Assessing the Security Benefits of Defence in Depth

City, University of London
School of Mathematics, Computer Science and Engineering

Areej A. Algaith

Submitted in accordance with the requirements for the degree of
Doctor of Philosophy

Under the Supervision of
Dr. Ilir Gashi

February 2019

Intellectual Property and Publication Statements

The candidate confirms that the work submitted is her own, except where work has formed part of jointly-authored publications. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

- Areej Algaith, Ilir Gashi, Bertrand Sobesto, Michel Cukier, Selman Haxhijaha, Gazmend Bajrami (2016). “Comparing Detection Capabilities of AV Products: An Empirical Study with Different Versions of Products from the Same Vendors”. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, pp. 48-53. (Details in Chapter 4 of the thesis). Available at: http://openaccess.city.ac.uk/15503/

The data analysis, discussion, conclusions and first authorship of the paper is that of the candidate. The data collection was done at the Universities of Maryland and University for Business and Technology, Prishtina who kindly shared the data with the candidate.

- Pasha Shahegh, Tommy Dietz, Michel Cukier, Areej Algaith, Attila Brozik, Ilir Gashi: “AV and Malware Analysis Tool”. NCA 2017, pp. 365-368. (Details in Chapter 5 of the thesis). Available at: http://openaccess.city.ac.uk/18334/

The candidate contributed in all the discussion regarding the inception, requirements and design of the tool. She also analysed the data from test reports shared by the developers at the University of Maryland, and provided detailed bug reports, which helped with the tool debugging. The candidate also contributed towards the writing of the paper, specifically the tool design and testing sections, as well as providing comments on several versions during the paper writing process. The implementation of the AVAMAT tool was done at the University of Maryland.

- Areej Algaith, Ivano Alessandro Elia, Ilir Gashi, Marco Vieira: “Diversity with intrusion detection systems: An empirical study”. NCA 2017, pp. 19-23. (Details in Chapter 6 of the thesis). Available at: http://openaccess.city.ac.uk/18335/

The data analysis, discussion, conclusions and first authorship of the paper is that of the candidate. The data collection was done at the University of Coimbra, who kindly shared the data with the candidate.

- Areej Algaith, Paulo Nunes, Jose Fonseca, Ilir Gashi, Marco Viera (2018). “Finding SQL Injection and Cross Site Scripting Vulnerabilities with Diverse Static Analysis Tools”. 14th European Dependable Computing Conference (EDCC’18) (pp.57-64). (Details in Chapter 7 of the thesis). Available at: http://openaccess.city.ac.uk/20065/

The data analysis, discussion, conclusions and first authorship of the paper is that of the candidate. The data collection, and plug-in analysis were done at the University of Coimbra and Polytechnic Institute of Guarda, who kindly shared the data with the candidate.

Acknowledgements

First of all, I would like to thank Allah who always guided me to resolve crisis during my PhD research.

This research would not see the light without my parents encouragements, with their prayers, love and support. Thank you, mother, Laila Aljalajel. Thank you, father, Abdulrahman Algaith. Your sacrifices made my path always light and shine.

I am sincerely thankful to the government of Saudi Arabia for awarding me the scholarship and the opportunity they have given me.

I am extremely grateful to my supervisor Dr. Ilir Gashi for his supervision, patience, advice and commitment throughout whole this PhD research. This work would not have been possible without him.

I would also like to thank Professor Lorenzo Strigini for his discussion and useful input in the optimal adjudication function.

I am also very grateful to Dr. Vladimir Stankovic for the insightful advice on the MPhil-to-PhD Transfer Report.

A special thanks also to Dr. Michel Cukier from University of Maryland in the USA for his guidance and making the AntiVirus data available.

I would also like to thank Mr. Paulo Nunes and Dr. Jose Fonseca from the Department of Computer Science, Polytechnic Institute of Guarda (IPG), and Dr. Ivano Elia and Professor Marco Vieira from the University of Coimbra, Portugal for their valuable discussion and making data available.

I am indebted to all my family members, my friends and everyone who supported and helped me along my Journey towards the completion of my study. Thank you all.

Abstract

Most modern computer systems are connected to the Internet. This brings many opportunities for revenue generation via e-commerce and information sharing, but also threats due to the exposure of these systems to malicious adversaries. Therefore, almost all organisations deploy security tools to improve overall detection capabilities. However, all security tools have limitations: they may fail to detect attacks, fail to uncover all vulnerabilities or generate alarms for non-malicious traffic or non-vulnerable code. Using terminology from signalling theory, we can state that security tools suffer from two types of failures: failure to correctly label a malicious event as malicious (False Negatives); and failure to correctly label a non-malicious event as non-malicious (False Positive). These failures may vary from one tool to another, since security tools are diverse in their weaknesses as well as their strengths. Therefore, an obvious design paradigm when deploying these defences is Diversity or Defence in Depth: the expectation is that employing multiple tools increases the chance of detecting malicious behaviour. This thesis presents research to assess the benefits (or harm) from using diversity. This thesis begins with a literature review on defence in depth, diversity and fault tolerance while identifying areas for further research. This review is followed by the presentation of the overall methodology that we have used to perform the diversity assessment for three types of defence tools namely AntiVirus (AV) products, Intrusion Detection Systems (IDS) and Static Analysis Tools (SAT). The context of this project is inspired by the EPSRC D3S[1] project in the Centre for Software Reliability (CSR) at the City, University of London as well as the previous work on diversity conducted at the same centre, but also elsewhere in the world. This thesis presents the results using the well-known metrics for binary classifiers: Sensitivity and Specificity; and assesses the various forms of adjudication that may be used: 1-out-of-N (1ooN – raise an

[1] http://www.city.ac.uk/news/2015/march/researchers-at-citys-centre-for-software-reliability-are-the-recipients-of-a-563,089

alarm as long as ANY of the defences do so), N-out-of-N (NooN – raise an alarm only if ALL the defences do so), majority voting (raise an alarm where a MAJORITY of the defences do so) or optimal adjudication (raise an alarm in such a way that it minimises the overall loss to the system from a failure).

The first study compares the detection capabilities of nine different AV products. Additionally, for each vendor, the detection capabilities of the version of the product that is available for free in the VirusTotal[2] platform are compared with the full capability version of that product that is available from the same vendor's website. Counterintuitively, the free version of AVs from VirusTotal performed better (in most cases) than the commercial versions from the same vendor.

The second study compares the detection capabilities of IDS when deployed in a combined configuration. The functionally diverse combinations are shown to increase the true positive rate significantly while experiencing smaller increases in false positive rate.

The third study analyses the improvements and deteriorations of using diverse SATs to detect web vulnerabilities. The largest improvements in sensitivity, with the least deterioration in specificity was observed with the 1ooN configurations, in NooN configurations there is an improvement in specificity compared with individual systems, and there is a deterioration in sensitivity.

Finally, the benefits of "optimal adjudication" were also investigated: the result shows that the total loss that can result from the two types of failures considered (False Positives and False Negatives) can be significantly reduced with optimal adjudication configurations compared with more conventional methods of adjudication such as 1ooN, NooN or majority voting.

In conclusion, using diverse security protection tools is shown to be beneficial to improving the detection capability of three different families of products and optimal adjudication techniques can help balance the benefits of improved detection while lowering the false positive rates.

[2] VirusTotal is a web service providing online malware analysis based on several AV products, available at https://www.virustotal.com/

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

1ooN	One-out-of-N
ACD	Anomalous Character Distribution
ADJ	Adjudication
API	Application Programming Interface
AV	AntiVirus
AVAMAT	AntiVirus and Malware Analysis Tool
AVFC	AntiVirus Full Capability
AVSB	AntiVirus Signature Based
CA	Crawling Action
CAS	Centre for Assured Software
CMS	Content Management System
DF	Detection/Failure
DID	Defence in Depth
EP	Entry Point
FC	Full Capability AV
FN	False Negative
FP	False Positive
FR	Failure Rate
IDS	Intrusion Detection System
IE	Internet Explorer
KVM	Kernel-based Virtual Machine
MW	Malware
NooN	N-out-of-N

NVLOC	Non-Vulnerable Line of Code
ROC	Receiver Operating Characteristic
RooN	R-out-of-N (majority voting)
SA	Successful Attack
SAT	Static Analysis Tool
Sen.	Sensitivity
Spec.	Specificity
SQL	Structured Query Language
SQLi	SQL injection
SS	Sensitive Sink
TN	True Negative
TP	True Positive
VT	Virus Total
VLOC	Vulnerable Line of Coe
XSS	Cross-site Scripting

(1) INTRODUCTION

1.1 Context of the research

An important part of design for security is defence in depth, consisting of “layers” of defence that reduce the probability of successful attacks. Guidance documents now advocate defence in depth as an obvious need[3]; but their qualitative guidance ignores the decision problems. Crucially, these questions concern diversity: defences should be diverse in their weaknesses. Any attack that happens to defeat one defence should with high probability be stopped or detected by another one. Ultimately, *diversity* and *defence in depth* are two facets of the same defensive design approach. Another term used in the literature is *defence in breadth*: the difference between the two is often (either explicitly or implicitly) understood to be that defence in depth is applying diversity between the layers and the defence in breadth across the same layer. In this thesis we mainly use the terms diversity and defence in depth, and unless explicitly stated otherwise, we use them interchangeably.

The important questions are not about defence in depth and diversity being "a good idea", but about whether a set of specific defences would improve security more than another set; and about – if possible – quantifying the security gains.

The security community is aware of diversity as potentially valuable (e.g. references in (Littlewood & Strigini, 2004) and (Garcia et al., 2014)). Discussion papers argue the general desirability of diversity among network elements, like communication media, network protocols, operating systems etc. Research projects studied distributed systems using diverse off-the-shelf products for intrusion tolerance (e.g. the U.S. projects Cactus, HACQIT (Reynolds et al., 2002) and SITAR[4]; the EU MAFTIA project[5]). New uses of diversity appear every now and then, e.g., diverse AV software in a commercial E-mail scanner[6] and “in the cloud” (Oberheide, Cooke and Jahanian, 2008), metrics for effectiveness of defence in depth[7], etc.

[3] https://www.iad.gov/iad/library/ia-guidance/archive/defense-in-depth.cfm

[4] http://www.cs.arizona.edu/cactus/ , http://people.ee.duke.edu/~kst/sitar.html

[5] http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/maftia/

[6] http://www.gfi.com/products-and-solutions/email-and-messaging-solutions/gfi-mailessentials/specifications/up-to-five-AV-engines

[7] http://ids.cs.columbia.edu/sites/default/files/law2011-aldr-final2.pdf

But there has been only sparse research (e.g. Gupta, 2003; Singh et al., 2003) on how to choose among alternative layered defences; occasionally, unsuitable models appear relying on the naive assumption of independent failures (Wang et al., 2011). Official guidance shows trust in defence in depth but glosses over the need for quantification. For example, justifications like "Even the best available Information Assurance products have inherent weaknesses... it is only a matter of time before an adversary will find an exploitable vulnerability"[8] ignore the probabilistic nature of security. What matters is not that adversaries will eventually break through – this applies to layered defences too! It is how soon they are likely to break through. Added layers of defence postpone that moment, directly by requiring extra steps, and indirectly by allowing for detection and defensive steps. But by how much? Thus the main outcomes of the research we present in this thesis will be methods for measuring security to drive rational decisions, and quantitative assessment of diversity with widely used security tools.

1.2 Aims and objectives

We note from our review of diversity in security (cf. Chapter 2) that although there is plentiful literature on defence in depth models and some literature on mitigation strategies, there is insufficient research on assessing the benefits (and harm) of defence in depth. In this research we address this gap by presenting results from three empirical studies with different types of defence. At a high level, our aim is to analyse:

- the variety of architectural options about how diverse security controls are assembled and their responses combined ("adjudication"). For instance, a security architecture may have multiple defences: the adjudication can be such as to make this architecture a "1-out-of-N" system (successful at stopping attacks if just one of the N "layers'" succeeds), or weaker but less likely to report "false alarms" – blocking non-hostile activities (e.g. if it only treats an activity as an attack if a quorum of the defences flags it as such);

the interplay between the risks of failing to react to true attacks and of false alarms ("false negative" and "false positive" failures, where "failure" may mean different things – penetration, lack of detection, etc. – depending on the

[8] www.nsa.gov/ia/_files/support/defenseindepth.pdf

function concerned). Receiver Operating Characteristic (ROC) curves (Egan, 1975) are often used to describe this trade-off for each detector and type of attack, but not fully studied for complex compositions of subsystems (Ulvila, Gaffney and Jr, 2003)

We have conducted experiments with three types of defence tools: AV products, Intrusion Detection Systems (IDS) and Static Analysis Tools (SAT). We present the results using the well-known metrics for binary classifiers: namely Sensitivity and Specificity; and we assess the various forms of adjudication that may be used when configuring diverse tools: 1-out-of-N (raise an alarm as soon as ANY of the defences do so), N-out-of-N (raise an alarm only if ALL the defences do so), majority voting (raise an alarm where a MAJORITY of the defences do so) or optimal adjudication (raise an alarm in such a way that it minimises the overall loss to the system from a failure).

The main goals of our research are:

1. Analyse the security benefits (or harm) of defence tools individually; presented in Chapters 4, 5, 6 and 7.
2. Analyse the benefits (or harm) of the use of defence tools in diverse configurations; presented in Chapters 4, 5, 6, 7 and 8.
3. Analyse the diversity that exists in the vulnerabilities of different applications – three web applications in Chapter 6 and a content management system with many plugins presented in Chapter 7.
4. Analyse the benefits of “optimal adjudication” (Giandomenico and Strigini, 1990); presented in Chapter 8.
5. Provide an analysis methodology for assessing the *performance* (in terms of lower false negative and false positive rates) of N-version diverse security decision support systems presented in Chapter 3.

1.3 Contributions of the research

The research presented in this thesis describes a number of novel analyses to help us quantify the possible benefits (and harm) of diversity for security, and hence help improve decision making for security. The main contributions are as follows:

- An assessment of detection capabilities of different AV products from different vendors, and the different versions that those vendors provide (i.e. the free version of the AV that is made available through services like VirusTotal and their commercial counterparts). We presented three types of analysis:

  - Analysis of the detection rate of full capability AV products compared to the detection rate provided by the free version of the same product.
  - Analysis of the detection time difference, in other words, if both AV products are able to detect the same malware, which version manages to detect the malware first and calculate the time difference between them.
  - Analysis of the labels that an AV version assigns to a given malware. The labels may be important to provide a security analyst with extra information on what type of malware they are dealing with, which may help them with the diagnosis. Analyses are achieved first by checking whether the labels assigned by the two versions of a given AV product were exactly the same, and second by analysing the classifications that the AV products used for these different types of malware.

- An assessment of the performance of diverse IDS configurations: We used a previously published dataset (Elia et al., 2010) that used nine IDS configurations, when they were monitoring 3 web applications that were subjected to SQL injection attacks and benign crawling actions. We present the results using the well-established measures for binary classifiers: sensitivity, specificity and accuracy, and use ROCs to visualise the results.
- An assessment of the performance of diverse Static Analysis Tool (SAT) configurations: We used a previously published dataset composed of five diverse SATs aimed to find SQL Injections (SQLi) and Cross-Site Scripting (XSS) vulnerabilities in 134 WordPress plugins (Nunes et al., 2017). We present the results using ROCs, sensitivity and specificity, for all possible diverse combinations that can be constructed using these five SATs.
- Assessing the benefits of optimal adjudication: Using the IDS and SAT datasets described above we evaluate the security tools in different adjudication setups (1ooN, NooN, majority voting and Optimal

adjudication), and quantify how much better optimal adjudication performs in reducing total loss, compared with the other adjudication setups.

- We provide empirically-supported guidance on which combination of tools is better at detecting most vulnerabilities, and with low false alarms, depending on the requirements of the decision maker.
- We provide an analysis method that may help other researchers with their analysis of diversity and defence in depth. This should prove useful to other researchers and organisations to assess diversity in their setups.

In addition to the assessment work outlined above, we also designed, implemented and tested a tool for assessing diverse AV products called AVAMAT (presented in Chapter 5). This work was a joint effort with University of Maryland, with the author of this thesis being primarily involved in the design and testing of AVAMAT (implementation was done primarily at the University of Maryland).

1.4 Thesis outline structure

The second Chapter provides a review of literature in defence in depth for security, background on diversity research, as well as empirical and experimental studies for assessing diversity. The third Chapter present the analysis methodology that we have used in the thesis to assess the diversity with AV, IDS and SAT products. The fourth Chapter presents the results of analysis that compare the detection capabilities of full-capability AV products and signature-based AV products. The fifth Chapter presents AVAMAT: a tool for assessing full-capability diverse AV products. The sixth Chapter presents the analysis of a study in which we have assessed the detection capabilities of intrusion detection systems when deployed in diverse, defence in depth configurations. The seventh Chapter presents the result of an experiment in which five diverse SATs have been used to detect SQLi and XSS vulnerabilities in 132 plug-ins of the WordPress content management system. The eighth Chapter presents an analysis on the use of optimal adjudication with the SATs and IDSs and their comparison with the conventional forms of adjudication (namely 1-out-of-N, N-out-of-N and majority voting). Finally, Chapter 9 summarises the conclusions of the research and provisions for further work.

1.5 Publications

- Areej Algaith, Ilir Gashi, Bertrand Sobesto, Michel Cukier, Selman Haxhijaha, Gazmend Bajrami (2016). "***Comparing Detection Capabilities of AntiVirus Products: An Empirical Study with Different Versions of Products from the Same Vendors"***. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (pp. 48-53). IEEE DSN-W. ISBN 978-1-5090-3688-2. (Details in Chapter 4 of the thesis)
  Available at: http://openaccess.city.ac.uk/15503/

- Pasha Shahegh, Tommy Dietz, Michel Cukier, Areej Algaith, Attila Brozik, Ilir Gashi: ***"AntiVirus and Malware Analysis Tool***". IEEE NCA 2017 (pp. 365-368). (Details in Chapter 5 of the thesis)
  Available at: http://openaccess.city.ac.uk/18334/

- Areej Algaith, Ivano Alessandro Elia, Ilir Gashi, Marco Vieira: ***"Diversity with intrusion detection systems: An empirical study"***. IEEE NCA 2017 (pp. 19-23). (Details in Chapter 6 of the thesis)
  Available at: http://openaccess.city.ac.uk/18335/

- Areej Algaith, Paulo Nunes, Jose Fonseca, Ilir Gashi, Marco Viera (2018). "***Finding SQL Injection and Cross Site Scripting Vulnerabilities with Diverse Static Analysis Tools***". 14th European Dependable Computing Conference (EDCC'18) (pp.57-64) 10-14 September, Iasi, Romania. (Details in Chapter 7 of the thesis)
  Available at: http://openaccess.city.ac.uk/20065/

(2) BACKGROUND AND RELATED WORK

2.1 Background on fault tolerance, diversity and defence in depth

A discussion on fault and intrusion tolerance usually begins with the definition of the "bad" events that the system designer wants to tolerate. The definitions of the terms **fault** (or bug), **error** and **failure** given in this section are based on (Avižienis et al., 2004). A failure is said to occur when the system stops performing its required functions. An error is an erroneous (defective) internal state in the system, which propagates through the system and causes the failure. A fault is the triggering condition which when activated causes an error. Therefore the event of the system failure lies in the end of a causal chain that begins with the activation of a fault under certain operating conditions followed by the propagation of an erroneous internal state through the system.

The definitions of the fault tolerance terms and mechanisms defined in this section are based on (Lee and Anderson, 1990). A **fault-tolerant** system is a system that can continue in operation after some system faults have manifested themselves. Fault tolerance is therefore based on the premise that faults exist and that it is possible for the computer system to handle them without external interventions. The goal of fault tolerance is to ensure that system faults do not result in system failure. Fault tolerance can be achieved through both software and hardware, but throughout this thesis the software mechanisms will be discussed unless otherwise stated. Apart from (Lee and Anderson, 1990), other references which provide extensive coverage of software fault tolerance are (Bishop, 1995; Pullum, 2001).

The MAFTIA project[9] extended the definitions of fault tolerance when reasoning about malicious events and hence tolerance mechanisms aimed at ensuring system security. So the fault-error-failure model described in (Avižienis *et al.*, 2004) is further extended by the concepts of: **attack** – a malicious interaction *fault,* through which an attacker aims to deliberately violate one or more security properties (i.e. an *intrusion* attempt); **vulnerability** – a fault created during development of the system, or during operation, that could be exploited to create an *intrusion;* **intrusion** – a *malicious*, externally-induced *fault* resulting from an *attack* that has been successful in exploiting a *vulnerability.* This relationship between attack, vulnerability and intrusion is referred to as the **AVI security model** in MAFTIA

[9] http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/maftia/

terminology (Verıssimo et al., 2003), and an intrusion-tolerant system is one that can continue in operation in the presence of vulnerabilities and attacks. The goal of **intrusion tolerance** is to ensure that system vulnerabilities and attacks do not result in successful intrusions (cf. analogy with the fault tolerance definition from the previous paragraph).

The central theme of this thesis is the use of *diverse* security products to increase the security of a system. Software *design diversity* is the phenomenon of *bespoke development* or *reuse* of multiple diverse versions of a software *program* (or existing *product*) from a common requirement specification with the goal of increasing the system reliability or availability. The intuitive underlying principle of design diversity is the simple longstanding belief that "two heads are better than one" and its advocacy for use with computer systems may be thought of as first being proposed by Charles Babbage(Babbage, 1982)[10] although by computer he meant a person.

The main reason for employing design diversity in software is due to software suffering from *design faults* (Littlewood et al., 2001) and not *physical faults* (such as wear-and-tear for example) which are hardware specific. A *design fault* in its simplest definition is a fault that is introduced in the software during its development (hence the word *design* in this context is used for the whole software *development* process). If non-diverse redundant copies of the same software product are used then these design faults will be simply replicated across the copies. Such replication of faulty software elements fails to enhance the fault tolerance of the system with respect to design faults.

The ideal goal of employing design diversity is to achieve *negative dependence* between the failure modes of the software products (i.e. whenever one fails the other one does not). *Independent failure* modes of the channels that constitute the diverse system would also be highly desirable as they would enable an assessor to easily calculate the probability of failure of the diverse system: the product of the failure probabilities of the individual channels in the diverse system would give the failure probability of the diverse system. However virtually all of the experimental studies performed for measuring the benefits of design diversity (Chen &wiki Avizienis, 1995; Kelly & Avizienis, 1983; Knight & Leveson, 1986) have found faults, which cause

[10] The paper from Charles Babbage titled "*On the Mathematical Powers of the Calculating Engine (Unpublished manuscript, December 1837)*" can be found on the aforementioned edited book by Brian Randell.

coincident failures in more than one version with probability significantly higher than would be expected if the versions truly had independent failure modes.

Additionally, in the context of security tools which rely on updates of **rules/signatures** to detect malicious events, the diversity in their behaviour may come from the diversity in their rules and the frequency with which the tool vendors update these rules.

In summary, there is a lot of literature on design diversity: a more thorough review of the effectiveness of design diversity (both experimental results and probabilistic modelling) is given in (Littlewood et al., 2001); design aspects are discussed in (Strigini, 2005). Diversity for security is reviewed more thoroughly in (Littlewood and Strigini, 2004), and in a more recent study in diversity for security in (Garcia et al., 2014). But there has been only sparse research (e.g. Gupta, 2003; Singh et al., 2003) on how to choose among alternative layered defences.

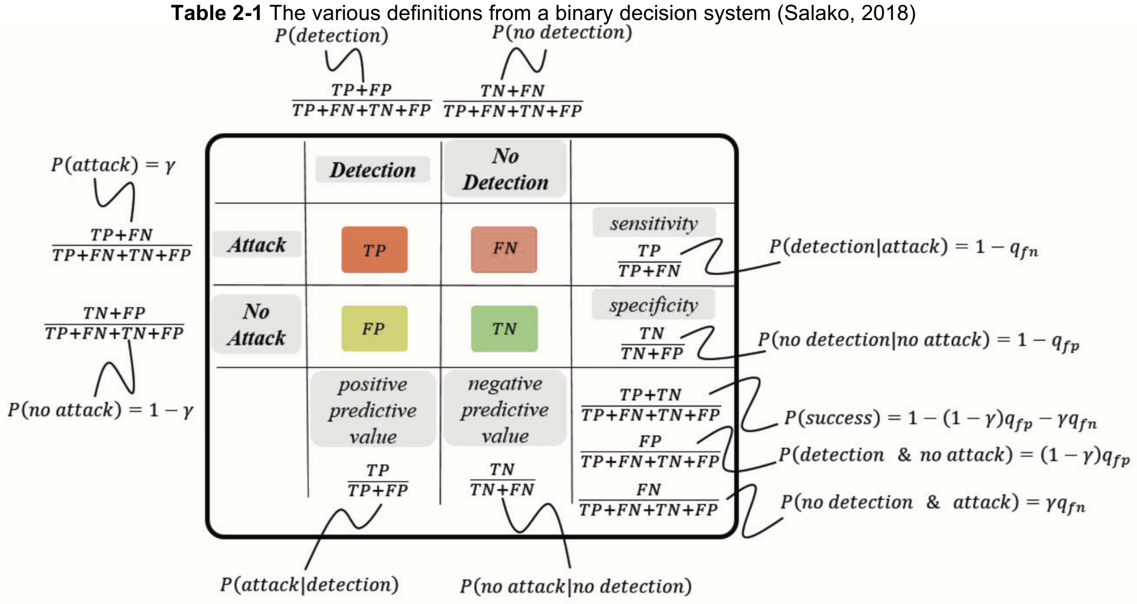
2.2 Assessment of a binary decision system

A binary decision system (or a binary classifier) is a system which given some input (e.g. piece of code, network traffic etc.) produces an output selected from two (binary) choices: e.g. true or false, alarm or no alarm etc. For most of the assessment work we will present in this thesis, the inputs are divided into two categories, e.g. malicious or non-malicious, vulnerable or non-vulnerable etc. In that context, if there are two outputs for each of these two inputs, then there can be four different categories of outcomes. Using an example of code, and the outputs from an SAT that labels the code as vulnerable or non-vulnerable, then these four categories are:

- For code that is not vulnerable:
  - **False Positive (FP)**: the binary decision system incorrectly determines that the code is vulnerable;
  - **True Negative (TN)**: the binary decision system correctly determines that the code is not vulnerable.
- For code that is vulnerable:
  - **False Negative (FN)**: the binary decision system incorrectly determines that the code is not vulnerable;
  - **True Positive (TP)**: the binary decision system correctly determines that the code is vulnerable.

From the four categories above, various measures can be calculated. A concise summary is given in Table 2-1 (Salako, 2018) below.

Table 2-1 The various definitions from a binary decision system (Salako, 2018)



	Detection	No Detection	
Attack	TP	FN	sensitivity $\frac{TP}{TP+FN}$
No Attack	FP	TN	specificity $\frac{TN}{TN+FP}$
	positive predictive value $\frac{TP}{TP+FP}$	negative predictive value $\frac{TN}{TN+FN}$	$\frac{TP+TN}{TP+FN+TN+FP}$ $\frac{FP}{TP+FN+TN+FP}$ $\frac{FN}{TP+FN+TN+FP}$

$P(detection)$
$\frac{TP+FP}{TP+FN+TN+FP}$
$P(no\ detection)$
$\frac{TN+FN}{TP+FN+TN+FP}$
$P(attack) = \gamma$
$\frac{TP+FN}{TP+FN+TN+FP}$
$\frac{TN+FP}{TP+FN+TN+FP}$
$P(no\ attack) = 1 - \gamma$
$P(detection|attack) = 1 - q_{fn}$
$P(no\ detection|no\ attack) = 1 - q_{fp}$
$P(success) = 1 - (1-\gamma)q_{fp} - \gamma q_{fn}$
$P(detection\ \ \&\ no\ attack) = (1-\gamma)q_{fp}$
$P(no\ detection\ \ \&\ \ attack) = \gamma q_{fn}$
$P(attack|detection)$
$P(no\ attack|no\ detection)$

Given the nature of the systems that are presented in this thesis (IDSs, AVs and SATs), the outputs of which can usually be classified as true or false, and the inputs are of two types (malicious or benign), then we looked into the best practice in assessment of these types of systems and the types of measures that are used in the literature. We found that ROC graphs are used extensively for visualising the performance of different systems, and the two axes in the ROC curve that are used more extensively are sensitivity and specificity (or sensitivity vs 1-specificity). In what follows we will give a more detailed description of ROC graphs, their origins and their applications.

2.2.1 ROC background

A Receiver Operating Characteristic (ROC) is a graphical plot for illustrating, organizing and choosing classifiers based on their performance. Decision makers plot the (ROC) curves for each system of interest when analysing the decisions of a binary classifier. ROC graphs are used in signal detection theory to represent the trade-off between false alarms and true alarms (P. Egan, 1975; Swets, Dawes and Monahan, 2000).

The first use of ROC was during World War II to analyse the signals of radars. ROC analysis was developed to overcome the minor radar discrepancies between different types of aircraft, mainly to correctly detect Axis (in this case Japanese) aircraft from radar signals (Hopley and Schalkwyk, 2011).

ROC graph is a two-dimensional graph, where the usual practice is for the X-axis to plot the false positive rate (FPR) and the Y-axis to plot the true positive rate (TPR). An ROC graph shows the relation between the benefits (in TP) with the costs (in FP). Figure 2-1 illustrates an ROC graph and the five points (A, B, C, D and E) represent different classifiers.

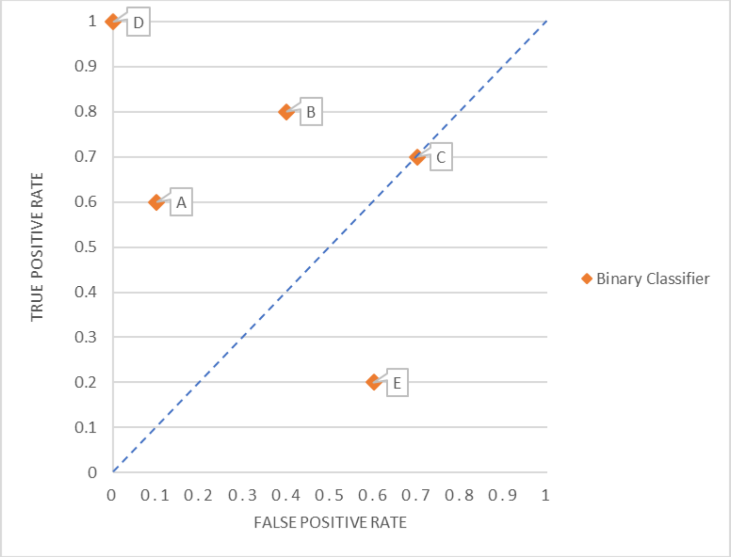


Figure 2-1 ROC graph

Each classifier provides a “*FPR*, *TPR”* pair corresponding to a single point in the ROC graph. The lower left corner (0, 0) represents the no positive classification, means no false positive error and no true positives also. In the upper right corner (1,1) represents the point where there are no false negatives, but also no true negatives (i.e. the system alerts for all demands). (0,1) is the optimal classification point (no false negatives and no false positives): in the ROC graph above system D is perfect. The ROC graph allows an analyst to visually compare the performance of the different classifiers. For example; classifiers on the left-hand side and close to the X-axis can be conservative because they produced low true positive and few positive errors (FP). However, classifiers on the upper right-hand can be liberal because they produced high false positive rate and few true positive errors (TP). In the figure classifier A is more conservative than B.

In this thesis we have primarily used sensitivity and specificity measures, as they tend to be the measures most commonly used in literature. Though the other measures listed in Table 2-1 above can be derived from the FP, TP, FN and TN counts which we provide in Chapters 6, 7 and 8.

2.2.2 Applications of ROC graph analysis

ROC analysis for visualising the diagnostic systems behaviour (Swets, 1988) is to a large extent based on the experience from the extensive literature in the medical decision making. ROC graphs are used extensively in the medical field. A few examples are: (Rangayyan et al., 1997) where ROC graphs are used to analyse the results of different cases (benign and malignant) mammogram scans for early diagnosis of breast cancer; in (Buscombe et al., 2001) ROC graph analysis was used to investigate if a combination of x-ray of mammography and scintimammography[11] was more accurate than a single test alone for patients with suspected primary breast cancer; etc.

In the computer science area ROC analysis is used extensively in the machine learning area. An example is in (Spackman, 1989) where ROC analysis was used to compare and evaluate different machine learning algorithms. The use of ROC graphs in machine learning field is particularly prominent because of a need for a more holistic analysis than just using accuracy metric, which is considered a poor metric for measuring performance (Provost and Fawcett, 1997).

2.3 Threats

There are a multitude of threats that affect computer systems, from vulnerabilities in applications, to attacks from various adversaries (from script kiddies to nation states). In this thesis we have looked into malware, and attacks and vulnerabilities that affect web applications (including XSS and SQLi). We concentrated on these types of malware and attacks as they are the most prevalent types of threats that affect systems exposed to the Internet.

2.3.1 Malware collection and analysis

Malware, or malicious software, is any software or file that typically disrupts, damages, takes control of computer, gains unauthorised access to a computer system or gathers information from a computer. The malware can be categorized into types depending on the malware's method of operation (Aycock, 2006).

Several dynamic malware analysis solutions exist on the Internet as online services. They offer a website and sometimes an API to submit

[11] **Scintimammography** is a type of breast imaging test that is used to detect cancer cells.

malware samples. A report of the observed behaviour is generated and provided to the user. The authors in (Bayer *et al.*, 2006) introduced a tool to dynamically analyse the execution behaviour of Windows executable files which is called Anubis. This tool was made available online (Seclab, 2011). (Willems et al., 2007) introduced Malware Analysis CWSandbox (CWSandbox, 2013) which is currently available at (VMRay, 2014) and is another online service for malware analysis which later became the commercial solution called Threat Analyzer[12]. VMRay is another example of a commercial tool (VMRay, 2014). Malwr.com provides a public web interface for Cuckoo sandbox, a tool dedicated to malware dynamic analysis.

Over the past several years, researchers and practitioners have used honeypots to learn about attacks, attackers and malware. These systems can be categorised as security tools whose value lies in being probed, attacked, or compromised (Spitzner, 2002). These carefully monitored systems allow security researchers to attract hackers, analyse their actions and profile them (Ramsbrock et al., 2007). Additionally they can also be used to collect the latest malware samples that attackers use. Honeypot systems can be found at different scales: from a single host to more complex honeypot networks. These networks, also called honeynets, can be deployed on a few IP addresses within a local network. The project Leurre.com (Pouget, 2005), SGNET (Leita and Dacier, 2008) and the honeynet initiative from CAIDA (Vrable et al., 2005) are all examples of distributed honeypot networks that have been used in the past and deployed in different locations. Not all the data that is collected in the honeynets is necessarily harmful – for example, there may be test data that the attackers have uploaded, or malware may be “malformed” during the uploading process making the files innocuous. Hence experiment designers need to be careful to analyse and if necessary filter the data before using the collected data as malicious inputs in experiments.

2.3.2 SQL Injection

SQL Injection vulnerabilities exist due to user inputs that are not adequately validated (HTTP POST and GET variables, cookies, server variables, database values, etc.). In practice, when such inputs are not correctly sanitised, an attacker may be able to exploit them to maliciously inject new SQL commands and/or modify the logic of the existing ones. In this way, the attackers may read sensitive data from the database, modify it, and

[12] https://www.threattrack.com/malware-analysis.aspx

even execute administration commands (Barnett, 2015; W3Schools, 2016; OWASP, 2017). According to several reports (IBM Security Solutions, 2013; Acunetix, 2015; OWASP, 2017) SQL Injection attacks are considered one of the most dangerous types of attack for web applications and at the same time among the most common.

Given the relevance of SQL injection attacks countermeasures have been investigated (Halfond et al., 2006). The best defence from SQL injection attacks is through preventive identification of vulnerabilities. However, the ubiquity of such vulnerabilities (IBM Security Solutions, 2013; Acunetix, 2015) shows that this approach is often disregarded due to time and resource constraints. For this reason, systems capable of runtime detection and prevention of intrusions have received a lot of attention (Buehrer, Weide and Sivilotti, 2005; Halfond and Orso, 2005). In fact, a significant amount of research has been undertaken to create novel methods for detecting SQL Injection attacks (Lee et al., 2012; Sharma et al., 2012; Shar and Tan, 2013). However, security practitioners are still concerned with the effectiveness of IDSs (Richardson, 2011).

2.3.3 Cross-site scripting (XSS)

Cross-site scripting vulnerabilities appeared in 1996, in the first few years of the World Wide Web (Fogie et al., 2007). In 1999, Georgi Guninski (security researcher) was working on finding flaws in Internet Explorer’s (IE) security model. David Ross (security researcher at Microsoft) was influenced by the work of Georgi Guninski. Ross published the first paper on the cross-site scripting vulnerabilities “Script Injection” (Fogie et al., 2007). Ross explained that the script content can be used to bypass the same security controls and the fault can exist on the server-side. In the XSS vulnerability the attacker can exploit the vulnerabilities by injecting malicious scripts in the new web page. The browser renders the page and executes the scripts in the victim’s machine as a trusted script which can hijack user sessions, deface web sites, or redirect the user to malicious sites (OWASP, 2017)

2.3.4 Examples of SQLi and XSS vulnerabilities

OWASP provides the top ten most critical web application security risks. SQLi and XSS vulnerabilities are in the list, and are particularly damaging (Halfond and Orso, 2005; OWASP, 2017). According to several reports (IBM Security Solutions, 2013; Acunetix, 2015; OWASP, 2017) , SQL Injection attacks are considered one of the most dangerous types of attack for web applications and at the same time among the most common. An SQLi vulnerability occurs when untrusted data flowing from Entry Points (EPs, e.g. user input) with inadequate input validation (i.e. inadequate analysis of the data against a predefined patterns) is used for constructing SQL queries. An attacker may explore these data flows and execute queries not expected by the application developer or may access sensitive data without proper authorisation (OWASP, 2017). These vulnerabilities occur whenever data input coming into applications from untrusted EPs is not validated, sanitised or escaped and flows through the application reaching Sensitive Sinks (SSs) (see Figure 2-2). An SS is a call of a function that exposes private data to external systems. An example for SQLi is the PHP mysql_query function, which executes an SQL query and returns the results. The PHP print function that outputs HTML/JavaScript to the browser is an example of an SS for XSS. XSS vulnerabilities occur whenever an application includes untrusted data in a new web page without proper validation, sanitisation or escaping (OWASP, 2017).

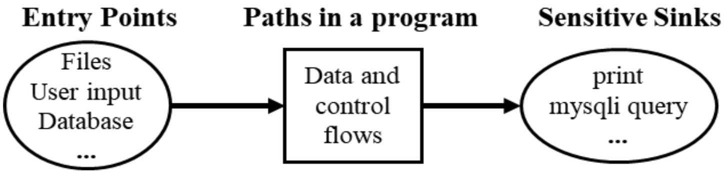
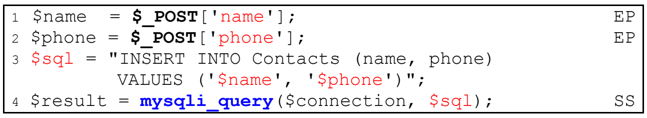


Figure 2-2 Data flow vulnerabilities

To explain how SQLi vulnerabilities occur, and how they can be exploited, we use a PHP script example (Figure 2-3). The script inserts contacts data (name and phone) in a database without any validation. In this script, there are two EPs, (`$_POST` array at lines 1 and 2) and one SS (line 4). The data flowing from the EPs to the SS are not validated, so there is one SQLi vulnerability in line 4.



```
1 $name  = $_POST['name'];                            EP
2 $phone = $_POST['phone'];                           EP
3 $sql  = "INSERT INTO Contacts (name, phone)
          VALUES ('$name', '$phone')";
4 $result = mysqli_query($connection, $sql);          SS
```

Figure 2-3 PHP code for inserting contacts in a database.

The PHP script in Figure 2-4 shows an example of an XSS vulnerability. It searches contacts by name in the database and displays the results in an HTML page. The user provides the name (EP, line 1) to be searched through the `$_GET` array parameter. The script outputs the value of the parameter without any proper escaping (line 2). In this case, there is one *reflected* or *first order* XSS vulnerability. In reflected XSS the untrusted data coming from the user is immediately written back. The exploitation of this class of vulnerabilities requires some kind of social engineering by the attacker to convince the victim to click in the crafted URL. In line 3, the script makes use of the same parameter (`$name`) to build the SQL query to be sent to the database server (line 4). In this case, there is also one SQLi vulnerability. Similarly to the data flowing from the user input, the database is also a source of untrusted data due to inappropriate validation when inserted in the database, as shown in Figure 2-3. In fact, any attacker can insert in the database malicious code instead of a valid contact name. Therefore, the PHP statement in line 7 is an EP in the application that retrieves untrusted data from the database. These data are outputted in lines 9 and 10 without any escaping, hence two *stored* or *second order* XSS vulnerabilities exist. This class of vulnerabilities is especially dangerous because it does not require any kind of social engineering to trick the victim and a single piece of malicious code stored in the database can be executed in the browser of all users visiting the website.

```
1  $name = $_GET['name'];                                        EP
2  print("<h1>Your search for: $name</h1>");                     SS
3  $sql="SELECT * FROM Contacts where name like '%$name%'";
4  $result=mysqli_query($connection, $sql);                      SS
5  echo '<table><tr><th>Name</th><th>Phone</th></tr>';
6  $n=0;
7  while($row = mysqli_fetch_array($result)) {                   EP
8    echo '<tr>';
9    echo '<td>' . $row[name] . '</td>';                         SS
10   echo "<td>{$row['phone']}</td>";                            SS
11   echo '<tr>'; $n++;
12 }
13 printf("Total records: %d", $n);                              SS
```

Figure 2-4 PHP code for searching contacts in a database

For a given class of vulnerability one *line of code* (LOC) is potentially vulnerable if it contains an SS function call with at least one parameter (Nunes et al., 2017). A vulnerable line of code (VLOC) is a LOC with an SS and a variable with data coming from EPs without any validation. A non-vulnerable line of code (NVLOC) is a LOC with an SS where all variables are sanitised (Rutar et al., 2004; Kiezun et al., 2009; Backes et al., 2017) Lines 2, 4, 9 and 10 of the script in Figure 2-4 are examples of VLOCs and line 13 is an example of a NVLOC.

2.4 Related work on assessing AntiVirus (AV) products

AV products are one of the most widely used security protection systems. They are usually deployed as the last line of defence on desktop, laptop, tablet and smartphone devices for both home and business use. Studies that compare their detection capabilities are widely available[13].

[13] av-comparatives.org/, av-test.org/, virusbtn.com/index

There are two major platforms that allow for suspicious files to be uploaded for scanning by multiple AV products, namely VirusTotal and Metadefender[14].

VirusTotal is an online service that hosts (at the time of writing) 56 signature-based detection engines from different AV vendors. It is a service that is widely used by both academia and industry to submit and inspect malware samples. It also provides an Application Programming Interface (API) through which multiple malware samples can be submitted.

Metadefender is also an online service that hosts (at the time of writing) 42 signature-based detection engines. It provides a service similar to VirusTotal, and it also provides an API for submitting malware samples.

Both of these services provide a valuable resource to malware researchers and the security community. But they have some limitations when it comes to more in depth analysis of AV detection capabilities:

- Both platforms run signature-based detection engines of these AV products, rather than the full capability products that would run on an end-point, making comparisons with full capability versions of these products difficult. Metadefender states the following on its "Statistics" page[15] " *Please note that the detection data comes from Software Development Kit (SDK) and Command Line Interface (CLI) package versions of these anti-malware engines, using static analysis only, and not from endpoint desktop applications which may be capable of enhanced behavioural and other dynamic analysis, so detection results may differ significantly from commercial endpoint performance. The data below should not be used for comparing performance of desktop or server anti-malware applications".*

  VirusTotal states the following on its "About" page: "*VirusTotal's AV engines are command-line versions, so depending on the product, they will not behave exactly the same as the desktop versions: for instance, desktop solutions may use techniques based on behavioural analysis and count with personal firewalls that may decrease entry points and mitigate propagation*".

[14] https://www.metadefender.com/

[15] https://www.metadefender.com/stats#!/1

- Both platforms are essentially “black box” testing platforms. In other words, the user submits a file, and gets a response on whether the file was detected as malicious, which AV products detected it as such, and the signature used for the detection. But they do not provide more detail about *when* the file was detected (e.g. “on entry” – before being downloaded on the endhost; after it was downloaded but without forcing a scan; only after a scan is performed), or on which operating system the AV product was running when it detected (or not) a file as malicious. This makes it more difficult to assess the potential damage that a file may cause on the end host before malware is actually detected, and whether it would have caused any damage at all on a given operating system (some malware could be malicious on one operating system, but innocuous on another).
- Both platforms, on their free versions, put restrictions on the number of files that can be uploaded at any given time (which is reasonable to allow fair use). For researchers wishing to upload thousands of malware files on a daily basis, they need to pay a fee, which can run into several tens of thousands of dollars per year.

Empirical analyses of the benefits of diversity with diverse AV products are rare. We know of three published studies that have looked at the problem.

A study with a deployment of the Cloud-AV implementation in a university network over a six month period is given in (Oberheide, Cooke and Jahanian, 2008) . For the executable files observed in the study, the network overhead and the time needed for an AV product to make a decision are relatively low. This is because the processes running on the local host, during the observation period, could make a decision on the maliciousness of the file in more than 99% of the cases they examined a file. The authors acknowledge that the performance penalties could be much higher if more types of files than just executables are examined, or if the number of new files observed on the host is high (since the host will need to forward the files for examination to the network service more often).

The work in (Bishop et al., 2011; Gashi et al., 2009) uses the VirusTotal service for the analysis. The analysed dataset is composed of 1599 malware samples collected by a real world honeypot deployment – SGNET (Leita and Dacier, 2008) SGNET is a distributed honeypot deployment for the observation of server-side code injection attacks. The dataset in (Bishop et al., 2011; Gashi et al., 2009) was collected in the period February to August

2008. Each sample in the SGNET dataset was resubmitted to VirusTotal on a daily basis for a period of no more than 30 days. Further results on diversity of AV products, with newer datasets and different data collection environments, were also presented in (Gashi et al., 2013). Since these studies are based on results obtained from interaction with VirusTotal, they suffer from the imitations highlighted earlier, namely that VirusTotal uses the signature based detection engines of these AV products, rather than the full capability versions; VirusTotal gives the results on whether an AV product detected a malware or not (and when it detects it, the signature that the AV product assigned to that malware), but does not tell us when the malware was detected, or on which operating system the AV product was running when the detection happened etc.

2.5 Related work on assessing Intrusion Detection Systems (IDSs)

A very extensive survey on evaluation of intrusion detection systems is presented in (Milenkoski et al., 2015). This survey analyses and systematises a vast number of research works in the field. The main features analysed in the survey are the workloads used to test the IDSs, the metrics utilised for the evaluation of the collected experimental data, and the used measurement methodology. The survey demonstrates that IDS evaluation is a key research topic and that one of the main benefits that IDS evaluation can bring is related with guidelines on how to improve IDS technologies.

Several works try to provide a comprehensive analysis of IDSs for SQL injection (Tajpour et al., 2013), (Shrivastava and Tripathi, 2012). However differently from what is suggested in (Milenkoski et al., 2015), in most cases the comparison is limited to a review of the features and characteristics of the IDSs and does not deal with the actual verification of their real effectiveness by testing their capabilities.

2.6 Related work on assessing diverse configurations of Static Analysis Tools (SATs)

(Rutar, Almazan and Foster, 2004) studied five well-known SATs on a small set of Java programs with different sizes, and from various domains. They concluded that the results of each tool are highly correlated with the techniques used for finding bugs, and that no single tool can be considered the best to detect defects. They proposed a meta-tool to automatically combine and correlate SAT outputs. This meta-tool is based on a set of scripts that combine the results of the various tools in a common format. The bugs found were not manually reviewed, thus, there is no distinction between TP and FP. The metric used to evaluate and compare the tools was the number of bugs that each SAT found.

(Na *et al.*, 2008) proposed an approach to merge the results of multiple SATs. The user specifies the programs to be analysed and chooses the classes of bugs to be scanned. After determining which tools can search for the specified class of bugs, it generated the necessary tool configurations, ran the tools, combined the outputs in a single report, and applied two prioritizing policies to rank the results. They used this approach to conclude that developers could benefit by using more than one SAT. However, neither were the SAT outputs classified as TP and FP nor did the authors propose any metric to evaluate the approach. The workload was composed of a small Java program that is not representative of real applications. Therefore, with such limited validation, it is very difficult to assess the strength and drawbacks of the solution.

(Wang *et al.*, 2008) proposed an approach that combines multiple SATs in a simple Web Service. The user has the possibility to upload the source code and auxiliary information such as the programming language and the classes of bugs to be scanned. The tools perform the analysis of the source code and the results are merged in such a way that the same defect is reported only once. The experiments were quite limited, having just a single Java test case, and the approach was evaluated in terms of the running time when combining two SATs, lacking the validation of the effectiveness of the vulnerability detection.

The NSA Centre for Assured Software (CAS) specified a methodology, the CAS Static Analysis Tool Study Methodology, that measures and rates the effectiveness of SATs and combination of SATs in a standard and repeatable manner (NIST, 2018). The metrics used are Precision, Recall, F-Score, and Discrimination Rate (DR). A discrimination occurs if a SAT reports a vulnerability in the vulnerable test case (TP) and keeps quiet in the non-vulnerable test case (TN). The CAS created a collection over 81,000 synthetic C/C++ and Java programs with known flaws, which was called the Juliet Test Suite (NIST, 2018). Each test case is a slice of artificial code having exactly one flaw and at least one non-flaw construct similar to the vulnerability. In

2011, the CAS conducted a study with the purpose of determining the capabilities of five SATs for C/C++ and Java (Britton, 2011). In this study, the authors proposed the combination of two SATs to show that adding a second SAT might complement the first one. However, the evaluation of the combinations is limited because it is based on the Recall and DR metrics. The Recall does not consider the number of FP reported, and the DR severely penalises the SATs that report both many vulnerabilities and many FP. Furthermore, they also evaluated the overall coverage (Recall) of four combinations of five different SATs. They concluded that the Recall increases as the number of tools increases.

(Nunes et al., 2017) studied the problem of combining the outputs of diverse static analysis tools to detect web vulnerabilities. They argued that the use of two or more SATs might be helpful, as more vulnerabilities may be reported. However, the drawback is that the number of FPs may at the same time increase. Moreover, they claim that the acceptable/expected outcome of the static analysis process (in terms of TPs and FPs) depends on the software development scenario. They considered four software development scenarios with different goals and constraints, ranging from low budget to high-end (e.g. business critical) applications. For each scenario, they used one main metric to rank the combinations of the tools and a tiebreaker metric used only when there is a tie. For example, for the high-end scenario the main metric is the Recall and the tiebreaker metric is the Precision. In this scenario, the goal is to find the highest number of vulnerabilities at any cost. Therefore, the Recall metric captures this goal. They evaluated all 32 1-out-of-N adjudicator combinations of the outputs of five free SATs finding SQLi and XSS vulnerabilities in a workload composed by 134 WordPress plugins organised by scenarios. The results showed that the best solution depends on the class of the vulnerability and on the scenario. In fact, the best solution never includes all the SATs and in some cases, a single SAT performs better than the best combination of SATs. The approach is based on a considerable amount of software in production characterised in terms of vulnerable and non-vulnerable lines of code for a more precise classification of the outputs of the tools with respect to TP and FP. One limitation of this work is that all the combinations are based on 1-out-of-N adjudicator combinations. For instance, in the less stringent scenario, where the resources for vulnerability fixing are very low, every FP is an important cause of concern. In this case, N-out-of-N adjudicator combinations might increase the confidence of the results. For instance, a potential vulnerability reported by more than one SAT has higher probability to be a true vulnerability than one reported only by one tool.

2.7 Gaps in the Literature

The previous sections gave an overview of the type of research in security in relation to design diversity and defence in depth. Few of the pervious works have provided empirical assessment of the benefits, or harm, resulting from defence in depth, or what adjudication mechanisms may be used when diverse systems are deployed. In particular, there is not enough research in previous studies on assessing the benefits of diversity and defence in depth for security of binary decision systems. The rest of the thesis provides details of the work done to address these gaps.

(3) METHODOLOGY

3.1 Introduction

In this chapter we present the analysis methodology that we have used in the thesis to assess the diversity with AV, IDS and SAT products. The methodology we used was influenced by the objectives we set-out at the start of the thesis. Namely, to study:

- the variety of architectural options about how diverse security controls are assembled and their responses combined ("adjudication");
- the interplay between the risks of failing to react to true attacks and of false alarms ("false negative" and "false positive" failures).

For the first objective above, we studied the most common types of adjudications reported in the literature, namely 1-out-N, N-out-of-N and majority voting. Additionally we also applied the "optimal adjudication" function, which was described in the papers by (Giandomenico and Strigini, 1990). For the second objective, we used the extensive prior work on the assessment of binary decisions systems (as referenced in Chapter 2 earlier) and used it to assess diversity in defence systems where the output from the defences can be reduced to a binary decision (e.g. true or false; alarm or no alarm etc).

Note that we could not apply the method in full to all three studies. This is because for the AV products we only had data about malicious events (malware). Hence we can only measure the false negative rate. For the other two studies (namely IDSs and SATs) we applied the methodology in full.

The rest of the chapter is organised as follows: section 3.2 outlines the method for assessing diversity for binary decision systems; section 3.3 presents a justification for why the AVs, IDSs and SATs were chosen in the thesis; and finally section 3.4 concludes the chapter.

3.2 Assessing diversity for binary decision systems

The same as for any binary decision system, we can classify the decisions of a binary decision security defence system into four classes:

- For benign inputs (e.g. benign crawling actions, non-vulnerable code, non-malicious files etc):
    - *False Positive (FP)*: the security defence system, incorrectly, determines that a benign input is malicious;

   - *True Negative (TN)*: the security defence system, correctly, determines that a benign input is not malicious.

- For malicious input (e.g. attacks, vulnerable code, malicious software etc.):
   - *False Negative (FN)*: the security defence system, incorrectly, determines that a malicious input is not malicious;
   - *True Positive (TP)*: the security defence system, correctly, determines that a malicious input is malicious.

In our work we extend the analysis from the viewpoint of diversity. We analysed all possible pairs, triplets,…, "N-plets", that can be constructed with the N versions of the security defences that we study[16]. The decision that a N-version defence system would make on a given input will depend on how it does the voting/adjudication on the results it receives from each of the individual systems. We studied the following adjudication schemes:

- 1-out-of-N (abbreviated 1ooN): an input is labelled as malicious as long as ANY one of the N versions in the defence system determines that the input is malicious;
- N-out-of-N (abbreviated NooN): an input is labelled as malicious only if ALL the N versions in the defence system determine that the input is malicious;
- r-out-of-N (abbreviated rooN[17]): an input is labelled malicious only if *r* out of the N versions in the defence system determine that the input is malicious;
- Optimal adjudication: an input is labelled malicious according to a cost function (optimal adjudicator) that minimises the overall cost of failure. More on this in Chapter 8.

In practice, each system is being asked whether a given input is malicious or not. Hence, we use the following conventional statistical measures of the performance of a binary classification test[18]:

[16] For AVs we decided to study the differences between versions of the product from the same vendor instead, as there has already been prior work in assessing diversity between products from different vendors, as we referenced in the previous chapter.

[17] Where r = (N/2)+1 when N is an even number; and r = ((N-1) / 2) +1 when N is an odd number.

[18] Very clear definitions with examples are provided in the following article: https://www.medcalc.org/manual/roc-curves.php

- **Sensitivity (True Positive rate)** – measures the performance of the security defence in detecting malicious inputs;
- **Specificity (True Negative rate)** – measures the performance of the security defence in not raising alarms for benign inputs;
- **Accuracy** – measures the combined performance of a security defence against malicious and benign inputs.

There are many other measures that can be used. We already provided some of them in the Literature Review (specifically section 2.2). A review of these and similar measures in the context of security is provided in (Antunes and Vieira, 2015). We chose the measures above as they are the most widely used and most easily understood, as evidenced from their widespread use in both security and medical fields. In any case, all the other measures are easily derived from the FP, FN, TP and TN counts, which we provide in full for each study. So practitioners can derive the other measures of interest directly from the counts above.

The analysis approach followed in this study provides an analyst with a useful methodology to help them with decision problems when configuring defence in depth architectures. Some of the problems for which we provide useful guidance are:

- How to choose diverse systems from a pool of similar products to improve the overall system security?
- If an organisation is already running a defence system and would like to add another that best complements the existing one (in terms of low false positives and low false negatives), which one should be chosen?
- What are the trade-offs between false positives and false negatives when analysing diverse defence systems in different applications and different attack profiles?

Our analysis methodology, though based on well-known and established techniques for analysing decision-based systems, should also prove useful to other researchers who need to analyse combinations of diverse protection systems (such as firewalls, or combinations of functionally different defences) etc.

Decision makers also plot the Receiver Operating Characteristic[19] (ROC) curves for each system of interest when analysing the decisions of a binary classifier. ROC curves are used to determine how a *threshold* should be set for a decision system to get an optimal configuration[20] that maximises the TP and minimises the FP rates. However, since the systems in our case are already pre-configured, the ROC plots show only a point for each system. By showing all the points for the single and diverse systems in the same plot, we can visualise which systems are configured optimally for a given application.

In summary, in our analysis we did the following for each study (and each application in the IDS and SAT studies):

- We calculate the FP, FN, TP, TN counts for each diverse configuration;
- We calculate the measures of interest (specificity, sensitivity and accuracy) or each diverse configuration, overall and by type of malicious input;
- We generate the ROC plots showing all the diverse configurations and the individual defence systems, overall, by type of cconfiguration and by type of malicious input;
- We calculate the differences in the measures of interest between diverse configurations and individual systems to measure the possible improvements or deteriorations from switching to a diverse system.

3.3 Justification for choosing to study diversity with AVs, IDSs and SATs

We have conducted experiments with three types of defence tools: AV products, Intrusion Detection Systems (IDS) and Static Analysis Tools (SAT). The main reasons for choosing to study these products are as follows:

- **Deployment in real systems**: AV products and IDSs are some of the most widely used security defence tools in both personal computing and commercial/organisational settings. The practical advice from any operating system vendor when starting up a machine is to make sure that an AntiVirus product is installed before connecting to the interne (or

[19] https://www.medcalc.org/manual/roc-curves.php

[20] What is “optimal” for a give system will inevitably depend on the relative cost that the decision maker assigns to the FP and FN failures.

connecting some external device to the machine). Similarly, IDSs are widely used in organisations as another layer of defence to complement the defences that are deployed on the end hosts. SATs are used at a different stage of a system lifecycle (i.e. during its development), but can also be used to test existing code bases for potential vulnerabilities. As such they are also widely used for software developers for vulnerabilities, especially if that software will be exposed to the web.

- **The availability of multiple products in the same product family**: since the primary aim of our research is the assess diversity, we chose product families that had multiple products from different vendors available for them, and for which there is a sufficient number of open source/freeware versions available. There are other product families for which there are multiple vendors, firewalls being the most clear example, but most of the products available in this family are prohibitively expensive for academic research.
- **The availability of datasets that facilitate assessment of diversity**: one way in which we thought we could make a substantial contribution in research, in a time efficient manner, was to leverage the very good work done by other researchers and build on top of it. This has the advantage of fostering collaboration between different institutions, and also making sure that the work is reviewed from multiple sites, enabling a higher level of scrutiny of the results and hence leading to better quality of research. To this end we approached colleagues in Europe and US who had extensive data collection infrastructures, who had published the datasets so that the data had already been scrutinised by the community, but who had not analysed the data from a diversity perspective. The three datasets we used were extensive, very well documented, included multiple products and had research colleagues who were available to answer questions we had about the datasets, how they were collected etc.

3.4 Conclusions

In this chapter we presented the methodology that we have used to perform the diversity assessment with AV, IDS and SAT products. We will reference this methodology in the subsequent chapters where the results of the diversity assessment with these products are presented in more detail. As we mentioned, the entire methodology could not be applied to the all the products, so we will make it clear in subsequent chapters which part of the methodology presented here applies in that context.

(4) DIVERSITY WITH ANTIVIRUS PRODUCTS (AV)

This chapter compares the full capability AV and signature based AV products.

Most sections in this chapter have been published in:

- Algaith, A., Gashi, I., Sobesto, B., Cukier, M., Haxhijaha, S. & Bajrami, G. (2016). ***"Comparing Detection Capabilities of AntiVirus Products: An Empirical Study with Different Versions of Products from the Same Vendors"***. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops. (pp. 48-53). IEEE. ISBN 978-1-5090-3688-2. Available at: http://openaccess.city.ac.uk/15503/

4.1 Introduction

In this chapter we report results of an empirical analysis of the detection capabilities of nine AV products when they were subjected to 3605 malware samples collected on an experimental network over a period of 31 days in November-December 2013. We compared the detection capabilities of the version of the AV products that the vendors make available for free in VirusTotal versus the full capability products that they make available via their own website. The analysis has been done using externally observable properties of the AV products: namely whether they detect a given malware, and, if they detect it, what label they assign to that malware.

The methodology we described in Chapter 3 could only be applied with respect to sensitivity assessment, since we only have malicious inputs (malware samples) that were sent to the AV products.

The rest of the chapter is organised as follows: section 4.2 outlines the objectives of the study; section 4.3 describes how the data was collected; section 4.4 presents a summary of the results; section 4.5 presents a comparison of the detection capabilities of the two versions of AV from each vendor; section 4.6 visualises the dataset over the three dimensions of analysis (dates, malware and AV), section 4.7 presents an analysis over time; section 4.8 presents an analysis of the labels used by AVs for malware; and finally section 4.9 presents a discussion, conclusions and provisions for further work.

4.2 Study objectives

The main objective of this experiment is to compare the detection rates of full capability products versus the version that the vendors make available for free via VirusTotal, which can help security researchers, who may have results from experimentations with VirusTotal alone, to compare and improve their estimates. Analysis of the labels/signature that AV products assign to the malware that they detect are also presented. The labels may be helpful to researchers and administrators with diagnosing the type of malware they are dealing with in their systems, and to check whether knowledge gained from analysing labels assigned by AVs in full capability versions are transferable to the labels that they assign to their VirusTotal versions, and vice versa.

4.3 The experimental design

Dionaea[21], a low-interaction honeypot tool that emulates common Internet services, was deployed by the University of Maryland on a distributed honeypot architecture to collect the malware analyzed in this study. 1136 public IP addresses have been used to implement Dionaea. These IP addresses are distributed across six different locations in four different countries: France, Germany, Morocco and the USA. The goal of the study is not to compare the malware collected on the different networks or locations. The goal is to study the detection capabilities for a malware set. The subnets do not have the same size and their configuration differs from one network to another. Note that none of the networks have the same security policy. Besides, these networks are not protected in the same way.

The implemented configuration of Dionaea exposes different Internet services and protocols. For each of these services and protocols, Dionaea emulates vulnerabilities used to trap malware attempting to exploit them. Because of the nature of the vulnerabilities and protocols emulated, we mainly collect Windows Portable Executable (PE) files[22].

Binary files can be captured in different ways and they can have different formats. Only Microsoft Windows PE files were kept for this study. This format is easily identifiable and executable on any Windows operating system.

Once a day, a Perl script fetches the malware and the information relative to their capture (stored in a SQLite database) from the different Linux virtual machines. This script then submits to VirusTotal the entire malware repository for analysis. When using the submission API, VirusTotal returns a scan key for each malware sample submitted. This scan key is composed of the binary's SHA1 hash and the submission timestamp. The Perl program stores the different scan keys returned by the website to later retrieve the analysis reports.

An additional Perl script is executed once all the malware has been submitted. This program uses the previously stored scan keys to fetch the analysis reports for all the malware in the repository.

[21] http://dionaea.carnivore.it

[22] https://docs.microsoft.com/en-us/windows/desktop/debug/pe-format

Everyday a new database entry is created for each malware. This entry contains the information related to the VirusTotal submission. The AV product's names, versions and the malware signature names are also uploaded in various tables and linked together with the file submission.

9 AV products (AntiVir, AVG, Comodo, F-Secure, Kaspersky, McAfee, Microsoft, Sophos and Symantec) were deployed at the University for Business and Technology, Prishtina. The same malware samples collected in the experimental architecture described above were sent to these AV products on the same dates as when they were sent to VirusTotal. These AV products were chosen because:

- they represent some of the most widely deployed AV products on the market;
- experience of using them in the past in experiments.

4.4 Summary results

We start our analysis by describing some initial statistics of the data. As previously mentioned, our empirical analysis is with two versions (VirusTotal version and full capability version) of AV products of nine different vendors when they were subjected to 3605 malware samples collected in our experimental infrastructure, over a 31 day period (11-November-2013–11-December-2013). Hence the inputs to our empirical analysis consist of a series of triplets {AV_i*2, $Malware_j$, Day_k}: a given malware *j* is inspected on a given date *k* by two versions of a given AV_i . For each of these triplets we observe a detection (stored as 0, or no failure), or no detection (stored as 1, or failure). For those triplets where we see a detection we also store the labels which a given AV version assigns to a given malware on a given date.

We send malware to the two AV versions every day from the first date a malware is observed in the honeypots in our infrastructure until the last day of the observation period. However, the total number of triplets we observed was less than 3605 Malware * 31 Days * 9 AVs * 2 Versions. This is because:

- Each day we observed new malware in the infrastructure and we could not send to VirusTotal a malware to be inspected by an AV product from an earlier date.

- Some of the AV products in VirusTotal are not active on certain dates for certain malware and hence we have no results for them, means they do not score neither zero nor one.

In this analysis we use all the triplets {AV$_i$*2, Malware$_j$, Day$_k$} for which we had observations from both versions of a given vendor allowing us to do a like-for-like comparison. Overall we had 958,972 * 2 such triplets, as shown in Table 4-1.

Table 4-1 Counts of detections and non-detections

Detection/Failure (DF)	Full Capability (FC)	VirusTotal (VT)
DF=0 - No failure: Detection	944,718	946,375
DF=1 - Failure: No Detection	14,254	12,597
Total	958,972	958,972

4.5 Comparison of the detection capabilities of the two versions from each vendor

Table 3-2 shows the counts of demands (detected and undetected) for the full capability and VirusTotal versions of the nine vendor products in our study. A demand is a pair {Malware$_j$, Day$_k$} which links a given *malware j* and the *date k* to a given version of an AV that inspected it. The total number of demands for the vendors are comparable, apart from F-Secure for which we did not get responses for several days in VirusTotal.

A surprising first observation is that for seven out of nine of these vendors, the version of their products that they had available in VirusTotal had a better detection rate compared with their full capability products (for each vendor we have highlighted in green which version gives the better detection rate).

Additionally, we were interested in finding out what were the detection rates for the AV products when they first had to inspect a newly observed malware in our honeypots. Table 4-3 shows these results. The ordering is similar to Table 4-2 with the exception of Comodo which fails to detect more malware the first time it encounters it in the full capability version compared with the VirusTotal version.

Table 4-2 Detections and non-detections for all demands

AV Name	DF_{FC}=0	DF_{FC}=1	FR_{FC}	DF_{VT}=0	DF_{VT}=1	FR_{VT}
AntiVir	108,141	455	0.004190	108,171	425	0.003914
AVG	108,051	61	0.000564	106,235	1877	0.017362
Comodo	108,423	148	0.001363	108,101	470	0.004329
F-Secure	91,339	1436	0.015478	91,383	1392	0.015004
Kaspersky	106,348	2146	0.019780	106,380	2114	0.019485
McAfee	103,969	4463	0.041159	106,392	2040	0.018814
Microsoft	105,817	2560	0.023621	105,974	2403	0.022173
Sophos	105,826	2612	0.024088	106,635	1803	0.016627
Symantec	106,804	373	0.003480	107,104	73	0.000681

FR_{FC} : Failure Rate of Full Capability Products

FR_{VT} : Failures Rate of Virus Total Products

Table 4-3 Detections and non-detections for the first inspection of a malware by an AV version in our experiment

AV Name	DF_{FC}=0	DF_{FC}=1	DF_{VT}=0	DF_{VT}=1
AntiVir	3590	15	3591	14
AVG	3603	2	3543	62
Comodo	3600	5	3603	2
F-Secure	3549	56	3550	55
Kaspersky	3535	70	3536	69
McAfee	3410	195	3538	67
Microsoft	3526	79	3526	79
Sophos	3459	146	3546	59
Symantec	3535	70	3604	1

DF_{FC} : Detection Failures of Full Capability Products

DF_{VT} : Detection Failures of Virus Total Products

The two preceding tables give a good overview of the detection capabilities of the versions separately. We then checked in more detail which demands are being detected in one version but not the other (and vice versa). Tables 4-4 and 4-5 give these numbers for all demands (Table 4-4) and for the first inspection of a given malware by a given AV version (Table 4-5). The total of the demands in the different columns of the two tables are as follows. First column: detected by both versions; second column: detected by the full capability product but not by VirusTotal; third column: detected by VirusTotal but not by the full capability product; and the fourth column: failed to be detected by both versions. For three products (AntiVir, McAfee and Sophos) the number of malware that have been detected by the full capability version but not detected by VirusTotal version is zero. For these products there seems to be no gain in detection capability from using the full capability product. For AVG, we have the opposite observation: the full capability product detected everything that the signature based detection engine detects in VirusTotal and more. For the other five vendors, the detection capabilities of the two versions of the product seem complementary: one version detected some demands that the other one cannot, and vice versa.

Table 4-4 Detections and non-detections for all demands on both versions

AV Name	**$DF_{FC}=0$ AND $DF_{VT}=0$**	**$DF_{FC}=0$ AND $DF_{VT}=1$**	**$DF_{FC}=1$ AND $DF_{VT}=0$**	**$DF_{FC}=1$ AND $DF_{VT}=1$**
AntiVir	108,141	0	30	425
AVG	106,235	1,816	0	61
Comodo	108,016	407	85	63
F-Secure	91,337	2	46	1,390
Kaspersky	106,334	14	46	2,100
McAfee	103,969	0	2,423	2,040
Microsoft	105,817	0	157	2,403
Sophos	105,820	6	815	1,797
Symantec	106,760	44	344	29

Table 4-5 Detections and non-detections for the first inspection of a malware by an AV version in our experiment – categorised by counts on both versions per vendor

AV Name	DF_{FC}=0 AND DF_{VT}=0	DF_{FC}=0 AND DF_{VT}=1	DF_{FC}=1 AND DF_{VT}=0	DF_{FC}=1 AND DF_{VT}=1
AntiVir	3,590	0	1	14
AVG	3,543	60	0	2
Comodo	3,600	0	3	2
F-Secure	3,549	0	1	55
Kaspersky	3,535	0	1	69
McAfee	3,410	0	128	67
Microsoft	3,526	0	0	79
Sophos	3,459	0	87	59
Symantec	3,575	0	29	1

4.6 Visualising the dataset over the three dimensions (AV, MW, Dates)

Next, we investigated more closely the overall distributions of the detection rates to analyse any patterns or anomalies in the detection capabilities of the different vendors. Figures 4-1, 4-2 and 4-3 show this visualisation. Each of the figures represents three dimensional plots, with the *x* and *y* axes representing any two of the three dimensions of interest (Malware, AV or Date), and the *z* axis (given by the intensity of the colour) represents the proportion of demands of the remaining third dimension that have detection failures. We will use Figure 4-1(a) for explanation: the x-axis contains the dates (ordered from start to finish) of the collection period; the y-axis shows the malware (ordered by MD5[23] – same ordering preserved in parts (a) and (b) of the figure to make the visual comparison easier). A cell on the plot shows the proportion of full capability AV products that failed to detect a given malware on a given date. The colours of the cells represent the proportion of failures: white colour means none of the AV products failed to detect a given malware at a given date (i.e. they all detected the malware); black colour means missing data; the range from light green to dark red represents the failure rates from greater than 0 to 1 (in this case the failure

[23] The MD5 hash of the file is the unique identifier “*signature*”.

rate is given as a proportion of AV versions that failed to detect a given malware at a given date). Figure 4-1(b) shows the same plot for the VirusTotal. The cells in Figures 4-2 show the proportion of dates in which a given AV failed to detect a given malware, whereas Figure 4-3 shows the proportion of malware that failed to be detected by a given AV on a particular date. The colour encoding is the same. More analysis for Figures (4-1 and 4-2) are provided in Appendix A.

Main observations from these plots:

- **Figure 4-1:** it is visually clear that the full capability versions have a higher failure rate (due to the greater prevalence of non-white cells in Figure 4-1 (a)). A number of malware have a high failure rate throughout the period for both setups (as is clear from the red lines that run across the data collection period).
- **Figure 4-2:** There is some visible diversity in the "difficulty" of the malware across different vendors: we have red lines that run across several AV products (maximum six AVs for full capability products; maximum eight AVs for VirusTotal versions). The detection rates that we observed in Table 4-3 are confirmed in the figure: Comodo and AVG have a lower number of coloured cells in part (a) of the figure compared with (b); vice versa for the others.
- **Figure 4-3:** Only the VirusTotal version of Symantec has a perfect detection rate of all malware on a few dates of the experiment (as seen from the white gaps in Figure 4-3(b) for Symantec). The rest of the versions all fail on at least one malware. None of the full capability versions of AVs had a perfect detection rate on all the malware on any dates of the collection period. Comodo has a few days in the experimental period with a high failure rate in VirusTotal (as can be seen from the red areas in the top part of Figure 4-2(b) for Comodo). It is not clear why this is as VirusTotal is a black box for us. We can speculate that during this period Comodo was not updating its signature database in VirusTotal (or the update led to it failing to detect malware that it had detected in the past).

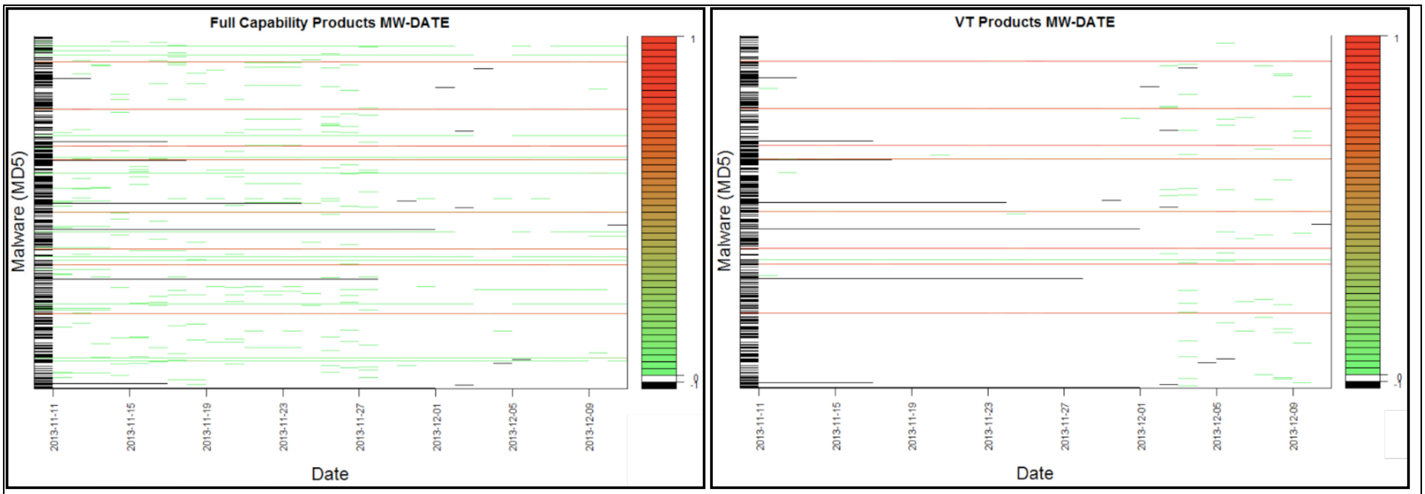


FIGURE 4-1(a)

FIGURE 4-1(b)

FIGURE 4-1 Date (x-axis), Malware (y-axis) and the proportion of AVs (given by the intensity of the colour in the plot) that fail to detect a malware on a given date. Figure 4-1(a) full capability versions; Figure 4-1(b) signature based detection engines as found in VirusTotal.

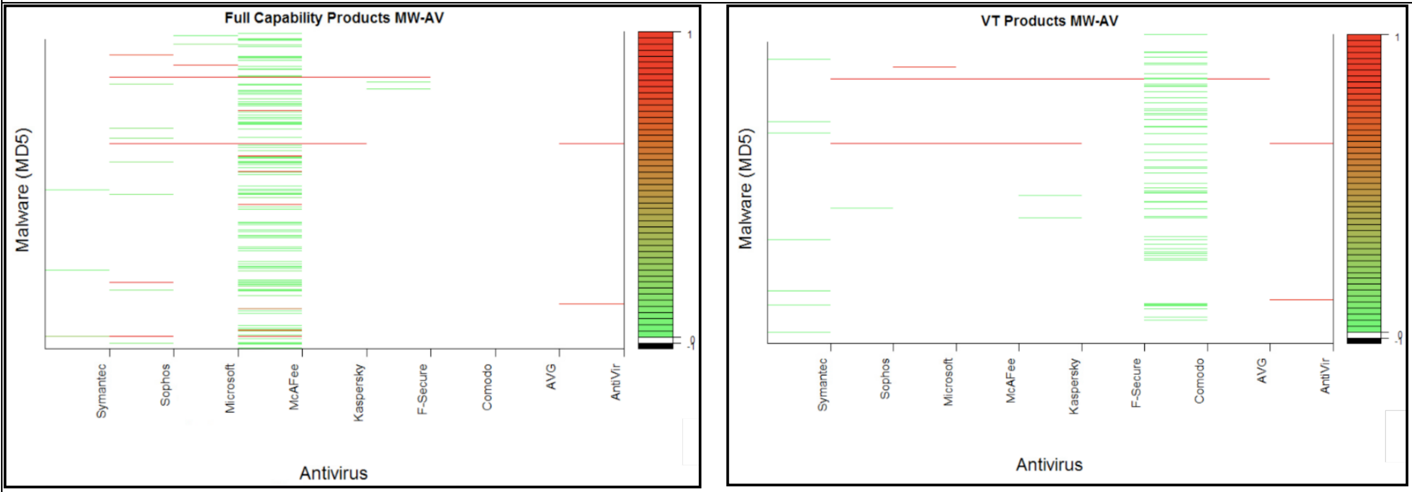


FIGURE 4-2 (a)

FIGURE 4-2 (b)

FIGURE 4-2 AV (x-axis), Malware (y-axis) and the proportion of days (given by the intensity of the colour in the plot) that a given AV failed to detect a given Malware. Figure 4-2(a) full capability versions; Figure 4-2(b) signature based detection engine as found in VirusTotal.

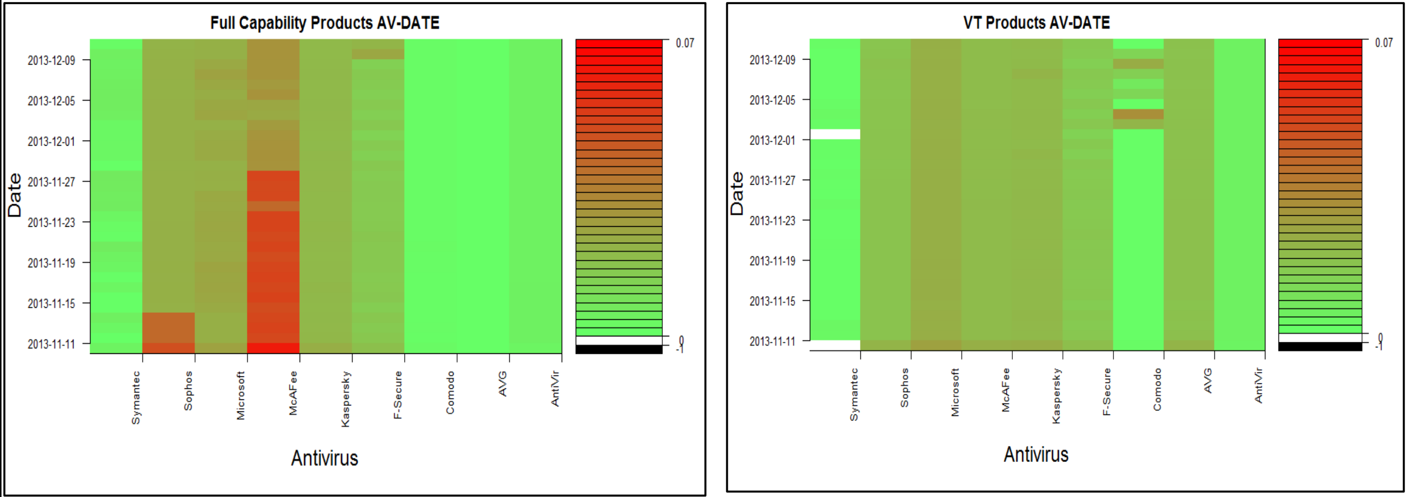


FIGURE 4-3 (a)

FIGURE 4-3(b)

FIGURE 4-3 AV (x-axis), Dates (y-axis) and the proportion of Malware (given by the intensity of the colour in the plot) that failed to be detected by a given AV on a given date. Figure 4-3(a) full capability versions; Figure 4-3 (b) signature based detection engines as found in VirusTotal.

4.7 Time lag analysis

Apart from looking at the detection capabilities of the two versions of the AV vendors in our study, we also looked at which version first detected a malware and what is the time-lag between detections of malware by the two versions. The results are given in Table 4-6. The first column (Difference in Days) represents the time lag in days: a 0 value means that both versions detected a malware on the same day; a positive value gives the difference in days between the first date that a full capability version detected a malware and the first time it was detected by the VirusTotal version (e.g. a value of 2 means the VirusTotal version detected the malware two days ahead of full capability); The subsequent columns to the column labelled as "Difference in Days" then give the counts of malware for each vendor. We should be clear that this is the malware for which both AV versions of a given vendor eventually did detect the malware: what we are measuring here is the difference in the time it took each vendor to first detect it in our collection period. It is worth to mention that the full capability products in this experiment are turned on, therefore the AVs are always using the latest updates when inspecting a malware. Most of the malware is either detected on the same date or VirusTotal detects them a day earlier: this might be because of the slight delay with which we send the malware to the two versions (the malware are sent to the VirusTotal versions on average two hours earlier than it is sent to the full capability versions, which should give a slight advantage to the full capability versions as they would work with a signature ruleset which is "fresher" by two hours). Row A shows the count of undetected malware from both versions, row B shows the count of undetected malware by full capability but detected by VirusTotal, and row C shows the count of the detected malware by full capability but not detected by VirusTotal. As we can see, only AVG of VirusTotal could not detect 60 malware while the full capability version was able to detect them.

Table 4-6 Time lag between detections by the two versions of an AV vendor

		AntiVir	AVG	Comodo	F-Secure	Kaspersky	McAfee	Microsoft	Sophos	Symantec
	A	14	2	2	54	69	67	79	59	1
	B	1	0	3	0	1	17	0	16	0
	C	0	60	0	0	0	0	0	0	0
Difference in Days	0	3590	3543	3600	3550	3535	3410	3526	3459	3575
	1						76			27
	2				1		9			1
	3								36	
	4								35	
	5						1			
	11						2			
	14									1
	17						2			
	22						11			
	23						10			

- ➢ **A :** Malware have not been detected by both versions
- ➢ **B :** Malware have been detected by **VT** version but not detected by FC
- ➢ **C :** Malware have been detected by **FC** version but not detected by VT
- ➢ **Difference in days**: number of days that VT detected the same malware before the FC version from the same vendor.

4.8 Difference in signature labels between full capability versions and VirusTotal versions

As we have mentioned before, when we submit a malware sample to the two versions of an AV per vendor, we can only observe whether that version i) detects or fails to detect a given malware on a given date; ii) if it does detect it, what is the label that the AV assigns to the malware. So far the results we presented concern analysis regarding part i). In the next section we analyse the labels that an AV version assigns to a given malware. The labels may be important to provide a system administrator with extra information on what type of malware they are dealing with etc., which may help them with diagnosis.

The first analysis we did was to check whether the labels assigned by the two versions of a given AV product were exactly the same. The first two columns of Table 4-7 show these results. Next, we analysed in more detail the labels that did not match, and checked whether there was a level of similarity between them. We observed two main categories: in some cases the full capability version just added further context to a label (see examples in Table

4-8) or the labels were completely different. The main observations from this analysis are:

- Microsoft is the only vendor which provides the same labels when detecting malware on both the full capability and VirusTotal versions; FSecure also has the same labels for all but one malware (which it identified differently on 23 days of our observation period);
- Kaspersky uses a different way of labelling malware in its full capability versions compared with the VirusTotal version;
- Four AV vendors (AntiVir, AVG, Comodo and Sophos) use mainly the same labels but add some extra information for the full capability version.

Table 4-7 Signature label counter

AV Name	Matching Signature Labels	Non Matching Signature Labels	Non Matching Because of Extra Info Added to the Label	Very different
AntiVir	0	108,141	108,139	2
AVG	0	106,235	105,384	851
Comodo	0	108,016	106,349	1,667
F-Secure	91,314	23	0	23
Kaspersky	0	106,334	0	106,334
McAfee	60,394	43,575	31	43,544
Microsoft	105,817	0	0	0
Sophos	0	105,820	105,635	185
Symantec	106,323	437	16	421

Table 4-8 Examples for signature label categorising

Signature Label Category		Vendor Name	Full Version Signature Label	Free Version Signature Label
Matching Signature Labels		F-Secure	Worm:W32/Downadup.gen!A	Worm:W32/Downadup.gen!A
Non Matching Signature Labels	***Non Matching Because Of Extra Info Added To The Label***	Comodo	NetWorm.Win32.Kido.A@132026498	NetWorm.Win32.Kido.A
	Totally Different	McAfee	W32/Conficker.worm.gen.a	Artemis!01273BEC3497

We also analysed the classifications that some of the AV products used for these different types of malware. We concentrated on the full capability versions. We compared the classifications used by different AV vendors. We found that the vendors seem to be categorising and classifying the malware differently from each other. Table 4-9 shows an example of this for two of the products (AntiVir and AVG). We can see from this table that AntiVir classifies 40,822 demands as Trojan horses, whereas AVG classifies 10,924 as such. Full results for all AVs are provided in Appendix A (section A-2)[24]. The lack of an interoperable standard for labelling the malware makes comparisons between vendors very difficult. This adds further confusion on diagnosis and recovery processes that a system administrator needs to perform when using diverse AVs (especially if they are used to the labelling and classification of malware by a different vendor than the one they need to administer presently).

Table 4-9 Malware type classification

AntiVir		**AVG**	
Malware type	**Count**	**Malware type**	**Count**
Backdoor	377	Trojan horse / Backdoor	10,924
Dropper	184	N/A	0
Root kit	30	N/A	0
Trojan horse	40,822	Trojan horse	1,208
Virus	6,541	Virus	5,649
Worm	60,187	"Virus identified –Worm"	88,464

4.9 Discussion, Conclusions and Limitations

In this section we summarise the main observations we have made from our experiment and discuss the possible implications they may have on product selection and administration.

Observation: Out of nine vendors in our study, only two of them had a full capability version which had a detection rate that was better than their VirusTotal counterpart. **Implication:** This suggests that for most of these products the free version they have in VirusTotal is perfectly suitable for

[24] The exception is Kaspersky. In our study the full capability version of Kaspersky labelled each detected malware by its MD5 value only, and gave no other label. So for Kaspersky we show the VirusTotal labels.

malware detection and may even perform better compared with a full capability one.

Observation: The full capability versions of some of these AV vendors detected a malware in some cases more than three weeks after their VirusTotal version had detected the same malware. **Implication:** This is a strange and counterintuitive observation. One would expect that the customers who have downloaded a paid version of a product would be served the signatures first. One possible explanation for this observation might be that the vendors are worried about false positives and want to first roll out a signature in VirusTotal. Only after they gain enough confidence that a file is indeed malicious do they roll it out to the full capability versions.

Observation: We sent all the malware which at least one of the versions in our data collection had failed to detect in our observation period to the malwr.com site which reported that some of these malware are not exhibiting malicious behaviour when run on a sandboxed environment. **Implication:** All the malware collected in our study have been uploaded to our honeypots. By definition, any interaction with a honeypot implies malicious behaviour, as honeypots do not have any legitimate production value. Some of the files may be "malformed" during the uploading process by the attacker – in those instances it is not clear whether an AV product should detect the malware or not (most AV products do detect them as malicious as they recognise at least a partially malicious payload). Other files are encrypted and the dynamic analysis may not be able to execute them. There are some advanced malware that have the ability to be aware that they are in a test of a sandbox environment, therefore they remain benign and do not exhibit any malicious behaviour when under observation[25].

Observation: The signature based versions and full capability versions of the same AV vendors seem to assign different labels to the malware for Kaspersky and McAfee. For the others is seems to be mainly the same, with some extra information added in the label of the full capability version. **Implication:** This implies that for most of these products (with the exception of Kaspersky and McAfee), it is the signature based detection engine that is mainly responsible for labelling the malware, with some extra information provided from the other components of the AV system which are found on a

[25] A further discussion on the drawbacks of sandboxing for detecting advanced threats is here: https://www.darkreading.com/risk/the-pros-and-cons-of-application-sandboxing/d/d-id/1138452

full capability version. Hence the free VirusTotal versions of these products are useful for system administrators to get information on the malware, and for most they are not likely to differ from the full capability versions.

Observation: There are differences between AV vendors in how they classify malware. For example, what may be a Trojan horse for one vendor, could be a worm for another. **Implication:** The lack of interoperability and consistency between vendors for labelling the malware adds further confusion on diagnosis and recovery processes that system administrators need to perform. The administrators may have experience with the labelling and classification of malware with a different vendor and that experience is not directly transferable to another vendor’s products.

The main conclusions we can draw from our analysis are:

- For most vendors in our study (seven out of nine) the VirusTotal version has a better detection rate than their full capability version. This would imply that investment in the full capability version of an AV product may not be worthwhile.
- Some of the full capability versions of the AV vendors in our study only detected some malware more than three weeks after the VirusTotal version of the same vendor has detected the same malware. This seems to imply that vendors for some malware are testing their detection signatures in their VirusTotal versions first before propagating them to the full capability versions, which may also explain the higher detection rates of the VirusTotal versions of some of these vendors.
- There are differences between the vendors in the way in which they classify malware. This lack of consistency between the vendor malware classification schemes makes it more difficult for system administrators to transfer their malware analysis expertise from one vendor’s system to another.
- Finally, we have tried to clearly position our work compared with related work. To the best of our knowledge this is the first study that has specifically compared the VirusTotal versions of AVs with full capability versions.

The main limitations of our conclusions, and provisions for further work:

- The malware samples are Windows portable executable files. More files types would allow for more general conclusions.

- All our analysis so far has been with malware samples, which means we cannot get any measurements on false positive rates.
- We have looked at nine vendors over a one-month period. A longer data collection time with more vendors may allow for stronger conclusions.
- The malware are collected on honeypots that are distributed in three continents and were uploaded to our honeypots in the period that we stated. For the assessment of the diversity that exists in the AV products, and the different versions of the products from the same vendor, one month of data is reasonably representative, We also note that the results we observed are consistent with previous studies that have assessed diversity with AV products (Bishop et al., 2011; Gashi et al., 2009), even though the primary focus of our research was in assessing the diversity that exists between different versions of the same product.
- The dataset is collected via the Dionaea project, which biases the dataset towards self-propagating malware. These malware are not necessarily representative of the overall malware population, which mainly propagates "passively" via drive-by exploits or social engineering attacks.

(5) AVAMAT: AN ANTIVIRUS AND MALWARE ANALYSIS TOOL

This chapter presents the AVAMAT tool that can be used to analyse the malware detection capabilities of existing AV (AV) products on different operating systems. Most sections in this chapter have been published in:

- Pasha Shahegh, Tommy Dietz, Michel Cukier, Areej Algaith, Attila Brozik, Ilir Gashi: ***"AVAMAT: AntiVirus and Malware Analysis Tool*"**. NCA 2017: 365-368

5.1 Introduction

While working with the VirusTotal and exploring the MetaDefender platforms, we noted several limitations of these platforms (cf. discussion in section 2.4). To overcome these limitations a tool called AV and Malware Analysis Tool (AVAMAT) was built: a multi-purpose tool that can be used for analysing different malware and AV product capabilities running on different operating systems. Currently AVAMAT supports eight full capability AV products (i.e. the full versions of AV products, rather than just signature based detection engines), namely: AVG, Comodo, F-Secure, Kaspersky, FProt, Trend, Avira and Emisoft. These AV products are run, where available, on three versions of the Microsoft Windows operating systems: XP[26], 7 and 8. The candidate contributed in all the discussion regarding the inception, requirements and design of the tool. The candidate also analysed the data from test reports shared by the developers at the University of Maryland, and provided detailed bug reports, which helped with the tool debugging. The candidate also contributed towards the writing of the paper that resulted from this work, specifically the tool design and testing sections, as well as providing comments on several versions during the paper writing process. The implementation of the AVAMAT tool was done at the University of Maryland. It was difficult to delineate the precise contribution of the candidate compared to those of the other collaborators on this part of the work. For this reason, we will only summarise the AVAMAT tool here, as it relates to the work that the candidate performed and leave the details of the tool itself out of the thesis – the interested reader can find out the details in the published paper.

The rest of the chapter is organised as follows: section 5.2 outlines the objectives of the study; section 5.3 describes the AVAMAT architecture; section 5.4 shows some results with AVAMAT; section 5.5 describes lessons learnt with AVAMAT; and finally section 5.6 presents a discussion, conclusions and provisions for further work.

[26] Avira and Emisoft AVs no longer support XP, so we could not run those products on XP.

5.2 Study objectives

AVAMAT was built to enable a researcher to:

- analyse the diversity of detection capabilities between *different* AV software on *different* operating systems. Being able to analyse the same malware on machines with the same AV yet different operating system allows us to investigate the operating system's effect on AV and malware behaviour. And analysing the detection capabilities of different AV products allows us to compare the benefits of combining multiple diverse AV products in a *diverse defence in depth* setup.
- analyse *when* an AV product detects a malware it encounters. We classify the detection in four stages depending on when a malware is detected: on entry; after a short wait; on a full scan; or after malware execution. This allows us to, for example, better classify whether the malware will be detected and prevented from running on the end-host machine, or whether the malware would run first before being executed, hence potentially requiring a clean-up and full scan of the machine. Again, we will analyse the diversity that exists in these classifications between different AV products and different OSs.

In the rest of the chapter we describe AVAMAT in more detail.

5.3 AVAMAT architecture

The AVAMAT architecture is built on top of open-source software and uses custom-developed scripts to allow us to test whether an AV *a,* running on a given OS *o,* detects a given malware *m* on a given date *d*, and, if it detects it, *when* does it do so. We will use the shorthand *VM*(*a*,*o*) to refer to a given virtual machine that runs an AV *a* on an OS *o*. There are four main components of AVAMAT:

Skeleton: interfaces with custom-developed scripts on each *VM(a,o)*. The skeleton chooses the specific script on *VM(a,o)*. Once selected, the skeleton uses the functions in the custom-developed script to perform an analysis on malware *m*;

Updaters: at the start of each experimental campaign, updates the OS and AV with the latest updates and patches available for each *VM(a,o);*

Snapshot Manager: at the start of each experimental campaign, takes a snapshot of each virtual machine *VM(a,o)*; after the virtual machine is finished inspecting a given malware *m* it reverts back to the last snapshot (ensuring that all malware in a given experimental campaign are executed by the same AV *a* and OS *o*);

Experiment Scheduler: the administrator of the experiment can specify how many times an experimental campaign should be repeated. This may be important to allow, for example, assessment of the capability of an AV product to continue detecting a malware or for testing how long it takes for an AV product to detect a malware it has not detected in the past. For each repetition of the experiment:

- We run the **Updaters** *once* for each *VM(a,o)*;
- We run the **Snapshot manager** to take a snapshot of each *VM (a,o) once* at the start of the experiment (i.e. before sending any malware to it). We then revert back to this clean snapshot of a given *VM(a,o)* after each malware *m* is inspected by that *VM(a,o)*;
- We run the **Skeleton** once after each malware *m* is sent to a *VM(a,o)*.
- Figure 5-1 shows the AV products and OSs currently supported in AVAMAT (so each coloured box represents one *VM(a,o)*).

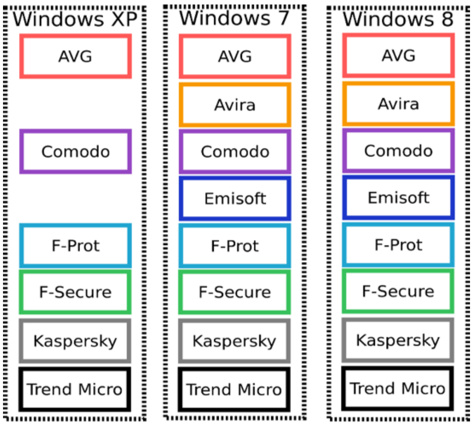


Figure 5-1 The AV products and OSs currently supported in AVAMAT. Each box is one virtual machine (we refer to a box as *VM(a,o)*). Avira and Emisoft no longer support Windows XP versions, hence they are missing from the figure above ;Windows XP on left-side of box.

5.4 Some results with AVAMAT

AVAMAT is currently a prototype. In what follows we show some examples of analysis using results obtained from testing AVAMAT. The malware we used during testing have been collected from research honeypots from the University of Maryland.

Figure 5-2 shows a 3D plot of the detection capabilities of Comodo AV, running on Windows 7 OS, when subjected to 5,855 malware (y-axis) over a seven day period (x-axis). The different colours on the plot show the stages of detection (light green means detected on entry (the best possibility); dark green for detecting after 10 seconds; dark blue for detecting on scan; orange for detecting on execution), failure to detect that malware at all (red), or missing data (black). Since this data was collected during the testing of AVAMAT, it was useful for us to debug when malware were not being sent to the tool (as is evident in day 5). But for research purposes it is useful to analyse when the different malware are being detected by the AV product, and what risk the machine is exposed to if the malware is not detected immediately on entry.

Figure 5-3 is a 3D plot showing the rate of malware detection (z-axis) by the different *VM(a,o)* (y-axis) [27] per stage of detection (x-axis; non-detection is step 1111). This graph illustrates the type of analysis we can do with results from AVAMAT to show the diversity that exists in malware detection, and stages of malware detection, between the different AV products and different OSs. For example the rate of malware detection by diverse 20 systems in the first step (x-axis 0: on entry) equals 0.8 but for the last step (x-axis 1111: non-detection) almost equals zero.

[27] Note that this analysis does not have all 22 *VM(a,o)* in the y-axis, as not all of them were operational when this part of testing was performed.

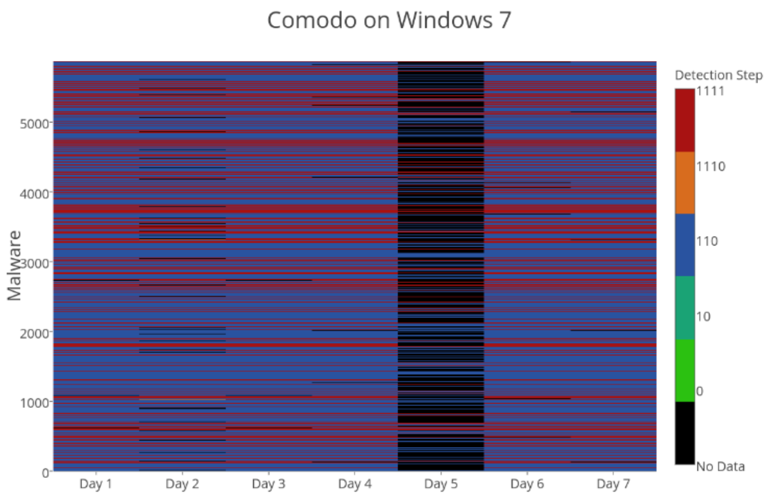


Figure 5-2 3D plot of the detection capabilities of Comodo, running on Windows 7 OS, when subjected to 5855 malware (y-axis) over a 7 day period (x-axis)

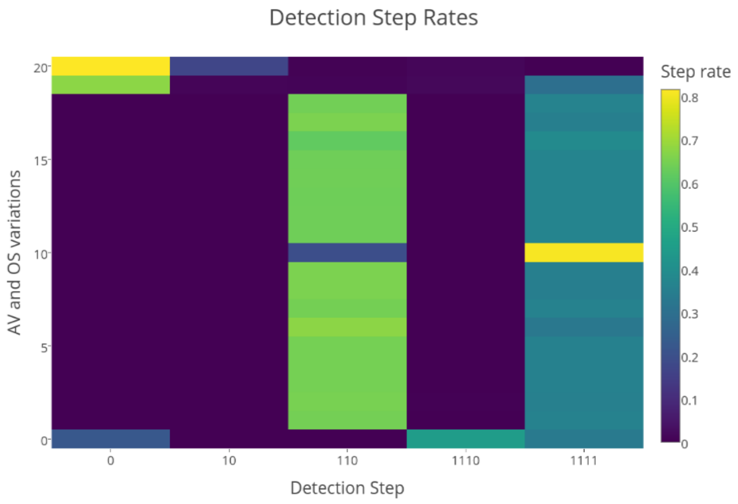


Figure 5-3 3D plot showing the rate of malware detection (z-axis) by the different *VM(a,o)* (y-axis) per stage of detection (x-axis; non-detection is step 1111)

5.5 Lessons learnt

AVAMAT is being built to overcome the limitations of existing malware testing platforms, such as VirusTotal and Metadefender, that do use multiple AV products, but only their command line interfaces that have limited functionality. These platforms also do not provide details on when an AV product actually detected the malware (on entry, on scan, once malware executes, etc.). AVAMAT enables researchers to analyse different malware and AV product capabilities running on different OSs. Currently AVAMAT supports eight full capability AV products (i.e. the full versions of AV products, rather than just signature based detection engines) that are run, where available, on three versions of the Microsoft Windows OS: XP, 7 and 8. We have chosen the Windows platform because the majority of malware samples collected from our previous experiments with honeypots were Windows executable files. Moreover, based on the literature surveyed, we found that Windows is the most “target-rich” environment for malware samples. We have used XP, 7 and 8 as these were the most popular Windows versions when we started work on AVAMAT. However, support for Windows 10 is also planned in the future versions of AVAMAT.

AVAMAT allows running experimental campaigns to help answer research questions such as the following (the list is not exhaustive):

- What are the differences in the detection capabilities of different AV products?
- Are there differences in detection capabilities depending on which OS platform the AV product runs?
- Are there differences by type of malware the AV products inspect?
- Do the AV products continue to detect a malware over time, or are there cases of regressions in detection behaviour?
- How do we combine multiple AV products to improve detection capabilities against malware?
- What are the false positive rates of AV products when subjected to benign files? Are there differences in these rates: by OS platform? By type of file? Etc.

Once the product is ready for wider release, there are two options of making the tool publicly available for other researchers to use:

- Release the code so that users can build their own version of AVAMAT, with their own AV products and licenses in their own environments;
- Provide an API through which users can submit malware samples for analysis to AVAMAT (similar to the way in which VirusTotal and Metadefender can be used). This is the preference of the authors, as is to make the tool free for use. But there are inevitable infrastructure costs for deploying a tool such as this, so the exact deployment and use model for AVAMAT remains to be decided.

5.6 Discussion, Conclusions and Further Work

We presented AVAMAT: AV and Malware Analysis Tool - a tool for analyzing the malware detection capabilities of AV (AV) products running on different operating system (OS) platforms. Even though similar tools are available, such as VirusTotal and MetaDefender, they have several limitations which motivated us to create of our own tool. With AVAMAT analyses of stages of inspection than an AV on a given OS detects a malware is possible. This allows a researcher to run experimental campaigns to answer various research questions, ranging from the detection capabilities of AVs on OSs, to optimal ways in which AVs could be combined to improve malware detection capabilities.

Current work and future enhancement for AVAMAT include:

- Building support for more AV products;
- Building support for more operating systems;
- Improving the code so that it is more modular, enabling easier maintenance;
- Producing a user manual and demonstration for use of the tool;
- Building a front-end extension (or a separate tool that connects to AVAMAT) for analysing data obtained from AVAMAT.

(6) DIVERSITY WITH INTRUSION DETECTION SYSTEMS (IDSs)

This chapter presents results of analysing the performance of diverse Intrusion Detection Systems (IDSs) configurations. Most sections in this chapter have been published in:

Areej Algaith, Ivano Alessandro Elia, Ilir Gashi, Marco Vieira: ***"Diversity with intrusion detection systems: An empirical study"***. NCA 2017: pp: 19-23

6.1 Introduction

In this chapter we provide empirical results on the assessment of diversity with Intrusion Detection Systems (IDSs). We utilise a dataset published by (Elia et al., 2010), who have helpfully made their data publicly available[28]. An attack injection methodology was used, consisting of injecting realistic vulnerabilities in three web applications (MyReferences, phpBB and TikiWiki) and sending attacks that attempt to exploit those vulnerabilities. In order to protect these web applications, four diverse IDS products were deployed (Apache Scalp, Anomalous Character Distribution (ACD) monitor, GreenSQL and Snort). Some of these IDSs were configured in different ways (depending, for example, on the rulesets they used and the threshold for identifying malicious requests), which produced nine different IDS configurations in total. While the authors in Elia et al. (2010) performed various analyses to assess the performance of the IDSs individually, they did not explore whether the different IDS products could be combined to improve the performance. This is the focus of our research and in this chapter we report our analysis and main conclusions.

We apply in full the methodology we described in Chapter 3. Namely, we calculate the FP, FN, TP, TN counts for each diverse configuration; we calculate the measures of interest (specificity, sensitivity and accuracy) or each diverse configuration, overall and by type of attack; we generate the ROC plots showing all the diverse configurations and the individual defence systems, overall, by type of configuration and by type of malicious input; we calculate the differences in the measures of interest between diverse configurations and individual systems to measure the possible improvements or deteriorations from switching to a diverse system.

The rest of the Chapter is organised as follows: section 6.2 outlines the objectives of the study; section 6.3 describes the dataset; section 6.4 provides the results for single version systems; section 6.5 presents the results from analysing two-version diverse systems; section 6.6 presents results from analysing diverse systems with more than two versions; and finally section 6.7 presents a discussion, conclusions and provisions for further work.

[28] Full dataset is available from: https://goo.gl/MDOhsw

6.2 Study objectives

Compared with the previous chapters, where we only had malicious inputs, in this chapter we present results from assessing diversity when both malicious and benign inputs have to be considered by security protection tools. We first did an analysis of the simplest possible diverse configuration: i.e. a two-version system, before we expanded it to consider configurations with more systems (up to N). With two version systems, there are two possible configurations of the voter/adjudicator:

- **One-out-of-two (1oo2):** the system flags an input as malicious (i.e. raises an alarm) as long as either one of the two IDSs flags that input as malicious.
- **Two-out-of-two (2oo2):** the system flags an input as malicious (i.e. raises an alarm) only if both of the IDSs flag that input as malicious.
- Since we have nine different IDS product configurations, we can construct 36 distinct two-version IDS combinations ($^9C_2$ = 36). In practice, each system is being asked whether a given input[29] is malicious or not. Hence, we used the sensitivity, specificity and accuracy measures as described in Chapter 3.

We analysed these measures for each of the 36 combinations of IDSs in the context of each of the three web applications (MyReferences, phpBB and TikiWiki). Additionally, we classified and analysed the results by:

- **Type of IDS**, i.e. whether an IDS is primarily configured to monitor for attacks at the Application, Database or Network levels;
- **Type of attacks**, distinguishing between HTML POST and HTML GET attacks.

We then extended this analysis to consider diversity with a larger value of N(triplets, quadruples etc.)

6.3 Dataset, IDSs, and web applications

The data used for the diversity analysis presented here is the result of the experimental campaign conducted by the authors of Elia et al. (2010).

[29] The 4 IDSs (and the 9 configurations that ensue from their use) are targeting different types of web-traffic, from 3 web applications. In reality the IDS is subjected to different type of attacks not only SQLi, though SQLi attacks are some of the most prominent, as discussed in Chapter 2.

Their campaign aimed at testing a set of IDSs in terms of their capability of detecting SQL Injection attacks. In order to produce realistic SQL Injection attacks to test the IDSs a Vulnerability and Attack Injection technique was used. This technique allowed the authors of Elia et al. (2010) to introduce realistic vulnerabilities in the code of a web application by code mutation (Vulnerability injection) and afterwards to automatically exploit those vulnerabilities by performing SQL injection attacks (Attack Injection). The injected vulnerabilities were considered realistic because they were based on an extensive field study on real web application vulnerabilities (Fonseca et al., 2014).

The vulnerability and attack injection tool[30] used in Elia et al. (2010) runs on an Ubuntu virtual machine, configured to inject vulnerabilities in a set of three web applications. The IDSs under test were deployed in the same virtual machine, and exposed to attacks generated by the attack injector and to non-malicious interactions carried out through a web crawler. In the remainder of this section, we briefly introduce the IDSs and web applications used in Elia et al. (2010).

6.3.1 Intrusion Detection Systems

The experimental setup in Elia et al. (2010) includes four different IDSs, some of which were tested using different configurations, leading to a total of nine distinct deployments. The IDSs tested in these experiments are composed of both well-known security tools and implementation of detection approaches proposed in research papers. The tested IDSs are also diverse in terms of both detection approach (anomaly-based, signature-based) and monitored layer (application level, database level, network level). Table 6-1 outlines the details.

Table 6-1 Classification of the Types of IDSs in our Study

Tool	Architectural Level monitored	Detection Approach	Data Source
ACD	Application	Anomaly Based	Apache Log
Apache Scalp	Application	Signature Based	Apache Log
Snort	Network	Signature Based	Network Traffic
GreenSQL	Database	Signature Based	SQL Proxy Traffic

[30] https://github.com/JoseCarlosFonseca/Vulnerability-and-Attack-Injector

The **Anomalous Character Distribution (ACD)** monitor is an anomaly-based tool that works at the application level. It analyses the Apache access log using the detection approach described in Kruegel and Vigna (2003), which is based on the character distribution in the URLs of the HTTP requests sent to the web application. The user must define the deviation threshold that separates the requests identified as malicious from those considered benign. In Elia et al. (2010) the configurations of threshold values were: 1, 3, 10, 30, and 100. In the rest of the analysis here we refer to these IDSs configurations as ACD1, ACD3, ACD10, ACD30 and ACD100, and in the graphs we assign them labels 1A-5A respectively (“A” being a shorthand for “Application” type IDS).

GreenSQL *(version 1.2.2)* is an IDS that focuses on the detection of attacks targeting a database. The experiments were performed on the open source version of the tool[31] (a commercial version of the tool). It was deployed as a proxy standing between the front-end and back-end of the web applications in order to monitor the SQL communications. It evaluates each SQL query by associating a risk-scoring matrix defining the probability of it being malicious. In our graphs we assigned this IDS the label 6D (“D” standing for “Database” IDS).

Apache Scalp *(version 0.4)*[32] is another Apache access log analyser. It uses a signature-based approach that compares the requested URLs in the access log against a set of attack signatures for web application attacks like SQL Injection, Cross Site Scripting, Cross Site Request Forgery and Path Traversal, etc. In Elia et al. (2010) the authors considered only the signatures for SQL injection attacks. In the rest of the analysis in this chapter we refer to this IDSs as SCALP sqlia, and in the graphs we assigned it the label 7A.

Snort *(version 2.8.4.1)*[33] is a signature based network level IDS. In practice, Snort is a network sniffer and thus has access to both the HTTP and SQL traffic. The detection approach is based on a very large set of attack signatures (the Snort Rules) maintained by the community. These signatures are then evaluated against the network traffic collected by the sniffer. The users are allowed to customise their set of Snort rules. In Elia et al. (2010) the

[31] https://github.com/larskanis/greensql-fw

[32] https://code.google.com/archive/p/apache-scalp/

[33] https://www.snort.org/

analysis was performed using two configurations: using only the official community rule set; and using a set of experimental Customised Rules (later identified with CR) provided in Mookhey and Burghate (2004). In our graphs we assigned these two configurations the labels 8N and 9N ("N" standing for "Network" IDS).

6.3.2 Web applications

The experimental setup in Elia et al. (2010) includes three web applications: MyReferences, TikiWiki, and phpBB, which allowed the authors to assess IDS performance with different types of applications (MyReferences is small; TikiWiki and phpBB are large open source projects).

- ***MyReferences*** is a publications and bibliographic references management web application. The application was developed by the authors of Elia et al. (2010). It provides functionality for managing (editing, querying and displaying) documents and publications metadata (title, authors, year of publication, etc.).
- ***TikiWiki***[34] is a groupware/CMS (Content Management System) platform that allows collaborative contribution over the website contents in a wiki style. It is one the most widely used applications of this type.
- ***phpBB***[35] is a one of the most widely used open-source forum solutions.

Table 6-2 shows the total number of benign demands (labelled below as Crawling actions) and successful attacks[36] (i.e. attacks that exploited a vulnerability) for each application, as reported in Elia et al. (2010). Each of the nine IDS[37] configurations got to inspect the same traffic for each application. These data form the basis of our analysis.

[34] https://tiki.org

[35] https://www.phpbb.com/

[36] The authors state that they had some attacks that, when sent to the applications, were "unsuccessful", i.e. they did not lead to an exploit of the vulnerability, and others that were "Successful". In this study we only consider the successful attacks , since we were uncertain on how to classify the behaviour of an IDS if it does not raise an alarm for an attack that is not successful.

[37] Including the functionally diverse IDSs

Table 6-2 The counts of crawling actions traffic and successful attacks traffic per web application

Web Application	IDS	Crawling Actions	Successful Attack
MyReferences	9	45	136
phpBB	9	97	245
TikiWiki	9	80	76

6.4 Analysis of single version systems

Table 6-3 presents the nine configurations of the IDSs and the labels we will use to refer to them in the graphs in the rest of this chapter. Next to them we also note the specificity and sensitivity for each of the three applications. The authors of Elia et al. (2010) already presented these classifications, and the results for each IDS individually.

Table 6-3 The nine distinct IDS deployments

Label	IDS Name	MyReferences		phpBB		TikiWiki	
		Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
1A	ACD1	0.89	0.76	0.93	0.37	0.49	0.48
2A	ACD3	0.61	0.84	0.22	0.68	0.24	0.75
3A	ACD10	0.35	1	0.07	0.99	0.20	0.99
4A	ACD30	0.27	1	0.04	1	0.20	1
5A	ACD100	0.10	1	0.02	1	0.00	1
6D	GREENSQL	0.12	1	0.63	1	1	1
7A	SCALP sqlia	0.25	0.91	0	1	0.21	1
8N	SNORT 2.8	0	1	0	1	0	1
9N	SNORT 2.8 plus CR	0.59	1	1	1	0.5000	1

In our work we extend the analysis from the viewpoint of diversity. We analysed all possible pairs, triplets etc., that can be constructed with the 9 IDS[38].

[38] Including the functionally diverse see examples in Table 6-4 to 6-6

6.5 Diversity analysis for two version systems

In this section we present the results of our analysis:

- *Summary results and analysis for each application*: FP, TN, FN, TP rates, ROC plots, and differences in the sensitivity, specificity and accuracy for each application, categorised by type of IDS combination;
- *Analysis by type of attack*: differences in sensitivity by type of attack (i.e. *GET* and *POST* attacks) for each application;
- *Differences in sensitivity, specificity and accuracy* between diverse and single IDS systems for each application;
- *Averages:* averages for sensitivity, specificity and accuracy for individual systems and the different types of IDS configurations, for each application.

6.5.1 Overall summary and analysis by type of IDS combination

Table 6-4 shows the full results for the MyReferences application. The table should be read as follows: the first column shows the label we assign to a particular combination (the label is meant to be short but meaningful: so 1-2 AA means a combination of IDS 1A and IDS 2A from Table 6-4 etc.); the second and third columns show the two IDS systems in that combination; and the subsequent columns show the FP, TN, FN and TN values for: the first system in the pair; the second system in the pair; the 1oo2 system and the 2oo2 system configurations. In the table we have highlighted cases where the diverse combination does better (green cells) than the best single system in that pair, or worse (red cells) than the worst single system in that pair. In cases where there is no improvement compared with the best single version (or no deterioration compared with the worst single system) we do not highlight any cells in a row. We also show the specificity and sensitivity values for each pair for the 1oo2 and 2oo2 cases.

The table is split into two sections:

- The first 16 rows show all the combinations that can be built when the two individual systems in the pair are of the same type (so Application-only (AA) – first 15 rows; or Network-only (NN) – the 16th row). We only have one DB IDS so we cannot build any DB-only pair). We labelled these as “Functionally redundant” pairs.
- The remaining 20 rows are for the pairs of IDSs that are of different types (e.g. Application and Network (AN) – 12 pairs; Application and Database

(AD) – 6 pairs: and Database and Network (DN) – 2 pairs). We labelled these as “Functionally diverse” pairs.

We have also built these tables for the other two applications phpBB and TikiWiki see Tables 6.5 and 6.6.

From the results in the table we can already see some patterns emerging: the 1oo2 systems are better at detecting attacks (higher TP and lower FN rates), compared with best individual systems; on the other hand, 2oo2 systems are better at correctly labelling benign traffic (higher TN and lower FP rates). This is to be expected as:

- 1oo2 systems will in *all* cases perform:
    - *better or equal* to the best single system in the pair for *malicious traffic*, as any alarm from any of the two systems will lead to an alarm in a 1oo2 system;
    - *equal or worse* than the worst single system in the pair for *benign traffic*, as any alarm from either single system for benign traffic will be incorrectly labelled as malicious.
- 2oo2 systems will in *all* cases perform:
    - *better or equal* to the best single system for *benign traffic* as the 2oo2 system only raises an alarm for benign traffic if both the single systems in the pair raise an alarm;
    - *equal or worse* than the worst single system in the pair for *malicious traffic*, as the 2oo2 system will only label an attack as malicious if both the single systems in the pair label it as such.

What is important is *how much* better, or *how much worse*, would a diverse pair perform in these setups, and the results in Table 6-4 already give us some indications about this. But looking at these numbers in isolation makes it difficult to make a decision about the overall system performance. The ROC plots help with this.

Table 6-4 Counts of detections and failures for benign traffic (False Positive and True Negative) and for legitimate attack (False Negative and True Positive) for each IDS and for the 36 combinations of 2-version systems 1oo2 and 2oo2 for **MyReferences** and taking into account the type of IDS

				FP				TN				FN				TP				Sensitivity		Specificity	
	Combined System Label	A	B	FP (A)	FP (B)	FP 1oo2	FP 2oo2	TN (A)	TN (B)	TN 1oo2	TN 2oo2	FN (A)	FN (B)	FN 1oo2	FN 2oo2	TP (A)	TP (B)	TP 1oo2	TP 2oo2	1oo2 (A,B)	2oo2 (A,B)	1oo2 (A,B)	2oo2 (A,B)
Functionally Redundant	1-2 AA	1A	2A	11	7	15	3	34	38	30	42	15	53	15	53	121	83	121	83	0.89	0.61	0.67	0.93
	1-3 AA	1A	3A	11	0	11	0	34	45	34	45	15	89	15	89	121	47	121	47	0.89	0.35	0.76	1
	1-4 AA	1A	4A	11	0	11	0	34	45	34	45	15	99	15	99	121	37	121	37	0.89	0.27	0.76	1
	1-5 AA	1A	5A	11	0	11	0	34	45	34	45	15	123	15	123	121	13	121	13	0.89	0.10	0.76	1
	1-7 AA	1A	7A	11	4	13	2	34	41	32	43	15	102	8	109	121	34	128	27	0.94	0.20	0.71	0.96
	2-3 AA	2A	3A	7	0	7	0	38	45	38	45	53	89	53	89	83	47	83	47	0.61	0.35	0.84	1
	2-4 AA	2A	4A	7	0	7	0	38	45	38	45	53	99	53	99	83	37	83	37	0.61	0.27	0.84	1
	2-5 AA	2A	5A	7	0	7	0	38	45	38	45	53	123	53	123	83	13	83	13	0.61	0.10	0.84	1
	2-7 AA	2A	7A	7	4	11	0	38	41	34	45	53	102	46	109	83	34	90	27	0.66	0.20	0.76	1
	3-4 AA	3A	4A	0	0	0	0	45	45	45	45	89	99	89	99	47	37	47	37	0.35	0.27	1	1
	3-5 AA	3A	5A	0	0	0	0	45	45	45	45	89	123	89	123	47	13	47	13	0.35	0.10	1	1
	3-7 AA	3A	7A	0	4	4	0	45	41	41	45	89	102	73	118	47	34	63	18	0.46	0.13	0.91	1
	4-5 AA	4A	5A	0	0	0	0	45	45	45	45	99	123	99	123	37	13	37	13	0.27	0.10	1	1
	4-7 AA	4A	7A	0	4	4	0	45	41	41	45	99	102	82	119	37	34	54	17	0.40	0.13	0.91	1
	5-7 AA	5A	7A	0	4	4	0	45	41	41	45	123	102	89	136	13	34	47	0	0.35	0	0.91	1
	8-9 NN	8N	9N	0	0	0	0	45	45	45	45	136	56	56	136	0	80	80	0	0.59	0	1	1
Functionally Diverse	1-8 AN	1A	8N	11	0	11	0	34	45	34	45	15	136	15	136	121	0	121	0	0.89	0	0.76	1
	1-9 AN	1A	9N	11	0	11	0	34	45	34	45	15	56	5	66	121	80	131	70	0.96	0.51	0.76	1
	2-8 AN	2A	8N	7	0	7	0	38	45	38	45	53	136	53	136	83	0	83	0	0.61	0.00	0.84	1
	2-9 AN	2A	9N	7	0	7	0	38	45	38	45	53	56	10	99	83	80	126	37	0.93	0.27	0.84	1
	3-8 AN	3A	8N	0	0	0	0	45	45	45	45	89	136	89	136	47	0	47	0	0.35	0	1	1
	3-9 AN	3A	9N	0	0	0	0	45	45	45	45	89	56	28	117	47	80	108	19	0.79	0.14	1	1
	4-8 AN	4A	8N	0	0	0	0	45	45	45	45	99	136	99	136	37	0	37	0	0.27	0.00	1	1
	4-9 AN	4A	9N	0	0	0	0	45	45	45	45	99	56	36	119	37	80	100	17	0.74	0.13	1	1
	5-8 AN	5A	8N	0	0	0	0	45	45	45	45	123	136	123	136	13	0	13	0	0.10	0	1	1
	5-9 AN	5A	9N	0	0	0	0	45	45	45	45	123	56	43	136	13	80	93	0	0.68	0	1	1
	7-8 AN	7A	8N	4	0	4	0	41	45	41	45	102	136	102	136	34	0	34	0	0.25	0	0.91	1
	7-9 AN	7A	9N	4	0	4	0	41	45	41	45	102	56	56	102	34	80	80	34	0.59	0.25	0.91	1
	1-6 AD	1A	6D	11	0	11	0	34	45	34	45	15	120	12	123	121	16	124	13	0.91	0.10	0.76	1
	2-6 AD	2A	6D	7	0	7	0	38	45	38	45	53	120	47	126	83	16	89	10	0.65	0.07	0.84	1
	3-6 AD	3A	6D	0	0	0	0	45	45	45	45	89	120	73	136	47	16	63	0	0.46	0	1	1
	4-6 AD	4A	6D	0	0	0	0	45	45	45	45	99	120	83	136	37	16	53	0	0.39	0	1	1
	5-6 AD	5A	6D	0	0	0	0	45	45	45	45	123	120	107	136	13	16	29	0	0.21	0	1	1
	6-7 DA	6D	7A	0	4	4	0	45	41	41	45	120	102	93	129	16	34	43	7	0.32	0.05	0.91	1
	6-8 DN	6D	8N	0	0	0	0	45	45	45	45	120	136	120	136	16	0	16	0	0.12	0	1	1
	6-9 DN	6D	9N	0	0	0	0	45	45	45	45	120	56	48	128	16	80	88	8	0.65	0.06	1	1

Table 6-5 Counts of detections and failures for benign traffic (False Positive and True Negative) and for legitimate attack (False Negative and True Positive) for each IDS and for the 36 combinations of 2-version systems 1oo2 and 2oo2 for **phpBB** and taking into account the type of IDS

				FP				TN				FN				TP				Sensitivity		Specificity	
	Combined System Label	A	B	FP (A)	FP (B)	FP 1oo2	FP 2oo2	TN (A)	TN (B)	TN 1oo2	TN 2oo2	FN (A)	FN (B)	FN 1oo2	FN 2oo2	TP (A)	TP (B)	TP 1oo2	TP 2oo2	1oo2 (A,B)	2oo2 (A,B)	1oo2 (A,B)	2oo2 (A,B)
Functionally Redundant	1-2 AA	1A	2A	61	31	61	31	36	66	36	66	16	192	16	192	229	53	229	53	0.93	0.22	0.37	0.68
	1-3 AA	1A	3A	61	1	61	1	36	96	36	96	16	227	16	227	229	18	229	18	0.93	0.07	0.37	0.99
	1-4 AA	1A	4A	61	0	61	0	36	97	36	97	16	234	16	234	229	11	229	11	0.93	0.04	0.37	1.00
	1-5 AA	1A	5A	61	0	61	0	36	97	36	97	16	239	16	239	229	6	229	6	0.93	0.02	0.37	1.00
	1-7 AA	1A	7A	61	0	61	0	36	97	36	97	16	245	16	245	229	0	229	0	0.93	0.00	0.37	1.00
	2-3 AA	2A	3A	31	1	31	1	66	96	66	96	192	227	192	227	53	18	53	18	0.22	0.07	0.68	0.99
	2-4 AA	2A	4A	31	0	31	0	66	97	66	97	192	234	192	234	53	11	53	11	0.22	0.04	0.68	1.00
	2-5 AA	2A	5A	31	0	31	0	66	97	66	97	192	239	192	239	53	6	53	6	0.22	0.02	0.68	1.00
	2-7 AA	2A	7A	31	0	31	0	66	97	66	97	192	245	192	245	53	0	53	0	0.22	0.00	0.68	1.00
	3-4 AA	3A	4A	1	0	1	0	96	97	96	97	227	234	227	234	18	11	18	11	0.07	0.04	0.99	1.00
	3-5 AA	3A	5A	1	0	1	0	96	97	96	97	227	239	227	239	18	6	18	6	0.07	0.02	0.99	1.00
	3-7 AA	3A	7A	1	0	1	0	96	97	96	97	227	245	227	245	18	0	18	0	0.07	0.00	0.99	1.00
	4-5 AA	4A	5A	0	0	0	0	97	97	97	97	234	239	234	239	11	6	11	6	0.04	0.02	1.00	1.00
	4-7 AA	4A	7A	0	0	0	0	97	97	97	97	234	245	234	245	11	0	11	0	0.04	0.00	1.00	1.00
	5-7 AA	5A	7A	0	0	0	0	97	97	97	97	239	245	239	245	6	0	6	0	0.02	0.00	1.00	1.00
	8-9 NN	8N	9N	0	0	0	0	97	97	97	97	245	0	0	245	0	245	245	0	1.00	0.00	1.00	1.00
Functionally Diverse	1-8 AN	1A	8N	61	0	61	0	36	97	36	97	16	245	16	245	229	0	229	0	0.93	0.00	0.37	1.00
	1-9 AN	1A	9N	61	0	61	0	36	97	36	97	16	0	0	16	229	245	245	229	1.00	0.93	0.37	1.00
	2-8 AN	2A	8N	31	0	31	0	66	97	66	97	192	245	192	245	53	0	53	0	0.22	0.00	0.68	1.00
	2-9 AN	2A	9N	31	0	31	0	66	97	66	97	192	0	0	192	53	245	245	53	1.00	0.22	0.68	1.00
	3-8 AN	3A	8N	1	0	1	0	96	97	96	97	227	245	227	245	18	0	18	0	0.07	0.00	0.99	1.00
	3-9 AN	3A	9N	1	0	1	0	96	97	96	97	227	0	0	227	18	245	245	18	1.00	0.07	0.99	1.00
	4-8 AN	4A	8N	0	0	0	0	97	97	97	97	234	245	234	245	11	0	11	0	0.04	0.00	1.00	1.00
	4-9 AN	4A	9N	0	0	0	0	97	97	97	97	234	0	0	234	11	245	245	11	1.00	0.04	1.00	1.00
	5-8 AN	5A	8N	0	0	0	0	97	97	97	97	239	245	239	245	6	0	6	0	0.02	0.00	1.00	1.00
	5-9 AN	5A	9N	0	0	0	0	97	97	97	97	239	0	0	239	6	245	245	6	1.00	0.02	1.00	1.00
	7-8 AN	7A	8N	0	0	0	0	97	97	97	97	245	245	245	245	0	0	0	0	0.00	0.00	1.00	1.00
	7-9 AN	7A	9N	0	0	0	0	97	97	97	97	245	0	0	245	0	245	245	0	1.00	0.00	1.00	1.00
	1-6 AD	1A	6D	61	0	61	0	36	97	36	97	16	91	16	91	229	154	229	154	0.93	0.63	0.37	1.00
	2-6 AD	2A	6D	31	0	31	0	66	97	66	97	192	91	45	238	53	154	200	7	0.82	0.03	0.68	1.00
	3-6 AD	3A	6D	1	0	1	0	96	97	96	97	227	91	80	238	18	154	165	7	0.67	0.03	0.99	1.00
	4-6 AD	4A	6D	0	0	0	0	97	97	97	97	234	91	80	245	11	154	165	0	0.67	0.00	1.00	1.00
	5-6 AD	5A	6D	0	0	0	0	97	97	97	97	239	91	85	245	6	154	160	0	0.65	0.00	1.00	1.00
	6-7 DA	6D	7A	0	0	0	0	97	97	97	97	91	245	91	245	154	0	154	0	0.63	0.00	1.00	1.00
	6-8 DN	6D	8N	0	0	0	0	97	97	97	97	91	245	91	245	154	0	154	0	0.63	0.00	1.00	1.00
	6-9 DN	6D	9N	0	0	0	0	97	97	97	97	91	0	0	91	154	245	245	154	1.00	0.63	1.00	1.00

Table 6-6 Counts of detections and failures for benign traffic (False Positive and True Negative) and for legitimate attack (False Negative and True Positive) for each IDS and for the 36 combinations of 2-version systems 1oo2 and 2oo2 for **TikiWiki** and taking into account the type of IDS

				FP				TN				FN				TP				Sensitivity		Specificity	
	Combined System	A	B	FP (A)	FP (B)	FP 1oo2	FP 2oo2	TN (A)	TN (B)	TN 1oo2	TN 2oo2	FN (A)	FN (B)	FN 1oo2	FN 2oo2	TP (A)	TP (B)	TP 1oo2	TP 2oo2	1oo2 (A,B)	2oo2 (A,B)	1oo2 (A,B)	2oo2 (A,B)
Functionally Redundant	1-2 AA	1A	2A	42	20	42	20	38	60	38	60	39	58	39	58	37	18	37	18	0.49	0.24	0.48	0.75
	1-3 AA	1A	3A	42	1	42	1	38	79	38	79	39	61	39	61	37	15	37	15	0.49	0.20	0.48	0.99
	1-4 AA	1A	4A	42	0	42	0	38	80	38	80	39	61	39	61	37	15	37	15	0.49	0.20	0.48	1.00
	1-5 AA	1A	5A	42	0	42	0	38	80	38	80	39	70	39	70	37	6	37	6	0.49	0.08	0.48	1.00
	1-7 AA	1A	7A	42	0	42	0	38	80	38	80	39	60	34	65	37	16	42	11	0.55	0.14	0.48	1.00
	2-3 AA	2A	3A	20	1	20	1	60	79	60	79	58	61	58	61	18	15	18	15	0.24	0.20	0.75	0.99
	2-4 AA	2A	4A	20	0	20	0	60	80	60	80	58	61	58	61	18	15	18	15	0.24	0.20	0.75	1.00
	2-5 AA	2A	5A	20	0	20	0	60	80	60	80	58	70	58	70	18	6	18	6	0.24	0.08	0.75	1.00
	2-7 AA	2A	7A	20	0	20	0	60	80	60	80	58	60	52	66	18	16	24	10	0.32	0.13	0.75	1.00
	3-4 AA	3A	4A	1	0	1	0	79	80	79	80	61	61	61	61	15	15	15	15	0.20	0.20	0.99	1.00
	3-5 AA	3A	5A	1	0	1	0	79	80	79	80	61	70	61	70	15	6	15	6	0.20	0.08	0.99	1.00
	3-7 AA	3A	7A	1	0	1	0	79	80	79	80	61	60	53	68	15	16	23	8	0.30	0.11	0.99	1.00
	4-5 AA	4A	5A	0	0	0	0	80	80	80	80	61	70	61	70	15	6	15	6	0.20	0.08	1.00	1.00
	4-7 AA	4A	7A	0	0	0	0	80	80	80	80	61	60	53	68	15	16	23	8	0.30	0.11	1.00	1.00
	5-7 AA	5A	7A	0	0	0	0	80	80	80	80	70	60	56	74	6	16	20	2	0.26	0.03	1.00	1.00
	8-9 NN	8N	9N	0	0	0	0	80	80	80	80	76	38	38	76	0	0	0	0	0.50	0.00	1.00	1.00
Functionally Diverse	1-8 AN	1A	8N	42	0	42	0	38	80	38	80	39	76	39	76	37	0	37	0	0.49	0.00	0.48	1.00
	1-9 AN	1A	9N	42	0	42	0	38	80	38	80	39	38	20	57	37	38	56	19	0.74	0.25	0.48	1.00
	2-8 AN	2A	8N	20	0	20	0	60	80	60	80	58	76	58	76	18	0	18	0	0.24	0.00	0.75	1.00
	2-9 AN	2A	9N	20	0	20	0	60	80	60	80	58	38	31	65	18	38	45	11	0.59	0.14	0.75	1.00
	3-8 AN	3A	8N	1	0	1	0	79	80	79	80	61	76	61	76	15	0	15	0	0.20	0.00	0.99	1.00
	3-9 AN	3A	9N	1	0	1	0	79	80	79	80	61	38	31	68	15	38	45	8	0.59	0.11	0.99	1.00
	4-8 AN	4A	8N	0	0	0	0	80	80	80	80	61	76	61	76	15	0	15	0	0.20	0.00	1.00	1.00
	4-9 AN	4A	9N	0	0	0	0	80	80	80	80	61	38	31	68	15	38	45	8	0.59	0.11	1.00	1.00
	5-8 AN	5A	8N	0	0	0	0	80	80	80	80	70	76	70	76	6	0	6	0	0.08	0.00	1.00	1.00
	5-9 AN	5A	9N	0	0	0	0	80	80	80	80	70	38	34	74	6	38	42	2	0.55	0.03	1.00	1.00
	7-8 AN	7A	8N	0	0	0	0	80	80	80	80	60	76	60	76	16	0	16	0	0.21	0.00	1.00	1.00
	7-9 AN	7A	9N	0	0	0	0	80	80	80	80	60	38	38	60	16	38	38	16	0.50	0.21	1.00	1.00
	1-6 AD	1A	6D	42	0	42	0	38	80	38	80	39	0	0	39	37	76	76	37	1.00	0.49	0.48	1.00
	2-6 AD	2A	6D	20	0	20	0	60	80	60	80	58	0	0	58	18	76	76	18	1.00	0.24	0.75	1.00
	3-6 AD	3A	6D	1	0	1	0	79	80	79	80	61	0	0	61	15	76	76	15	1.00	0.20	0.99	1.00
	4-6 AD	4A	6D	0	0	0	0	80	80	80	80	61	0	0	61	15	76	76	15	1.00	0.20	1.00	1.00
	5-6 AD	5A	6D	0	0	0	0	80	80	80	80	70	0	0	70	6	76	76	6	1.00	0.08	1.00	1.00
	6-7 DA	6D	7A	0	0	0	0	80	80	80	80	0	60	0	60	76	16	76	16	1.00	0.21	1.00	1.00
	6-8 DN	6D	8N	0	0	0	0	80	80	80	80	0	76	0	76	76	0	76	0	1.00	0.00	1.00	1.00
	6-9 DN	6D	9N	0	0	0	0	80	80	80	80	0	38	0	38	76	38	76	38	1.00	0.50	1.00	1.00

Figure 6-1 shows the three ROC plots, one for each of the three applications (MyRefereces, phpBB and TikiWiki). On each plot we have the following:

- The blue diamonds represent the single IDS system (and we have labelled them in the plot);
- The orange squares represent the 1oo2 systems;
- The green triangles represent the 2oo2 systems.

The optimal system in a ROC plot is one that appears on the top left-hand corner (i.e. one that has a true positive rate of 1 (it detects all attacks) and a false positive rate of 0 (it never raises an alarm for benign traffic)). We have one such system for phpBB and TikiWiki, but not for MyReferences (i.e. there were no IDSs with perfect performance for MyReferences). From Figure 5-1 in general we observe a lot of 1oo2 systems outperforming the individual IDS on the true positive rates, and 2oo2 systems outperforming individual IDS on the true negative rates. This is consistent across all three applications. Figure 6-2 gives ROCs per application again, but now we have split the points of the “functionally redundant” (subfigures a-c) and “functionally diverse” (sub-figures d-f) pairs. Overall we see that the functionally diverse pairs are performing better than functionally redundant pairs (as evident by the higher number of 1oo2 systems (squares) appearing in the top half of the plot, and a higher number of 2oo2 systems (triangles) appearing on the left of the plot in subfigures d-f, compared with those in a-c).

Next, we measured the differences in the Accuracy (Figure 6-3), Specificity (Figure 6-4) and Sensitivity (Figure 6-5) of the diverse systems versus the best individual IDS in those respective pairs. The goal is to understand how much better, or how much worse, a given diverse system performs compared with the best IDS in that pair. So we subtract from a given measure (accuracy, specificity or sensitivity) of a 1oo2 or 2oo2 system, the corresponding measure of the best individual IDS in the pair.

We kept the ordering in the x-axis the same as in Table 6-4. Hence the leftmost 16 points in the x-axis are for the functionally redundant pairs, and the remaining ones for functionally diverse pairs. The y-axis shows the differences. A positive value means that the diverse system has a better performance than the best single IDS of the respective pair; a negative value means the performance of the pair is worse than the best single system of the respective pair; and a zero value in the y-axis means the performance of the new diverse pair is the same as the best single system in the respective pair.

In summary:

- **For accuracy**: 1oo2 systems perform better overall for MyReferences and phpBB applications. The picture is less clear for TikiWiki, where 2oo2 systems perform at least no worse than the best single IDS for most of the application-only IDSs, though for diverse pairs 1oo2 systems perform better or at least no worse than the best IDS in the pair. We should note that Accuracy may not be a fair measure for MyReferences and phpBB applications in the dataset we have used, as the traffic that is being inspected for these applications is attack intensive (hence favouring 1oo2 configurations). The traffic sent to TikiWiki was more balanced between attacks and benign traffic and this is reflected in the smaller differences between 1oo2 and 2oo2 setups for that application in most cases.
- **Specificity**: we would expect gains only in 2oo2 setups. For this dataset we only get improvements in some cases for MyReferences, and in all cases they are application-only setups. This is because the specificity of an individual system is already very high. We are much worse off for 1oo2 setups, especially for some pairs of application-only 1oo2 systems.
- **Sensitivity**: we see improvements in sensitivity of 1oo2 systems, especially for application-network (specifically of ACD IDSs with 9N – “Snort 2.8 plus CR”) and application-database pairs. Diversity in failure behaviour against malicious traffic, benefits 1oo2 setups considerably. 2oo2 systems on the other hand perform very badly in most cases for sensitivity compared with the best individual IDS in the pair.

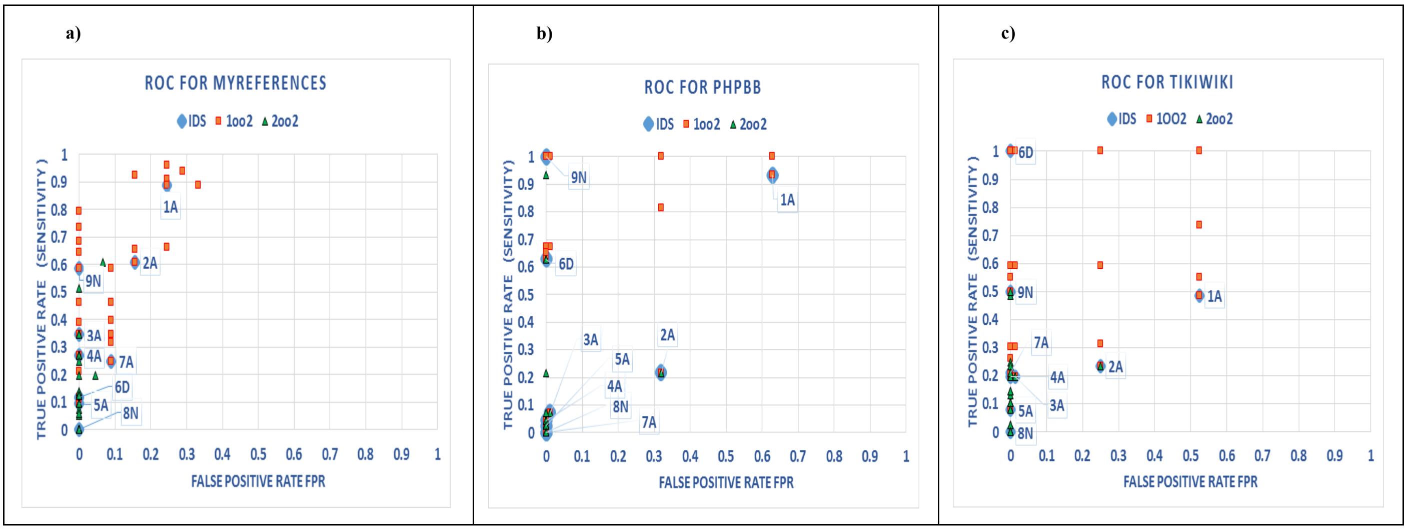


Figure 6-1 The ROC plot showing the individual IDSs, 1oo2 and 2oo2 configurations. charts a)-c) show the ROC for each application

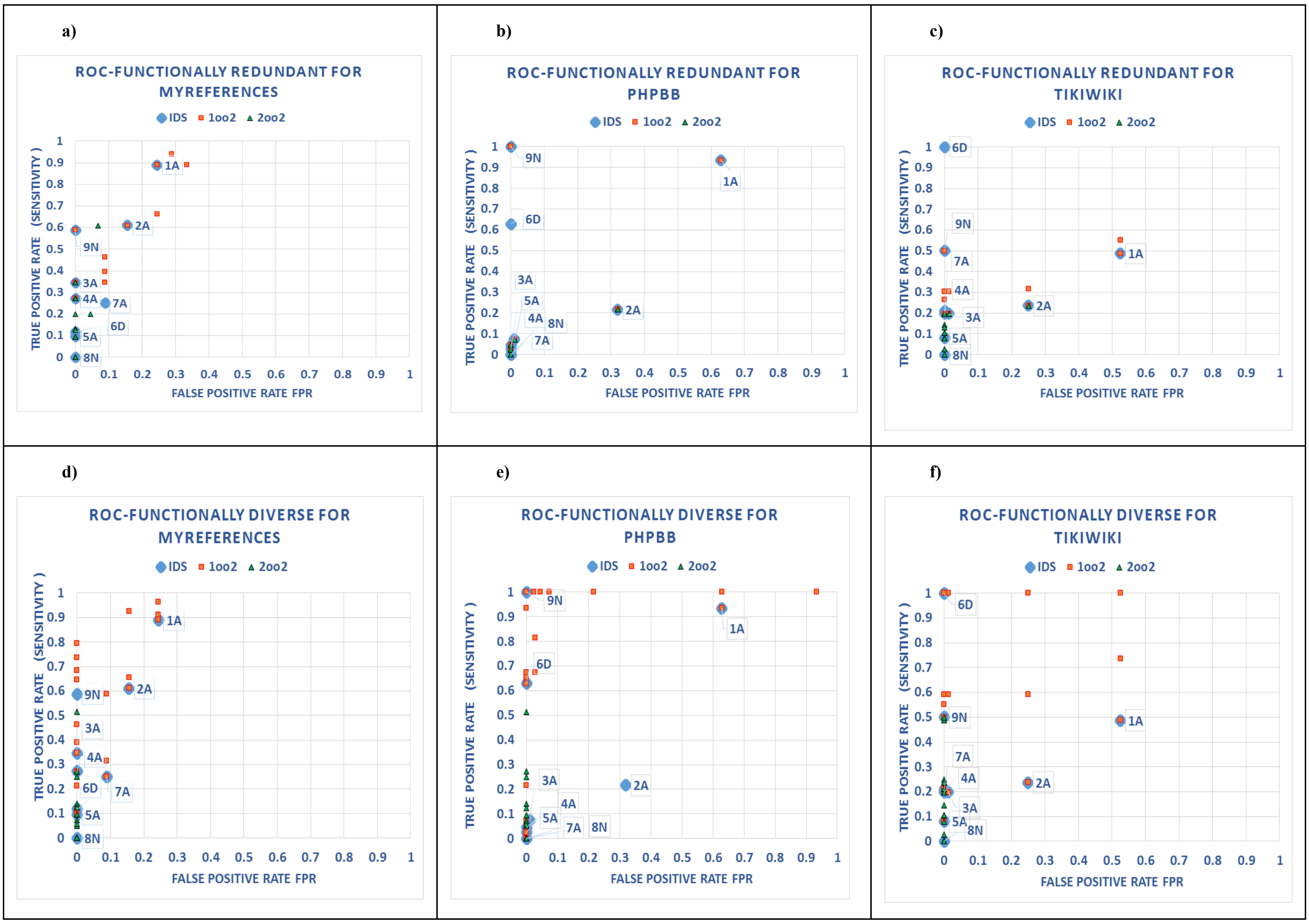


Figure 6-2 The ROC plots showing the individual IDSs, 1oo2 and 2oo2 configurations: charts a)-c) the ROCs for each application when the pairs were functionally redundant; charts d)-f) the ROCs for each application when the pairs were functionally diverse

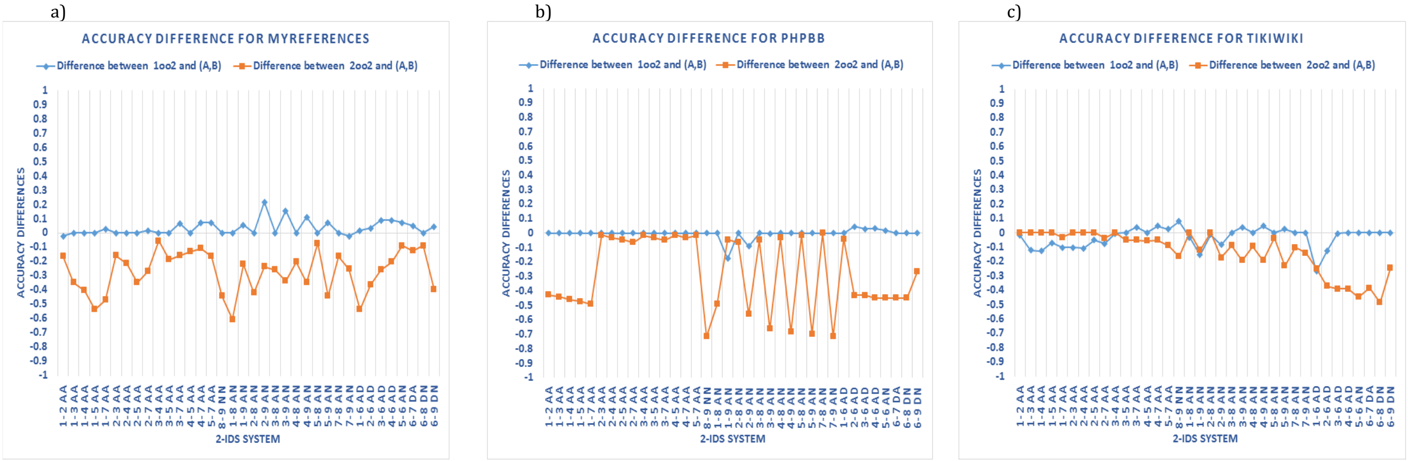


Figure 6-3 The accuracy difference of 1oo2 and 2oo2 configurations, compared with the best single system in the respective pair, charts a)-c) show these differences for each application when the pairs were functionally redundant or diverse

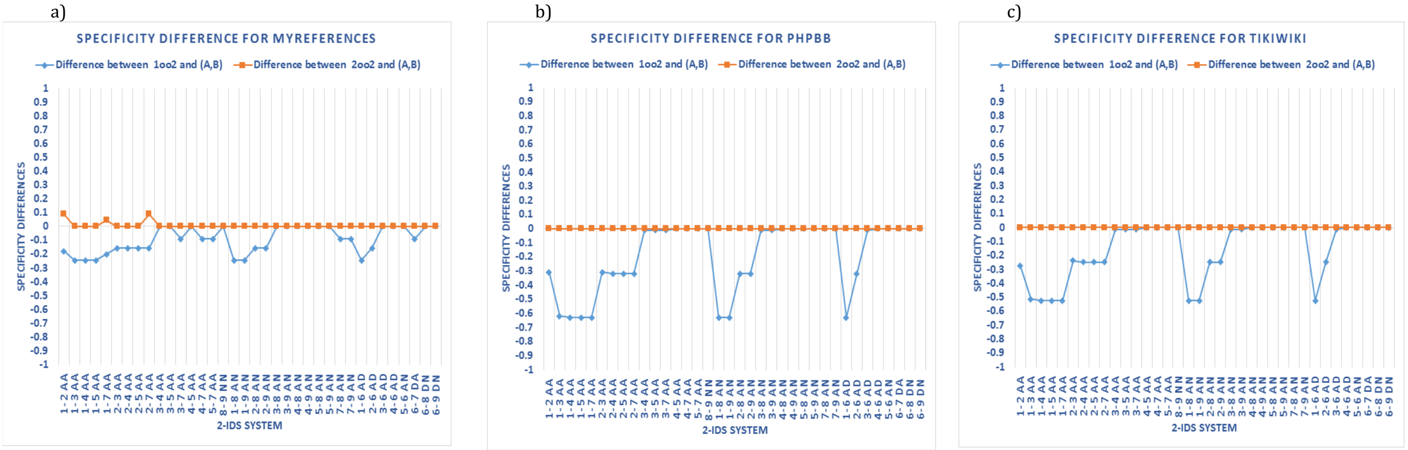


Figure 6-4 The specificity difference of 1oo2 and 2oo2 configurations, compared with the best single system in the respective pair, charts a)-c) show these differences for each application when the pairs were functionally redundant or diverse.

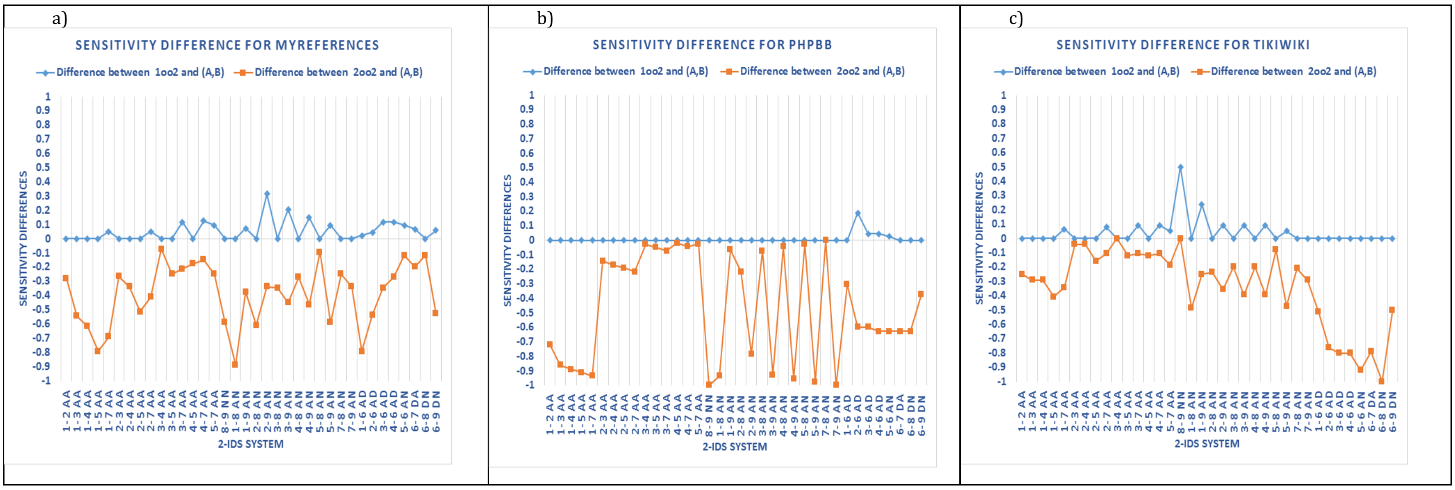


Figure 6-5 The sensitivity difference of 1oo2 and 2oo2 configurations, compared with the best single system in the respective pair, charts a)-c) show these differences for each application when the pairs were functionally redundant or diverse

6.5.2 Analysis by HTTP method

The authors in Elia et al. (2010) also classified the attacks based on the HTTP method used to carry out the attack: through a GET or a POST. We also analysed this aspect in Table 6-7, which gives the counts of successful attacks using GET and POST for each Web application.

Table 6-7 The counts of successful attacks by using GET and POST attack method per web application

Web Application	GET Method	POST Method
MyReferences	99	37
phpBB	77	168
TikiWiki	33	43

Since we are only dealing with attack traffic, we will concentrate on sensitivity differences only. These are given in Figure 6-6 Plots a)-c) of this figure give the differences in Sensitivity for GET attacks; whereas plots d)-f) give the differences in Sensitivity for POST attacks.

It becomes clear from these difference plots that for these applications the gains in diversity (in the 1oo2 setups) are almost entirely from GET attacks (and this seems to be mainly for MyReferences and TikiWiki applications). We see no improvements in most cases for POST attacks. This is likely due to the way some of these IDSs monitor the traffic. As reported in Elia et al. (2010) the Scalp and ACD IDSs monitor the Apache access log. Only the malicious payload of GET based attacks are stored in the access log, thus making POST based attacks almost undetectable by these tools and their variants.

We should note that since the differences in improvements / deteriorations are between the 1oo2 / 2oo2 systems and the best system in a given pair, the results presented in figure 6-6 may seem at first glance to not be consistent with those in Figure 6-5. For example, for phpBB, we see no improvements compared with the best system in the pair when we look at the demands separately in Figure 6-6, but we did see improvements when we looked at all the demands in Figure 6-5. We checked the results and we found that for those pairs (usually ACD versions with GreenSQL) all the improvements in sensitivity for 1oo2 for GET demands were from ACD versions of the IDS. Whereas for POST demands the improvement were due to GreenSQL. So when we look at the demands separately we see no gain compared with the best system in the pair, but when we look at all demands we do see the gains.

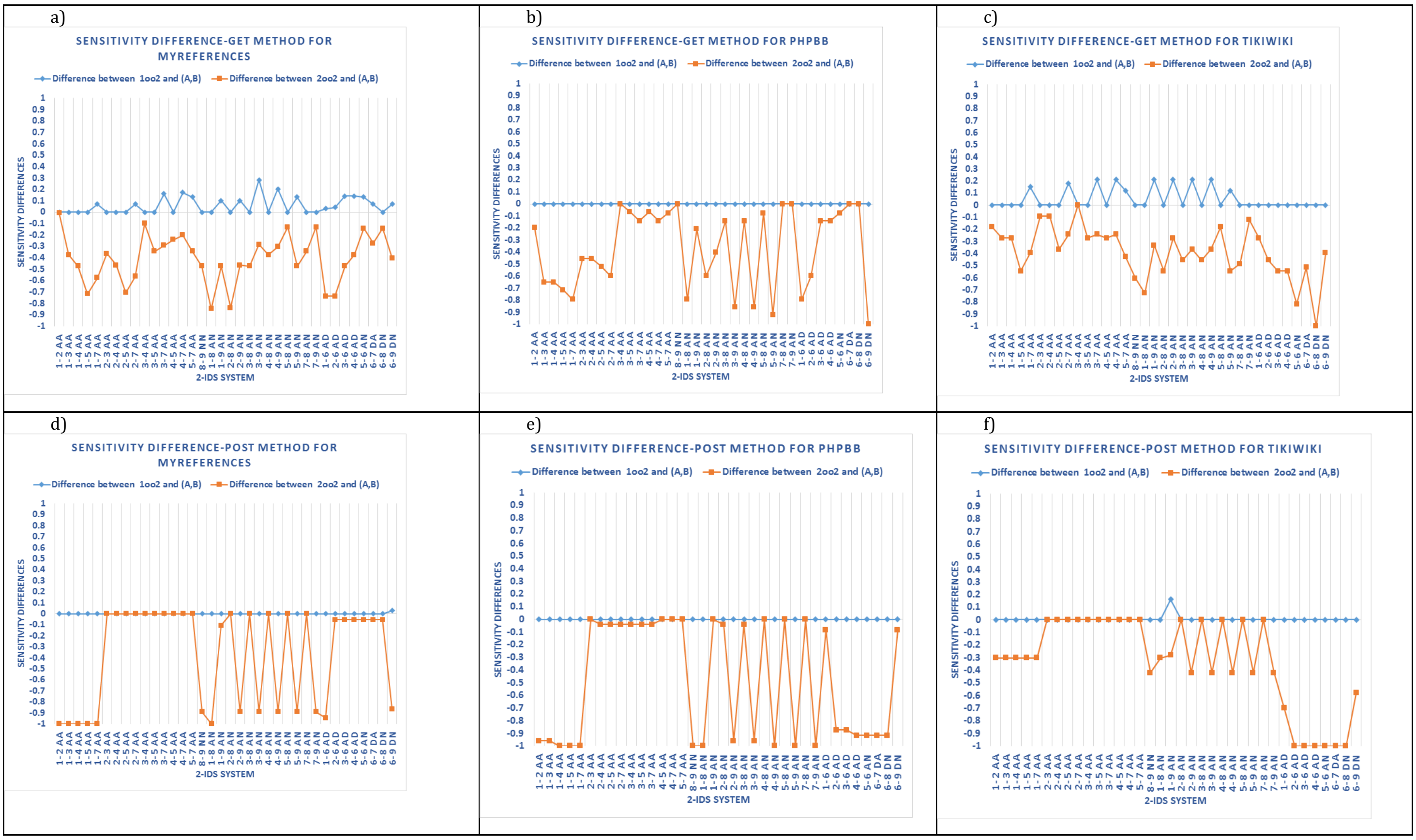


Figure 6-6 The sensitivity difference of 1oo2 and 2oo2 configurations compared with the best single system in the respective pair, charts a)-c) show these differences for each application when attacks were Read (Get); whereas charts d-f), show them when they were Write (Post).

6.5.3 Differences over a single IDS setup

So far the results are useful for a decision maker that is deciding which two IDSs to choose for a diverse setup. However, organisations may already be using an IDS, and costs of switching to a different pair of IDSs may be prohibitively high (in terms of licensing, re-training staff, etc.). For these organisations it may be useful to know which IDS B they should choose to run alongside their existing product A in a diverse 1oo2 or 2oo2 AB setup. Figure 6-7 shows results for this type of comparison for MyReferences application. For each IDS, we show the improvements (positive values in the y-axis - shown in orange bars), or deterioration (negative values in the y-axis – shown in blue bars), in sensitivity and specificity for 1oo2 (sub-figure *a*), and 2oo2 (sub-figure *b*). Each IDS can be paired with eight other IDSs in our study. Using the left-most box in subfigure a) as an example, the first blue bar shows the deterioration in specificity that a user of ACD1 IDS would observe if they switched to a 1oo2 setup ACD1-ACD3. Since there is no corresponding orange bar it means there is no improvement in sensitivity for users of ACD1 from switching to a 1oo2 ACD1-ACD3 configuration. Note that for users of

ACD3 the observation could be different, and indeed it is, as can be seen from the first stacked bar in the second dashed-line-box (ACD3): we can see that these users would see a sizeable sensitivity improvement from switching to a 1oo2 setup (though a specificity deterioration as well). Whether this improvement in sensitivity would be worth it for the organisation that uses ACD3 would of course depend on the relative costs that this organisation places on false positives and false negatives.

From Figure 6-7 we observe the following for **1oo2 setups**:

- For application-based IDSs, pairing with a functionally diverse IDS ("GreenSQL" or "Snort 2.8 plus Custom Rules"), in a 1oo2 setup brings considerable improvements in sensitivity at no cost to specificity.
- For GreenSQL (database IDS) pairing with the functionally-diverse "Snort 2.8 plus Custom Rules" in a 1oo2 setup brings considerable improvements in sensitivity at no cost to specificity.
- The same is also true when pairing with the functionally-diverse anomaly-based ACD with higher thresholds (ACD10, ACD30 and ACD100). When pairing with ACD1 and ACD3 and Scalp, there are improvements in sensitivity but there is also some deterioration in specificity.
- For "Snort 2.8 plus Custom Rules" pairing with the functionally-diverse ACD with a higher threshold (ACD10, ACD30 and ACD100), or with GreenSQL leads to improvements in sensitivity at no cost to specificity.
- The largest improvements in sensitivity from switching to a diverse 1oo2 configuration would be observed by organisations that are using an anomaly-based application IDS with a high threshold value (ACD30, ACD100) and they are paired with another anomaly-based application IDS that has a low threshold value (ACD1 and ACD3 for example). But for these organizations it would be more cost effective to just use a single IDS with a lower threshold value, if sensitivity is their primary concern.

In **2oo2 setups** (part b) of Figure 6-8 we observe some improvements in specificity but these are in most cases far outweighed by the significant deteriorations in sensitivity.

The observations above about functional diversity pairings are consistent also for the other two applications (phpBB and TikiWiki), as we can observe from the remaining Figures (6-9 to 6-12); for example in phpBB all the 245 attacks are detected by Snort 2.8 Plus Custom Rules, accordingly when pairing any single functionally diverse IDS with Snort 2.8 Plus it improves sensitivity, see Figure (6-9).

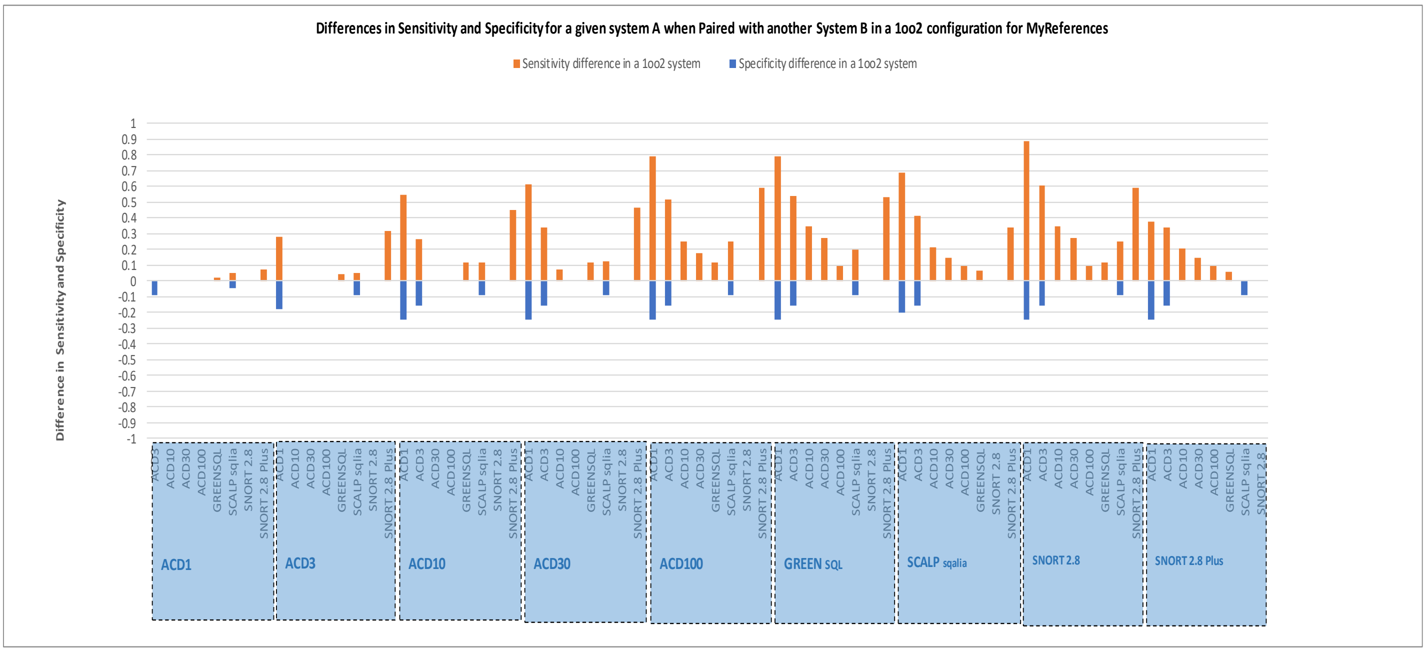


Figure 6-7 Differences in sensitivity and specificity for a given system A when paired with another System B for MyReferences for a 1oo2 configuration

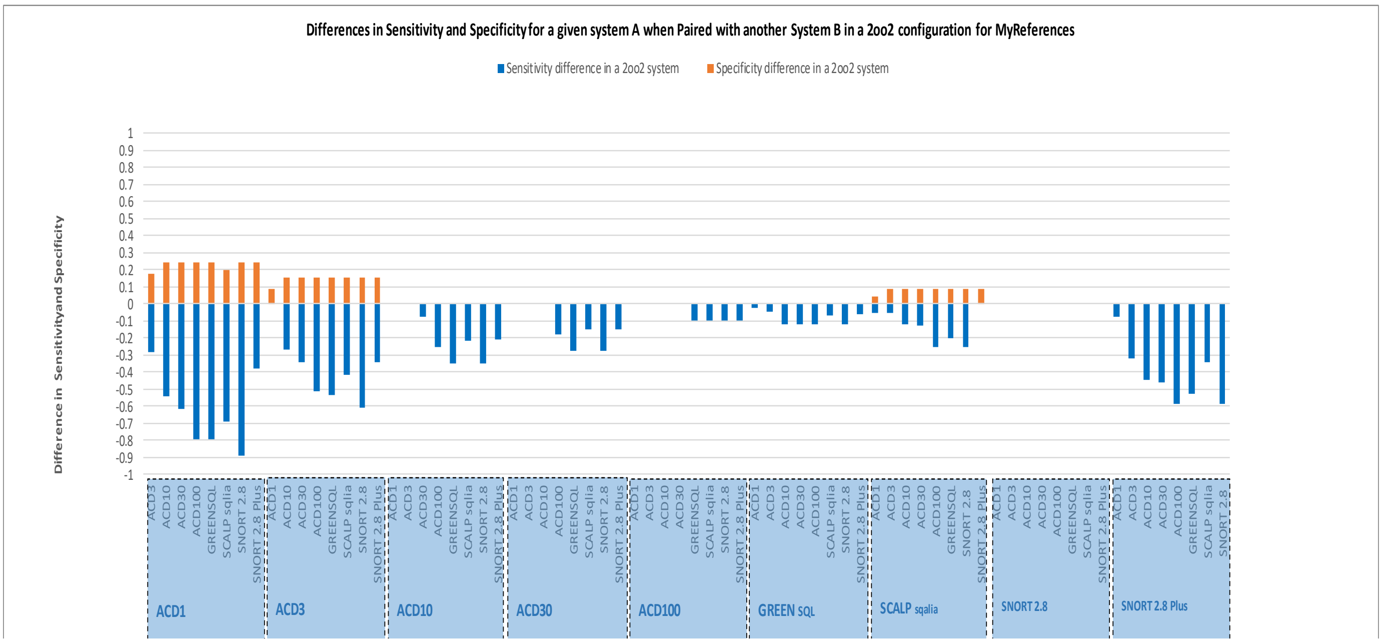


Figure 6-8 Differences in sensitivity and specificity for a given system A when paired with another System B for MyReferences for a 2oo2 configuration

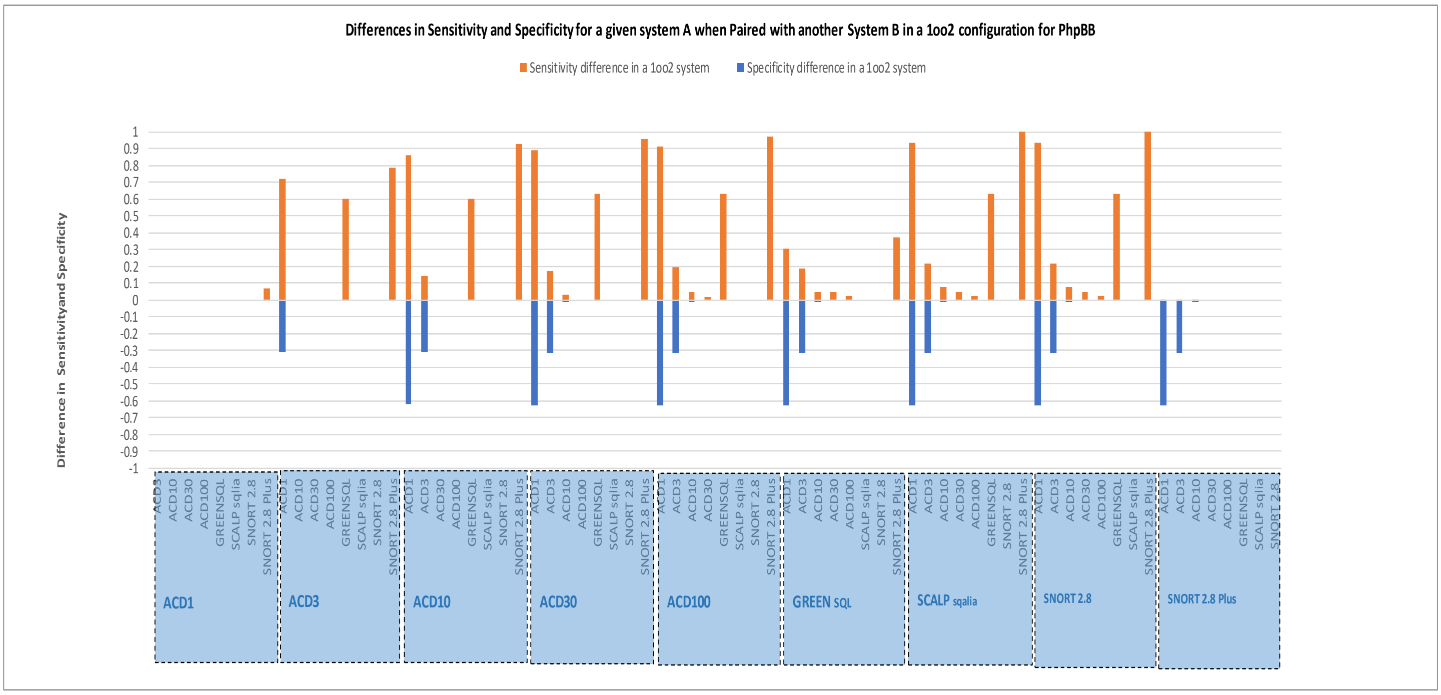


Figure 6-9 Differences in sensitivity and specificity for a given system A when paired with another System B for phpBB for a 1oo2 configuration

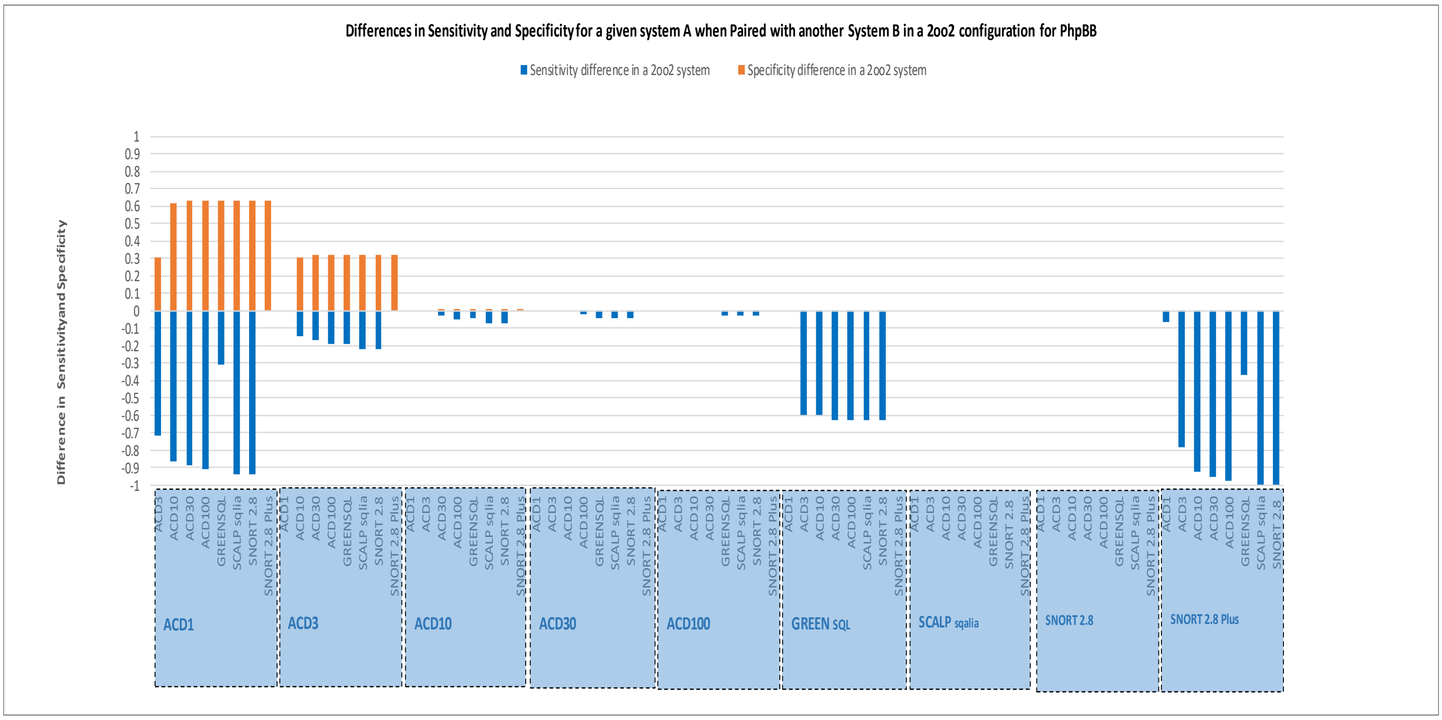


Figure 6-10 Differences in sensitivity and specificity for a given system A when paired with another System B for phpBB for a 2oo2 configuration

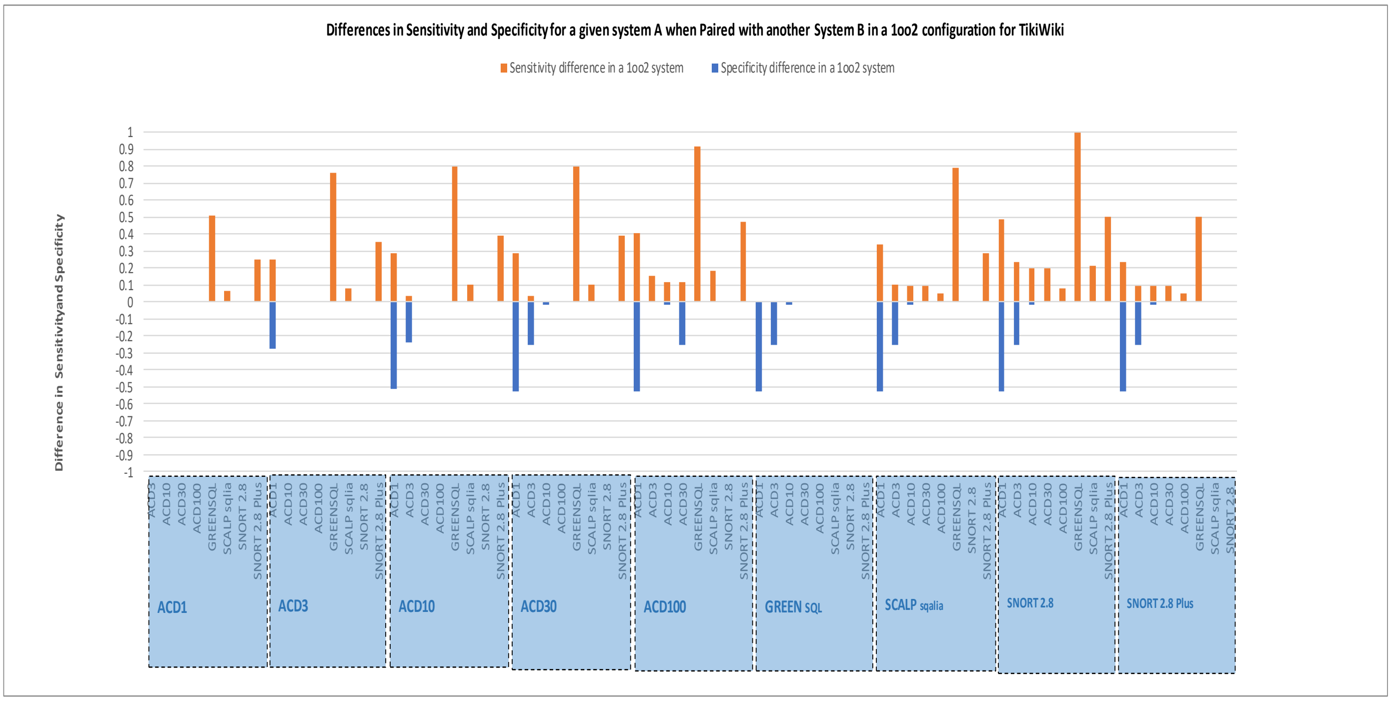


Figure 6-11 Differences in sensitivity and specificity for a given system A when paired with another System B for TikiWiki for a 1oo2 configuration

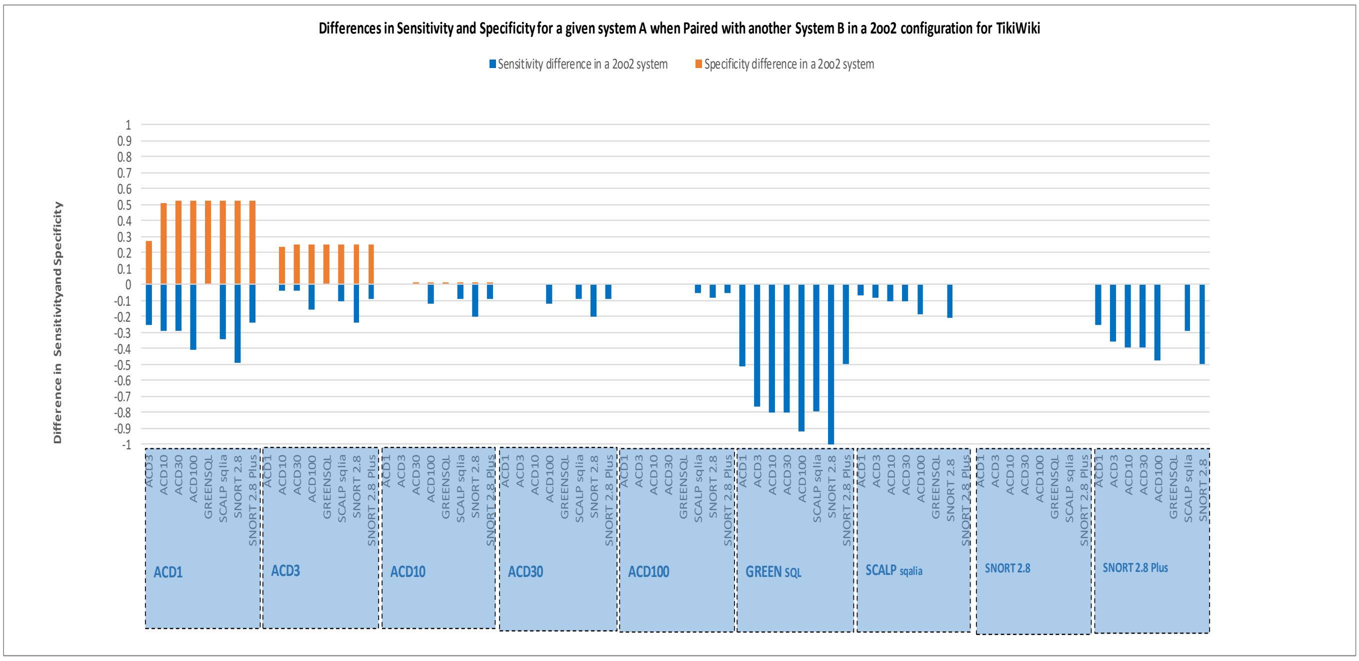


Figure 6-12 Differences in sensitivity and specificity for a given system A when paired with another System B for TikiWiki for a 2oo2 configuration

6.5.4 Averages for different two-version diverse setups

We conclude our analysis of two-version systems with a summary table (Table 6-8) showing the average Sensitivity, Specificity and Accuracy for single IDSs compared with these same averages for 1oo2 and 2oo2 configurations. The averages are sub-divided by type of configurations and application. These results confirm the observations shown so far:

- For 1oo2 systems: improvements in sensitivity compared with individual systems are around 40% on average for the three applications, but come with around 10% specificity deterioration on average.
- For 1oo2 systems: the largest improvements in sensitivity, with the least deterioration in specificity are from functionally-diverse pairs of IDSs;
- For 2oo2 systems: improvements in specificity compared with individual systems are around 10% on average for the three applications, but come with around 60% sensitivity deterioration on average;
- For 2oo2 systems: the largest improvements in specificity, with the least deterioration in sensitivity, are from functionally-diverse pairs of IDSs (specifically Application & Database, and Network & Database).

Table 6-8 The average sensitivity, specificity and accuracy for single IDS and the 1oo2 and 2002 pairs, per type and per web application

Averages			MyReferences			PhpBB			TikiWiki		
			Sensitivity	Specificity	Accuracy	Sensitivity	Specificity	Accuracy	Sensitivity	Specificity	Accuracy
Single System			0.32	0.91	0.63	0.32	0.89	0.49	0.32	0.91	0.63
1oo2	*Overall 1oo2*		0.51	0.83	0.68	0.56	0.80	0.63	0.51	0.83	0.68
	Functionally Redundant	*Application-only*	0.33	0.76	0.55	0.39	0.70	0.48	0.33	0.76	0.55
		Network-only	0.50	1	0.76	1	1	1	0.50	1	0.76
	Functionally Diverse	*Application & Network*	0.41	0.87	0.65	0.61	0.84	0.67	0.41	0.87	0.65
		Application & Database	1	0.87	0.93	0.73	0.84	0.76	1	0.87	0.93
		Network & Database	1	1	1	0.81	1	0.87	1	1	1
2oo2	*Overall 2oo2*		0.13	0.99	0.57	0.09	0.99	0.34	0.13	0.99	0.57
	Functionally Redundant	*Application-only*	0.13	0.98	0.57	0.04	0.98	0.31	0.13	0.98	0.57
		Network-only	0	1	0.51	0	1	0.28	0	1	0.51
	Functionally Diverse	*Application & Network*	0.07	1	0.55	0.11	1	0.36	0.07	1	0.55
		Application & Database	0.23	1	0.63	0.11	1	0.37	0.23	1	0.63
		Network & Database	0.25	1	0.63	0.31	1	0.51	0.25	1	0.63

6.6 Diversity analysis for configurations with more than two versions

So far we presented the results for two-version systems. We then expanded the analysis to consider all the possible combinations we can build with the 9 IDSs. We calculate the sensitivity and specificity for each of the diverse combinations with the nine IDSs, for three types of adjudication setups considered (namely 1ooN, simple Majority vote – e.g. 2-out-of-3, 3-out-of-5 etc., and NooN).

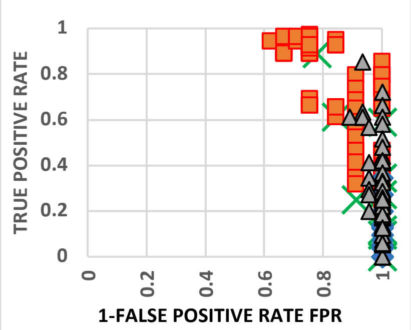
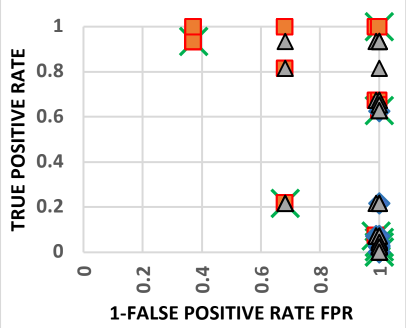
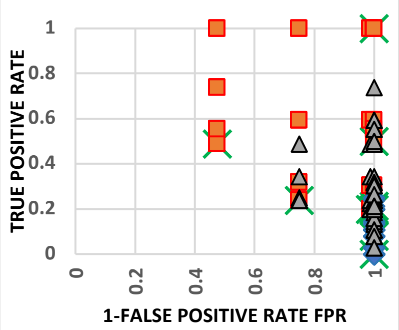
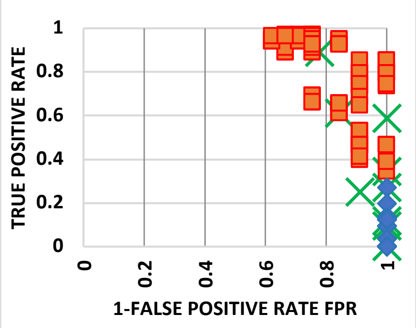
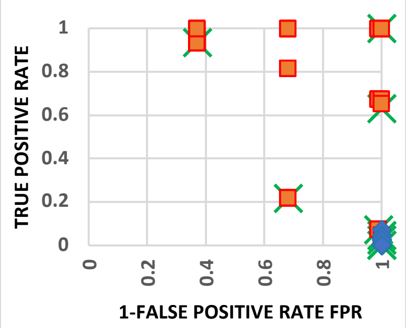
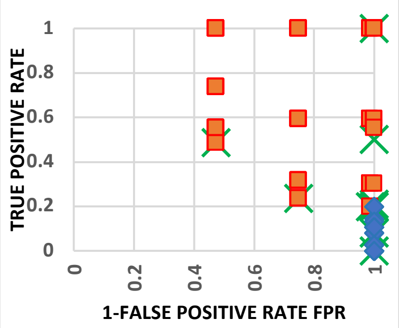
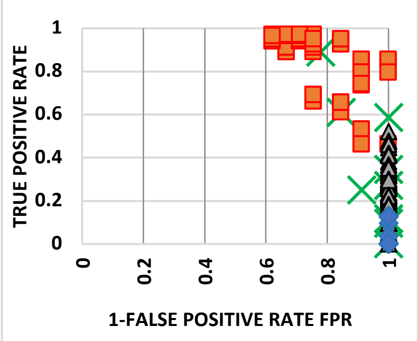
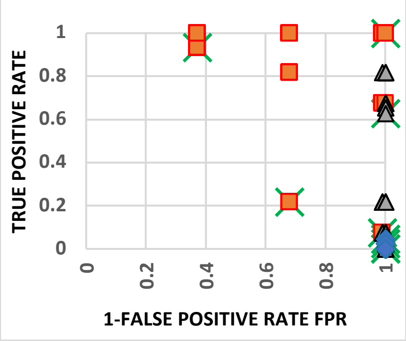
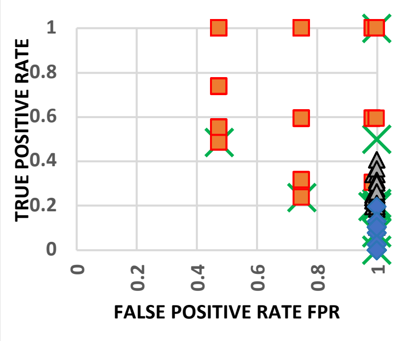
Figure 6-13 shows the ROC plots, one for each of the three web applications (MyRefereces, phpBB and TikiWiki). On each plot we have the following:

- The green X symbols represent the single IDS system;
- The orange squares represent the 1ooN systems;
- The blue diamonds represent the NooN systems.

- The green triangles with the black border represent the RooN (majority vote) systems, which were calculated for odd values of N, as they allow a simple majority vote to be calculated.

The best system in an ROC plot is one that appears on the top right-hand corner (i.e. one that has a true positive rate of 1 (it detects all attacks) and a false positive rate of 1 (it never raises an alarm for benign traffic)). From Figure 6-13 we observe a similar picture to what we have seen with two-versions systems so far: 1ooN systems outperforming the individual IDS on sensitivity, and NooN systems outperforming individual IDS on specificity.

✕ IDS 🟥 1ooN ▲ RooN ◆ NooN

N	MyReferences	PHBBB	TikiWiki
3, 9C3= 84			
4, 9C4 = 126			
5, 9C5 = 126			

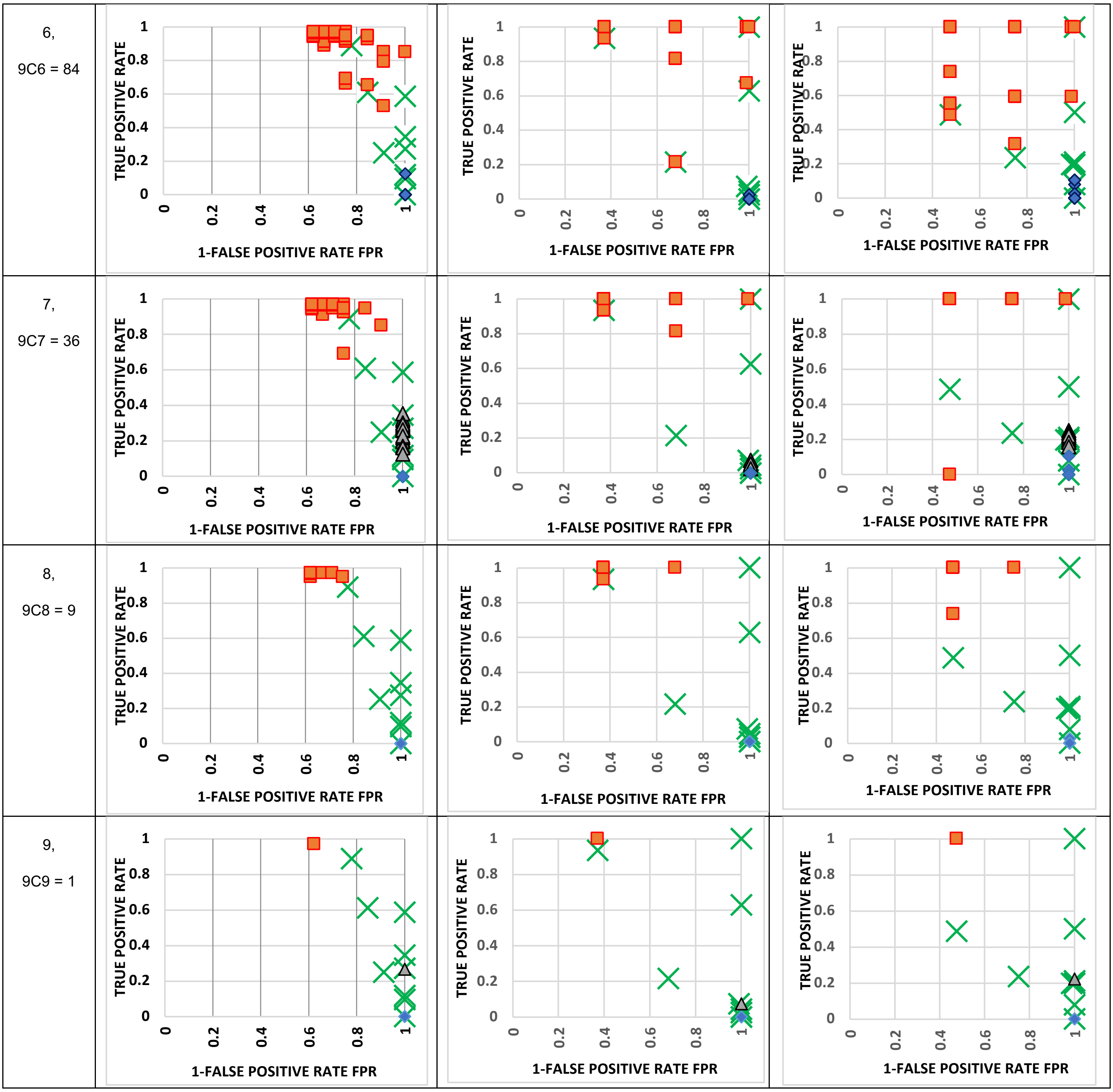


Figure 6-13 The ROC plot showing the individual IDSs, 1ooN and NooN configurations for each application.

6.6.1 Averages for different diverse setups

Tables 6-9 to 6-11 below show the average sensitivity and specify for each application for each of the main adjudication setups. Figure 6-14 then shows three graphs (one for each application) that visualise the numbers from the preceding three tables. The results are consistent with what we observed so far:

- Improvements in sensitivity for 1ooN systems as we increase N.
- Improvements in specificity for NooN systems as we increase N.
- Deterioration in specificity for 1ooN systems as we increase N.
- Deterioration in sensitivity for NooN systems as we increase N.

The tables below allow us to calculate more precisely just what those improvements (or deteriorations are).

Table 6-9 The averages of Sensitivity (TPR) and Specificity (1-FPR) for the combined IDSs, in 1ooN, RooN and NooN configurations, for **MyReferences**

N	TPR 1ooN	1-FPR 1ooN	TPR RooN	1-FPR RooN	TPR NooN	1-FPR NooN
2	0.5727	0.8944			0.1315	0.9969
3	0.7145	0.8463	0.2890	0.9907	0.0528	1
4	0.8049	0.7991			0.0211	1
5	0.8745	0.7593	0.2533	1	0.0071	1
6	0.9194	0.7204			0.0015	1
7	0.9485	0.6846	0.2504	1	0	1
8	0.9649	0.6519			0	1
9	0.9706	0.6222	0.2647	1	0	1

Table 6-10 The averages of Sensitivity (TPR) and Specificity (1-FPR) for the combined IDSs, in 1ooN, RooN and NooN configurations, for **PhpBB**

N	TPR 1ooN	1-FPR 1ooN	TPR RooN	1-FPR RooN	TPR NooN	1-FPR NooN
2	0.5604	0.7964			0.0890	0.9906
3	0.7247	0.7086	0.2260	0.9719	0.0204	0.9999
4	0.8288	0.6301			0.0048	1
5	0.9159	0.5606	0.1415	0.9988	0.0016	1
6	0.9595	0.5001			0.0003	1
7	0.9822	0.4485	0	1	0	1
8	0.9927	0.4055			0	1
9	1	0.3711	0.0735	1	0	1

Table 6-11 The averages of Sensitivity (TPR) and Specificity (1-FPR) for the combined IDSs, in 1ooN, RooN and NooN configurations, for **TikiWiki**

N	TPR 1ooN	1-FPR 1ooN	TPR RooN	1-FPR RooN	TPR NooN	1-FPR NooN
2	0.5128	0.8326			0.1334	0.9924
3	0.6430	0.7603	0.2523	0.9774	0.0739	1
4	0.7404	0.6952			0.0471	1
5	0.8205	0.6374	0.2221	0.9985	0.0279	1
6	0.8830	0.5866			0.0160	1
7	0.9722	0.5427	0.2127	1	0.0080	1
8	0.9708	0.5056			0.0029	1
9	1	0.4750	0.2237	1	0	1

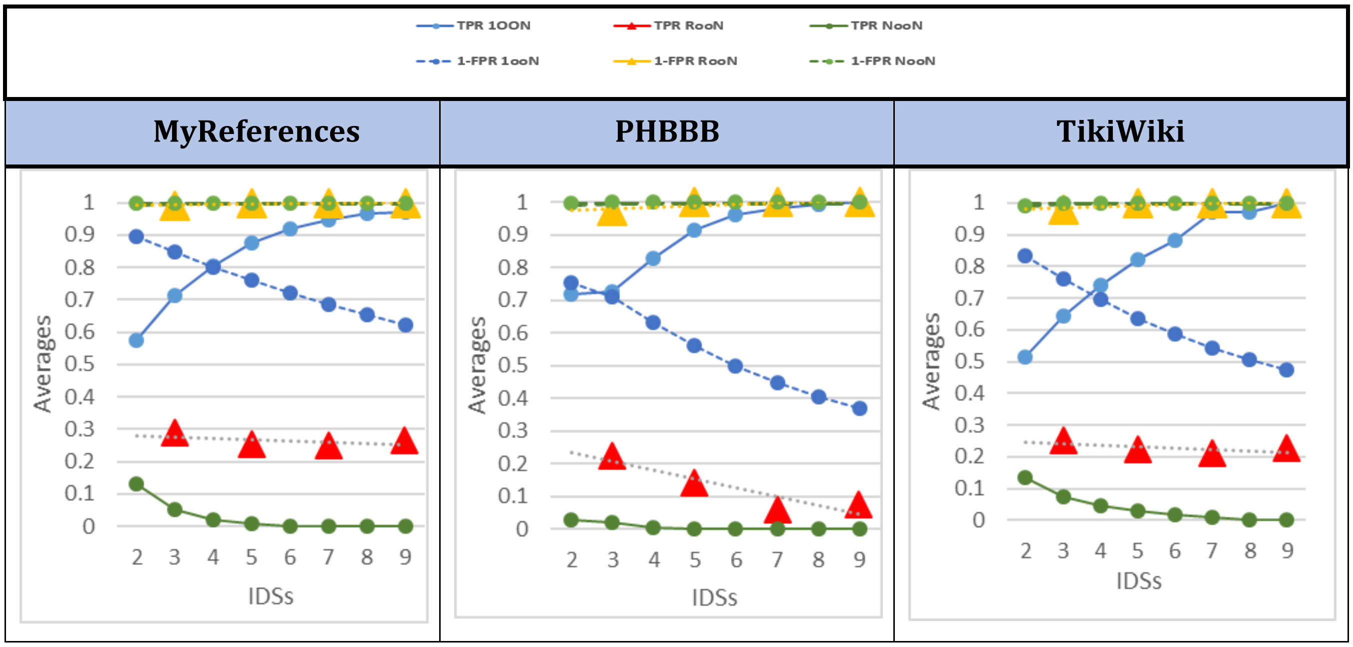


Figure 6-14 Plots show the averages of TPR and 1-FPR for the combined IDSs, in 1ooN, RooN and NooN configurations for all the web applications

6.6.2 Averages for functionally-redundant and diverse setups

In the preceding section we gave the averages for any diverse configuration. Similarly to what was presented for two-versions systems, we also did an analysis of the functionally redundant and functionally diverse configurations. Since we have 9 IDS, but only 4 IDS function types, then we have differentiated the different "degrees" of diversity for systems with more than two-versions.

Table 6-12 shows the average sensitivity, specificity and accuracy for single IDSs. This is the base reference. Tables 6-13, 6-14, 6-15 and 6-16 show averages for 1ooN, RooN and NooN configurations for each application for N=3, N=5, N=7 and N=9. We have subdivided these averages by the degree of diversity that exists between the types of IDS (A: Application, D: Database and N: Network), for each application. So for example AAD, means we have two IDSs of application type, and one IDS of Database type, in the three-version configuration. For 3-version and 5-version systems, since we have 6 Application level IDSs, we can still have functionally-redundant configurations (of application-only types). For 7-version and 9-version systems we can only have functionally-diverse configurations. In the tables, on each column, we highlight the best (green coloured cell) and worst (red coloured cell) performing configuration on average for each application and each adjudication scheme.

The results show that:

- On the Sensitivity measure:
  - For 1ooN systems, the functionally diverse systems are always performing best on average in all cases;
  - For majority vote and NooN, the functionally redundant systems perform best on average for MyReferences in most cases.
- On the Specificity measure:
  - Functionally diverse systems are always performing best on average in all cases, for all adjudicators.

Table 6-12 the average sensitivity, specificity and accuracy for single IDS and per web application

	MyReferences			PhpBB			TikiWiki		
IDS	**Sensitivity**	**Specificity**	**Accuracy**	**Sensitivity**	**Specificity**	**Accuracy**	**Sensitivity**	**Specificity**	**Accuracy**
Single System	0.45	0.94	0.1	0.32	0.89	0.49	0.32	0.91	0.63

Table 6-13 The average sensitivity and specificity for (1oo3,2oo3 and 3oo3) per web application

		MyReferences						PhpBB						TikiWiki					
		Sensitivity			Specificity			Sensitivity			Specificity			Sensitivity			Specificity		
IDS Combination Types		1oo3	2oo3	3oo3	1oo3	2oo3	3oo3	1oo3	2oo3	3oo3	1oo3	2oo3	3oo3	1oo3	2oo3	3oo3	1oo3	2oo3	3oo3
Overall		0.71	0.29	0.05	0.10	0.99	1.00	0.72	0.23	0.02	0.71	0.97	1.00	0.64	0.25	0.07	0.76	0.98	1.00
Functionally redundant		0.76	0.38	0.10	0.00	0.99	1.00	0.73	0.24	0.02	0.60	0.95	1.00	0.59	0.24	0.11	0.67	0.96	1.00
Functionally diverse	*Overall*	0.71	0.26	0.03	0.87	0.99	1.00	0.78	0.27	0.02	0.75	0.98	1.00	0.72	0.27	0.07	0.79	0.99	1.00
	AAN	0.70	0.28	0.05	0.85	0.99	1.00	0.67	0.20	0.01	0.72	0.98	1.00	0.62	0.25	0.07	0.77	0.98	1.00
	AAD	0.74	0.31	0.05	0.84	0.99	1.00	0.73	0.17	0.02	0.70	0.96	1.00	0.56	0.20	0.05	0.76	0.97	1.00
	ANN	0.77	0.28	0.02	0.86	1.00	1.00	0.83	0.42	0.10	0.74	1.00	1.00	0.73	0.38	0.09	0.78	1.00	1.00
	NND	0.32	0.05	0.00	0.91	1.00	1.00	0.63	0.00	0.00	1.00	1.00	1.00	1.00	0.21	0.00	1.00	1.00	1.00
	*fully diverse:***ADN**	0.65	0.17	0.01	0.91	1.00	1.00	0.81	0.27	0.01	0.81	1.00	1.00	0.82	0.29	0.05	0.85	1.00	1.00

Table 6-14 The average sensitivity and specificity for (1oo5,3oo5 and 5oo5) per web application

		MyReferences						PhpBB						TikiWiki					
		Sensitivity			Specificity			Sensitivity			Specificity			Sensitivity			Specificity		
IDS Combination Types		1oo5	3oo5	5oo5	1oo5	3oo5	5oo5	1oo5	3oo5	5oo5	1oo5	3oo5	5oo5	1oo5	3oo5	5oo5	1oo5	3oo5	5oo5
Overall		0.87	0.25	0.01	0.76	1.00	1.00	0.92	0.14	0.00	0.56	1.00	1.00	0.82	0.22	0.03	0.64	1.00	1.00
Functionally redundant		0.89	0.35	0.04	0.67	1.00	1.00	0.81	0.06	0.00	0.42	0.99	1.00	0.50	0.21	0.05	0.52	0.99	1.00
Functionally diverse	*Overall*	0.87	0.25	0.01	0.76	1.00	1.00	0.92	0.15	0.00	0.57	1.00	1.00	0.84	0.22	0.03	0.64	1.00	1.00
	AAAAN	0.88	0.32	0.02	0.72	1.00	1.00	0.84	0.08	0.00	0.49	1.00	1.00	0.57	0.20	0.03	0.58	1.00	1.00
	AAAAD	0.84	0.32	0.00	0.72	1.00	1.00	0.89	0.06	0.00	0.49	1.00	1.00	1.00	0.23	0.07	0.58	1.00	1.00
	AAANN	0.93	0.25	0.00	0.76	1.00	1.00	1.00	0.09	0.00	0.54	1.00	1.00	0.67	0.16	0.00	0.62	1.00	1.00
	AAAND	0.84	0.21	0.00	0.79	1.00	1.00	0.93	0.19	0.00	0.61	1.00	1.00	0.98	0.25	0.03	0.68	1.00	1.00
	AANND	0.91	0.14	0.00	0.84	1.00	1.00	1.00	0.28	0.00	0.68	1.00	1.00	1.00	0.26	0.01	0.74	1.00	1.00

Table 6-15 The average sensitivity and specificity for (1oo7,4oo7 and 7oo7) per web application

		MyReferences						PhpBB						TikiWiki					
		Sensitivity			Specificity			Sensitivity			Specificity			Sensitivity			Specificity		
IDS Combination Types		1oo7	4oo7	7oo7	1oo7	4oo7	7oo7	1oo7	4oo7	7oo7	1oo7	4oo7	7oo7	1oo7	4oo7	7oo7	1oo7	4oo7	7oo7
Functionally diverse	*Overall*	0.95	0.25	0.00	0.68	1.00	1.00	0.98	0.06	0.00	0.45	1.00	1.00	0.93	0.21	0.01	0.54	1.00	1.00
	AAAAAND	0.93	0.28	0.00	0.67	1.00	1.00	0.96	0.06	0.00	0.42	1.00	1.00	1.00	0.23	0.02	0.52	1.00	1.00
	AAAAAAN	0.95	0.32	0.00	0.62	1.00	1.00	0.97	0.06	0.00	0.37	1.00	1.00	0.64	0.21	0.01	0.48	1.00	1.00
	AAAANND	0.96	0.20	0.00	0.72	1.00	1.00	1.00	0.06	0.00	0.49	1.00	1.00	1.00	0.21	0.00	0.58	1.00	1.00
	AAAAANN	0.96	0.28	0.00	0.67	1.00	1.00	1.00	0.06	0.00	0.42	1.00	1.00	0.71	0.18	0.00	0.52	1.00	1.00
	AAAAAAD	0.95	0.32	0.00	0.62	1.00	1.00	0.93	0.07	0.00	0.37	1.00	1.00	1.00	0.22	0.03	0.48	1.00	1.00

Table 6-16 the average sensitivity and specificity for (1oo9,5oo9 and 9oo9) per web application

	MyReferences						PhpBB						TikiWiki					
	Sensitivity			Specificity			Sensitivity			Specificity			Sensitivity			Specificity		
IDS Combination Types	1oo9	5oo9	9oo9	1oo9	5oo9	9oo9	1oo9	5oo9	9oo9	1oo9	5oo9	9oo9	1oo9	5oo9	9oo9	1oo9	5oo9	9oo9
AAAAAAANND	0.97	0.26	0.00	0.62	1.00	1.00	1.00	0.07	0.00	0.37	1.00	1.00	1.00	0.22	0.00	0.48	1.00	1.00

6.7 Discussion, Conclusions and Limitations

In this chapter we presented results of analysing the performance of diverse IDS configurations. The analysis is performed using a previously published dataset by the authors of (Elia et al., 2010), which used nine individual IDS configurations to monitor three web applications, that were subjected to SQL injection attacks and benign crawling actions. From the nine individual IDSs we built all the possible diverse pairs, triplets, quadruples etc., of IDSs. While analysing the results we considered three possible configurations of the adjudicator: 1-out-of-N (raise an alarm when any one of the IDS in the N configuration raise it); majority vote (raise an alarm only when a majority of the systems in an N configuration raise it; when N is even, we use the minimum number needed to reach a majority, e.g. 3oo4, 4oo6 etc) and N-out-of-N (raise an alarm only when all IDSs in the N configuration raise it). We presented the results using the well-established measures for binary classifiers: sensitivity, specificity and accuracy.

The main conclusions from our analysis are:

- **For 1oo2 systems**: improvements in sensitivity compared with individual systems are around *40% on average* for the three applications, but come with around *10% specificity deterioration* on average. The largest improvements in sensitivity, with the least deterioration in specificity are from *functionally-diverse* pairs of IDSs;
- **For 2oo2 systems**: improvements in specificity compared with individual systems are around *10% on average* for the three applications, but come with around *60% sensitivity deterioration* on average;
- **For 1ooN systems:** on average there is an improvement in sensitivity compared with individual IDS, and a deterioration in specificity ;
- **For NooN systems:** specificity can be perfect in most setups, but with severe deterioration in sensitivity on average;

- **Majority voting systems** usually offer a compromise between the extremes of 1ooN and NooN setups, but for these setups they tended to negatively impact the sensitivity measures, with marginal gains in specificity.

We also explored the level of "functional diversity" that exists in the IDSs and how that impacts the sensitivity and specificity measures. On average we found that the more functionally diverse the system is the better the sensitivity and specificity on average, though not in all cases.

Apart from the results presented, we also provide a well-documented, step-by-step analysis methodology for assessing the performance of N-version diverse security decision support systems. This should prove useful to other researchers and organisations to assess diversity in their setups.

There are a few limitations of the dataset we have used which prevent us from making more generalised conclusions on the possible benefits of diversity with IDSs:

- The dataset is from 2009, hence it may not accurately reflect the prevalence of current attacks. However, we should stress that we are matching *like* with *like*: i.e. attacks and IDSs at some snapshot in time, in this case 2009. And SQL Injection attacks still remain largely prevalent and topical for web applications today. Good datasets are difficult to find, but future work could involve analysis with more current attack mechanisms and IDS versions. This will allow comparison with our results and hence analyse diverse IDS performance over time;
- The dataset is for web applications only. Ideally we would want datasets for a wider array of applications. However, web applications dominate the market, and because they are directly exposed to attackers, are the ones security engineers spend the most effort trying to protect;
- The attacks are limited to SQL Injection. A wider array of attacks would be preferable. Though we should stress that SQL Injection attacks are some of the most dangerous and widely used attacks for database driven web applications, hence the focus on them is valid;
- The data is generated by a vulnerability and attack injection tool and may not be representative of operational scenarios used by different organisations. Though we should stress that it is difficult to get operational datasets, as organisations rarely share them, so authors in (Elia et al.,

2010) state that they took extra care to select attacks that were representative of those seen in the field;

- Dataset for MyReferences and phpBB is dominated by attacks. As mentioned before, the accuracy measure is likely to be dominated by TP and FN in these circumstances. So we recommend using the sensitivity measures for these applications.
- As we also mentioned for the AV study, we should emphasise that despite the limitations stated above our main aim is to assess diversity between tools in a particular snapshot in time. Our results show that diversity can be effective and we have quantified this effectiveness.

(7) DIVERSITY WITH STATIC ANALYSIS TOOLS (SATs)

This chapter presents results of analysing the performance of diverse Static Analysis Tools (SATs) configurations. Most sections in this chapter have been published in:

Algaith, A., Nunes, P., Fonseca, J., Gashi, I. and Viera, M. (2018). "***Finding SQL Injection and Cross Site Scripting Vulnerabilities with Diverse Static Analysis Tools***". 14th European Dependable Computing Conference (EDCC'18), pp.57-64.

7.1 Introduction

Static analysis tools are used to inspect software looking for vulnerabilities, without executing the code. Since they can cover all the source code effectively, they are a valuable tool to help security researchers to automate the task of discovering some type of vulnerabilities. However, as with any other binary decision system, SATs also suffer from false negative (FN) errors (missing vulnerabilities in the code they inspect) and false positive (FP) errors (incorrectly labelling code as containing a vulnerability when in fact it does not).

There are many SATs available, and each one has its own strengths and weaknesses. Rather than using just one tool, several diverse SATs can be used for finding vulnerabilities to reduce the probability of vulnerabilities remaining undetected. However, for diversity to be effective, the SATs should be diverse in their design. This way, a vulnerability undetected by one SAT should, with high probability, be detected by another one, while at the same time not increasing prohibitively the number of false positives. The important questions are whether a specific set of SATs would improve vulnerability detection more than another set; quantifying these gains; and quantifying the false positives.

We provide empirical results to help with the problem of deciding which combination of SATs to use. We present results of analysing the performance of diverse SAT configurations based on a previously published dataset (Nunes et al., 2017). The dataset consists of five SATs that were individually used to find two types of vulnerabilities, namely SQL Injection (SQLi) and Cross-Site Scripting (XSS), in 134 plugins of the WordPress Content Management System (CMS). WordPress powers 30% of the web and represents 60% of all CMSs. According to the Hacked Website Report, WordPress is the most infected CMS (Sucuri Remediation Group, 2017) it accounted for 74% of all CMS infections in Q3 of 2016, and 83% of all CMS infections in 2017.

We apply in full the methodology we described in Chapter 3. Namely, we calculate the FP, FN, TP, TN counts for each diverse configuration with SAT tools; we calculate the measures of interest (specificity, sensitivity and accuracy) or each diverse configuration, overall and by type of vulnerability; we generate the ROC plots showing all the diverse configurations and the individual defence systems, overall, by type of configuration and by type of malicious input; we calculate the differences in the measures of interest between diverse configurations and individual systems to measure the possible improvements or deteriorations from switching to a diverse system.

The rest of the chapter is organised as follows: section 7.2 outlines the objectives of the study; section 7.3 describes the dataset; section 7.4 describes the analysis methodology; section 7.5 presents the results from analysing diverse systems with the different configurations; section 7.6 presents an analysis of the plug-ins in which the vulnerabilities were found; and finally section 7.7 presents a discussion, conclusions and provisions for further work.

7.2 Study objectives

In this study, we investigate all the possible diverse configurations that we can build with the five individual SATs: 10 diverse pairs, 10 diverse triplets, five diverse quadruples and one diverse quintet SAT system. We considered various configurations for the adjudicator: 1-out-of-N (raise an alarm for a vulnerability when any of N SATs in the diverse configuration does so); N-out-of-N (raise an alarm for a vulnerability only when all N SATs in the diverse configuration do so); and simple majority (raise an alarm for a vulnerability when the majority of the N SATs in a diverse configuration do so).

Results are presented using the well-established measures for binary classifiers: sensitivity (measures the performance of the SAT to find vulnerabilities) and specificity (measures the performance of the SAT to not raise false alarms). These measures capture the main requirements of practitioners when selecting SATs: a tool that finds most vulnerabilities without raising too many false alarms.

We analysed the measures for all possible two-SAT, three-SAT, four-SAT and five-SAT diverse configurations. We found that none of the SATs, or combinations of SATs, was able to find all the vulnerabilities in the target plugins. But, we found that some of the SATs exhibit considerable diversity in their ability to detect the types of vulnerabilities analysed. We then provide empirically supported guidance on which combination of SATs provide the most benefits in the ability to detect vulnerabilities, with a reduced false positive rate. Hence, this study provides a significant new contribution compared with the previous work in (Nunes et al., 2017) on which only 1-out-of-N configurations were analysed. 1-out-of-N systems raise an alarm as long as any one of the SATs in the system raise an alarm. One limitation of these configurations is the potential increase of FPs, which may be unacceptable in many situations. In the present work, we look at all the possible N-out-of-N and majority voting configurations. This way a security researcher has more evidence on the interplay between FPs and FNs in diverse SAT configurations.

7.3 Dataset

A standard way to evaluate and compare the effectiveness of SATs is to make them search for vulnerabilities in a set of applications (i.e. the workload), followed by the computation of the evaluation metrics. The workload strongly determines the results, so it should be representative of all applications. Unfortunately, this is very hard to attain. To make the problem treatable, the workload can be built for a particular domain. However, the selection of a set of representative applications in a given domain is still a difficult task. Another difficulty is the characterization of the applications in the workload (especially if they are real applications) concerning the vulnerable (VLOC) and non-vulnerable (NVLOC) lines of code (LOC). Moreover, the computation of several evaluation metrics requires the outputs of all the tools to be classified into FP, FN (False Negatives), TP and TN (True Negatives). To compare the results of two or more SATs we need their outputs to be in a common format with detailed data about the vulnerabilities such as the LOC, the SS, the vulnerable variables, the chains of data/control dependencies of the vulnerable variables from the Entry points (EPs) to the Sensitive Sink (SS) to prove that the user input reaches the SS. Unfortunately, SATs report the vulnerabilities they find in different formats with varying degrees of detail. For example, some SATs report data in HTML pages and others in a GUI. Although these data are human readable, they need to be converted to a common format. To accomplish this in a seamless way (Nunes et al., 2017) developed a tool able to automate the process.

The workload in (Nunes et al., 2017) builds on previous study (Na et al., 2008). In that work (Na et al., 2008), the study proposed an approach to select applications based on public repositories of vulnerabilities that include confirmed vulnerabilities in real software. It applied this methodology to the domain of WordPress plugins and for SQLi and XSS vulnerabilities based on the online WPScan Vulnerability Database (WPVD) (wpvulndb, 2018). The workload is a set of 134 plugins composed of 4,975 PHP files, 1,339,427 LOC and where each plugin has at least one SQLi and/or one XSS VLOC (i.e. a

LOC with at least one vulnerable SS). As identifying all VLOCs and NVLOCs (i.e. a LOC with all SSs non-vulnerable) in the workload is a hard task that requires a thorough review by security experts, our approach to find more VLOCs than those present in the WPVD was based on searching for further vulnerabilities in the workload with one or more SATs, followed by a manual review to confirm if they are TPs or FPs. The merge of all TPs with the vulnerabilities of the WPVD becomes the list of VLOCs in the workload (which is, nevertheless, a best-effort subset of all of them). Therefore, the list of NVLOCs is obtained from all LOCs with a SS with at least one variable, excluding those that were reported by the tools and confirmed manually as TP.

To detect the SQLi and XSS vulnerabilities in the plugins, the following five SATs were used: RIPS v0.55 (Dahse et al., 2014), Pixy v3.03(2007) (Jovanovic et al., 2006), phpSAFE (Na et al., 2008), WAP v2.0.1(Medeiros et al., 2014) and WeVerca v20150804 (Hauzar & Kofroň, 2015)

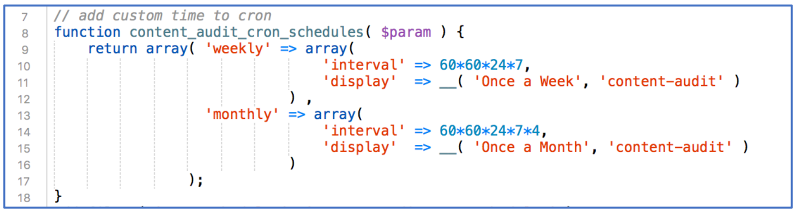
RIPS performs static taint analysis and string analysis. RIPS and Pixy are two of the most referenced PHP SATs in the literature, but they are not ready for Object Oriented Programming (OOP) analysis. RIPS has been developed as open source until 2014, and only its recently released commercial version is able to fully analyse OOP code. WAP, phpSAFE, and WeVerca are recent tools under active development and they are prepared for OOP code.

Static analysis is a complex task, and the tools may be unable to fully process some files of the workload. Overall, phpSAFE was unable to analyse 130 files, RIPS could not analyse 2179 files, Pixy did not process 1473 files, and WeVerca was not able to analyse a total of 20 files. To make the analysis comparable we consider only the results obtained from files that could be successfully analysed by all five tools.

Overall, the plugins contain 713,456 LOC and 402,218 logical LOC (LLOC, i.e. commented and whitespace lines are excluded), as can be seen in Table 6-1. The counting of the LOC and LLOC was performed using the *phploc*[39]. WordPress plugin are constructed with the PHP language which allows a mixture of OOP (Object Oriented Programming) and POP (Procedure

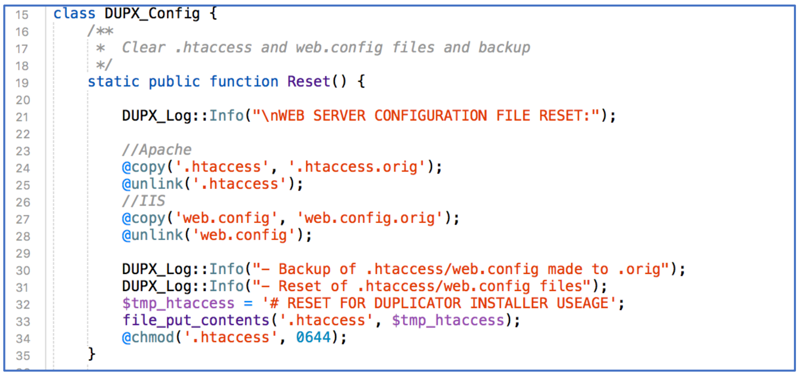
[39] https://phar.phpunit.de/phploc.phar

Oriented Programming) code. The authors in (Nunes et al., 2017) provided a classification of vulnerable and non-vulnerable POP and OOP code in the plug-ins that they analysed. Examples are provided in the code extracts in the two figures below Figure (7-1) and (7-2)



```
// add custom time to cron
function content_audit_cron_schedules( $param ) {
    return array( 'weekly' => array(
                            'interval' => 60*60*24*7,
                            'display'  => __( 'Once a Week', 'content-audit' )
                        ) ,
                'monthly' => array(
                            'interval' => 60*60*24*7*4,
                            'display'  => __( 'Once a Month', 'content-audit' )
                        )
                );
}
```

Figure 7-1 Example POP code in the analysed plugins.



```
class DUPX_Config {
    /**
     *  Clear .htaccess and web.config files and backup
     */
    static public function Reset() {

        DUPX_Log::Info("\nWEB SERVER CONFIGURATION FILE RESET:");

        //Apache
        @copy('.htaccess', '.htaccess.orig');
        @unlink('.htaccess');
        //IIS
        @copy('web.config', 'web.config.orig');
        @unlink('web.config');

        DUPX_Log::Info("- Backup of .htaccess/web.config made to .orig");
        DUPX_Log::Info("- Reset of .htaccess/web.config files");
        $tmp_htaccess = '# RESET FOR DUPLICATOR INSTALLER USEAGE';
        file_put_contents('.htaccess', $tmp_htaccess);
        @chmod('.htaccess', 0644);
    }
```

Figure 7-2 Example OOP code in the analysed plugins.

In summary, (Nunes et al., 2017) considered a module to be OOP if that module had a definition of a class. However within the same module, there can be code, outside functions and methods of a class which would be POP. Hence a LoC in their classification is always either POP or OOP, but not both. But within a module there can be a mixture of OOP and POP LoCs.

Since programming orientation may be relevant for the performance of the SATs, Table 7-1 also shows the LLOCs that are classified as POP code and those that are classified as OOP code[40].

[40] A LOC cannot be categorised as both OOP or POP.

Table 7-1 Plugin Information

SQLi				XSS			
		LLOC				LLOC	
Plug.	Files	POP	OOP	Plug.	Files	POP	OOP
117	2168	120917	46617	130	3401	175747	58937

Table 7-2 shows the VLOCs and NVLOCs for SQLi and XSS for the workload in Table 7-1. In this, we can see more XSS than SQLi data, but that is also usual in real life applications.

Table 7-2 Dataset

Vulnerability	Code Type	VLOC	NVLOC	Total
SQLi	POP	138	605	743
	OOP	509	5574	6083
	Total	**647**	**6179**	**6826**
XSS	POP	965	1370	2335
	OOP	3384	18525	21909
	Total	**4349**	**19895**	**24244**

7.4 Analysis of single version systems

Table 7-3 presents the five SATs, the labels (in brackets) we use to refer to them in the rest of the study, and the FP, TN, FN and TP counts respectively, for each of the two classes of vulnerabilities. Table 7-4 presents the sensitivity and specificity measures for each SAT.

Table 7-3 The five SATs and the FP, TN, FN and TP counts

SAT	SQLi				XSS			
	FP	TN	FN	TP	FP	TN	FN	TP
phpSAFE (A)	53	6126	268	379	213	19682	2293	2056
RIPS (B)	116	6063	465	182	454	19441	1469	2880
WAP (C)	0	6179	492	155	25	19870	3964	385
Pixy (D)	31	6148	583	64	172	19723	3313	1036
WeVerca (E)	4	6175	608	39	24	19871	3488	861

Table 7-4 The five SATs and the sensitivity (sens.) and specificity (spec.) measures for each SAT

SAT	SQLi		XSS	
	Sens.	Spec.	Sens.	Spec.
phpSAFE (A)	0.586	0.991	0.473	0.989
RIPS (B)	0.281	0.981	0.662	0.977
WAP (C)	0.240	1.000	0.089	0.999
Pixy (D)	0.099	0.995	0.238	0.991
WeVerca (E)	0.060	0.999	0.198	0.999

In the present study, we extend the analysis of (Nunes *et al.*, 2017) from the viewpoint of diversity. From the five SAT configurations, we can build a total of:

- 10 two-version combinations ($^5C_2$),
- 10 three-version combinations ($^5C_3$),
- 5 four-version combinations ($^5C_4$), and
- 1 five-version combination ($^5C_5$).

7.5 Results

In this section, we present the results of our analysis of the diversity in the SAT tools.

7.5.1 Visualising diversity

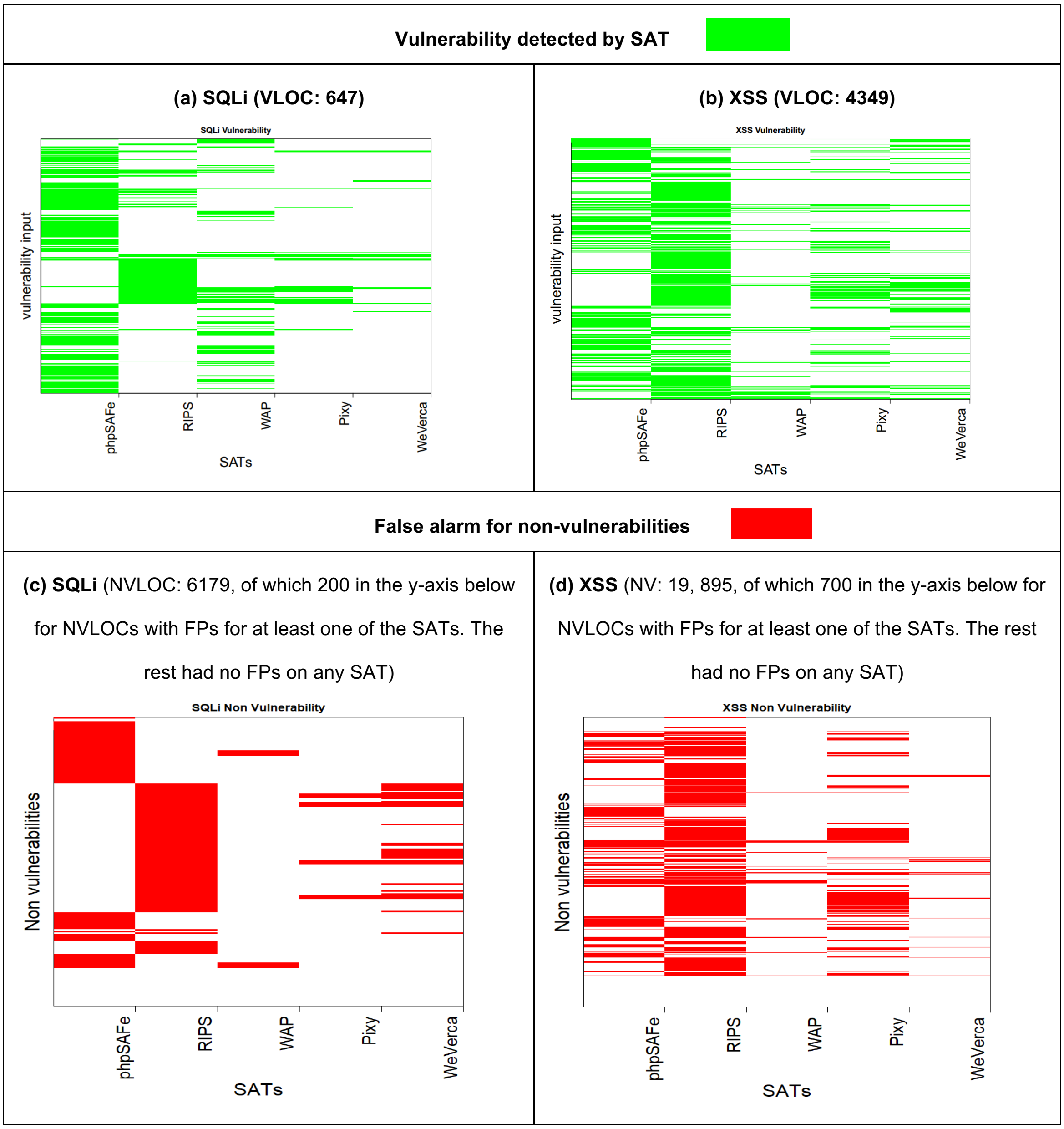
We begin our analysis with a simple visualisation that shows the commonality and diversity of the tools on the vulnerable and non-vulnerable code. Figure 7-3 contains these plots for the two classes of vulnerabilities. We will use Figure 7-3(a) to illustrate what each plot shows:

- The x-axis lists the five SAT tools;
- The y-axis lists the VLOCs (647 in total for SQLi).

A green cell in the plot shows for each SAT whether they detected the vulnerable code as such (i.e. the green cells represent true alarms (TPs); the white cells represent no alarms (in this case, FNs)).

Figure 7-3(b) is the same but for the XSS vulnerabilities. Figures 7-3(c) and 7-3(d) are similar but in these plots we visualise the responses from SATs on code that was not vulnerable – NVLOC (hence an alarm is a false positive (FP) represented by a red coloured cell; no alarms are again represented as white cells (in this case they are TNs)) for the SQLi and XSS vulnerabilities, respectively. In 7.3(c) and 7.3(d) we show only the NVLOCs on which at least one of the SATs reports an FP.

From these plots, we can observe that there is noticeable diversity between some of the SATs (e.g. considerable diversity for both SQLi and XSS between phpSAFE and RIPS, as is evident by the limited overlap in their alarms in the graphs).



Vulnerability detected by SAT 🟩

(a) SQLi (VLOC: 647)

(b) XSS (VLOC: 4349)

False alarm for non-vulnerabilities 🟥

(c) SQLi (NVLOC: 6179, of which 200 in the y-axis below for NVLOCs with FPs for at least one of the SATs. The rest had no FPs on any SAT)

(d) XSS (NV: 19, 895, of which 700 in the y-axis below for NVLOCs with FPs for at least one of the SATs. The rest had no FPs on any SAT)

Figure 7-3 Diversity between SATs for SQL Injection and Cross Site Scripting (XSS)

7.5.2 Sensitivity, specificity and ROCs for diverse SATs

We then proceeded to calculate the sensitivity and specificity for each of the diverse combinations with the five SATs, for the three types of adjudication setups considered (namely 1ooN, simple Majority vote (2oo3, 3oo4 and 3oo5) and NooN). Table 7-5 presents the results of this analysis for all the possible two-version, three-version, four-version and five version combinations for SQLi. Table 7-6 shows the results for XSS.

From Tables 7-5 and 7-6 we can see some patterns emerging: the 1ooN systems are better at finding vulnerabilities (better sensitivity), compared with the best individual SATs; on the other hand, NooN systems are better at correctly labelling non-vulnerable code (higher specificity). This is to be expected since (similarly to what we also stated for the IDS systems in the preceding chapter):

- 1ooN systems will in *all* cases perform:
    - *better or equal* to the best single SAT in the diverse combination N for *vulnerable code*, as any “alarm” from any of the N SATs systems will lead to an alarm in a 1ooN system;
    - *equal or worse* than the worst single SAT in the diverse combination N for *non-vulnerable code*, as any “alarm” from any single SAT will lead to this code being incorrectly labelled as vulnerable.
- NooN systems will in *all* cases perform:
    - *better or equal* to the best single SAT for *non-vulnerable code* as the NooN system only raises an “alarm” for non-vulnerable code if ALL the SATs in the diverse configuration do so;
    - *equal or worse* than the worst single SAT system in the diverse configuration N for *vulnerable code*, as the NooN system will only label code as vulnerable if ALL the single SATs in the diverse configuration do so.
- Majority voting setups usually balance out these extremes, as they are not as “trigger happy” as 1ooN setups in raising alarms, but also not as conservative as NooN setups in remaining silent.

What is important to understand is *how much* better, or *how much worse*, would a diverse configuration perform in these setups, and the results in Tables 7-5 and 7-6 provide us with some interesting observations.

Table 7-5 Sensitivity (sens.) and specificity (Spec) for the 1ooN, Majority vote and NooN configurations for N between 2 and 5 for SQLi

SQLi	1ooN		Majority		NooN	
	Sens.	Spec.	Sens.	Spec.	Sens.	Spec.
(a, b)	0.782	0.976	-	-	0.085	0.996
(a, c)	0.770	0.991	-	-	0.056	1.000
(a, d)	0.655	0.987	-	-	0.029	0.999
(a, e)	0.624	0.991	-	-	0.022	1.000
(b, c)	0.444	0.981	-	-	0.077	1.000
(b, d)	0.289	0.981	-	-	0.091	0.995
(b, e)	0.297	0.981	-	-	0.045	0.999
(c, d)	0.280	0.995	-	-	0.059	1.000
(c, e)	0.277	0.999	-	-	0.023	1.000
(d, e)	0.111	0.995	-	-	0.048	0.999
(a, b, c)	0.901	0.976	0.193	0.999	0.012	1.000
(a, b, d)	0.787	0.976	0.153	0.998	0.026	0.999
(a, b, e)	0.796	0.976	0.111	0.994	0.020	1.000
(a, c, d)	0.784	0.987	0.138	0.999	0.003	1.000
(a, c, e)	0.787	0.991	0.097	0.999	0.002	1.000
(a, d, e)	0.668	0.987	0.056	0.998	0.022	1.000
(b, c, d)	0.447	0.981	0.119	0.998	0.045	1.000
(b, c, e)	0.457	0.981	0.102	0.994	0.022	1.000
(b, d, e)	0.301	0.981	0.094	0.994	0.045	0.999
(c, d, e)	0.292	0.995	0.083	0.998	0.023	1.000
(a, b, c, d)	0.901	0.976	0.087	1.000	0.003	1.000
(a, b, c, e)	0.913	0.976	0.051	1.000	0.002	1.000
(a, b, d, e)	0.799	0.976	0.053	0.998	0.020	1.000
(a, c, d, e)	0.796	0.987	0.045	1.000	0.002	1.000
(b, c, d, e)	0.459	0.981	0.079	0.998	0.020	1.000
(a, b, c, d, e)	0.913	0.976	0.094	0.998	0.000	1.000

Table 7-6 Sensitivity (sens.) and specificity (spec) for the 1ooN, Majority vote and NooN configurations for N between 2 and 5 for XSS

XSS	1ooN		Majority		NooN	
	Sens.	Spec.	Sens.	Spec.	Sens.	Spec.
(a, b)	0.963	0.970	-	-	0.172	0.9963
(a, c)	0.522	0.989	-	-	0.040	0.9991
(a, d)	0.631	0.982	-	-	0.080	0.9985
(a, e)	0.586	0.988	-	-	0.084	0.9997
(b, c)	0.687	0.977	-	-	0.064	0.9990
(b, d)	0.683	0.976	-	-	0.218	0.9921
(b, e)	0.733	0.977	-	-	0.127	0.9989
(c, d)	0.271	0.991	-	-	0.056	0.9995
(c, e)	0.251	0.998	-	-	0.035	0.9998
(d, e)	0.334	0.991	-	-	0.102	0.9994
(a, b, c)	0.981	0.970	0.208	0.996	0.034	0.9992
(a, b, d)	0.967	0.970	0.342	0.989	0.064	0.9991
(a, b, e)	0.986	0.970	0.309	0.995	0.037	0.9998
(a, c, d)	0.655	0.982	0.113	0.998	0.032	0.9997
(a, c, e)	0.613	0.988	0.133	0.999	0.013	0.9999
(a, d, e)	0.681	0.982	0.189	0.998	0.039	0.9998
(b, c, d)	0.702	0.976	0.236	0.991	0.051	0.9996
(b, c, e)	0.751	0.977	0.169	0.998	0.029	0.9998
(b, d, e)	0.747	0.976	0.257	0.991	0.095	0.9995
(c, d, e)	0.357	0.990	0.143	0.999	0.025	0.9999
(a, b, c ,d)	0.981	0.970	0.088	0.998	0.031	0.9997
(a, b, c, e)	0.998	0.970	0.073	0.999	0.013	0.9999
(a, b, d, e)	0.990	0.970	0.139	0.998	0.032	0.9999
(a, c, d, e)	0.696	0.982	0.071	0.999	0.012	1.0000
(b, c, d, e)	0.759	0.976	0.125	0.999	0.025	0.9999
(a, b, c, d, e)	0.998	0.970	0.154	0.998	0.003	1.0000

Sensitivity: Combining SATs phpSAFE (A), RIPS (B) and WAP (C) in a 1ooN setup (meaning we identify code as vulnerable as soon as any one of these tools identifies it as such) gives very large gains in sensitivity for both SQL Injection and XSS. Sensitivity for the best of these tools for SQL injection is 0.56. 1oo3 configuration of these three tools (as listed in the row (a,b,c)) is 0.9. Adding the remaining two SATs (Pixy and WeVerca) improves sensitivity a little bit more (to 0.91) in a 1oo5 setup (row (a,b,c,d,e)). For XSS, phpSAFE

(A) and RIPS (B) in a 1oo2 setup have a sensitivity score of 0.96 (individually RIPS (B) had the best sensitivity at 0.66). Combining all five tools in a 1oo5 setup meant all the XSS vulnerabilities in the plugins we considered were detected. As we would expect, we see large deteriorations in sensitivity for NooN setups. We also observe poor sensitivity results for majority voting setups.

Specificity: We see gains in specificity in NooN setups (meaning we only label code as vulnerable if all N tools in the setup agree that the code is vulnerable). Many configurations never raise false alarms in these configurations. However, they also have very poor sensitivity values. As expected, majority voting setups do better for sensitivity compared with NooN, but worse for specificity.

ROC plots also help a decision maker to visualise these results and compare the performance of the different systems. Figure 7-4 shows the eight ROC plots, one for each vulnerability (SQLi and XSS), and for each configuration of N, 2≤N≤5. In addition to the 1ooN, simple majority (1oo3, 3oo4 and 3oo5), and NooN setups that we showed in Tables 7-5. and 7-6, we also calculated the remaining voting setups (2oo4, 2oo5, 4oo5) not shown in those tables.

The optimal system in an ROC plot is one that appears on the top right-hand corner (i.e. one that has both sensitivity and specificity of 1, since it detects all vulnerabilities and never raises an alarm for code that does not contain vulnerabilities). We have no such system in the configurations in our examples. As we have seen from the results in Tables 7-5 and 7-6, most of the results in our configurations have extremes of high sensitivity (1ooN) or high specificity (NooN). The ROC plots make it easier to identify configurations that lie between these extremes.

7.5.3 Averages for different diverse setups

We conclude our analysis with a summary table (Table 7-7) showing the average Sensitivity and Specificity for non-diverse setups (abbreviated “1v” in the first row of the table) compared with the averages for the different diverse configurations. These results confirm the observations we have shown so far:

- For 1ooN systems: more than 70% improvements in sensitivity on average in a 1oo2 setup compared with average individual SATs. More than three times the improvements in sensitivity on average on a 1oo5 setup compared with individual SATs. However, this comes at a correspondingly high deterioration in specificity;

- For NooN systems: almost perfect specificity can be achieved when using NooN setups (especially for configurations of N > 2). But this comes at a large deterioration in sensitivity;
- Simple majority voting setups on average lead to a deterioration in sensitivity (of between 30-65%) but with some improvements in specificity.

	SAT	phpSAFE	RIPS	WAP	Pixy	WeVerca
	SAT Label	A	B	C	D	E

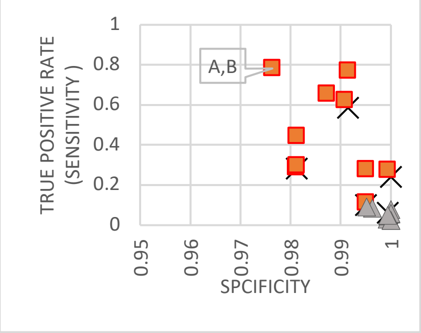
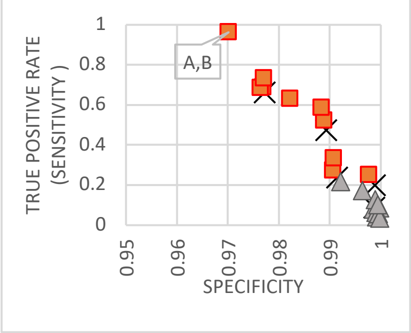
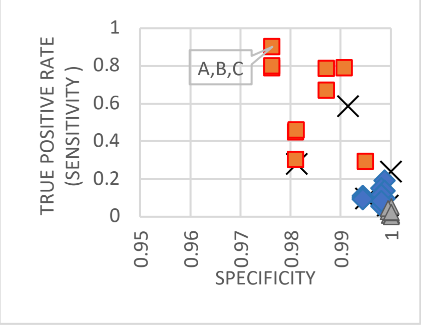
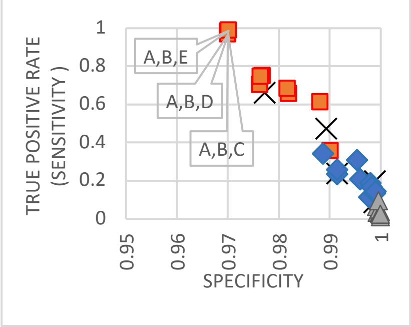
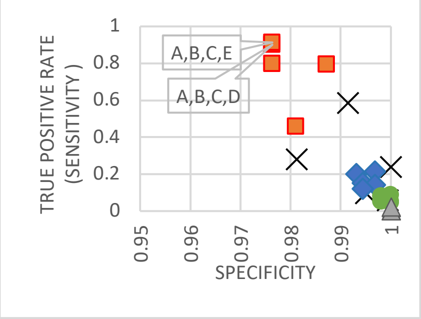
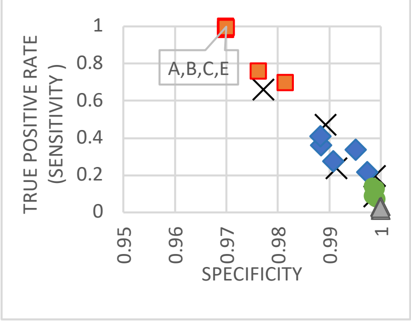
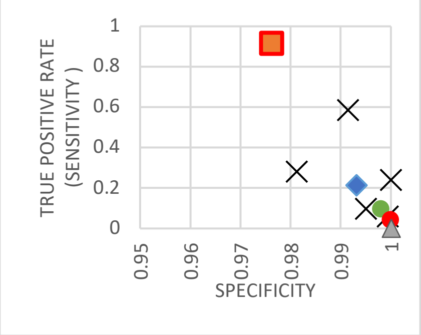
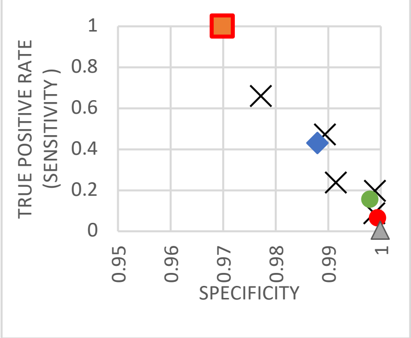
N	SQLi	XSS
N=2, 5C2 = 10 × SAT □ 1oo2 ▲ 2oo2		
N=3, 5C3= 10 × SAT □ 1oo3 ◆ 2oo3 ▲ 3oo3		
N=4, 5C4 = 5 × SAT □ 1oo4 ◆ 2oo4 ● 3oo4 ▲ 4oo4		
N=5, 5C5 = 1 × SAT □ 1oo5 ◆ 2oo5 ● 3oo5 ● 4oo5 ▲ 5oo5		

Figure 7-4 - ROC plots for the different diverse combinations and the two classes of vulnerabilities.

Table 7-7 Average Sensitivity and Specificity for each diverse version and each class of vulnerabilities

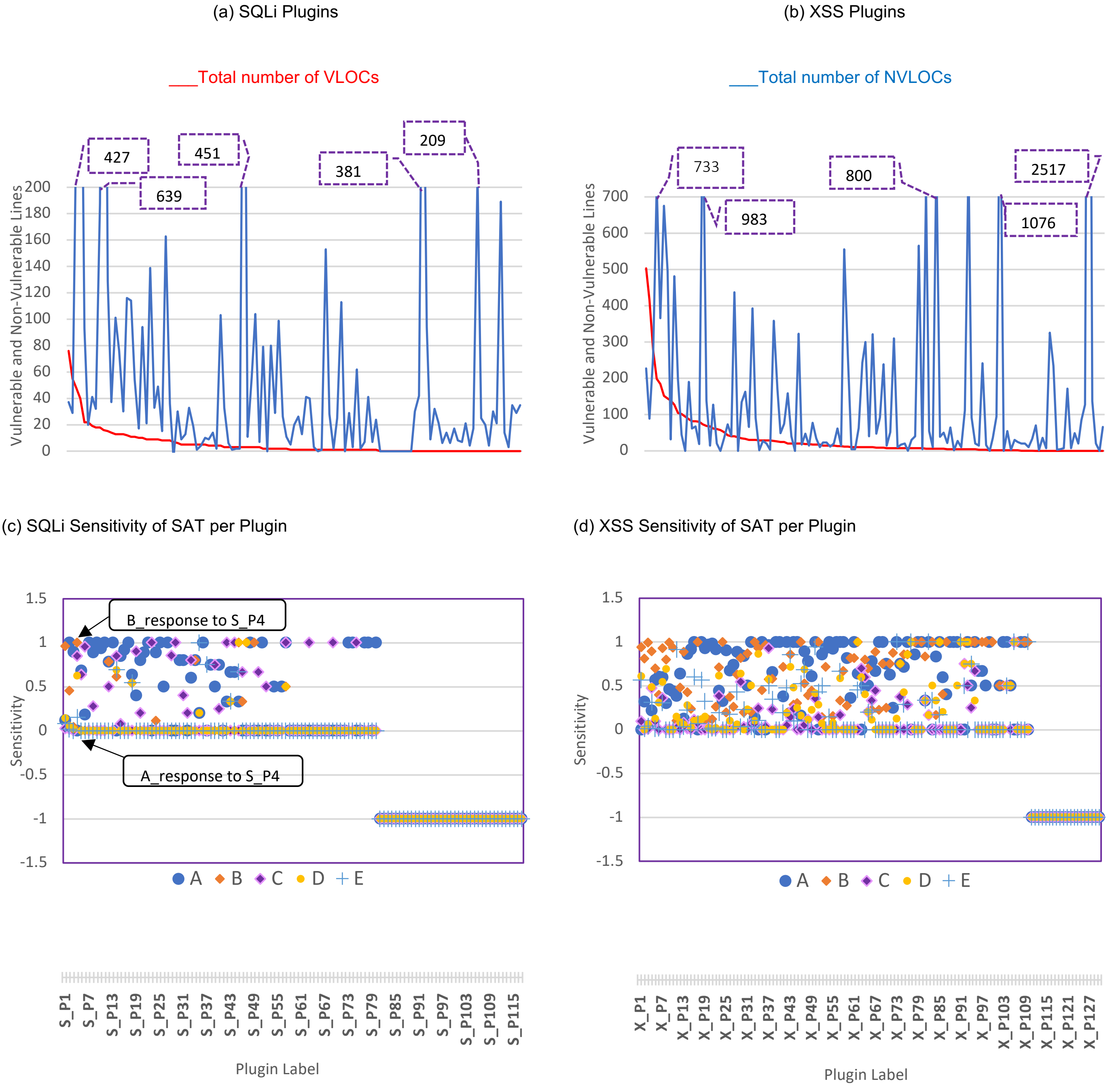
	SQLi		XSS	
SAT Configurations	**Sens.**	**Spec.**	**Sens.**	**Spec.**
1v	0.25	0.99	0.33	0.99
1oo2	0.45	0.99	0.57	0.98
1oo3	0.62	0.98	0.74	0.98
1oo4	0.77	0.98	0.89	0.97
1oo5	0.91	0.98	0.99	0.97
2oo2	0.05	0.99	0.10	0.99
3oo3	0.02	0.99	0.04	0.99
4oo4	0.01	1.00	0.02	0.99
5oo5	0.00	1.00	0.003	0.99
2oo3	0.11	0.99	0.21	0.99
2oo4	0.17	0.99	0.32	0.99
2oo5	0.21	0.99	0.43	0.99
3oo5	0.09	0.99	0.15	0.99
4oo5	0.04	1.00	0.06	0.99

7.6 Analysis of the plugin

In this section we present the results of our analysis by the plug-ins of WordPress that the SAT tools analysed. We did this to understand better the observed diversity in detection capabilities of the SAT tools.

Figures 7-5(a) and (b) show the ordering of the plugins by total number of VLOCs (left to right, those with most VLOCs are on the left of the graph) for SQLi and XSS respectively. Figure 7-5(c) and 7-5(d) shows the sensitivity of each SAT for each of these plugins (the order in the x-axis of figures 7-5(c) and 7-5 (d) corresponds to the order in Figures 7-5(a) and 7-5(b) respectively). We see that there is considerable diversity in the sensitivity of the tools for the different plugins. For example, Figure 7-5 (d) shows a large cluster of orange diamonds (RIPS (B)) in the top left, which indicates that this tool was outperforming phpSAFE (A) on sensitivity for these plug-ins, even though phpSAFE was better on average overall. RIPS (B) for SQLi reports many vulnerabilities in the levelfourstorefront.8.1.14 plugin (S_P4) and the phpSAFE (A) reports none (we highlighted this plugin in Figure 7-5(c)). However, for the sendit.2.1.0 plugin (S_P7) the SAT phpSAFE reports many

VLOCs and the other SATs none.



(a) SQLi Plugins

(b) XSS Plugins

(c) SQLi Sensitivity of SAT per Plugin

(d) XSS Sensitivity of SAT per Plugin

Figure 7-5 Vulnerable and non-vulnerable lines count per plugin and sensitivity measures, for SQLi and XSS.

7.7 Discussion, Conclusions and Limitations

In this study, we presented results of analysing the performance of diverse Static Analysis Tools (SATs) configurations. The analysis is performed using a previously published dataset, where five SATs were used for finding two types of vulnerabilities, namely SQL Injections (SQLi) and Cross-Site Scripting (XSS), in 134 WordPress plugins. From the five individual SATs, we built 10 diverse pairs, 10 diverse triplets, 5 diverse quadruples and one diverse quintet SAT system. When analysing the results, we considered various configurations of the adjudicator: 1ooN (raise an alarm for a vulnerability when any of N SATs in the diverse configuration do so); NooN (raise an alarm for a vulnerability only when all N SATs in the diverse configuration do so); and simple majority (raise an alarm for a vulnerability when the majority of the N SATs in a diverse configuration do so). We presented the results using the well-established measures for binary classifiers: sensitivity and specificity. The main conclusions from our analysis are:

- **For 1ooN systems:** improvements in sensitivity compared with individual SAT are from 70% on average for 1oo2 systems, to more than 300% for 1oo5 systems, but come with a corresponding specificity deterioration on average. The largest improvements in sensitivity, with the least deterioration in specificity are from combining phpSAFE with WAP SATs in a diverse 1oo2 configuration;
- **For NooN systems:** specificity can be perfect in most setups, but with severe deterioration in sensitivity on average;
- **For simple majority voting setups:** average deterioration in sensitivity (of between 30-65%) but with some improvements in specificity.

For organisations primarily interested in detecting vulnerabilities (improved sensitivity) and that are willing to invest resources in sifting through alarms to separate out the false alarms from true alarms, diverse setups in a 1ooN adjudication setup can be very beneficial. In particular, phpSAFE, RIPS and WAP SATs exhibit considerable diversity in vulnerability detection.

Work conducted by colleagues from Universities of Coimbra and Polytechnic Institute of Guarda provided further insight into the observed diversity between the SAT tools. They found that some of the tools are better at detecting vulnerabilities in certain code constructs than others (for example the way in which they analyse arrays, control flow constructs etc.), which helps explain the observed benefits in vulnerability detection overall that we presented in the previous sections.

There are some limitations to the conclusions we can draw from the research and several provisions for further work:

- Lack of sufficient automation for extracting the slices of code and deriving the test cases. Hence automating this process is further work. We found that using small test cases derived from the plugins is one helpful way to find strengths and weaknesses of the SATs;
- The analysis is based on plug-ins of WordPress which are written in php. Hence the conclusions are limited primarily to web applications written in php. Using SATs to analyse vulnerabilities in code other that PHP, and with a wider range of applications than just WordPress plug-ins, is useful further work;
- The conclusions are limited to SQLi and XSS vulnerabilities. Using SATs that can analyse different types of vulnerabilities other than SQLi and XSS would be more beneficial. However we should state that SQLi and XSS vulnerabilities are some of the most widely spread and most dangerous vulnerabilities, as evidenced from them being consistently ranked in the top 10 most critical web application security risks by The Open Web Application Security Project (OWASP) in the last 5 years (OWASP, 2017).

As we also mentioned for the AV and IDS studies in preceding chapters, we should emphasise that despite the limitations stated above our main aim is to assess diversity between tools in a particular snapshot in time. Our results show that diversity can be effective and we have quantified this effectiveness.

(8) OPTIMAL ADJUDICATION

8.1 Introduction

In this chapter we describe how an algorithm that ensures "optimal adjudication" – an adjudication function which deterministically combines the random outputs of several tools to give outputs that guarantee no other scheme can be shown to be quantifiably better – can be used with SATs and IDSs. The algorithm is the work of (Giandomenico & Strigini, 1990) and is based on the observation that if we have a criterion for choosing between two adjudication functions which one is better to deploy (given what we know about the replicas with which it will work, and our loss functions or the various kinds of erroneous adjudicated output), then we implicitly know how to specify an adjudicator function that is optimal with respect to that criterion. We illustrate that use of the optimal adjudication with the IDS and SAT datasets that we outlined in the preceding two chapters (namely, Chapters 6 and 7).

The rest of the chapter is organised as follows: section 8.2 outlines the objectives of the study; section 8.3 describes the optimal adjudication function; section 8.4 illustrates the use of optimal adjudication with an example; section 8.5 describes the analysis methodology; section 8.6 presents the results of the analysis of optimal adjudication and its comparison with other forms of adjudication (namely 1ooN, majority vote and NooN) for the two datasets; section 8.7 shows the average losses for each type of adjudicator (1ooN, majority vote, NooN and "optimal adjudicator") when we treat each loss as equal, as well as when we assign different weights to the losses; and finally section 8.8 presents a discussion, conclusions and provisions for further work.

8.2 Study objectives

The commonly used solutions for adjudication are some variation of "voting", the simplest being majority voting: if out of – say – three opinions, two are in agreement and disagree with the third one, then the two that agree are more likely to be correct than the third one. This is a reasonable assumption, for instance, in many cases of redundancy used against hardware failures and it yields a very effective design. These kinds of considerations are often behind the choice of **voting as an adjudication function** in diverse-redundant systems. However, for many applications of redundancy and diversity, including most security applications, these obvious considerations do not necessarily hold. Taking the example of attack sensors, we use diversity to give ourselves a chance that if one sensor cannot detect a certain attack, another sensor might. We expect all sensors to have flaws – i.e., that they will not detect some zero-day attacks, or even known attacks for which the designers failed to specify a fully effective detection method, or even that the implementation of a good specification has some bugs. With luck, and as demonstrated in the empirical studies so far in this thesis, these weaknesses will not be common to all the sensors we deploy; likewise, among different sensors only some will raise an alarm on any given situation of innocuous traffic: diversity "works". But some of the systematic errors will be common to more than one sensor. In principle, we could have two sensors that consistently fail to detect exactly the same attacks, while the third sensor correctly flags them; and thus, a 2-out-of-3 vote will produce a FN error. We will be no better off than if we had used that third sensor alone. Of course similar perverse alignments between errors may affect non-attacks as well, so that a 2-out-of-3 voter might also frequently produce FPs. Does this mean that diversity will not work in practice despite its intuitive attractiveness? Can we better realise its potential by more refined adjudication? To this effect we used a technique called optimal adjudication outlined in a paper by (Giandomenico & Strigini, 1990) and applied it to the IDS and SAT datasets. We assessed how much better can optimal adjudication do compared with other conventional forms of adjudication we described so far (1ooN, majority voting or NooN) in reducing the total loss from the two types of failures (FPs and FNs).

8.3 Optimal adjudication

The optimal adjudication described in (Giandomenico & Strigini, 1990) is based on the observation that if we have a criterion for choosing between two adjudication functions which one is better to deploy (given what we know about the replicas with which it will work, and our loss functions or the various kinds of erroneous adjudicated output), then we implicitly know how to specify an adjudicator function that is optimal with respect to that criterion. For instance, a reasonable criterion is minimising expected loss. To assess the expected loss due to a certain adjudication function, we need the probability of each event of interest: e.g., probability of real attack and false negative output of the adjudicator, of no attack and false positive output, etc. If the adjudicator is specified to calculate a function of the outputs of all the replicas, it can be assessed – we can calculate the expected loss from one decision on the set of outputs of a certain set of replicas, and therefore we can compare different adjudication functions and choose the best one – given the probabilities of each replica (each sensor in our case), pair of replicas, etc., giving a correct answer on each kind of input.

The process of specifying an optimal adjudicator, as described in (Giandomenico & Strigini, 1990), is:

1) for each possible combination of sub-component outputs to adjudicate – *syndrome*, assess the two probabilities of it occurring in the presence of an attack and in the presence of a non-attack situation (0,1);
2) the adjudicator function is defined as a simple table, which for each syndrome will contain what output the adjudicator should issue if it receives that syndrome as input;
3) to choose this specified adjudicated output, calculate the values of expected loss that would be associated with that particular syndrome if the adjudicator indicated “attack” and if it were to output “non-attack”; choose the output with lower expected loss.

8.4 Illustration of the use of optimal adjudication

This section illustrates the use of optimal adjudication described in (Giandomenico & Strigini, 1990) using a practical example with Intrusion Detection Systems. We will use an example with three IDSs, using actual numbers which we will present in more detail in next sections.

Table 8-1 shows the eight possible syndromes of a three version system. We label the output from each IDS as either one (i.e. raises an alarm) or zero (i.e. no alarm raised). The syndromes are the input of the adjudication function.

We then check each syndrome against the actual inputs to the IDSs. In our examples we have two types of inputs: Attacks (labelled as A) and non-attacks (labelled as NA). Table 8-2 shows a small excerpt of this input table for some of the inputs. In this example we have a total of 136 Attack inputs, and 45 Non-Attack inputs. In this simple example we have assumed that the losses associated with False Positives (i.e. alarms raised for Non Attacks) and False Negatives (i.e. no alarms for Attacks) are the same. So, we associate a loss unit of 1 for each FP or FN error.

Now that we have a look up table for all inputs to the IDSs, the syndromes (i.e. the outputs of the IDSs), and the loss values for the two types of failures we can proceed with calculating the optimal adjudicated output.

Table 8-1 The Syndromes of three IDSs.

Output from SCALP sqlia	**Output from SNORT 2.8**	**Output from SNORT 2.8 Plus**	**Syndromes**
0	0	0	0
0	0	1	1
0	1	0	10
0	1	1	11
1	0	0	100
1	0	1	101
1	1	0	110
1	1	1	111

Table 8-2 The optimal adjudication lookup table for the three-version system of Table 8-1

Syndromes →	**0**	**1**	**10**	**11**	**100**	**101**	**110**	**111**	Total
Attack	56	46	0	0	0	34	0	0	136
Non-Attack	41	0	0	0	4	0	0	0	45
Total Demands	97	46	0	0	4	34	0	0	**181**
P(Syndrome)	0.54	0.25	0	0	0.02	0.19	0	0	
Adjudged Output	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	
P(attack|Si)	0.58	1	0	0	0	1	0	0	
P(fn | Si)	0	0	0	0	0	0	0	0	
P(fp | Si)	0.42	0	1	1	0	0	1	1	
Total Probability of FP Errors (sum of product of P(fp | Si) * P(Syndrome))									0.227
Total Probability of FN Errors (sum of product of P(fn | Si) * P(Syndrome))									0
Total probability of error: Total Probability of FP Errors + Total Probability of FN Errors									**0.227**

As we stated previously, we do this by first calculating the values of expected loss that would be associated with that particular syndrome if the adjudicator indicated “attack” and if it were to output “non-attack” and then choosing the output with lower expected loss. Table 8-2 shows these results for the example with the three IDSs. Let us explain the table in more detail:

the first column represents syndrome 0: i.e. all three IDSs did not raise an alarm. From the lookup table we found that all three IDSs did not raise an alarm for 56 attacks, and 41 non-attacks. Raising an alarm would lead to 41 FPs, but not raising an alarm would lead to 56 FNs. Since we associated the same loss value to both FNs and FPs, then we chose the output that minimises the overall loss: in this case (counterintuitively), when all three IDSs do not raise an alarm, the optimal choice for the adjudicator is to actually raise an alarm, as indicated by the "1" in the "Adjudged Output" row. We proceed to do the same for each of the other syndromes. When we have no data (as in examples for Syndromes 10, 11, 110 or 111) we can set the optimal adjudicator to either 1 or 0. If we are more concerned about FNs, we would rather set it to 1 in those cases (i.e. in this case we are stating that even if we have not seen any inputs so far with that syndrome, if we do see them in the future, we would rather raise an alarm so that an administrator can investigate them). As we observe more inputs in the future, we would update the lookup table which could then lead the optimal adjudication outputs to also change.

Based on the adjudged output then (Giandomenico & Strigini, 1990) also describe how to calculate the total probability of FP errors, and the total probability of FN errors. These may be smaller or larger than other forms of adjudication for a given system, but the total probability of error for the system (the sum of these two errors) will always be smallest for the optimal adjudicator. In the example above, we correctly labelled all attacks as such (i.e. we have 0 FNs) but have failed to correctly label 41 non-attacks as such (i.e. we have 41 FPs) out of a total of 181 demands. So, 41 / 181 is = 0.227 .

8.5 Analysis methodology

Using the method of calculating the probability of errors illustrated in the previous section we proceeded to calculate these probabilities for the optimal adjudicator for all the possible combinations of diverse systems for both IDSs and SATs. In this chapter we show the total probability of error for all four adjudicators (1ooN, majority vote, NooN and "optimal adjudication"), for the two studies (IDSs and SATs) and all their corresponding applications (3 for IDSs and 2 for SATs) to enable us to compare the different options.

In summary, we did the following:

- We calculated the FP, FN, TP, and TN counts for each diverse configuration (as previously explained in Chapter 3);

- We calculated the total probability of error for each configuration and each adjudicator, as explained in section 8.4;
- We calculated the average total probability of error for each configuration and each adjudicator.

In the rest of this chapter we will present the results.

8.6 Results of the analysis of optimal adjudication with the two datasets

Figures 8-1 and 8-2 below show the total probability of error calculated for the different types of adjudicators we are studying (1ooN, simple majority vote, NooN and optimal adjudicator), for the two datasets (IDS and SATs) for each of the applications (three for IDSs and two for SATs). To enable a fair comparison, we chose the N values that allow for a simple majority vote to be calculated and where we have more than one diverse IDS system (i.e. N=3, 5, and 7 for the IDS; N=3 for the SATs). Tables 8-3 and 8-4 show the total probability of errors for the 9-version IDS and 5-version SAT for each of the adjudicators, for each application (since there is a single such system that we can calculate there was no need to draw a graph).

As expected, we can observe that the "optimal adjudicator" always performs best, or equally best compared with any of the other adjudicators. In some cases it can outperform the other adjudicators by a large amount.

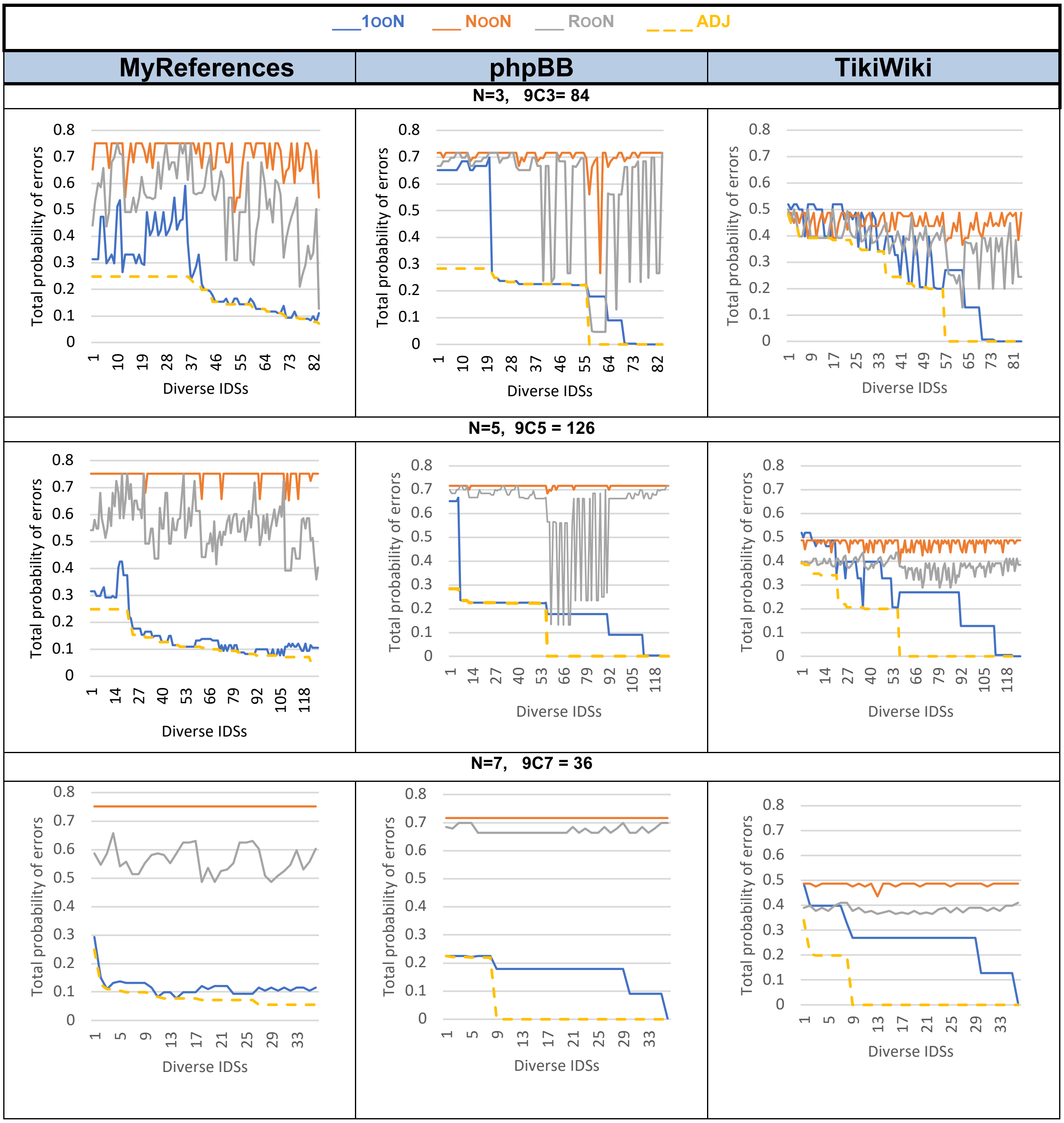


Figure 8-1 Total loss probability of the diverse IDSs for the three applications, for N=3, 5 and 7.

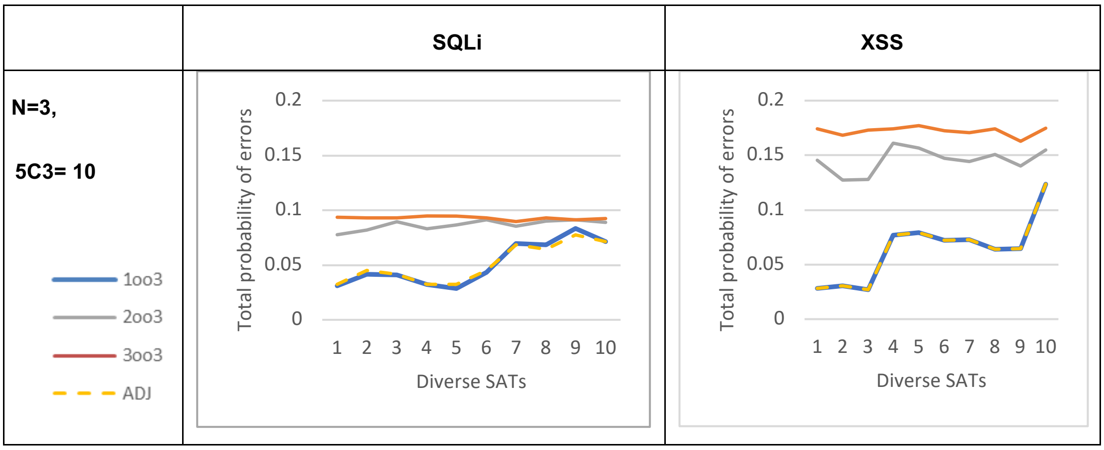


Figure 8-2 Total loss probability of the diverse SATs for the two applications, for N=3.

Table 8-3 – Total loss probability of the diverse 9-version IDSs for each application

	MyReferences	phpBB	TikiWiki
1oo9	0.116	0.178	0.525
5oo9	0.552	0.664	0.776
9oo9	0.751	0.716	1
Optimal Adjudication	0.055	0	0

Table 8-4 - Total loss probability of the diverse 5-version SATs for each application

	SQLi	XSS
1oo5	0.033	0.025
3oo5	0.088	0.153
5oo5	0.499	0.499
Optimal Adjudication	0.029	0.025

8.7 “Weighted loss” analysis

The total probabilities of error we have shown do not discriminate between the different loss values that an organisation may associate with an FP or an FN error. For example the loss that an organisation associates with an FN error may be 10 times higher than for an FP error. We did a simple experiment where we associate different loss “weights” to different errors and compare how much better does an optimal adjudicator perform compared with the other adjudicators as we vary the weights (or in other words by how much the optimal adjudicator can reduce total loss compared with the other adjudicators).

Figure 8-3 shows the average losses (y-axis) for all the possible 3-version, 5-version, 7-version and 9-version IDS systems, for each application. Each row in the table shows a different weight for the FPs and FNs: the first row, shows the examples where we weigh the losses with same weight; and the other rows show the weight we associate with FPs and FNs. Figure 8-5 shows a similar table for the SATs (3-version and 5-version).

We observe that the ordering of the best systems, as well as the gains from optimal adjudication can vary depending on the losses associated with the different errors. The average added losses compared with optimal adjudication for the 1ooN, majority voting and NooN setups are given in the stacked-bar charts in Figures 8-4 for IDSs and Figures 8-6 for SATs (for each application, respectively). So these figures provide an organisation with measures on the savings they could make by reducing losses due to FP or FN errors if they switch from a conventional adjudicator (1oon, majority vote or NooN) to an optimal adjudicator setup.

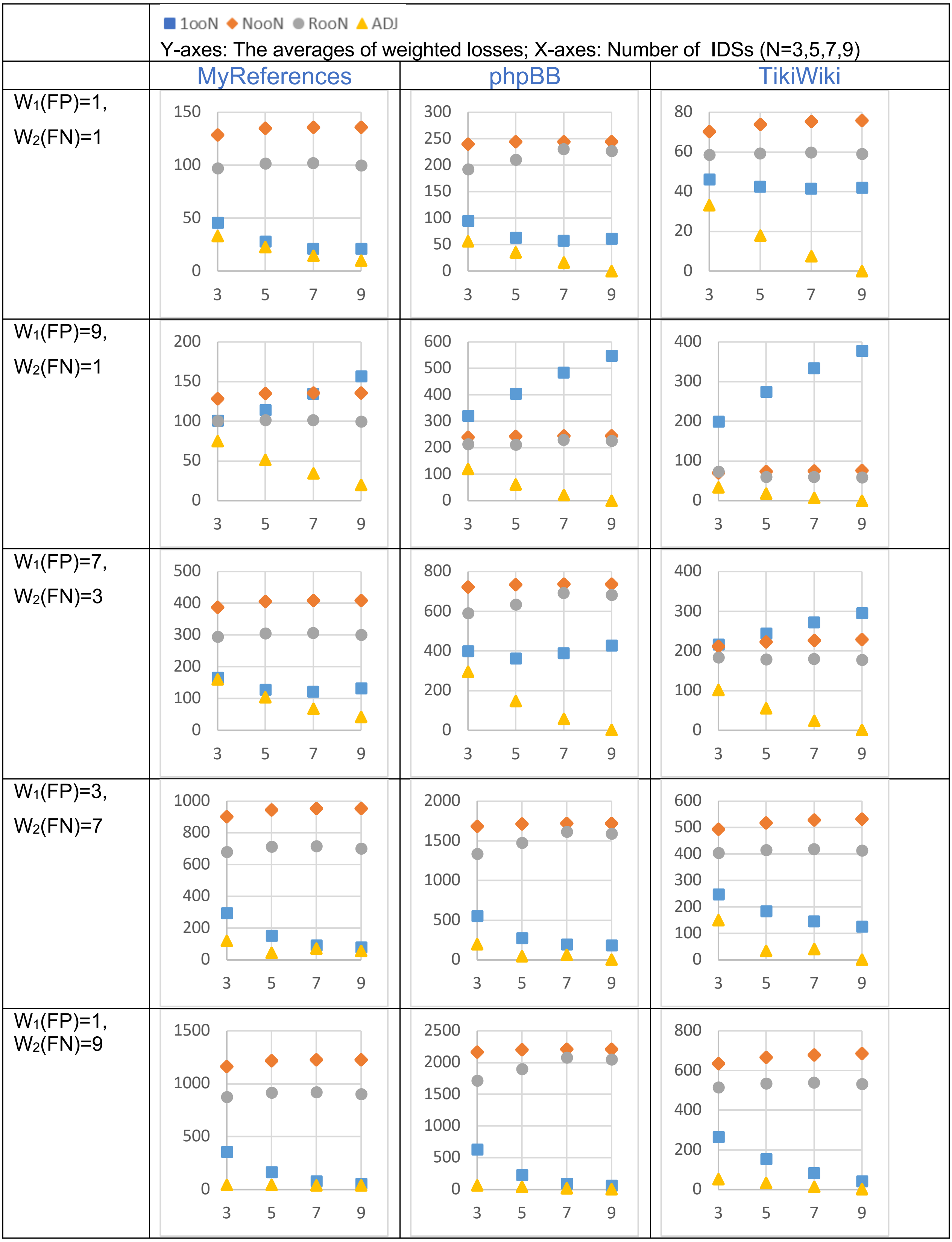


Figure 8-3 The average loss per diverse IDS setup for each application (where N=3,5, 7 and 9)

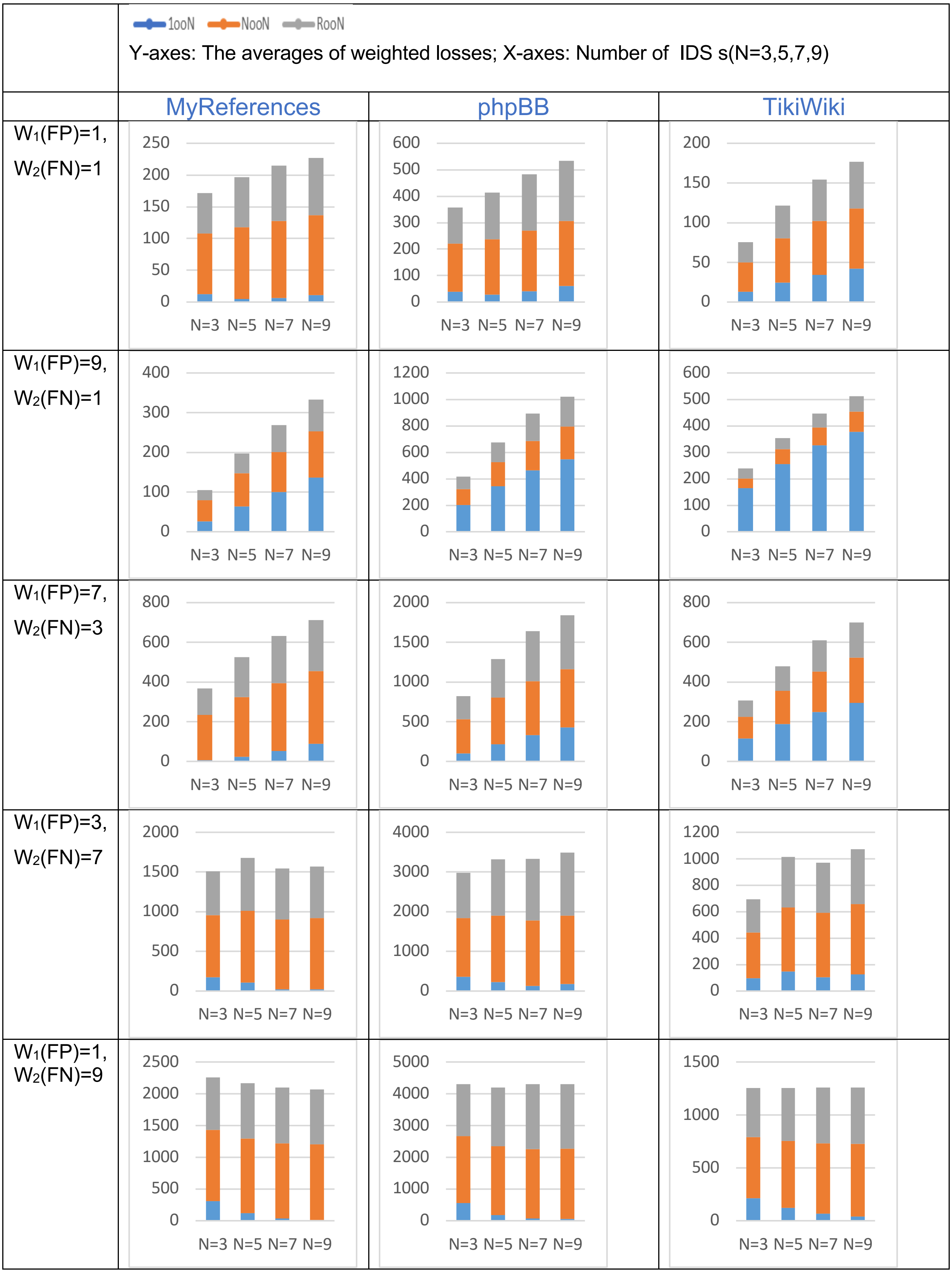


Figure 8-4 The average loss difference between the optimal adjudication function and the other diversty IDS setups (1ooN,majority vote and NooN) for each application. For N=3, 5, 7 and 9.

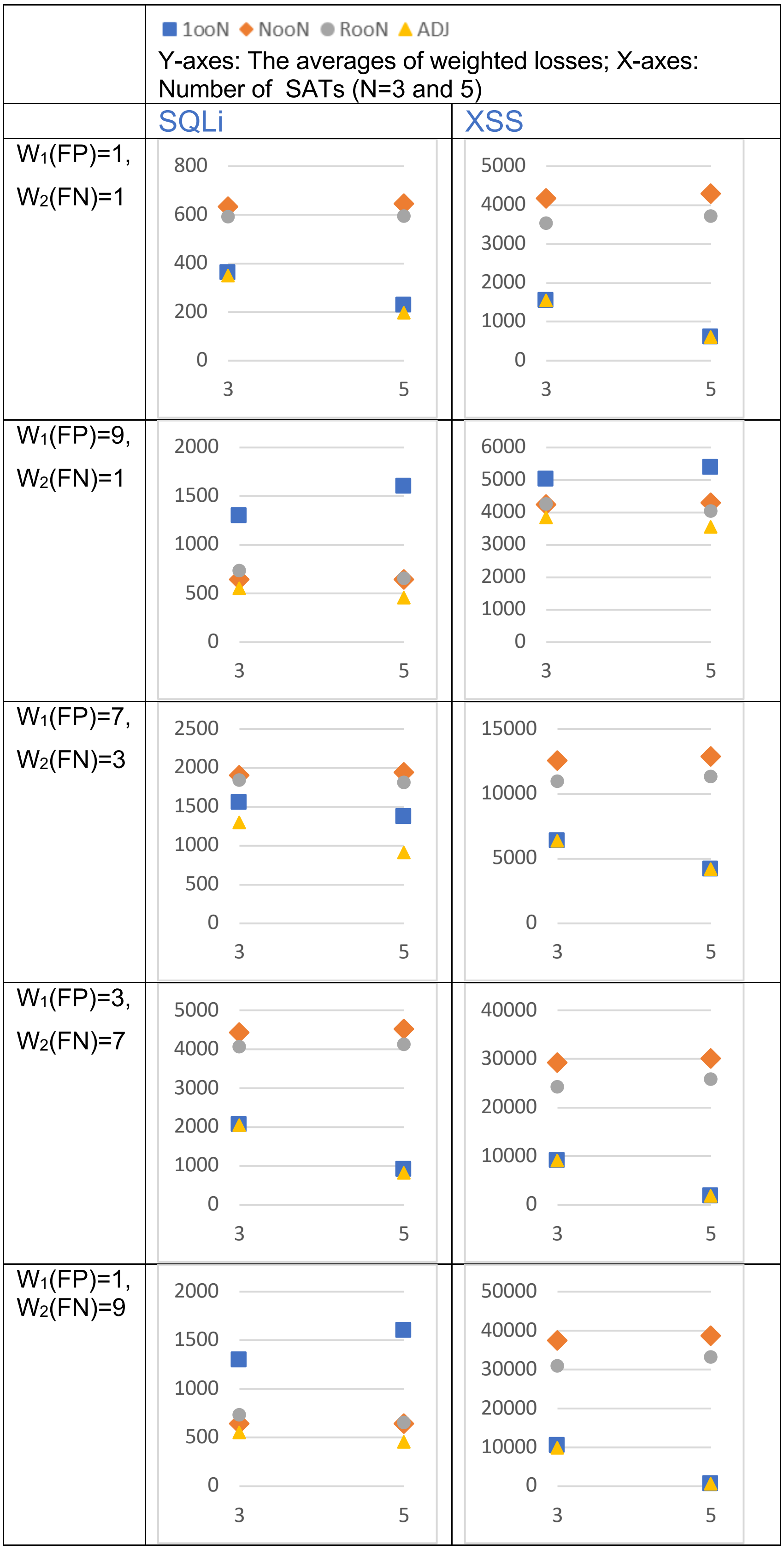


Figure 8-5 The average loss per SAT diverse setup for each application (where N=3 and 5)

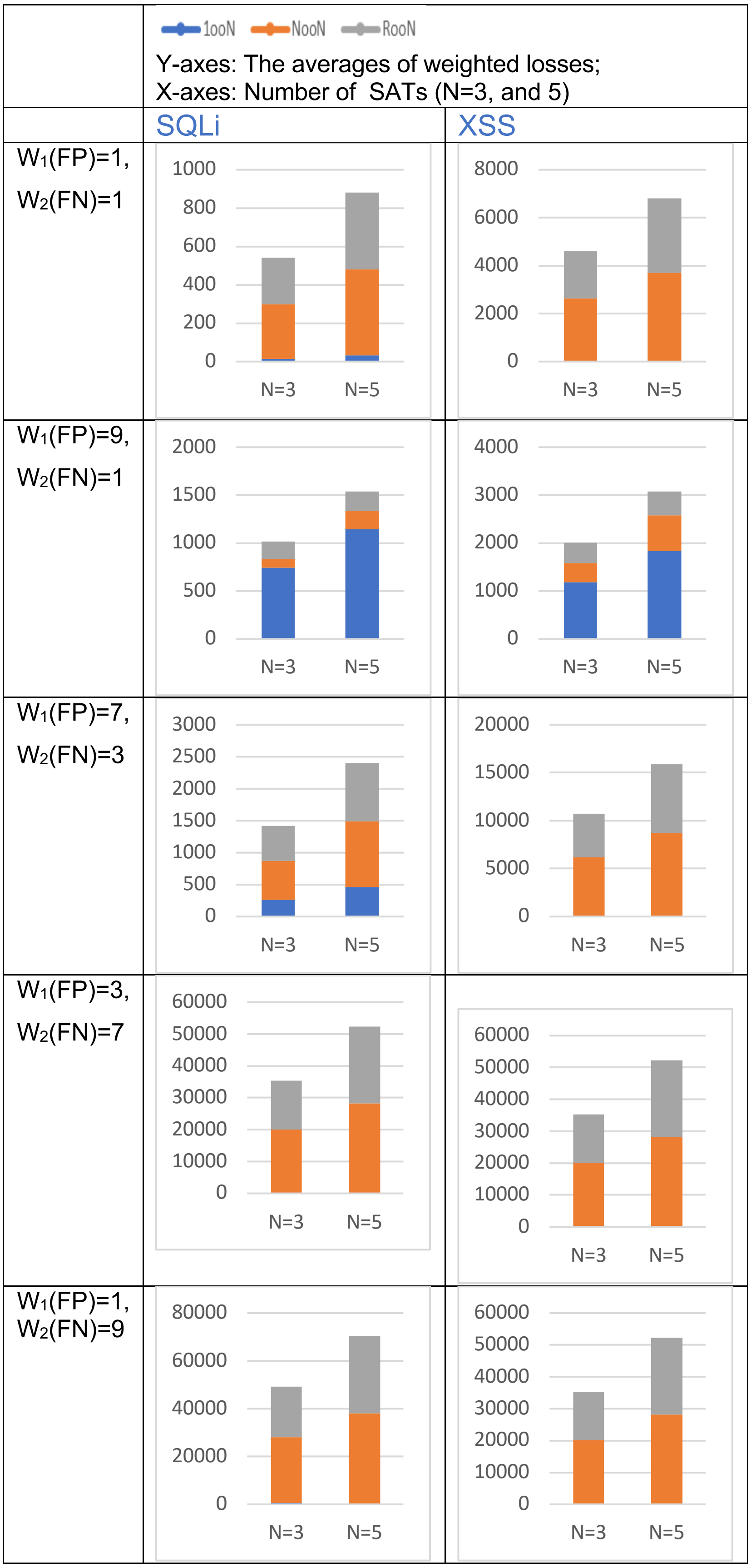


Figure 8-6 The average loss difference between the optimal adjudication function and the other diverse SAT setup (1ooN,majority vote and NooN) for each application.

8.8 Discussion, Conclusions and Limitations

We applied the optimal adjudication function from (Giandomenico & Strigini, 1990) in a practical security context with IDS and SAT datasets that we outlined in the preceding two chapters (namely, Chapters 5 and 6), and we show the gains that an organisation can obtain from switching from a "conventional" adjudicator (such as 1ooN, majority vote or NooN) to an "optimal adjudicator". While the preceding chapters showed the gains/or penalties from using diversity with binary decision systems, in this chapter we show the relative gains from optimal adjudication between different diverse setups. We also show how these gains change as the relative costs of the two types of failures vary in different scenarios.

A limitation of optimal adjudication is that we need to have labelled data on past failures before we decide what is the "optimal" setup. But this is true of any other adjudication scheme. We can blindly decide to use 1ooN, but we will not know if this is working well until we see the results.

As future work, it will be useful to apply the optimal adjudicator scheme to other types of datasets and systems to analyse the potential gains that may be obtained. In terms of engineering further work it will be interesting to have a version of an optimal adjudicator implemented for a working security system that uses multiple channels. This is currently planned as work in progress by other colleagues from City working on the EU DiSIEM project[1].

[1] http://disiem-project.eu/

(9) CONCLUSIONS AND FUTURE WORK

9.1 Introduction

Each of the chapters had their own Conclusions sections. The purpose of this chapter is to link and summarise those conclusions into a single coherent chapter, review the aims and objectives of the research and summarise the provisions for further work.

9.2 Summary of conclusions

We presented research that helps us quantify the possible benefits (and harm) of diversity for security, and hence help improve decision making for security. We conducted experiments with three types of defence tools: AV products, Intrusion Detection Systems (IDS) and Static Analysis Tools (SAT). We presented the results using well-known metrics for binary classifiers: namely Sensitivity and Specificity; we assessed the various forms of adjudication that may be used when configuring diverse tools: 1-out-of-N (raise an alarm as soon as ANY of the defences do so), N-out-of-N (raise an alarm only if ALL the defences do so), majority voting (raise an alarm where a MAJORITY of the defences do so) or optimal adjudication (raise an alarm in such a way that it minimises the overall loss to the system from a failure).

The main conclusions are as follows:

- For the study with **AV** products:

  - For most vendors in our study (seven out of nine) the VirusTotal version has a better detection rate than their full capability version. This suggests that for most of these products the free version they have in VirusTotal is perfectly suitable for malware detection and may even perform better compared with a full capability one;
  - Some of the full capability versions of the AV vendors only detected some of the malware more than three weeks after the VirusTotal version of the same vendor has detected the same malware. This seems to imply that vendors for some malware are testing their detection signatures in their VirusTotal versions first before propagating them to the full capability versions, which may also explain the higher detection rates of the VirusTotal versions of some of these vendors;

    - There are differences between the vendors in the way in which they classify malware. This lack of consistency between the vendor malware classification schemes makes it more difficult for system administrators to transfer their malware analysis expertise from one vendor's system to another;
    - The lack of a platform for assessing full capability products motivated research on building AVAMAT, in collaboration with researchers at the University of Maryland. A prototype of the tool has been built and is currently being further tested and improved.
- For the study with **IDSs**:
    - For 1ooN systems: improvements in sensitivity compared with individual IDS, with a corresponding specificity deterioration on average;
    - For NooN systems: specificity can be perfect in most setups, but with severe deterioration in sensitivity on average;
    - Majority voting systems usually offer a compromise between the extremes of 1ooN and NooN setups, but for these setups they tended to negatively impact the sensitivity measures, with marginal gains in specificity;
    - The more "functionally diverse" a system is the better the sensitivity and specificity on average, though not in all cases.
- For the study with **SATs**:
    - For 1ooN systems: improvements in sensitivity compared with individual SAT are from 70% on average for 1oo2 systems, to more than 300% for 1oo5 systems, but come with a corresponding specificity deterioration on average;
    - For NooN systems: specificity is perfect in most setups, but with severe deterioration in sensitivity;
    - For simple majority voting setups: average deterioration in sensitivity (of between 30-65%) but with improvements in specificity;
    - For organisations primarily interested in detecting vulnerabilities and that are willing to invest resources in analysis alarms to separate out the false alarms from true alarms, diverse setups in a

    1ooN adjudication setup can be very beneficial. In particular, we found phpSAFE, RIPS and WAP SATs exhibit considerable diversity in vulnerability detection;
  - Further analysis of the code in the plugins reveals the source of diversity in the SAT behaviour: some of the tools are better at detecting vulnerabilities in certain code constructs than others (for example the way in which they analyse arrays, control flow constructs etc.).
- The study with "**Optimal adjudication**":
  - The gains that an organisation can obtain from switching from a "conventional" adjudicator (such as 1ooN, majority vote or NooN) to an "optimal adjudicator" can be significant;
  - Hence we show not only the gains (or penalties) from using diversity with binary decision systems, but also the relative gains from optimal adjudication between different diverse setups.

Throughout the thesis we noted that there are limits to the conclusions that we can draw from some of the analysis due to the size, type and age of datasets, the type of applications used, the tools used etc. Despite these limitations the results we have presented should serve as enough justification for any organisation that wishes to try diversity in their setup. Additionally we have presented a well-documented, step-by-step analysis methodology for assessing the performance of N-version diverse security decision support systems. This should prove useful to other researchers and organisations to assess diversity in their setups.

9.3 Review of aims and objectives

In the Introduction of the thesis we stated that the two main Aims and Objectives of the research are to analyse:

1. the variety of architectural options about how diverse security controls are assembled and their responses combined ("adjudication");
2. the interplay between the risks of failing to react to true attacks and of false alarms ("false negative" and "false positive" failures, where "failure" may mean different things – penetration, lack of detection, etc. – depending on the function concerned).

The research we have presented in the five main chapters of this thesis (Chapters 4-8) demonstrate analysis of the adjudication options for combining diverse systems (hence fulfilling objective 1 above), and the interplay between the false positives and false negatives with three different widely used categories of security products, namely AntiVirus products, Intrusion Detection Systems and Static Analysis Tools (hence fulfilling objective 2 above). Additionally, as stated previously, we presented an analysis methodology for assessing the performance of N-version diverse security decision support systems.

9.4 Provisions for further work

There are several provisions for further work. We will outline these per study that we did:

- For the AV study:
    - Conduct a study with more files types than just portable executable files;
    - Extend the analysis with benign files to enable measurements of false positive rates for AV products;
    - A longer data collection time with more vendors;
    - Further improvements of AVAMAT and building support for more products and operating systems.
- For the IDS study:
    - Perform the study with a more recent dataset;
    - Extend the analysis with more IDS products;
    - Use a wider array of applications and types of attacks.
- For the SAT study:
    - Extend the analysis with more SATs;
    - Use a wider array of applications and types of vulnerabilities.
- For Optimal Adjudication study:
    - Apply the optimal adjudicator scheme to other types of datasets and systems to analyse the potential gains that may be obtained;

- Build an optimal adjudicator for a working security system that uses multiple channels (e.g. Security Information and Event Management (SIEM) systems).

We should also note that the thesis mainly deals with the problem of vulnerability/attack/”failure” etc., detection. There are other stages of fault tolerance that we have not looked at in more depth, in particular diagnosis and recovery. The datasets in our research were already labelled. In an operational context, the operators would need to spend time sifting through alarms and diagnosing the true alarms, as well as noting any attacks that have penetrated through the defences while not being caught by the defences. Machine learning algorithms may be able to help with this labelling, but they come with their own problems of diagnosis. Recovery is another concern when we are implementing an end-to-end solution: we need to be able to recover the state of the end-system that we are protecting. Standard forward or backward error recovery techniques may work in some contexts, but nevertheless they need to be studied further depending on the context where the system will be deployed.

Finally, we note that the studies we did were for one family of systems at a time (i.e. we had datasets for AVs, for IDSs and SATs separately). Most organisations deploy these systems in a defence in depth infrastructure. We note that the methodology we have presented in this thesis can also be used to assess the gains or losses from multiple systems in a diverse defence-in-depth setting. The main difficulty for the experimenter will be on deciding the level of granularity for the “demand”/”input” to the different systems that allows for an easy like-for-like comparison between different defences. If the granularity is too fine (e.g. at the network packet level) it may not make sense, for example, to assess AV products at that level. If it is done at the application layer, then some network level defences (network IDSs and Firewalls) may not provide alerts at that level. So the experimenter would need to decide on the right level of granularity to group the demands. And then the rest of the analysis can proceed with the same methodology we presented in this thesis.

9.5 Final remarks

Defence in depth is an important part of design for security. Defences should be diverse in their weaknesses and attacks that defeat one defence should with high probability be stopped or detected by another one. There is no need to emphasise that diversity is a "a good idea": the security community is well aware of that. But there is a need to assess whether, for example, a set of specific defences would improve security more than another set; and to quantifying the security gains. We hope that the research we present in this thesis will go some way to providing a method for measuring diversity for security to drive rational decisions, and quantify the benefits (and harm) from diversity with widely used security tools. We have presented a well-documented, step-by-step analysis methodology for assessing the performance of N-version diverse security decision support systems. This should prove useful to other researchers and organisations to assess diversity in their setups. To the best of our knowledge a comprehensive method, with illustrations using several different tools and multiple datasets has not been presented before.

References

Acunetix (2015) 'Web Application Vulnerability Report 2015'. Available at: https://www.infosecurityeurope.com/__novadocuments/237693?v (Accessed: 21 December 2018).

Antunes, N. and Vieira, M. (2015) 'On the metrics for benchmarking vulnerability detection tools', *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* 22-25 June, Rio de Janeiro, Brazil, pp. 505–516. IEEE, USA, doi: 10.1109/DSN.2015.30.

Avižienis, A., Laprie, J. C., Randell, B. and Landwehr, C. (2004) 'Basic concepts and taxonomy of dependable and secure computing', *IEEE Transactions on Dependable and Secure Computing*, 1(1), pp. 11–33. IEEE , USA, doi: 10.1109/TDSC.2004.2.

Aycock, J. D. (2006) 'Computer viruses and malware', 22, Springer, USA, pp. 27–203, ISBN 978-0-387-34188-0. doi: 10.1007/0-387-34188-9.

Babbage, C. (1982) 'On the mathematical powers of the calculating engine (unpublished manuscript, December 1837)', In: Randell, B. (eds) *The Origins of Digital Computers Selected Papers*. 3rd edn, pp. 17–52. Springer Verlag, Berlin, Germany, doi: 10.1007/978-3-642-61812-3_2.

Backes, M., Rieck, K., Skoruppa, M., Stock, B. and Yamaguchi, F. (2017) 'Efficient and flexible discovery of PHP application vulnerabilities', *Proceedings of the 2nd IEEE European Symposium on Security and Privacy (EuroS and P)* 26-28 May, Paris, France, pp. 334–349. IEEE, USA, doi: 10.1109/EuroSP.2017.14.

Barnett, R. (2015) 'The web application security consortium / SQL injection', Available at: http://projects.webappsec.org/w/page/13246963/SQL Injection (Accessed: 25 December 2018).

Bayer, U., Andreas, M., Kruegel, C. and Kirda, E. (2006) 'Dynamic analysis of malicious code', *Journal of Computer Virology*, 2(1), pp. 67–77. Springer Verlag, Berlin, Germany, doi: 10.1007/s11416-006-0012-2.

Bishop, P. (1995) 'Software fault tolerance', In: Lyu, R. (eds) *Software Fault Tolerance by Design Diversity*, vol1995, pp. 212–230. John Wiley & Sons Ltd., USA, ISBN 978-0471950684. Available at: http://www.cse.cuhk.edu.hk/~lyu/book/sft/pdf/chap9.pdf (Accessed: 16 December 2018).

Bishop, P., Bloomfield, R., Gashi, I. and Stankovic, V. (2011) 'Diversity for security: a study with off-the-shelf antivirus engines', *Proceedings of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE)* 2-5 November, Hiroshima, Japan, pp. 11–19. IEEE, USA, Available at: http://openaccess.city.ac.uk/524/.

Britton, W. (2011) ‘Media from Black Hat, BlackHat’. Available at: https://media.blackhat.com/bh-us-11/Willis/BH_US_11_WillisBritton _Analyzing_Static_Analysis_Tools_WP.pdf (Accessed: 16 December 2018).

Buehrer, G., Weide, B. and Sivilotti, P. (2005) ‘Using parse tree validation to prevent SQL injection attacks’, *The 5th International Workshop on Software Engineering and Middleware (SEM)* 5-6 September, Lisbon, Portugal, pp. 106–113. ACM, New York, NY, USA, doi: 10.1145/1108473.1108496.

Buscombe, J., Cwikla, J., Holloway, B. and Hilson, A. (2001) ‘Prediction of the usefulness of combined mammography and scintimammography in suspected primary breast cancer using ROC curves’, *Journal of Nuclear Medicine*, 42(1), pp. 3–8. doi: 10.1300/J199v06n02.

Chen, L. and Avizienis, A. (1995) ‘N-version programming: A fault-tolerance approach to reliability of software operation’, *‘Highlights from Twenty-Five Years’*, *Proceedings of the 25th IEEE International Symposium on Fault Tolerant Computing (FTCS-25)* 27-30 June, Pasadena, CA, USA, pp. 113–119. IEEE, USA, doi: 10.1109/FTCSH.1995.532621.

Dahse, J. and Holz, T. (2014) ‘Simulation of built-in PHP features for precise static code analysis’, *Proceedings of the Network and Distributed System Security (NDSS) Symposium* 23-26 February. San Diego, CA, USA, [no pagination]. doi: 10.14722/ndss.2014.23262.

Egan, J. (1975) ‘Signal detection theory and ROC-analysis’, Academic Press, New York. ISBN 0122328507

Elia, I. A., Fonseca, J. and Vieira, M. (2010) ‘Comparing SQL injection detection tools using attack injection: an experimental study’, *Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE)* 1-4 November, San Jose, CA, USA, pp. 289–298. IEEE, USA, doi: 10.1109/ISSRE.2010.32.

Fogie, S., Grossman, J., Hansen, R., Rager, A. and Petkov, P. D. (2007) ‘XSS attacks: cross site scripting exploits and defense’, In: Fogie, S. (eds) *Syngress Publishing*, Burlington, MA, USA. pp. 2–190. ISBN 978-1597491549, doi: 10.1007/s13398-014-0173-7.2.

Fonseca, J., Seixas, N., Vieira, M. and Madeira, H. (2014) ‘Analysis of field data on web security vulnerabilities’, *IEEE Transactions on Dependable and Secure Computing*. IEEE, USA, 11(2), pp. 89–100.

Garcia, M., Bessani, A., Gashi, I., Neves, N. and Obelheiro, R. (2014) ‘Analysis of operating system diversity for intrusion tolerance’, In: Buyya, R., Bishop, J., Cooper, K., Jones, R., Poggi, A. and Srirama, S.(eds), *Software: Practice and Experience*, John Wiley & Sons Ltd., USA, 44(6), pp. 735–770. doi: 10.1002/spe.2180.

Gashi, I., Sobesto, B., Stankovic, V. and Cukier, M. (2013) ‘Does Malware Detection Improve with Diverse Antivirus Products? An

Empirical Study', In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds), *Proceedings of the 32nd International Conference on Computer Safety, Reliability and Security (SAFECOMP)* 14-17 February*,* Toulouse, France, pp. 94–105. , Springer, Berlin, Heidelberg. doi: 10.1007/978-3-642-40793-2. Available at: http://openaccess.city.ac.uk/2338/1/Does Malware Detection Improve With Diverse.pdf (Accessed: 6 December 2014).

Gashi, I., Stankovic, V., Leita, C. and Thonnard, O. (2009) 'An experimental study of diversity with off-the-shelf antivirus engines', *Proceedings of the 8th IEEE International Symposium on Network Computing and Applications (NCA)* 9-11 July, Boston, MA, USA, pp. 4–11. IEEE, USA, doi: 10.1109/NCA.2009.14.

di Giandomenico, F. and Strigini, L. (1990) 'Adjudicators for diverse-redundant components', *Proceedings of the 9th IEEE Symposium on Reliable Distributed Systems (SRDS)* 9-11 October, Huntsville, AL, USA, pp. 114–123. IEEE, USA, doi: 10.1109/RELDIS.1990.93957.

Gupta, V., Lam, V., Ramasamy, H., Sanders, W. and Singh, S. (2003) 'Dependability and Performance Evaluation of Intrusion-Tolerant Server Architectures', In: de Lemos, R., Weber, T. and CamargoJr, J.(eds), *Proceedings of the 1st Latin American Symposium on Dependable Computing, Lecture Notes in Computer Science* 21-24 October, Sao Paulo, Brazil, 2847, pp. 81–101. Springer-Verlag, Berlin, Germany, doi: 10.1007/978-3-540-45214-0_9.

Halfond, W. G. and Orso, A. (2005) 'AMNESIA: Analysis and Monitoring for NEutralizing SQL-injection Attacks', *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering* 7-11 November, Long Beach, CA, USA, pp. 174–183. ACM Press, USA, doi: 10.1145/1101908.1101935.

Halfond, W., Viegas, J. and Orso, A. (2006) 'A classification of SQL injection attacks and countermeasures', *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Washington DC, USA, pp. 13–22. Available at: https://pdfs.semanticscholar.org/81a5/02b52485e52713ccab6d260f15871c2acdcb.pdf (Accessed: 16 December 2018).

Hauzar, D. and Kofroň, J. (2015) 'Framework for static analysis of PHP', In: Boyland, J., (eds) *Proceedings of the 29th European Conference on Object-Oriented Programming (ECOOP'15)* 5-10 July, Prague, Czech Republic, pp. 689–711. Schloss Dagstuhl, Germany, doi: 10.1145/1101908.1101935.

Hopley, L. and Schalkwyk, J. (2011) 'The magnificent ROC (Receiver Operating Characteristic curve)'. Available at: http://www.anaesthetist.com/mnm/stats/roc/Findex.htm (Accessed: 16 December 2018).

IBM Security Solutions (2013) 'IBM X-Force 2013 mid-year trend and risk report', *IBM Security Systems*. Available at: https://www.viftech.com/wp-content/uploads/2015/05/IBM-X-Force-

2013-Mid-Year-Trend-and-Risk-Report-Executive-Summary.pdf.

Jovanovic, N., Kruegel, C. and Kirda, E. (2006) ‘Pixy: a static analysis tool for detecting Web application vulnerabilities’, *Proceedings of the IEEE Symposium on Security and Privacy (S and P)* 21-24 May, Oakland, CA, USA, pp. 258–263. IEEE, USA, doi: 10.1109/SP.2006.29.

Kelly, J. and Avizienis, A. (1983) ‘A specification-oriented multi-version software experiment’, *Proceedings of the 13th International Symposium on Fault-Tolerant Computing* (FTCS) 28-30 June, Milan, Italy, pp. 121–126. IEEE, USA,

Kiezun, A., Guo, P. J., Jayaraman, K. and Ernst, M. D. (2009) ‘Automatic creation of SQL injection and cross-site scripting attacks’, *Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE)*16-24 May, Vancouver, BC, Canada, pp. 199–209. IEEE, USA, doi: 10.1109/ICSE.2009.5070521.

Knight, J. C. and Leveson, N. G. (1986) ‘An experimental evaluation of the assumption of independence in multiversion programming’. *IEEE Transactions on Software Engineering (TSE)*, 12(1), pp. 96–109. IEEE, USA, doi: 10.1109/TSE.1986.6312924.

Kruegel, C. and Vigna, G. (2003) ‘Anomaly detection of web-based attacks’, *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)* 27-30 October, Washington D.C., USA, pp. 251–261. ACM, USA. doi: 10.1145/948143.948144.

Lee, I., Jeong, S., Yeo, S. and Moon, J. (2012) ‘A novel method for SQL injection attack detection based on removing SQL query attribute values’,In Agarwal, R. et al.(eds), *Mathematical and Computer Modelling*, 55(1–2), pp. 58–68. Elsevier Ltd, Amsterdam, Netherlands. doi: 10.1016/j.mcm.2011.01.050

Lee, P. and Anderson, T. (1990) ‘Fault tolerance: principles and practice’, In: Avizienis, A., Kopetz, H., and Laprie, J. (eds), *Dependable Computing and Fault-Tolerant Systems*, 3, Springer-Verlag Wien, New York. ISBN 978-3-7091-8990-0, doi: 10.1007/978-3-7091-8990-0.

Leita, C. and Dacier, M. (2008) ‘SGNET: A worldwide deployable framework to support the analysis of malware threat models’, *Proceedings of the 7th European Dependable Computing Conference (EDCC)* 6-8 May, Kaunas, Lithuania, pp. 99–109. IEEE, USA, doi: 10.1109/EDCC-7.2008.15.

Littlewood, B., Popov, P. and Strigini, L. (2001) ‘Modeling software design diversity: a review’, *ACM Computing Surveys*, 33(2), pp. 177–208. doi: 10.1145/384192.384195.

Littlewood, B. and Strigini, L. (2004) ‘Redundancy and diversity in security’, In: Samarati, P., Ryan, P., Gollmann, D. and Molva, R. (eds) *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS)* 13-15 September, Sophia Antipolis, France, pp. 423–438. Springer-Verlag, Berlin, Germany, doi: 10.1007/978-3-540-30108-0_26.

Medeiros, I., Neves, N. F. and Correia, M. (2014) ‘Automatic detection and correction of web application vulnerabilities using data mining to predict false positives’, *Proceedings of the 23rd International Conference on World Wide Web (WWW ’14)* 7-11 April. Seoul, Korea, pp. 63–74. ACM, New York, NY, USA, doi: 10.1145/2566486.2568024.

Milenkoski, A., Vieira, M., Kounev, S., Avritzer, A. and Payne, B. D. (2015) ‘Evaluating computer intrusion detection systems: a survey of common practices’, *ACM Computing Surveys (CSUR)*, 48(1), pp. 1–41. ACM, New York, NY, USA, doi: 10.1145/2808691

Mookhey, K. and Burghate, N. (2004) ‘Detection of SQL injection and cross-site scripting attacks’, *Symantec Security Focus*. Available at: https://www.symantec.com/connect/articles/detection-sql-injection-and-cross-site-scripting-attacks (Accessed: 4 December 2018).

MwAnalysis (2013) ‘Malware Analysis CWSandbox’. Available at: http://www.mwanalysis.org. (Accessed: 16 December 2018).

Na, M., Qianxiang, W., Qian, W. and Hong, M. (2008) ‘An approach to merge results of multiple static analysis tools’, *Proceedings of the 8th International Conference on Quality Software 12-13 Aug*, Oxford, UK, pp. 169–174. IEEE, USA, doi: 10.1109/QSIC.2008.30.

NIST (2018) ‘CAS static analysis tool study methodology’, *SAMATE NIST*. Available at: https://www.semanticscholar.org/paper/Cas-Static-Analysis-Tool-Study-Methodology-Meade/966ed17141ad061c0500028a15d65cd4a0c551d7 (Accessed: 4 December 2018).

Nunes, P., Medeiros, I., Fonseca, J., Neves, N., Correia, M. and Vieira, M. (2017) ‘On Combining Diverse Static Analysis Tools for Web Security: An Empirical Study’, *Proceedings of the 13th European Dependable Computing Conference (EDCC)* 4-8 September, Geneva, Switzerland, pp. 121–128. IEEE, USA, doi: 10.1109/EDCC.2017.16.

Oberheide, J., Cooke, E. and Jahanian, F. (2008) ‘CloudAV: N-version Antivirus in The Network Cloud’, *Proceedings of the 17th conference on Security Symposium* 28 July-1 August, San Jose, CA, USA, pp. 91–106. USENIX Association, Berkeley, CA, USA, doi: 10.1109/ACSAC.2006.38.

OWASP (2017) ‘Top 10 - The ten most critical web application security risks’, *Owasp*. Available at: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (Accessed: 4 December 2018).

Pouget, F. (2005) ‘Leurre.com: on the advantages of deploying a large scale distributed honeypot platform’, Available at: http://www.eurecom.fr/publication/1558 (Accessed: 7 December 2014).

Provost, F. and Fawcett, T. (1997) ‘Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions’, *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD’97)* 14-17 August. Newport Beach, CA, USA, pp. 43–48. AAAI Press, USA, doi: 10.1007/978-3-642-25437-6_82.

Pullum, L. (2001) ‘Software fault tolerance techniques and implementation’, *Artech House*, Boston/London, ISBN 1-58053-470-8 ,Available at: http://index-of.co.uk/SE/Artech%20House%20-%20Software%20Fault%20Tolerance%20Techniques%20and%20Imple.pdf (Accessed: 4 January 2019).

Ramsbrock, D., Berthier, R. and Cukier, M. (2007) ‘Profiling attacker behavior following SSH compromises’, Proceedings of the *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)* 25-28 June, Edinburgh, UK, pp. 119–124. IEEE, USA, doi: 10.1109/DSN.2007.76.

Rangayyan, R., Shen, L., Shen, Y., Leo Desautels, J., Bryant, H., Terry, T. J., Horeczko, N. and Rose, M. (1997) ‘Improvement of sensitivity of breast cancer diagnosis with adaptive neighborhood contrast enhancement of mammograms’, *IEEE Transactions on Information Technology in Biomedicine*, 1(3), pp. 161–170. IEEE, USA, doi: 10.1109/4233.654859.

Reynolds, J., Just, J., Lawson, E., Clough, L., Maglich, R. and Levitt, K. (2002) ‘The design and implementation of an intrusion tolerant system’, *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN)* 23-26 June. Bethesda, MD, USA, pp. 285–290. IEEE, USA, doi: 10.1109/DSN.2002.1028912.

Richardson, R. (2011) ‘2010/2011 Computer crime and security survey’, computer security institute (15th)’, pp. 1–40. Available at: https://cours.etsmtl.ca/gti619/documents/divers/CSIsurvey2010.pdf (Accessed: 4 December 2018).

Rutar, N., Almazan, C. and Foster, J. (2004) ‘A comparison of bug finding tools for Java’, Proceedings of the *15th International Symposium on Software Reliability Engineering* (ISSRE) 2-5 November, Saint-Malo, France, pp. 245–256. IEEE, USA, doi: 10.1109/ISSRE.2004.1.

Salako, K. (2018) ‘D3.2 Probabilistic modelling of diversity for security and security trend (DiSIEM)’. Available at: http://disiem-project.eu/wp-content/uploads/2018/03/D3.2.pdf (Accessed: 4 December 2018).

Seclab (2011) ‘Anubis: Analyzing Unknown Binaries’. Available at: https://seclab.cs.ucsb.edu/academic/projects/projects/anubis/ (Accessed: 10 January 2019).

Shar, L. and Tan, H. (2013) ‘Defeating SQL injection’, *Computer*, 46(3), pp. 69–77. IEEE, USA, doi: 10.1109/MC.2012.283.

Sharma, P., Johari, R. and Sarma, S. (2012) ‘Integrated approach to prevent SQL injection attack and reflected cross site scripting attack', *International Journal of Systems Assurance Engineering and Management*. Springer India, 3(4), pp. 343–351.

Singh, S., Cukier, M. and Sanders, W. H. (2003) 'Probabilistic validation of an intrusion-tolerant replication system', *Proceedings of the International Conference on Dependable Systems and Networks (DSN)* 22-25 June, San Francisco, CA, USA. pp. 615–624. IEEE, USA, doi: 10.1109/DSN.2003.1209971.

Srivastava, S., Ranjan, R. and Tripathi, R. (2012) 'Attacks due to SQL injection & their prevention method for web-application', *International Journal of Computer Science and Information Technology (IJCSIT)*, 3(2), pp. 3615–3618. AIRCC Publishing Corporation, India, Available at: http://ijcsit.com/docs/Volume 3/Vol3Issue2/ijcsit2012030266.pdf.

Spackman, K. A. (1989) 'Signal detection theory: valuable tools for evaluating inductive learning', In: Segre, A.(eds) *Proceedings of the 6th International Workshop on Machine Learning* 26-27 June, Ithaca, NY, USA, pp. 160–163. Elsevier Inc, London, UK, doi: 10.1016/B978-1-55860-036-2.50047-3.

Spitzner, L. (2002) 'Honeypots: tracking hackers', *Addison-Wesley ,* ISBN 9780321108951. Available at: http://www.it-docs.net/ddata/792.pdf (Accessed: 5 January 2019).

Strigini, L. (2005) 'Fault tolerance against design faults', In: Diab, H. and Zomaya, A.(eds) *Dependable Computing Systems: Paradigms, Performance Issues and Applications*. pp. 213–241. Wiley & Sons, ISBN 0471674222.

Sucuri Remediation Group (2017) 'Hacked website report'. Available at: https://sucuri.net/reports/2017-hacked-website-report (Accessed: 4 December 2018).

Swets, J. (1988) 'Measuring the accuracy of diagnostic systems', *Science*,USA, 240 (4857), pp.1285-1293 doi: 10.1126/science.3287615.

Swets, J., Dawes, R. and Monahan, J. (2000) 'Better decisions through science', In: Ceci, S. and Bjork, R.(eds) *Scientific American,Inc. ,* USA 1(1), pp. 82–87, doi: 10.1038/scientificamerican1000-82. Available at: https://www.psychologicalscience.org/pdf/pspi/sciam.pdf (Accessed: 4 December 2018).

Tajpour, A., Ibrahim, S., Zamani, M. and Sharifi, M. (2013) 'Effective measures for evaluation of SQL injection detection and prevention tools', *Journal of Convergence Information Technology*, *Advanced Institute of Convergence Information Technology Research Center*, South Korea, 8(14), pp. 13–28.

Ulvila, J. and Gaffney, J. (2003) 'Evaluation of intrusion detection systems', *Journal of Research of the National Institute of Standards and Technology*, *National Institute of Standards and Technology,* USA*,* 108(6), pp. 453–473. doi: 10.6028/jres.

Verıssimo, P., Neves, N. and Correia, M. (2003) ‘Intrusion-Tolerant Architectures: Concepts and Design’, In: de Lemos, R., Gacek, C. and Romanovsky, A. (eds) Architecting Dependable Systems*. Lecture Notes in Computer Science, 2677*. pp. 3–36. Springer, Berlin, Heidelberg, doi: 10.1007/3-540-45177-3_1

VMRay (2014) ‘VMRay analyzer overview’. Available at: http://www.vmray.com/vmray-analyzer-overview/ (Accessed: 6 December 2014).

Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A. C., Voelker, G. M. and Savage, S. (2005) ‘Scalability, fidelity, and containment in the potemkin virtual honeyfarm’, in *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles - SOSP ’05*. ACM, New York, NY, USA, pp. 148–162. doi: 10.1145/1095810.1095825.

W3Schools (2016) ‘SQL injection’. Available at: https://www.w3schools.com/sql/sql_intro.asp (Accessed: 5 January 2019).

Wang, L., Islam, T., Long, T., Singhal, A. and Jajodia, S. (2008) ‘An attack graph-based probabilistic security metric’, In: Atluri, V. (eds) Data and Applications Security XXII, *Lecture Notes in Computer Science (DBSec) 13-16 July,* London, UK, 5094, pp. 283–296. Springer, Verlag,Germany,

Wang, L., Li, Z., Ren, S. and Kwiat, K. (2011) ‘Optimal voting strategy against rational attackers’, Proceedings of the 6th International *Risk and Security of Internet and Systems (CRiSIS)* 26-28 September, Timisoara, Romania, pp. 1–8. IEEE, USA, doi:10.1109/CRiSIS.2011.6061841

Willems, G., Holz, T. and Freiling, F. (2007) ‘Toward automated dynamic malware analysis using CWSandbox’, *IEEE Security and Privacy*, 5(2), pp. 32–39. IEEE, USA, doi: 10.1109/MSP.2007.45

wpvulndb (2018) ‘WPScan vulnerability database’. Available at: https://wpvulndb.com/ (Accessed: 5 January 2019).

Appendix A (Supporting Chapter 4 of the thesis)

This appendix provides further details to support the analysis provided in Chapter 4 of the thesis.

Appendix A-1: Analysing the Dataset over the Three Dimensions (AV, Malware, Dates)

Appendix A. Table 1 Analysis for Figure 4-1 (a). Each cell represents the number of malware in each of the failure rate categories, for that date: Full Capability products.

FC: MW-Date							
Date	**-1**	**FR=0**	**0>FR> 0.25**	**0.25>=FR> 0.50**	**0.50>=FR> 0.75**	**0.75>=FR> 1.0**	**FR=1**
11/11/2013	2060	1400	109	14	22	0	0
12/11/2013	68	3263	201	18	55	0	0
13/11/2013	68	3262	204	17	54	0	0
14/11/2013	67	3261	206	13	58	0	0
15/11/2013	68	3327	140	22	48	0	0
16/11/2013	67	3318	149	19	52	0	0
17/11/2013	66	3321	146	23	49	0	0
18/11/2013	59	3325	150	25	46	0	0
19/11/2013	59	3326	149	13	58	0	0
20/11/2013	56	3339	139	14	57	0	0
21/11/2013	51	3329	154	14	57	0	0
22/11/2013	46	3342	146	17	54	0	0
23/11/2013	41	3343	150	23	48	0	0
24/11/2013	41	3344	150	17	53	0	0
25/11/2013	35	3384	116	21	49	0	0
26/11/2013	31	3351	153	20	50	0	0
27/11/2013	26	3365	143	13	58	0	0
28/11/2013	26	3358	149	17	55	0	0
29/11/2013	17	3451	66	20	51	0	0
30/11/2013	17	3444	73	23	48	0	0
01/12/2013	14	3448	73	17	53	0	0
02/12/2013	8	3457	70	23	47	0	0
03/12/2013	24	3453	58	19	51	0	0
04/12/2013	12	3463	61	22	47	0	0
05/12/2013	4	3469	62	19	51	0	0
06/12/2013	4	3457	72	23	49	0	0
07/12/2013	0	3463	70	23	49	0	0
08/12/2013	0	3451	82	23	49	0	0
09/12/2013	0	3458	75	27	45	0	0
10/12/2013	2	3444	85	27	47	0	0
11/12/2013	8	3448	76	18	55	0	0

Appendix A. Table 1,2,3 and 4 represent the analysis for Figure 4-2 (a), 4-2 (b) 4-3 (a) and 4-3 (b): the three-dimensional plots (Malware, AV or Date) in the thesis: Appendix A. Table 1 and 3 for the FC products and Appendix A. Table 2 and 4 for the VT products.

Using Table Appendix A. Table 1 as illustration: the first column shows the date in which the AVs inspected the malware samples; the second gives the count of malware that were not inspected by the malware in a given date (missing data). The rest of the columns show the failure rate range for a given date; for example, there are 109 malware with failure rate greater than zero and less than 0.25 in the first date of the experiment (11/11/2013) (meaning between 0 and 25% of the AVs failed to detect them)

Appendix A. Table 2 Analysis for Figure 4-1 (b). Each cell represents the number of malware in each of the failure rate categories, for that date: VirusTotal products.

VT: MW -Date							
Date	**-1**	**FR=0**	**0>FR> 0.25**	**0.25>=FR> 0.50**	**0.50>=FR> 0.75**	**0.75>=FR> 1.0**	**FR=1**
11/11/2013	2060	1501	9	3	32	0	0
12/11/2013	68	3444	24	6	62	1	0
13/11/2013	68	3444	24	7	60	2	0
14/11/2013	67	3450	19	6	62	1	0
15/11/2013	68	3452	17	9	58	1	0
16/11/2013	67	3452	17	6	62	1	0
17/11/2013	66	3453	17	7	61	1	0
18/11/2013	59	3459	18	6	62	1	0
19/11/2013	59	3460	17	6	62	1	0
20/11/2013	56	3463	17	6	62	1	0
21/11/2013	51	3465	20	6	62	1	0
22/11/2013	46	3473	17	6	62	1	0
23/11/2013	41	3478	17	6	62	1	0
24/11/2013	41	3477	18	6	62	1	0
25/11/2013	35	3478	23	8	59	2	0
26/11/2013	31	3486	19	9	59	1	0
27/11/2013	26	3492	18	6	62	1	0
28/11/2013	17	3502	17	6	62	1	0
29/11/2013	17	3502	16	8	62	1	0
30/11/2013	17	3498	21	7	60	2	0
01/12/2013	14	3502	20	7	61	1	0
02/12/2013	8	3510	17	9	60	1	0
03/12/2013	24	3423	89	6	61	2	0
04/12/2013	12	3394	130	7	61	1	0
05/12/2013	4	3511	21	8	60	1	0
06/12/2013	4	3478	54	7	60	2	0
07/12/2013	0	3498	38	8	60	1	0
08/12/2013	0	3466	70	9	59	1	0
09/12/2013	0	3436	100	6	61	2	0
10/12/2013	2	3477	57	8	59	2	0
11/12/2013	8	3511	17	6	61	2	0

Appendix A. Table 3 Analysis for Figure 4-2 (a)

FC : MW- AV						
AV name	**FR=0**	**0>FR> 0.25**	**0.25>=FR> 0.50**	**0.50>=FR> 0.75**	**0.75>=FR> 1.0**	**FR=1**
avg	3603	0	0	0	0	2
antivir	3590	0	0	0	0	15
comodo	3598	2	0	0	0	5
F-Secure	3520	30	1	0	0	54
Kaspersky	3534	0	0	1	0	70
McAFee	2637	858	0	4	22	84
Microsoft	3470	49	7	0	0	79
Sophos	3456	71	0	0	3	75
Symantec	3557	29	8	2	8	1

Appendix A. Table 4 Analysis for Figure 4-2(b)

VT : MW- AV						
AV name	**FR=0**	**0>FR> 0.25**	**0.25>=FR> 0.50**	**0.50>=FR> 0.75**	**0.75>=FR> 1.0**	**FR=1**
avg	3543	0	0	0	0	62
antivir	3591	0	0	0	0	14
comodo	3214	389	0	0	0	2
F-Secure	3548	2	1	0	0	54
Kaspersky	3523	13	0	0	0	69
McAFee	3537	1	0	0	0	67
Microsoft	3526	0	0	0	0	79
Sophos	3540	6	0	0	0	59
Symantec	3559	45	0	0	0	1

Appendix A-2: Malware Classification

In this section we show more details about how each AV labelled the malware (full capability versions of the AV for each AV except for Kaspersky for which we used VirusTotal as the full capability version of Kaspersky labelled each detected malware by its MD5 value only). We counted the malware labels by classes of malware associated by the AV.

Appendix A. Table 5 Examples of how antivirus products classify malware differently. These tables provide further details to support the analysis provided in Table 4-9 in Chapter 4

Colour Code:

Back Door
Virus
Worm
Trojan
Virus identified - Worm
Trojan Horse - BackDoor
Worm -Trojan
Other

AntiVir

Malware Type	Counter
back-door program	377
DR/dropper	184
JS/ Java script virus	31
RKIT root kit	30
TR/ Trojan	40,822
W32/Windows virus	6,510
Worm.Conficker.Gen virus	62
WORM/ worm	60,125
TOTAL	108,141

AVG

Malware Type	Counter
Found Win32	152
Trojan horse	1,208
Trojan horse BackDoor.	10,924
Virus found	3,865
Virus identified -Worm	88,464
Virus identified Win32	1,622
TOTAL	106,235

Comodo

Malware Type	Counter
.UnclassifiedMalware	3
ApplicUnsaf.Win32	29
Backdoor.Win32	453
MalCrypt.Indus!@	90
Malware	1,302
NetWorm.Win32.Allaple.GEN	89,901
NetWorm.Win32.Trojan.Conficker	154
Packed.Win32.MUPX.Gen	118
Suspicious	362
TrojWare.Win32	7,148
Virus.Win32.	876
Worm.Win32	7,580
TOTAL	108,016

F-secure

Malware Type	Counter
Backdoor.	363
Gen:Heur.Zygug.6	26
Gen:Malware.Heur.	46
Gen:Trojan.	166
Gen:Variant.	546
Generic.Malware.	225
Generic.Sdbot.	28
GenPack:Generic.Malware.	23
GenPack:Generic.Mydoom.	51
GenPack:Trojan.Inject.	28
IRC-Worm.Generic.20088	77
MemScan:Trojan.Generic.	53
Net-Worm:W32/	15,651
Suspicious:W32/Malware!Gemini	15
Trojan	2,989
Win32.Virut.M	282
Win32.Worm.Downadup.Gen	2,805
Worm:	67,963
TOTAL	91,337

McAfee	
Malware Type	Counter
Artemis!	3,981
BackDoor-	212
Downloader.a!qr	774
Dropper-FED!	30
Generic	619
New	12
PWS-Zbot.gen.PMz	876
Ransom-AAY.gen.l	60
RDN/Generic.bfg!a	1,081
RDN/Sdbot.worm!bp	178
RemAdm-ProcLaunch	30
Trojan-FDCW!8342B1216DCC	29
VBObfus.da	28
W32/Autorun.worm!qq	149
W32/Conficker.gen	30
W32/Conficker.worm	76,553
W32/HAMweq.worm	19,327
TOTAL	103,969

Microsoft	
Malware Type	Counter
Backdoor:Win32/	577
Exploit:Win32/	272
PWS:Win32/Zbot	31
Trojan:Win32/	2,040
VirTool:Win32/	514
Virus:Win32/Parite.B	2,862
Worm:Win32/Allaple.A	99,521
TOTAL	105,817

Symantec	
Malware Type	Counter
Infostealer	61
Packed.Generic	241
Spyware	118
Suspicious	335
Trojan	3,841
W32	99,013
TOTAL	103,609

Sophos	
Malware Type	Counter
Mal/	80,379
'Troj/Agent-AAXV'.	3,279
'W32/Allaple	22,009
or	153
TOTAL	105,820

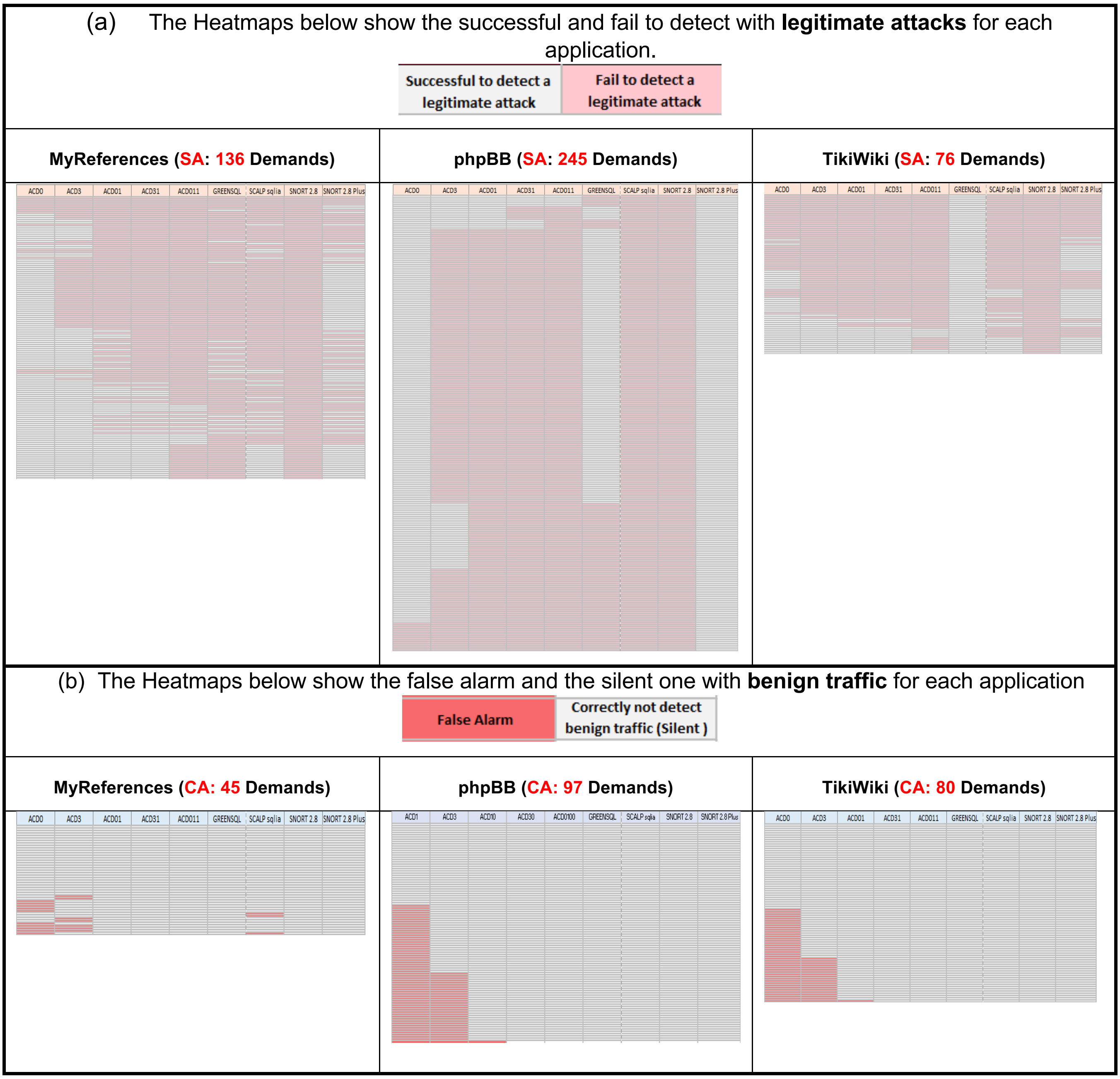
Kaspersky (VirusTotal version) - see footnote 18	
Malware Type	Counter
Backdoor.Win32	3,297
Email-Worm.Win32.Updater.n	180
HEUR:Trojan.Win32.Generic	993
HEUR:Worm.Win32.Generic	30
IM-Worm.Win32.Steckt.ae	30
Net-Worm.Win32.Agent.bk	94,704
Packed.Win32.Krap.hm	69
Trojan-Downloader.Win32.Agent.cuxe	5,458
Virus.Win32.Virut.n	1,359
Worm.Win32.AutoRun.edpn	214
TOTAL	106,334

Appendix B (Supporting Chapter 6 of the thesis)

This appendix provides further details to support the analysis provided in Chapter 6 of the thesis.

Appendix B-1 – Heatmaps

Heatmaps of detections and failures for successful attacks and benign traffic for Single IDS (the figure below show a detailed result of Table 6-2 in Chapter 6).



(a) The Heatmaps below show the successful and fail to detect with **legitimate attacks** for each application.

MyReferences (SA: 136 Demands)

phpBB (SA: 245 Demands)

TikiWiki (SA: 76 Demands)

(b) The Heatmaps below show the false alarm and the silent one with **benign traffic** for each application

MyReferences (CA: 45 Demands)

phpBB (CA: 97 Demands)

TikiWiki (CA: 80 Demands)

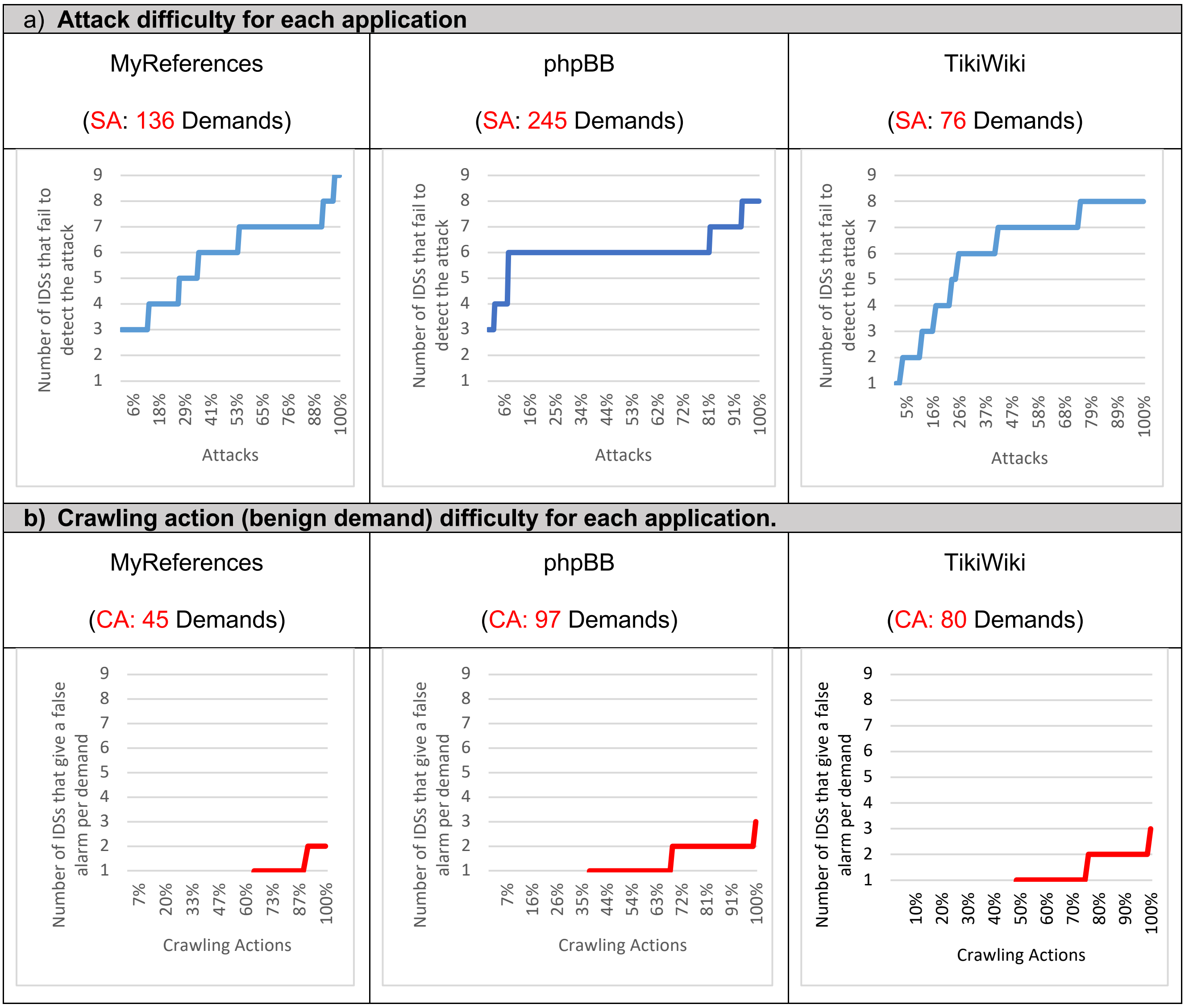
Appendix B. Figure 1 The heatmaps for successful attacks and benign traffic for each application

In Appendix B. Figure 1: (a) the first column shows the output result of the nine IDSs when subjected to attacks through My References web applications (the first row represents the name of the IDSs: ACD1,ACD3,ACD10, ACD30, ACD100, GreenSQL, SCALP scalia, SNORT 2 and SNORT 2.8 followed by the results of each IDS; the pink for failed to detect attack and the grey for successfully determining the attack). The second and the third table for the result of the same nine IDSs when subjected to attacks through phpBB web applications. Part (b) shows the result of the nine IDSs when subjected to benign traffic through My References (the first table in b) , then phpBB web application (the second table in b) , then TikiWiki (the third table in b), each cell in table represents the result of the IDS, the pink cells for falsely rising alarm and the grey for correctly determines that a benign input is not malicious.

Main observations from Appendix B. Figure 1: none of the IDSs in MyReferences web-application are able detect all the successful attacks. However, for phpBB web-application Snort plus special rule, manages to detect all the successful attacks, while in TikiWiki web-application GreenSQL mange to detect all the successful attacks. ACD1, ACD3 and Scalp Sqlia give false alarms in Myreferences. ACD1, ACD3 and ACD10 give false alarms in phpBB and TikiWiki.

Appendix B-2 Demands Difficulties

In relation to the sections 6.5 and 6.6 from Chapter 6: we present here the results of our analysis by the difficulty of the demands that the IDS flags as malicious or benign. The more IDSs fail to correctly classify a demand, the more “difficult” the demand is. Figure Appendix B. Figure 2 shows the difficulty for each type of demand and each application.



Appendix B. Figure 2 Demand difficulties of attack (a) and benign traffic (b) for each application.

Using the first graph from the left-side in **Appendix B. Figure 3 (a) as an illustration:** x-axis: ordered list of demands (from left to right: those that caused the least number of IDSs to fail to those that caused the most), y-axis: count of IDSs affected by a given demand. The y-axis is shown cumulatively. So we see that 54% of the demands caused 6 or less IDSs to fail.

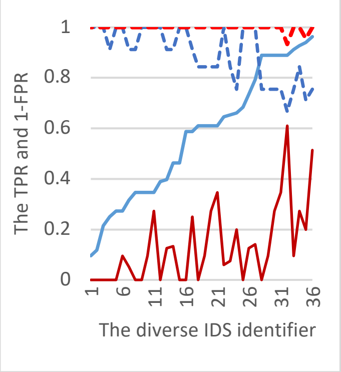
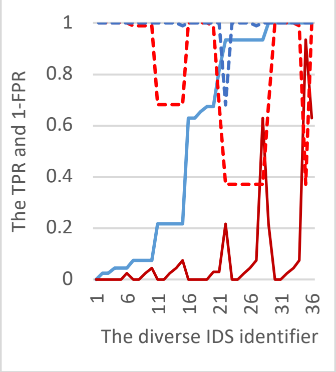
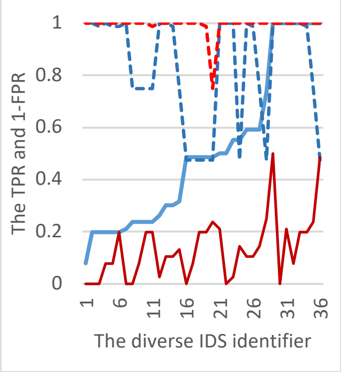
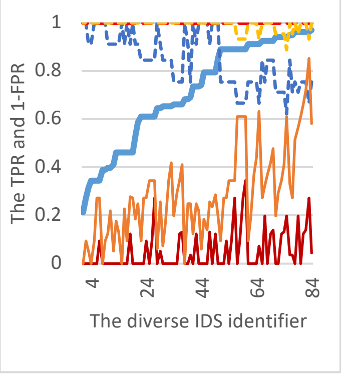
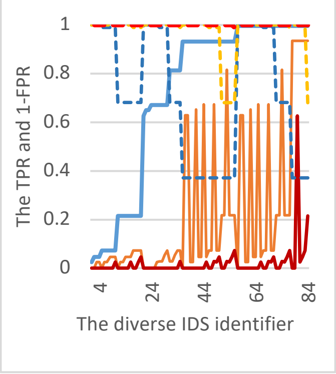
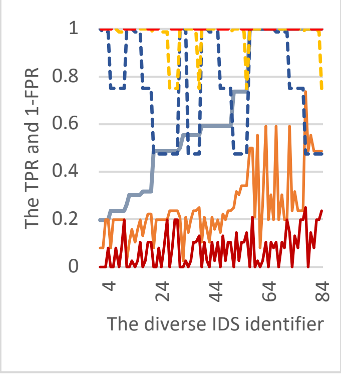
Main results from Appendix B. Figure 2

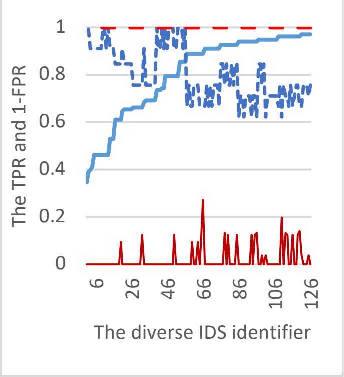
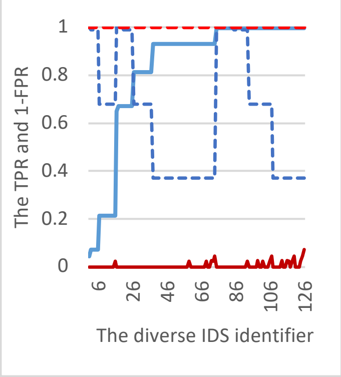
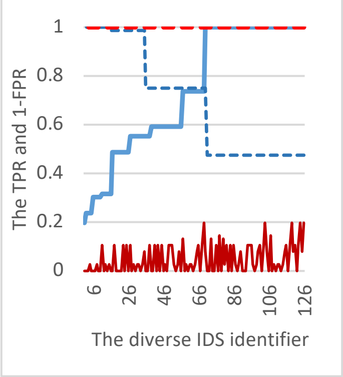
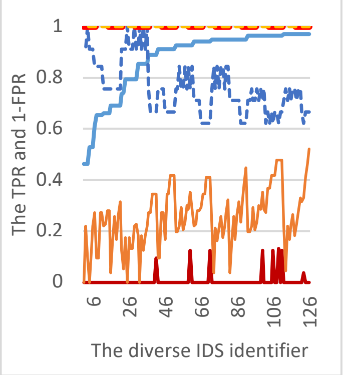
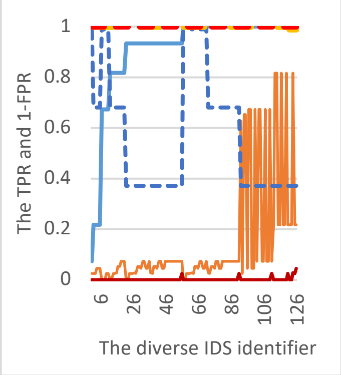
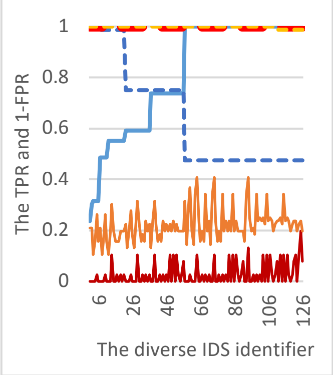
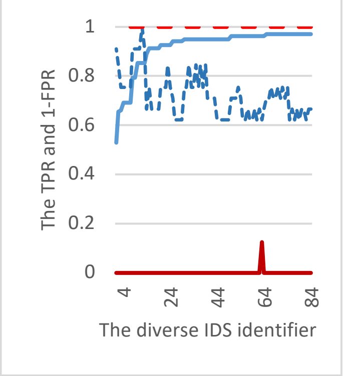
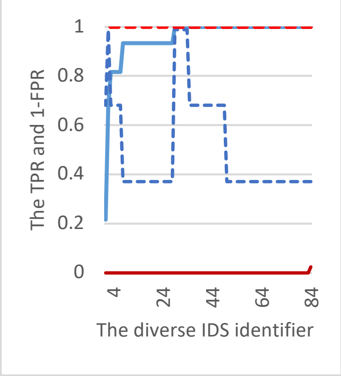
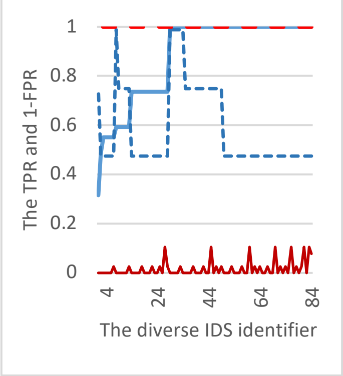
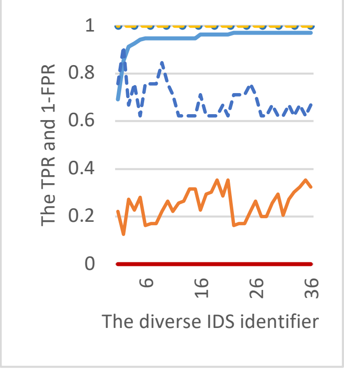
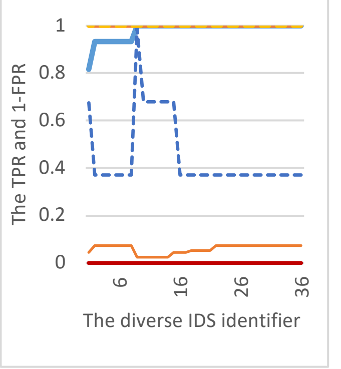
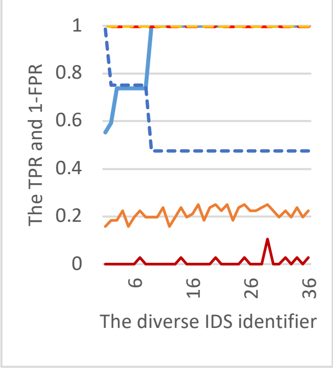
- There are 7 IDSs in Myreferences that fail to detect a large proportion of attacks. A small proportion of attacks are also “difficult” on all nine IDSs in MyReferences.
- In phpBB there are a large proportion of attacks not detected by 6 IDSs

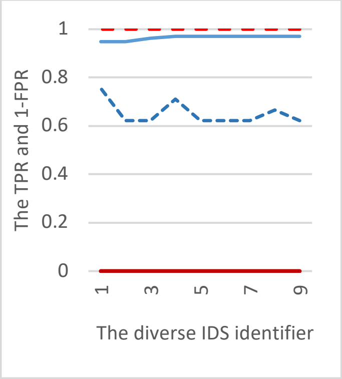
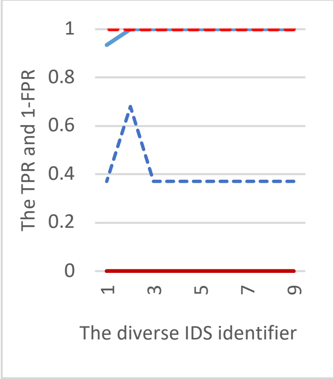
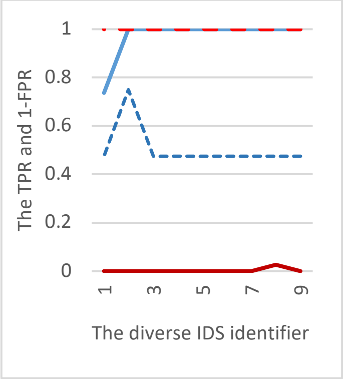
Appendix B-3: Diversity Analysis (Distribution of TPR and 1-FPR: ordered by TPR: further analysis for section 6.6 from Chapter 6)

The graphs below show the TPR and 1-FPR rates of each diverse systems for each adjudicator for N=2 to N=9.

TPR 1ooN — TPR RooN — TPR NooN
1-FPR 100N — 1-FPR RooN — 1-FRP NooN

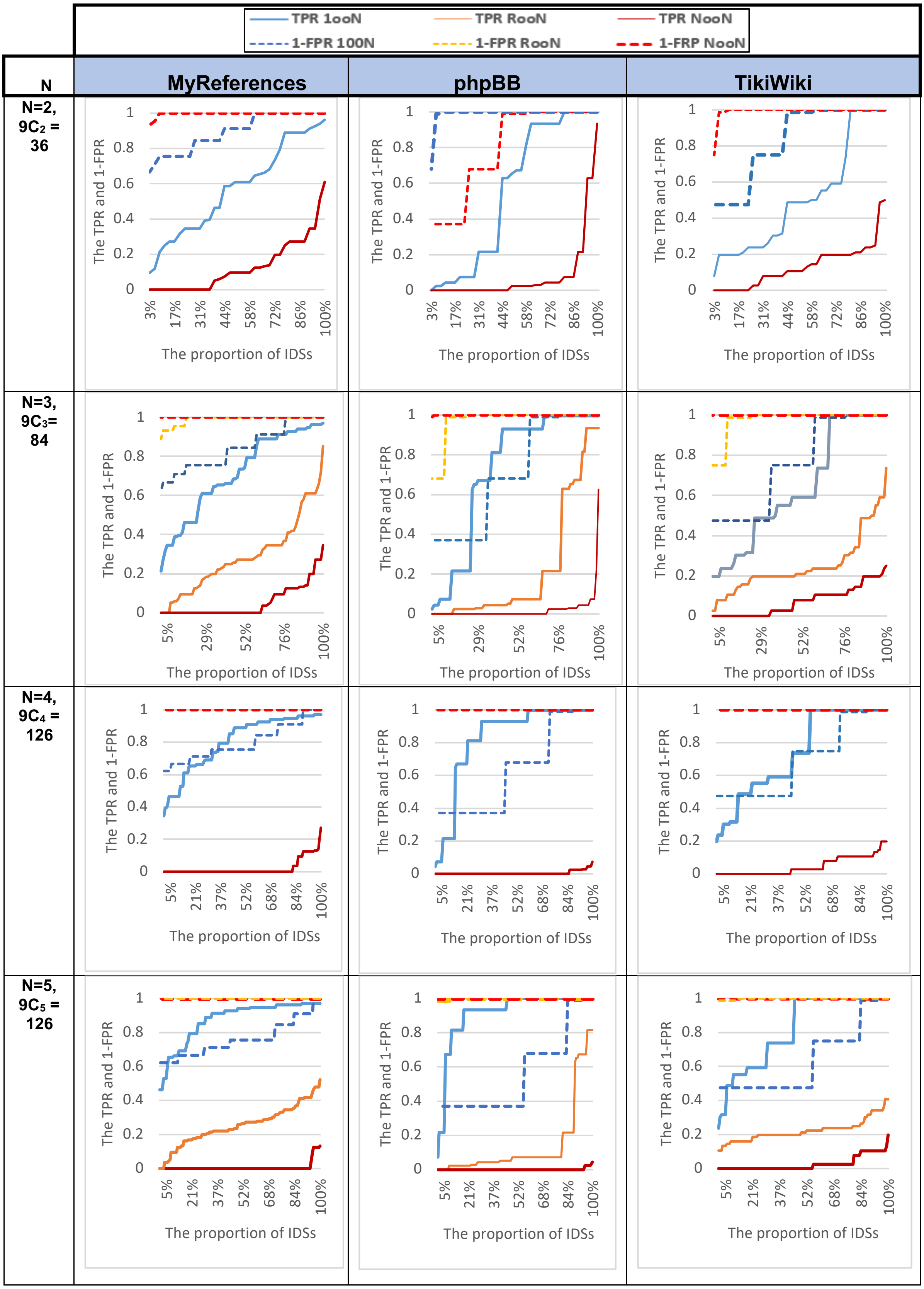
N	MyReferences	phpBB	TikiWiki
N=2, 9C$_2$ = 36			
N=3, 9C$_3$= 84			

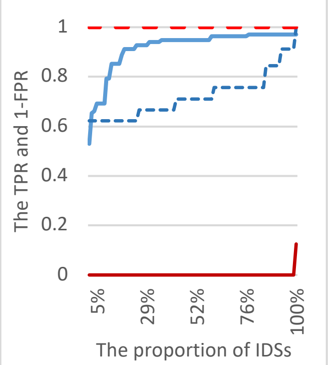
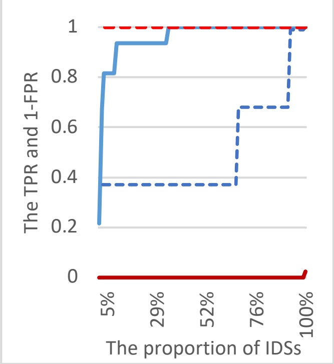
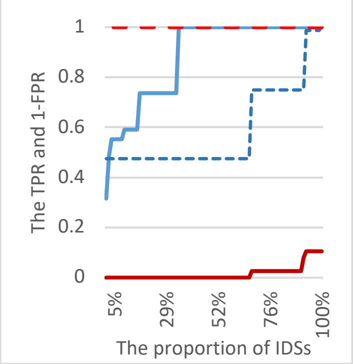
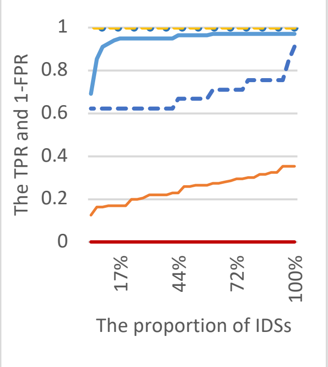
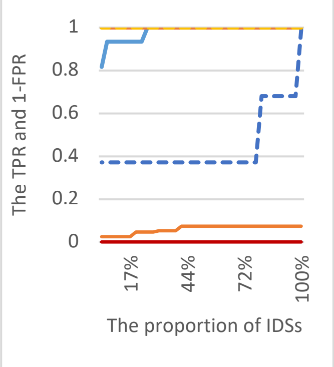
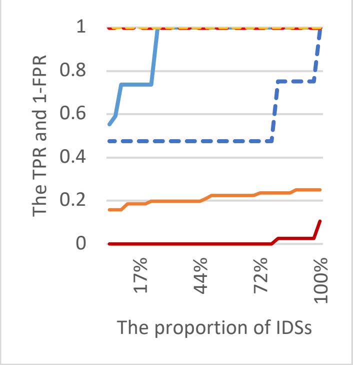
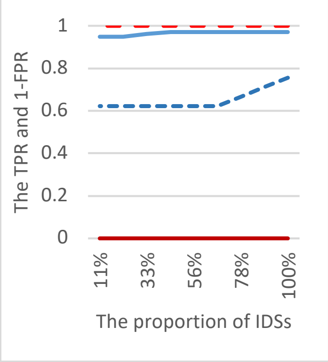
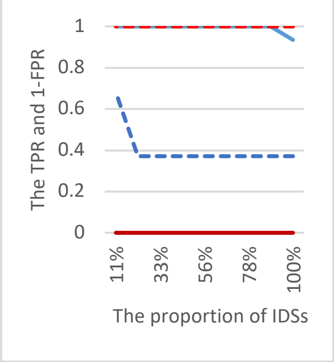
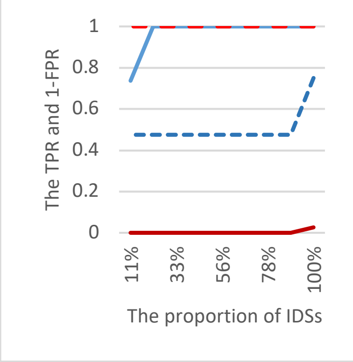
N=4, **$9C_4$ = 126**			
N=5, **$9C_5$ = 126**			
N=6, **$9C_6$ = 84**			
N=7, **$9C_7$ = 36**			

N=8, **$9C_8 = 9$**			

Appendix B. Figure 3 The TPR and 1-FPR Distribution Plots showing the combined IDSs, in 1ooN, RooN and NooN configurations. for each application ordered by TPR of 1ooN (so the x-value remains the same for each of the 6 lines on the plot). The x-axes represents the diverse IDSs for each configuration N.

The figure below shows, the same analysis as in the previous figure, but the order in the X-axis is no longer preserved (i.e. we show the ordering of each adjudicator on each measure independently).



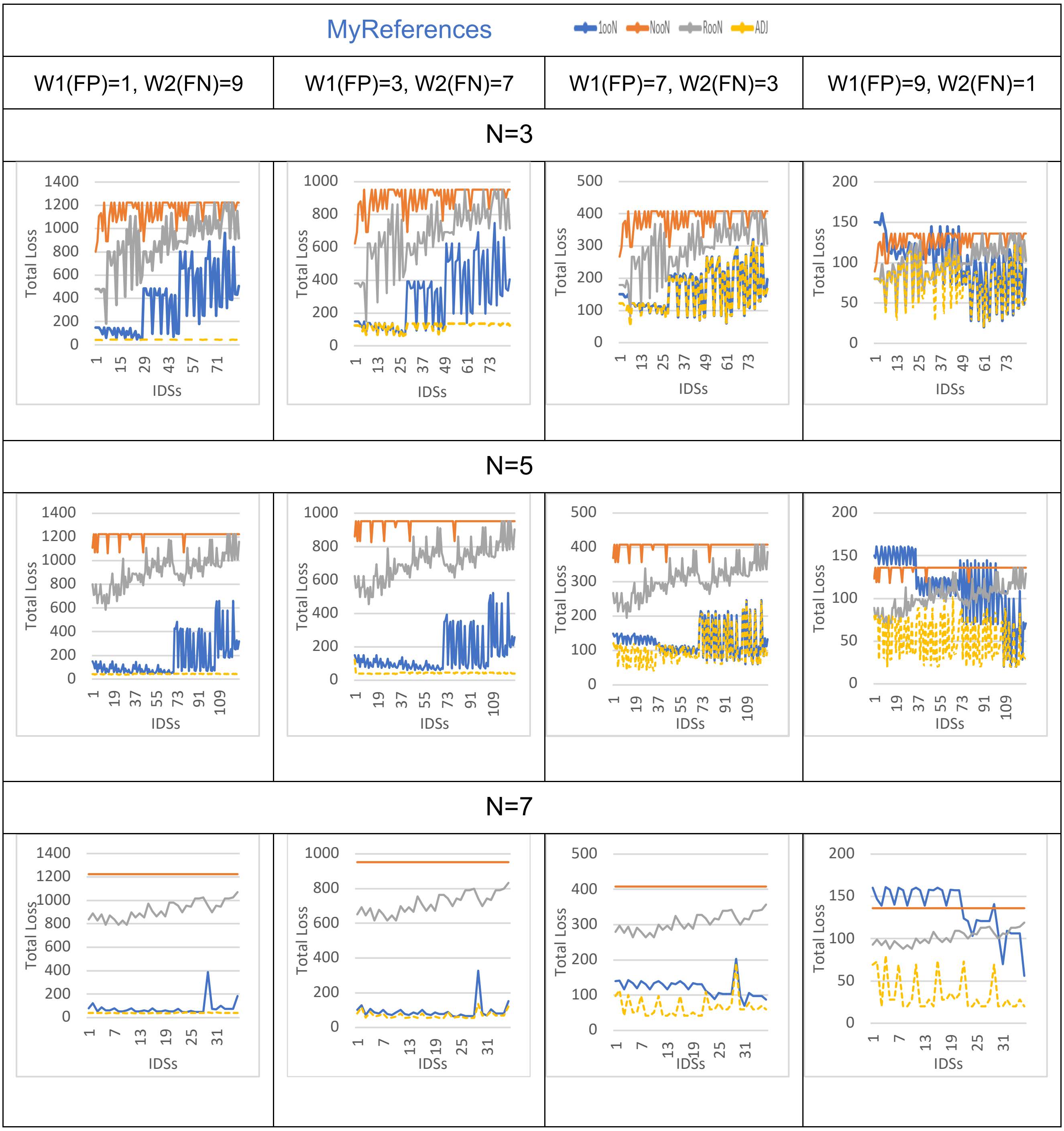
N=6, $9C_6$ = 84			
N=7, $9C_7$ = 36			
N=8, $9C_8$ = 9			

Appendix B .Figure 4 The TPR and 1-FPR Distribution Plots showing the combined IDSs, in 1ooN, RooN and NooN configurations. for each application ordered from low to high value. So each line is ordered from the best (left) to the worst (right), hence the actual value in the x-axis is not meaningful (though the proportion is).

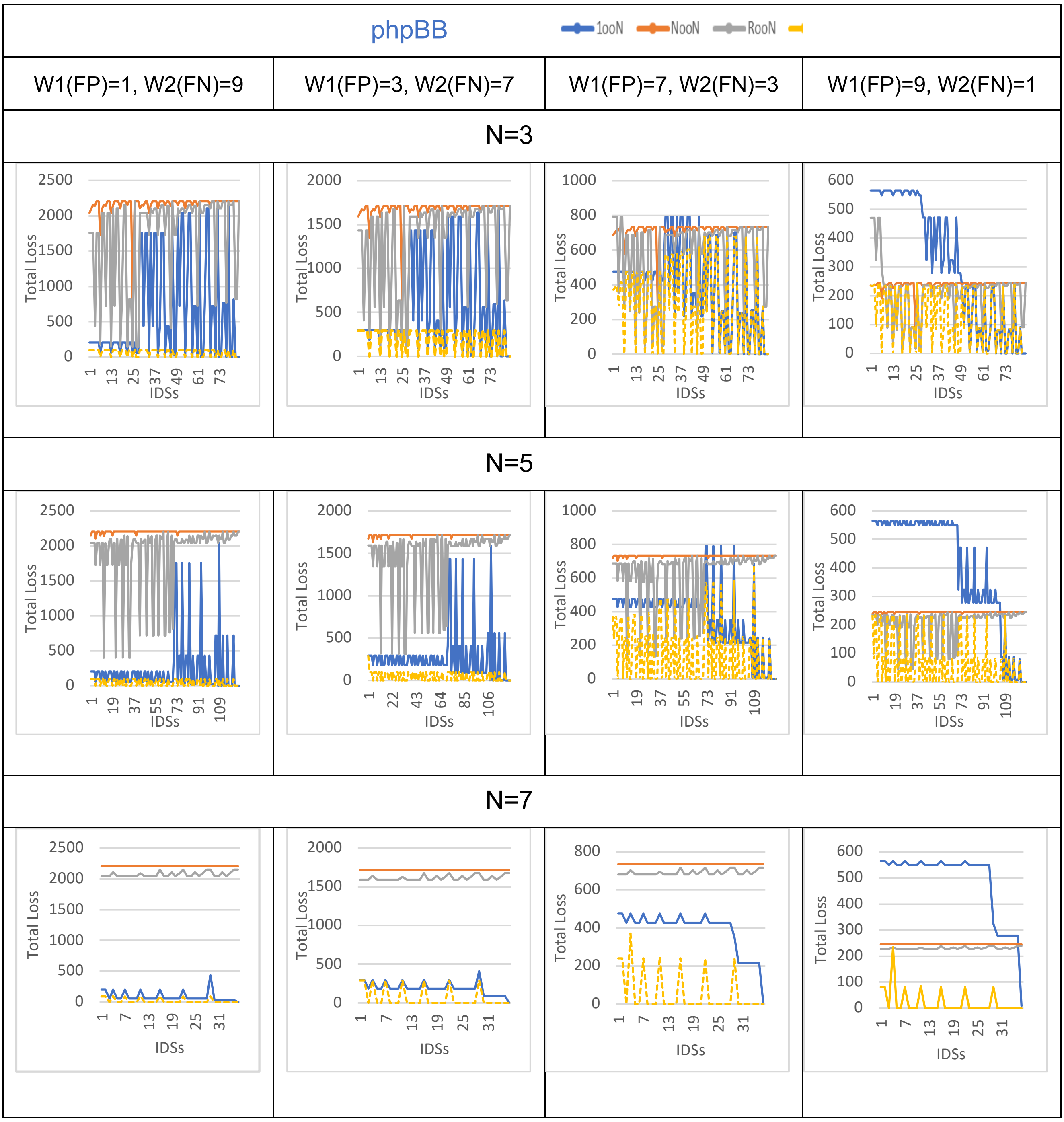
Appendix C (Supporting Chapter 8 of the thesis)

This appendix provides further details to support the analysis provided in Chapter 8 of the thesis.

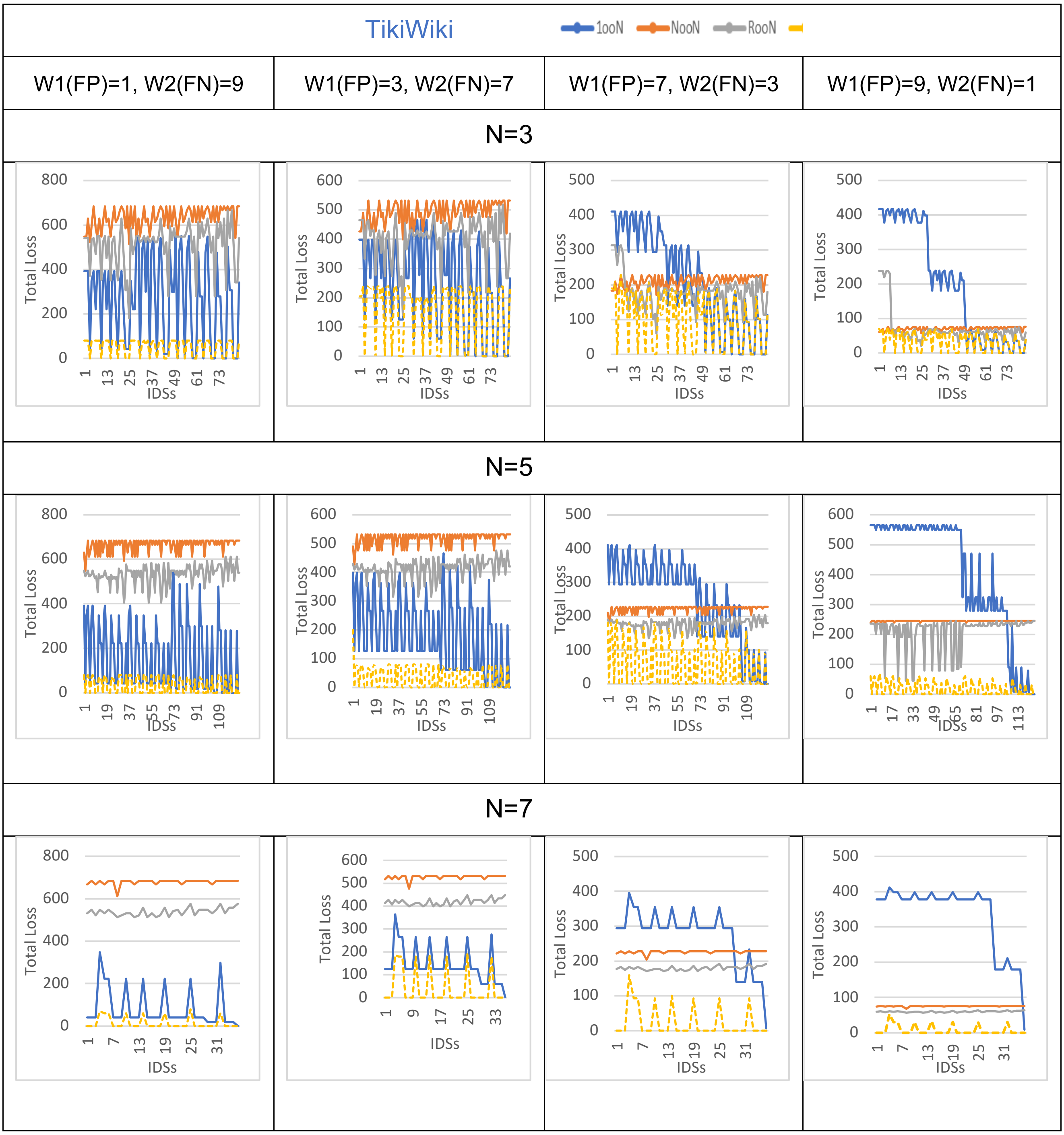
Appendix C-Figures 1, 2 and 3 (for IDSs), 4 and 5 (for SATs) are the loss values associated with the all diverse systems from which the averages in Figure 8-3 (for IDS) and 8-5 (for SATs) in Chapter 8 of the thesis are derived.



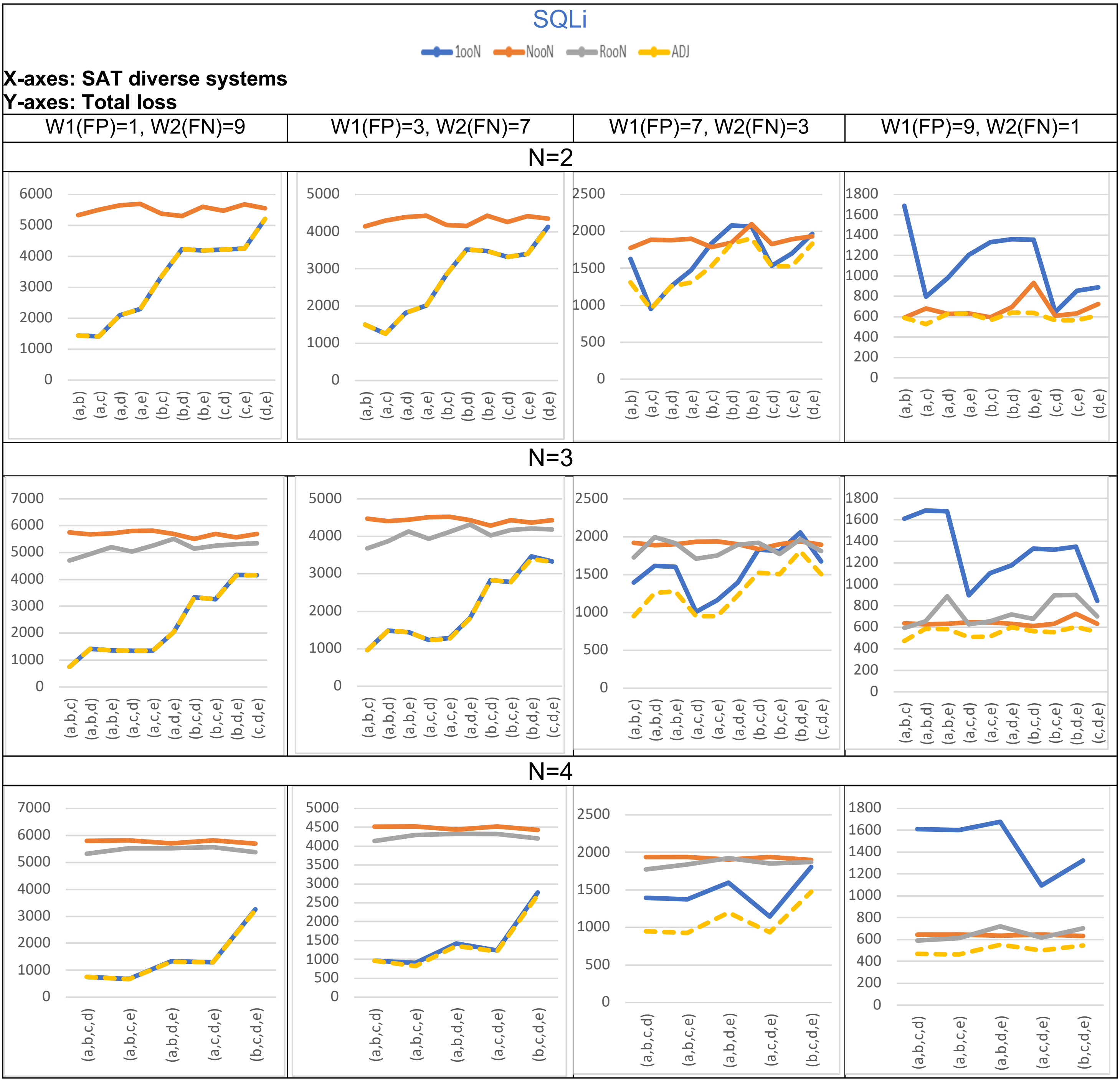
Appendix C-Figure 1 The weighted total loss (FP losses and FN losses) per diverse system configurations (1ooN, majority vote, NooN and optimal adjudication) for (MyReferences, N=3, 5 and 7)



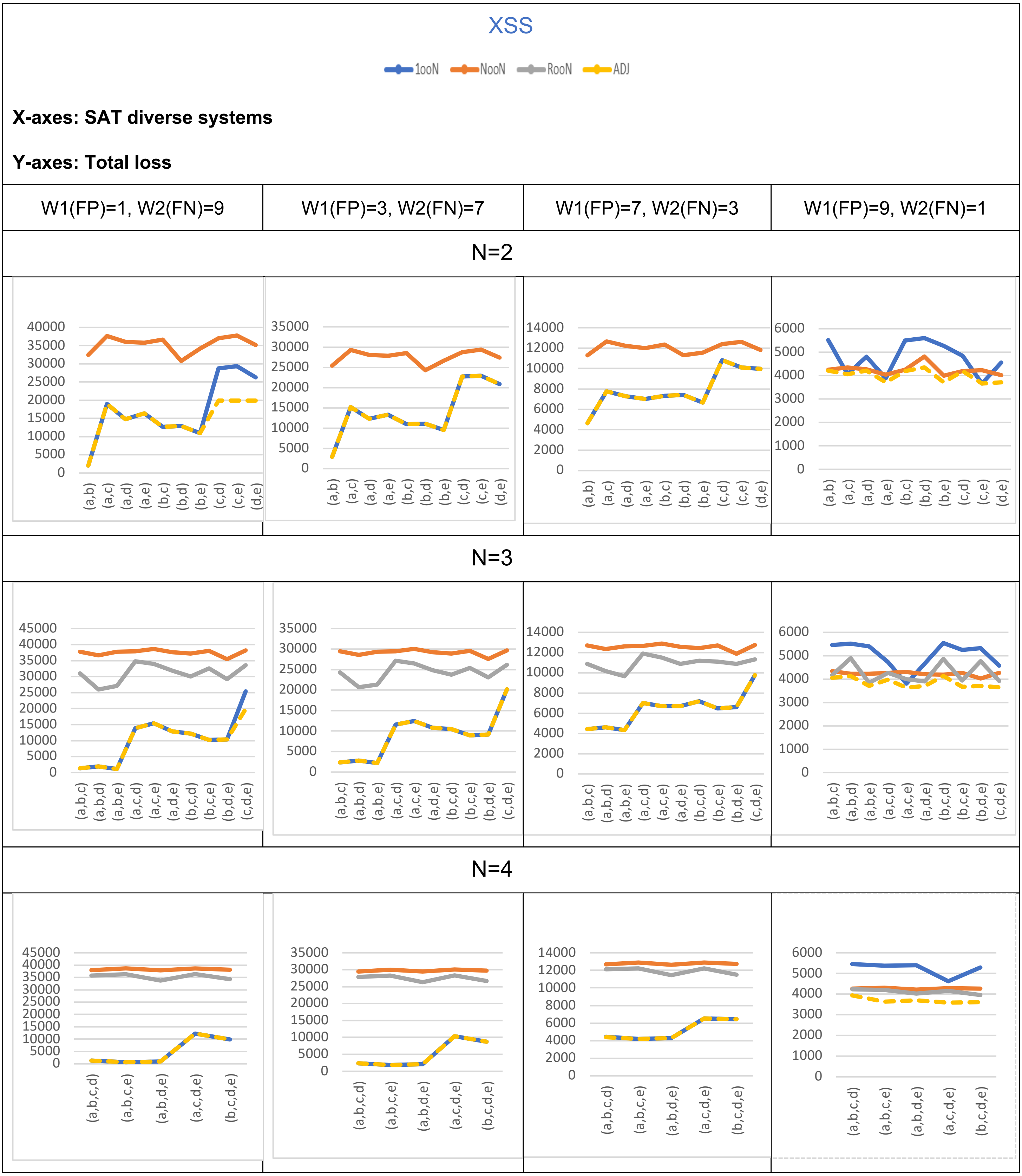
Appendix C-Figure 2 The weighted total loss (FP losses and FN losses) per diverse system configurations (1ooN, majority vote, NooN and optimal adjudication) for (phpBB, N=3, 5 and 7)



Appendix C-Figure 3 The weighted total loss (FP losses and FN losses) per diverse system configurations (1ooN, majority vote, NooN and optimal adjudication) for (TikiWiki, N=3, 5 and 7)



Appendix C-Figure 4 The weighted total loss (FP losses and FN losses) per diverse system configurations (1ooN, majority vote, NooN and optimal adjudication) for (SQLi, N=2, 3 and 4)



Appendix C-Figure 5 The weighted total loss (FP losses and FN losses) per diverse system configurations (1ooN, majority vote, NooN and optimal adjudication) for (XSS, N=2, 3 and 4)