. 2013), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-83r1.pdf (visited on 11/05/2022).

[21] datto. ‘Datto.’ (), [Online]. Available: https://www.datto.com/ (visited on 04/03/2022).

[22] J. Peterson. ‘Application security testing: Security scanning vs. runtime protection.’ (), [Online]. Available: https://www.whitesourcesoftware.com/resources/blog/ast-application-security-testing/ (visited on 25/02/2022).

[23] E. Katz. ‘Top 10 static application security testing (sast) tools in 2021.’ (8th Sep. 2021), [Online]. Available: https://spectralops.io/blog/top-10-static-application-security-testing-sast-tools-in-2021/ (visited on 10/02/2022).

[24] O. Harris. 'What do sast, dast, iast and rasp mean to developers?' (), [Online]. Available: https://www.softwaresecured.com/what-do-sast-dast-iast-and-rasp-mean-to-developers/ (visited on 10/02/2022).

[25] Synopsys. 'Static application scurity testing.' (), [Online]. Available: https://www.synopsys.com/glossary/what-is-sast.html (visited on 10/02/2022).

[26] A. Phadke. 'Sast vs. dast: What's the best method for application security testing?' (), [Online]. Available: https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference/ (visited on 10/02/2022).

[27] M. Focus. 'What is dynamic application security testing (dast)?' (), [Online]. Available: https://www.microfocus.com/en-us/what-is/dast (visited on 25/02/2022).

[28] positive technologies. 'Sast, dast, iast, and rasp: How to choose?' (2nd Aug. 2019), [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/knowledge-base/sast-dast-iast-and-rasp-how-to-choose/ (visited on 25/02/2022).

[29] Snyk. 'Interactive application security testing (iast).' (), [Online]. Available: https://snyk.io/learn/application-security/iast-interactive-application-security-testing/ (visited on 25/02/2022).

[30] Y. Pan. 'Interactive application security testing.' (2019), [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8901378/authors#authors (visited on 16/05/2022).

[31] R. Hat. 'What is a ci/cd pipeline?' (8th Jan. 2019), [Online]. Available: https://www.redhat.com/en/topics/devops/what-cicd-pipeline (visited on 25/02/2022).

[32] S. T. Help. 'What is rasp.' (3rd Feb. 2022), [Online]. Available: https://www.softwaretestinghelp.com/differences-between-sast-dast-iast-and-rasp/#What_Is_RASP (visited on 11/02/2022).

[33] Synopsys. 'Software composition analysis.' (), [Online]. Available: https://www.synopsys.com/glossary/what-is-software-composition-analysis.html (visited on 25/02/2022).

[34] D. Contributor. 'The cost of not building with security in mind.' (14th Apr. 2016), [Online]. Available: https://devops.com/cost-not-building-software-security-mind/ (visited on 02/03/2022).

[35] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio and W. Lee, 'Towards measuring supply chain attacks on package managers for interpreted languages,' Ph.D. dissertation, Georgia Institute of Technology, 2021.

[36] testim. 'Test environment guide.' (7th Nov. 2019), [Online]. Available: https://www.testim.io/blog/test-environment-guide/ (visited on 11/03/2022).

[37] A. Singhal, T. Winograd and S. Karen. 'Guide to secure web services.' (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf (visited on 11/05/2022).

[38] Veracode. 'Veracode manual penetration testing.' (), [Online]. Available: https://www.veracode.com/products/penetration-testing (visited on 11/03/2022).

[39] Synopsys. 'Open web application security project top 10 (owasp top 10).' (), [Online]. Available: https://www.synopsys.com/glossary/what-is-owasp-top-10.html (visited on 21/03/2022).

[40] A. Johnson, K. Dempsey, R. Ross, S. Gupta and D. Bailey. 'Guide for security-focused configuration management of information systems.' (Aug. 2011), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-128.pdf (visited on 12/05/2022).

[41] First. 'Common vulnerability scoring system sig.' (), [Online]. Available: https://www.first.org/cvss/ (visited on 12/05/2022).

[42] Snyk. 'How is a vulnerability's severity determined?' (), [Online]. Available: https://support.snyk.io/hc/en-us/articles/360001040078-How-is-a-vulnerability-s-severity-determined- (visited on 11/05/2022).

[43] D. Walter, S. Quinn, H. Booth, K. Scarfone and D. Prisaca. 'The technical specification for the security content automation protocol (scap).' (Feb. 2018), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-126r3.pdf (visited on 12/05/2022).

[44] D. Miessler. 'The difference between cwe and cve.' (17th Dec. 2019), [Online]. Available: https://danielmiessler.com/blog/difference-cve-cwe/ (visited on 12/05/2022).

[45] M. Kaczorowski. 'Using cwe and cvss scores to get more context on a security advisory.' (9th Feb. 2021), [Online]. Available: https://github.blog/2021-02-09-using-cwe-and-cvss-scores-to-get-more-context-on-a-security-advisory/ (visited on 12/05/2022).

[46] K. Dempsey, P. Eavy, G. Moore and E. Takamura. 'Automation support for security control assessments: Software vulnerability management.' (Apr. 2020), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8011-4.pdf (visited on 12/05/2022).

[47] Mitre. '2021 cwe most important hardware weaknesses.' (5th May 2022), [Online]. Available: https://cwe.mitre.org/ (visited on 12/05/2022).

[48] OWASP. 'Owasp risk rating methodology.' (), [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology (visited on 11/05/2022).

[49] M. Souppaya, J. Morello and K. Scarfone. 'Application container security guide.' (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf (visited on 11/05/2022).

[50] T. A. Limoncelli, S. R. Chalup and C. J. Hogan, *The Practice of Cloud System Administration*. Addison-Wesley Educational Publishers Inc, 2015.

[51] M. Mafy. 'Containers are inherently secure: Reality or myth?' (15th May 2020), [Online]. Available: https://www.paloaltonetworks.com/blog/2020/05/containers-are-inherently-secure-reality-or-myth/ (visited on 15/03/2022).

[52] S. T. Help. 'Top 10 best container software in 2022.' (4th May 2022), [Online]. Available: https://www.softwaretestinghelp.com/container-software/ (visited on 13/05/2022).

[53] Docker. 'Docker engine overview.' (), [Online]. Available: https://docs.docker.com/engine/ (visited on 13/05/2022).

[54] Checkmarx. 'Contact us.' (), [Online]. Available: https://checkmarx.com/contact/?checkmarx-offices (visited on 11/05/2022).

[55] Trivy. 'Aqua security/trivy.' (), [Online]. Available: https://github.com/aquasecurity/trivy (visited on 17/02/2022).

[56] PortSwigger. 'Portswigger.' (), [Online]. Available: https://portswigger.net/ (visited on 16/02/2022).

[57] PortSwigger. 'Burp suite is the choice of security professionals worldwide.' (), [Online]. Available: https://portswigger.net/burp (visited on 17/02/2022).

[58] C. Security. 'Contrast security.' (), [Online]. Available: https://www.contrastsecurity.com/ (visited on 27/02/2022).

[59] PortSwigger. 'Contact us.' (), [Online]. Available: https://portswigger.net/about/contact (visited on 21/02/2022).

[60] Crunchbase. 'Contrast security.' (), [Online]. Available: https://www.contrastsecurity.com/pricing (visited on 17/02/2022).

[61] A. Security. 'Contact us.' (), [Online]. Available: https://www.aquasec.com/about-us/contact-us/ (visited on 15/02/2022).

[62] A. Security. 'Aqua security - about us.' (), [Online]. Available: https://www.aquasec.com/about-us/ (visited on 15/02/2022).

[63] PortSwigger. 'About.' (), [Online]. Available: https://portswigger.net/about (visited on 15/02/2022).

[64] PortSwigger. 'Portswigger.' (), [Online]. Available: https://portswigger.net/burp/enterprise (visited on 16/02/2022).

[65] PortSwigger. 'Portswigger.' (), [Online]. Available: `https://portswigger.net/burp/pro` (visited on 16/02/2022).

[66] C. Security. 'Contrast security.' (), [Online]. Available: `https://www.contrastsecurity.com/customer-success` (visited on 17/02/2022).

[67] PortSwigger. 'Select your burp suite enterprise edition plan.' (), [Online]. Available: `https://portswigger.net/burp/enterprise/pricing` (visited on 18/02/2022).

[68] PortSwigger. 'Subscribe to burp suite professional.' (), [Online]. Available: `https://portswigger.net/buy/pro` (visited on 18/02/2022).

[69] C. Security. 'Contrast security plans and pricing.' (), [Online]. Available: `https://www.contrastsecurity.com/pricing` (visited on 17/02/2022).

[70] Detectify. 'Detectify.' (), [Online]. Available: `https://detectify.com/` (visited on 16/02/2022).

[71] Acunetix. 'About acunetix.' (), [Online]. Available: `https://www.acunetix.com/about/` (visited on 03/03/2022).

[72] Perforce. 'Klocwork.' (), [Online]. Available: `https://www.perforce.com/products/klocwork` (visited on 17/02/2022).

[73] Detectify. 'Detectify about.' (), [Online]. Available: `https://detectify.com/about` (visited on 17/02/2022).

[74] Invicti. 'Invicti.' (), [Online]. Available: `https://www.invicti.com/` (visited on 16/02/2022).

[75] Perforce. 'Contact us.' (), [Online]. Available: `https://www.perforce.com/contact-us` (visited on 21/02/2022).

[76] Netsparker. 'About netsparker.' (), [Online]. Available: `https://www.netsparker.com/about/` (visited on 16/02/2022).

[77] Perforce. 'About perforce.' (), [Online]. Available: `https://www.perforce.com/company` (visited on 15/02/2022).

[78] Perforce. 'Customers.' (), [Online]. Available: `https://www.perforce.com/customers?field_industry_target_id=All&field_product_line_target_id=1531&field_customer_resource_type_target_id=All` (visited on 21/02/2022).

[79] Detectify. 'Detectify pricing.' (), [Online]. Available: `https://detectify.com/pricing` (visited on 16/02/2022).

[80] D. Wendlandt. 'Nessus : A security vulnerability scanning tool.' (), [Online]. Available: `https://www.cs.cmu.edu/~dwendlan/personal/nessus.html` (visited on 18/02/2022).

[81] OWASP. 'Owasp zed attack proxy.' (), [Online]. Available: `https://www.zaproxy.org` (visited on 17/02/2022).

[82] Reshift. 'Welcome to reshift.' (), [Online]. Available: https://docs.reshiftsecurity.com/ (visited on 17/02/2022).

[83] Tenable. 'Contact tenable.' (), [Online]. Available: https://www.tenable.com/about-tenable/contact-tenable (visited on 16/02/2022).

[84] Crunchbase. 'Owasp foundation.' (), [Online]. Available: https://www.crunchbase.com/organization/owasp-foundation (visited on 17/02/2022).

[85] Crunchbase. 'Reshift security.' (), [Online]. Available: https://www.crunchbase.com/organization/reshift-security (visited on 16/02/2022).

[86] Tenable. 'About us.' (), [Online]. Available: https://www.tenable.com/about-tenable/about-us (visited on 15/02/2022).

[87] Psiinon. 'Zap is ten years old.' (6th Sep. 2020), [Online]. Available: https://www.zaproxy.org/blog/2020-09-06-zap-is-ten-years-old/ (visited on 17/02/2022).

[88] Tenable. 'Tenable customers.' (), [Online]. Available: https://www.tenable.com/customers (visited on 17/02/2022).

[89] Zap. 'Success stories.' (), [Online]. Available: https://www.zaproxy.org/success/ (visited on 17/02/2022).

[90] Reshift. 'Plans and pricing.' (), [Online]. Available: https://www.reshiftsecurity.com/pricing/ (visited on 16/02/2022).

[91] Tenable. 'Nessus.' (), [Online]. Available: https://www.tenable.com/products/nessus (visited on 17/02/2022).

[92] Snyk. 'What is snyk?' (), [Online]. Available: https://snyk.io/what-is-snyk/ (visited on 21/03/2022).

[93] StackHawk. 'Welcome to stackhawk.' (), [Online]. Available: https://docs.stackhawk.com/#welcome-to-stackhawk (visited on 16/02/2022).

[94] StackHawk. 'Find, triage, and fix security bugs.' (), [Online]. Available: https://www.stackhawk.com/product/ (visited on 17/02/2022).

[95] Veracode. 'The veracode solution.' (), [Online]. Available: https://www.veracode.com/products (visited on 17/02/2022).

[96] Crunchbase. 'Snyk.' (), [Online]. Available: https://www.crunchbase.com/organization/snyk (visited on 15/02/2022).

[97] StackHawk. 'About stackhawk.' (), [Online]. Available: https://www.stackhawk.com/about/ (visited on 15/02/2022).

[98] Veracode. 'Contact departments.' (), [Online]. Available: https://www.veracode.com/contact-us#mktoForm_326 (visited on 21/02/2022).

[99] Crunchbase. 'Veracode.' (), [Online]. Available: https://www.crunchbase.com/organization/veracode (visited on 21/02/2022).

[100] Snyk. 'Snyk.' (), [Online]. Available: https://snyk.io/ (visited on 16/02/2022).

[101] StackHawk. 'Stackhawk.' (), [Online]. Available: https://www.stackhawk.com/ (visited on 16/02/2022).

[102] Veracode. 'Veracode.' (), [Online]. Available: https://www.veracode.com/ (visited on 17/02/2022).

[103] Snyk. 'Plans and pricing.' (), [Online]. Available: https://snyk.io/plans/ (visited on 16/02/2022).

[104] StackHawk. 'Pricing.' (), [Online]. Available: https://www.stackhawk.com/pricing/ (visited on 15/02/2022).

[105] C. Security. 'Devsec with contrast.' (), [Online]. Available: https://contrastsecurity.dev/docs/getting-started/where-do-i-start/ (visited on 04/04/2022).

[106] C. Security. 'Integrations.' (), [Online]. Available: https://docs.contrastsecurity.com/en/integrations.html (visited on 06/04/2022).

[107] K. Srinivas. 'Detecting security vulnerabilities with contrast security.' (6th Sep. 2017), [Online]. Available: https://www.ibm.com/blogs/cloud-archive/2017/09/detecting-security-vulnerabilities-with-contrast-security/ (visited on 03/03/2022).

[108] C. Security. 'The contrast code security platform.' (), [Online]. Available: https://www.contrastsecurity.com/platform (visited on 03/03/2022).

[109] C. Security. 'Contrast scan.' (), [Online]. Available: https://www.contrastsecurity.com/contrast-scan?hsLang=en (visited on 03/03/2022).

[110] C. Security. 'Contrast sca.' (), [Online]. Available: https://www.contrastsecurity.com/contrast-sca (visited on 03/03/2022).

[111] C. Security. 'Contrast assess.' (), [Online]. Available: https://www.contrastsecurity.com/contrast-assess (visited on 03/03/2022).

[112] C. Security. 'Contrast serverless application security.' (), [Online]. Available: https://www.contrastsecurity.com/contrast-serverless (visited on 03/03/2022).

[113] C. Security. 'Contrast protect.' (), [Online]. Available: https://www.contrastsecurity.com/contrast-protect?hsLang=en (visited on 01/03/2022).

[114] C. Security. 'Sensitive data masking.' (), [Online]. Available: https://docs.contrastsecurity.com/en/sensitive-data-masking.html (visited on 24/03/2022).

[115] J. Watasseril. ‘Understanding and adjusting severity levels.’ (4th Feb. 2022), [Online]. Available: `https://support.contrastsecurity.com/hc/en-us/articles/360000449926-Understanding-and-Adjusting-Severity-Levels`.

[116] Contrast. ‘Application scoring guide.’ (), [Online]. Available: `https://docs.contrastsecurity.com/en/application-scoring-guide.html` (visited on 05/04/2022).

[117] Snyk. ‘Open source language and package manager support.’ (), [Online]. Available: `https://docs.snyk.io/products/snyk-open-source/language-and-package-manager-support` (visited on 06/04/2022).

[118] Snyk. ‘Snyk integrations.’ (27th Apr. 2022), [Online]. Available: `https://docs.snyk.io/integrations` (visited on 18/05/2022).

[119] Snyk. ‘Snyk learned.’ (), [Online]. Available: `https://learn.snyk.io/` (visited on 14/03/2022).

[120] Snyk. ‘Open source vulnerability database.’ (), [Online]. Available: `https://security.snyk.io/` (visited on 15/03/2022).

[121] Snyk. ‘Snyk open-source (sca).’ (), [Online]. Available: `https://docs.snyk.io/tutorials/springone-workshop/snyk-oss-for-developers` (visited on 15/03/2022).

[122] Snyk. ‘Snyk open source.’ (), [Online]. Available: `https://snyk.io/product/open-source-security-management/` (visited on 22/02/2022).

[123] Snyk. ‘Snyk container.’ (17th Feb. 2022), [Online]. Available: `https://docs.snyk.io/products/snyk-container` (visited on 01/03/2022).

[124] Snyk. ‘Snyk code.’ (), [Online]. Available: `https://snyk.io/product/snyk-code/` (visited on 22/02/2022).

[125] Snyk. ‘Snyk code.’ (28th Feb. 2022), [Online]. Available: `https://docs.snyk.io/products/snyk-code` (visited on 01/03/2022).

[126] Snyk. ‘Snyk infrastructure as code.’ (21st Feb. 2022), [Online]. Available: `https://docs.snyk.io/products/snyk-infrastructure-as-code` (visited on 22/03/2022).

[127] Snyk. ‘Sql injection.’ (), [Online]. Available: `https://learn.snyk.io/lessons/sql-injection/javascript/` (visited on 14/03/2022).

[128] Snyk. ‘How snyk handles your data.’ (), [Online]. Available: `https://docs.snyk.io/more-info/how-snyk-handles-your-data#product-specific-data-types` (visited on 18/03/2022).

[129] StackHawk. ‘Stackhawk and snyk are better together.’ (), [Online]. Available: `https://www.stackhawk.com/snyk/` (visited on 29/04/2022).

[130] StackHawk. 'Dynamic application security testing (dast): Overview and tooling guide.' (), [Online]. Available: https://www.stackhawk.com/blog/dynamic-application-security-testing-overview/ (visited on 14/03/2022).

[131] Zap. 'Owasp zed attack proxy (zap).' (), [Online]. Available: https://www.zaproxy.org/ (visited on 14/03/2022).

[132] StackHawk. 'Stackhawk integrations.' (), [Online]. Available: https://docs.stackhawk.com/workflow-integrations/ (visited on 06/04/2022).

[133] StackHawk. 'Dynamic application security testing.' (), [Online]. Available: https://www.stackhawk.com/solutions/dast/ (visited on 15/03/2022).

[134] StackHawk. 'Stackhawk platform.' (), [Online]. Available: https://docs.stackhawk.com/web-app/ (visited on 18/05/2022).

[135] StackHawk. 'Stackhawk privacy policy.' (), [Online]. Available: https://www.stackhawk.com/privacy-policy/ (visited on 06/04/2022).

[136] StackHawk. 'Risk.' (), [Online]. Available: https://docs.stackhawk.com/hawkscan/viewing-scan-results.html#risk (visited on 11/05/2022).

[137] Veracode. 'Supported languages and platforms.' (), [Online]. Available: https://docs.veracode.com/r/r_supported_table (visited on 04/04/2022).

[138] Veracode. 'Integrating with build and release management systems.' (), [Online]. Available: https://docs.veracode.com/r/c_integration_buildservs (visited on 06/04/2022).

[139] Veracode. 'Software development life cycle (sdlc).' (), [Online]. Available: https://www.veracode.com/security/software-development-lifecycle-sdlc (visited on 03/05/2022).

[140] Veracode. 'About veracode static analysis.' (), [Online]. Available: https://docs.veracode.com/r/c_static_overview (visited on 05/04/2022).

[141] Veracode. 'Veracode static analysis.' (), [Online]. Available: https://www.veracode.com/products/binary-static-analysis-sast (visited on 11/03/2022).

[142] Veracode. 'Getting started with veracode software composition analysis (sca).' (), [Online]. Available: https://docs.veracode.com/r/Getting_Started_with_Veracode_Software_Composition_Analysis_SCA (visited on 05/04/2022).

[143] Veracode. 'About veracode dynamic analysis.' (), [Online]. Available: https://docs.veracode.com/r/c_was_intro (visited on 05/04/2022).

[144] Veracode. 'Veracode discovery.' (), [Online]. Available: `https://www.veracode.com/products/discovery` (visited on 11/03/2022).

[145] Veracode. 'Understanding manual penetration testing.' (), [Online]. Available: `https://docs.veracode.com/r/c_understanding_manual` (visited on 16/05/2022).

[146] Veracode. 'About veracode elearning.' (), [Online]. Available: `https://docs.veracode.com/r/elearning_master` (visited on 04/05/2022).

[147] Veracode. 'Understanding veracode rules for data retention and archiving.' (), [Online]. Available: `https://docs.veracode.com/r/c_data_retention` (visited on 25/04/2022).

[148] Veracode. 'Scoring methodology.' (), [Online]. Available: `https://docs.veracode.com/r/review_scoringmethodology` (visited on 11/05/2022).

[149] Veracode. 'Using cvss versions.' (), [Online]. Available: `https://docs.veracode.com/r/Using_CVSS_Versions` (visited on 11/05/2022).

[150] C. Security. 'Java quick start guide.' (), [Online]. Available: `https://docs.contrastsecurity.com/en/java-quick-start-guide.html` (visited on 26/04/2022).

[151] lmat - Reinstate Monica. 'Powershell run java process problem.' (), [Online]. Available: `https://stackoverflow.com/questions/4685184/powershell-run-java-process-problem` (visited on 26/04/2022).

[152] C. Security. 'Contrast scan.' (), [Online]. Available: `https://www.contrastsecurity.com/contrast-scan` (visited on 06/04/2022).

[153] Veracode. 'Upload and scan applications with veracode software composition analysis.' (10th Jan. 2020), [Online]. Available: `https://www.youtube.com/watch?v=a4GDfRR7K4c&ab_channel=VERACODE` (visited on 07/04/2022).

[154] Veracode. 'Set up an agent to scan with veracode software composition analysis.' (12th Mar. 2020), [Online]. Available: `https://www.youtube.com/watch?v=fVxTD_EZ9tg` (visited on 07/04/2022).

[155] Veracode. 'About the veracode dynamic analysis workflow.' (), [Online]. Available: `https://docs.veracode.com/r/About_the_Veracode_Dynamic_Analysis_Workflow` (visited on 07/04/2022).

[156] Veracode. 'Run an authenticated dynamic analysis of a web application.' (), [Online]. Available: `https://docs.veracode.com/r/c_was_use_case2` (visited on 07/04/2022).

[157] Veracode. 'Veracode and github integration.' (22nd Nov. 2021), [Online]. Available: `https://www.youtube.com/watch?v=xalWwLf9bWM&ab_channel=VERACODE` (visited on 07/04/2022).

[158] Veracode. 'Generate veracode api credentials.' (), [Online]. Available: https://docs.veracode.com/r/t_create_api_creds (visited on 07/04/2022).

[159] Snyk. 'Code language and framework support.' (4th May 2022), [Online]. Available: https://docs.snyk.io/products/snyk-code/snyk-code-language-and-framework-support (visited on 05/05/2022).

[160] Invicti. 'A new era for modern application security: Netsparker is now invicti.' (8th Mar. 2022), [Online]. Available: https://www.invicti.com/blog/news/netsparker-is-now-invicti-new-era-modern-application-security/ (visited on 30/04/2022).

[161] StackHawk. 'First of its kind snyk integration correlates dynamic & static application security testing.' (27th Apr. 2022), [Online]. Available: https://www.stackhawk.com/blog/stackhawk-snyk-integration-press-release/?fbclid=IwAR0PT7SW-CVfyS2rwoM22vlDTVduQmIx0aZ_PF6Pmc-pFLmaCOXSE9nVcFk (visited on 30/04/2022).

[162] Snyk. 'Cross-site scripting.' (), [Online]. Available: https://learn.snyk.io/lessons/xss/javascript/ (visited on 14/03/2022).

[163] Veracode. 'Understanding language support for veracode sca upload scans.' (), [Online]. Available: https://docs.veracode.com/r/c_sc_supported_lang (visited on 25/03/2022).

[164] Veracode. 'Create and run an unauthenticated dynamic analysis.' (28th Jan. 2021), [Online]. Available: https://www.youtube.com/watch?v=El6xBMaC4yg&ab_channel=VERACODE (visited on 07/04/2022).

[165] Veracode. 'Best practices for gateway management.' (), [Online]. Available: https://docs.veracode.com/r/c_gateway_best_practices (visited on 07/04/2022).

[166] Veracode. 'Best practices for endpoint management.' (), [Online]. Available: https://docs.veracode.com/r/c_endpoint_best_practices (visited on 07/04/2022).

[167] Veracode. 'Configure a dynamic analysis of a web application for internal scanning.' (), [Online]. Available: https://docs.veracode.com/r/c_was_use_case3 (visited on 07/04/2022).

[168] Veracode. 'About veracode ide integrations.' (), [Online]. Available: https://docs.veracode.com/r/c_ide_intro (visited on 25/03/2022).

[169] Veracode. 'Install veracode for vs code to run ide scans.' (), [Online]. Available: https://www.youtube.com/watch?v=ow5fGyjKtug (visited on 26/04/2022).

[170] Veracode. 'Scanning source code using veracode for vs code.' (), [Online]. Available: https://www.youtube.com/watch?v=hal0pSJa5kM (visited on 26/04/2022).

[171] Veracode. 'Reviewing findings in veracode for vs code.' (), [Online]. Available: `https://www.youtube.com/watch?v=DbpeS4IWC28` (visited on 26/04/2022).

[172] Veracode. 'Request a static scan in the veracode platform.' (24th Sep. 2019), [Online]. Available: `https://www.youtube.com/watch?v=M-v0dodSavM&ab_channel=VERACODE` (visited on 07/04/2022).

[173] Veracode. 'Configuring an api credentials file.' (), [Online]. Available: `https://docs.veracode.com/r/c_configure_api_cred_file?tocId=x91jZOXBHfLA556x5w4xcQ` (visited on 25/03/2022).

[174] A. Aspøy. 'Forsvarsdepartementet.' (14th Oct. 2022), [Online]. Available: `https://snl.no/Forsvarsdepartementet`.

Eggestad, Høyaas, Løken, Stålevik

Software Security Testing