CYBER DIVERSITY INDEX

FOR

AUTONOMOUS ANOMALY MANAGEMENT

Michael Donevski

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Information Technology

Faculty of Business, Justice and Behavioural Sciences

School of Computing and Mathematics

Charles Sturt University, Australia

March 2020

Certificate of Authorship

I, Michael Donevski hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent as been accepted for the award of any other degree at Charles Sturt University or any other educational institution, except where due acknowledgement is made in the thesis.

Any contribution made to the research by colleagues with whom I have worked at Charles Sturt University or elsewhere during my candidature is fully acknowledged.

I agree that this thesis be accessible for the purpose of study and research in accordance with the normal conditions established by the Executive Director, Division of Library Services or nominee, for the care, loan and reproduction of theses.

Name Michael Donevski

Date 23/3/2020

Abstract

To effectively use Machine Learning (ML) in cybersecurity, automation is required. However, there cannot be true Intelligent Automation in cybersecurity if anomalies are managed manually, as is currently the case. As exponential technologies advance towards the widespread adoption of Internet of Things, Cyber Physical Systems, Internet+, Industry 4.0, and Society 5.0 there will be an influx of new cyber anomalies triggered by cyber attacks.

Detecting and managing these anomalies in complex Information and Operational Technology environments will require advanced autonomous anomaly management. This research defines cybersecurity preliminaries, stimulates questions to expose the complexities and identifies twelve cybersecurity automation challenges. One of these is that cybersecurity controls rely heavily on databases that require manual management.

First, a normal system state needs to be defined with any anomaly detection research, and anything outside of this state is considered to be an anomaly. Unfortunately, the normal system state or baseline is defined during the creation of these databases and datasets. Attempts are made by subject matter experts to keep these baselines operationally relevant with ad-hoc updates as part of their maintenance.

Anomaly detection is traditionally performed by querying, comparing and identifying an anomalous string from a database containing human readable entries, such as Indicators of Compromise, firewall rules, Intrusion Detection System signatures, application whitelisting, vulnerability management, and Anti-Virus rules.

Cybersecurity analysts manage anomalies using traditional as well as ML-infused security solutions, with database updates and manual data entries.

In this research anomaly is identified not as an anomalous entry in a database, but as an “unknown change” in concept of host or system combined with unique and incoherent applications' life-cycles. The application's integrity is monitored with a graph-based and real time measurable solution. A flexible application behavioural profile is developed as the base for system normality, that is dynamic. This is done by measuring change using entropy based anomaly detection.

The Biodiversity Index compares diverse relationships between species and communities in ecology, is used as the research contribution to compare diverse dynamic relationships between business applications that drive emerging technologies. Continuous sampling is used to track changes in application diversity, which contributes to both the applications' unique behavioural profiles for the novel Cyber Diversity Index (CDI), and dynamic normal baselines for the proposed theoretical autonomous anomaly management.

The CDI, is used to measure application system calls in real time and to monitor and detect unknown diversity changes, during normal operational environment and when placed under a zero-day cyber attacks. The purpose of CDI is to improve automation in cybersecurity by providing signature-less, self-managed and real time solution. In addition, CDI could provide a measurable solution, as to the level of cyber integration in a synthesised cyber ecological environment.

The research reports modest results from the experiments, however additional data points, different diversity indexes, and different types of entropy, could contribute towards future research.

Dedication

I am dedicating this dissertation to my mother, my wife, my kids and my father.

Acknowledgements

I would like to acknowledge and sincerely thank my supervisor, Dr Tanveer Zia, for his guidance and support throughout this long journey.

I would also like to thank Dr Peter White, Dr Sabih-Ur Rehman and Dr Kenneth Li-Minn Ang for their support during my research, and Dr Oliver Burmeister for his ethical and IT professionalism guidance. Sincere thanks to Dr Zahid Islam for his guidance and support in the field of data science. I would like to thank the examiners for their insightful comments and feedback.

I would like to acknowledge the ongoing study support from the Australian Commonwealth Government. I would also like to thank Proofed for their professional proofreading service.

My most sincere thanks to my wife for putting up with my 'anomalies'. Without her, this research would have not been possible. I would like to thank my kids for being patient and putting up with Dad's occasional grumpiness, my father for his lifetime support and my parents-in-law for always being super supportive.

List of Publications

- Donevski M., and Zia, T. (2018). A Survey of Anomaly and Automation from a Cybersecurity Perspective. 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, United Arab Emirates, 2018.

- Donevski M., and Zia, T. (2020). The dirty dozen of automation and cybersecurity: A survey. The Journal of Supercomputing. Paper submitted.

- Donevski M., and Zia, T. (2020). Cyber Diversity Index for sustainable self-control of machines. Cybernetics and Systems: An International Journal. Paper to be submitted.

List of Figures

List of Tables

Table of Contents

Chapter 1

1. Introduction

1.1 Current situation

The cybersecurity industry is in desperate need of automation. Machine Learning (ML) has attracted an unprecedented level of interest in cybersecurity research. Anomaly detection is a popular data science topic related to cybersecurity, there has been little research to solve a more complex cybersecurity problem, the automation of cyber anomaly management. This research has holistically identified anomalies generated by software, programs, and applications developed to support emerging technologies an area with a significant impact on cybersecurity research, and wide applicability to the business and manufacturing industry. It appears that adaptable and affordable solutions are not being developed fast enough to improve poor secure coding practices, inadequate software vulnerability management, fragmented operational environments, rigid security controls, poor organisational support for cybersecurity recommendations, and the shortage of skilled cybersecurity professionals. In the meantime, autonomous anomaly management solutions could provide some relief to cybersecurity professionals. Anomaly research in data science relates to data, and ML research consists of well-defined processes, tools, and progressive algorithms. Reputable anomaly based ML research is highly dependent on high-quality data and high-quality resulting models, whereas in cybersecurity, anomalies relates to data, systems, people, and processes. Therefore, the importance of digitising or correctly translating these dynamic business processes to relevant datasets is crucial for the effectiveness of the ML models, when transferring these to production environments.

Research conducted more than 20 years ago by Forrest, Hofmeyr, Somayaji, and Longstaff (1996) introduced the concept of Artificial Immune Systems (AIS) which set the path for many research papers that attempted to solve the complexities of autonomous systems by proposing a self-healing computer system that mimics the human immune system.

This research attracted a great deal of interest from data scientists who are improving algorithms and may one day discover the master algorithm (Domingos, 2018). Research that was undertaken at the University of New Mexico in conjunction with Carnegie-Mellon University led to the development of new advanced ML-infused cybersecurity solutions, such as the Enterprise Immune System (EIS) and Industrial Immune System (IIS) (Darktrace, 2018).

While anomaly based research is important for intrusion detection, it appears the major focus is on the development of algorithms and datasets (Buczak and Guven, 2016) rather than on the strategic management of the system normality and management of anomalies. Burgess, Haugerud, Straumsnes, and Reitan (2002) identified this problem and presented system normality to define, and measure this dynamic operational state from a system administration point of view. While both research groups identified the problem and the need for autonomous solutions, it appears there has been modest progress, perhaps due to the problem's complexity.

1.2 Preliminaries

1.2.1 Cybernetics and cybersecurity

Norbert Wiener (1961) coined the term, cybernetics, in 1948 in relation to control and communication in the animal and the machine describing self-regulating machines that use closed-loop feedback. Cybersecurity thus relates to continuous assurance of the control and secure communication of a human with self-regulating machine and further machine-to-machine (M2M) interactions. The term cybersecurity, a single word, is used throughout this paper, as defined by the Oxford Dictionary (2019).

1.2.2 Machine Learning

Machine learning is a sub-field of Artificial Intelligence (AI) that helps systems learn, without being explicitly programmed. Machine learning science automates discovery, and is also known as: pattern recognition, data mining, and knowledge discovery (Domingos, 2018).

Samuel (1959), coined the definition machine learning, in his research paper on computer game of checkers. The process of programming of a digital computer, needs to behave the same way a human learning process does. He notes, “computer can be programmed so that it will learn to play a better game of checkers, than can be played by the person who wrote the program” (Samuel, 1959, p. 1).

1.2.3 Zero day attacks

Unknown cyber attacks are also known as zero-day attacks, they target unknown or publicly undisclosed software vulnerabilities, the term refers to how many days the software vendor has known about the vulnerability (Ablon & Bogart, 2017). The effectiveness of a zero-day attack, is the lack of fix, patch or control in place, to defend against this attack. However, having systems with unpatched software and security controls that have not been updated for long time, inadvertently places the defender to defend against unknown attacks to the defender, which have the same impact of a zero-day attack. Additionally, when new patches are provided by software vendors, attackers can potentially reverse engineer the patch, identify exploitable vulnerabilities and develop an one-day exploit (Paganini, 2019). For the purpose of term consistency throughout this paper, the term zero-day, due to its impact, will be used, covering cyber attacks that successfully exploit vulnerabilities. The only plausible defensive solution against these attacks would be to use an anomaly based security control.

1.2.4 Anomalies

While the accepted definition of an anomaly is something that deviates from what is standard, normal or expected, in production environments, an anomaly is associated with an event of a change that is not truly understood.

Seeing the anomaly however is just the beginning of understanding the true normality of that environment. Dorothy E. Denning (1987) was the first to propose the detection of cyber attacks in intrusion detection expert systems by looking at anomaly records.

- The first most challenging aspect of anomaly based research is determining and defining the normal baseline in a changing and dynamic cyber environment (Burgess, Haugerud, Straumsnes, & Reitan, 2002).
- The second most challenging aspect is gaining or increasing the visibility of the anomalies and tremendous efforts have been made towards anomaly detection research, in which data scientists are working to improve ML models on given cybersecurity datasets. The challenge is to positively classify new malicious anomalies, this is highly dependent on input provided by cybersecurity experts (Dietterich, 2018).
- Would the improvement of ML models increase the number of detected anomalies and related alerts, so the demand for cybersecurity professionals could increase in the future and not decrease as expected with the usage of ML? This question leads us to the third challenge, autonomous anomaly management, a self-regulating system that manages its own anomalies and mitigates against the ones that were generated by cyber attacks.

1.2.5 System calls

A kernel is the core to any Operating System (OS). The kernel sits between applications and resources, such as Central Processing Unit (CPU), memory and devices. Applications and programs have lower privilege and make system calls to the kernel to get access to those resources. Applications cannot make direct calls and their requests must be processed by crossing the user-space and kernel boundary (IBM, 2019).

1.2.6 Entropy

A measure of uncertainty in a given system, has been identified by number of research papers as the solution for identifying anomalies, the entropy concept in information theory was first introduced by Claude Shannon (1948). Entropy is a concept in information theory that has also been applied to measure the diversity of collected data.

1.2.7 Biodiversity Index

The biodiversity index or diversity index, also known as the phylogenetic index is a scale or quantitative measure used in ecology to determine the diversity of species from a given sample in a community. The two most used indexes are the Simpson Diversity, and Shannon-Wiener Index which is based on the Shannon Entropy.

1.2.8 Cyber Ecology

Cyber ecology is a term that is used when ecological, immunity concepts, theories and methods are applied to cybersecurity research. Biological solutions that have been identified in nature are used to solve real complex cybersecurity problems and challenges.

1.3 Research question

The main research question is:

How can anomalies triggered by zero-day attacks be autonomously managed?

Sub-questions :

- What prevents the automation of anomaly management?
- How ML automates cybersecurity research and production environments?

The main hypothesis is that if a system is under a targeted zero-day attack

then it should differentiate from a system that is not, the anomalies generated by the vulnerable application could provide the visibility of a probable active system integrity exploitation and real time discovery of micro Indicators of Exploitation (mIoX) irrespective of the type of attack or targeted vulnerability. The system under attack should have a higher level of uncertainty, diversification or system entropy. Three supporting assumptions go towards this hypothesis:

- Hacking, black-box testing and fuzzing are messy.
- The defender knows the system normality better than the attacker.
- The attacker will have to change the core purpose of the victim system for their own gain.

In order to answer the main research question and find a solution to autonomously mitigate zero-day attacks, first we need to redefine what cyber anomalies are, and the challenges associated with possibly managing these autonomously, which is explored in the literature review. The second sub-question is related to the effectiveness, and the level of automation ML provides to cybersecurity. While the experiment in chapter 3 does not test any state of the art algorithms, it attempts to test AutoML against classifiers on a popular cybersecurity dataset. By running this standard experiment in Weka, not only do we show the effectiveness of the AutoML solution, but we also expose the manual processes that ML introduces to cybersecurity. Furthermore, the experiment exposes one of the automation challenges that is identified from the literature review, the manual creation and processing of ML datasets and models.

1.4 Research aim and key objectives

The aim of this research is to produce a dynamic and measurable system profile to efficiently manage application anomalies, to provide more granular solution in discovering of unknown cyber attacks and to simplify the processes required to maintain these controls autonomously and effectively in the future. The key objectives of this research are:

- To expose the limitations of existing security controls and identify the areas for improvement.
- To address the key complexities associated with automation, learning systems and cybersecurity.
- To examine the performance of security tools in relation to new technologies and look for continual improvements.
- To design a process that will improve visibility of advanced unknown cyber attacks by improving the understanding of system normality.
- To exploit different processes, technologies, tools and solutions in order to create an efficient, dynamic and autonomous security controls.

1.5 Research motivation

The main aim of chapter 2 is to identify automation challenges with cybersecurity learning systems and review research undertaken in these domains. Current overview of automation in the cybersecurity industry, and the major complexities are presented as the baseline for identifying the automation challenges in cybersecurity industry and research. Comprehensive but non-exhaustive list of challenges was developed to assist in development of a novel solution that fully or partially addresses these in this research. The motivation of this review was to identify an automation solution with high impact and wide industry applicability. Verification of integrity and anomaly management for business applications at run-time that drive exponential technologies, were identified as the most critical aspect of future zero-day attacks.

The main aim of chapter 3 is to test the level of automation, ML provides to the cybersecurity domain. One challenge that cybersecurity experts face is the selection of the best learner for a given cybersecurity dataset, which is due to their limited knowledge of data analytics. The effectiveness of AutoML with AutoWeka (Kotthoff, Thornton, Hoos, Hutter, & Leyton-Brown, 2017) and the automatic selection of learners was tested. The motivation for this experiment was to expose the manual aspects of commonly used ML practices and tools. Attempt is made to answer the second sub-question and identify if ML solutions in cybersecurity improve or hinder automation in cybersecurity.

The main aim of chapter 4 is to test entropy based solutions in on order to identify a unique application behavioural fingerprints needed for dynamic system normality, without using ML datasets. The motivation is to use Shannon Entropy and the most popular Biodiversity Indexes: Shannon Wiener Index and Simpson Index to measure the Application Integrity Diversification (AID).

The main aim of chapter 5 is to identify micro Indicators of Exploitation (mIoX) or non-human readable indicators, that are related to system calls, black-box analysis and entropic anomalies. The motivation is to develop a diversity measurable solution when applications are placed under a zero-day attack.

The main aim of chapter 6 is to develop a self-populating index, Cyber Diversity Index (CDI). The motivation is to monitor and verify the integrity of run-time business application.

The main aim of chapter 7 is to propose a future research for autonomous anomaly management and the motivation is to solve the automation challenge that relate to both cybersecurity and learning systems.

1.6 Research contribution

In Chapter 2, the main contributions are:
Review and background of research and industry automation challenges in cybersecurity. Compilation of a comprehensive but non-exhaustive list of automation challenges in cybersecurity using learning systems. The following has been addressed in more detail:

- The shortage of Subject Matter Experts (SME);
- The fragmentation issues, related to the Defense in Depth model (Small, 2019);
- The manual database management of Indicator of Compromise(IoC);
- The manual, slow and expensive creation of ML datasets;
- Risk management is a complex challenge and a manual process in cybersecurity;
- Trust assignment is a manual cybersecurity process and Zero Trust model and Adversarial Machine Learning (AML) introduce additional complexities.

- Technology diversity and software diversification increase resilience by introducing uncertainty to the attacker, but they also introduce complexities and hinder automation.
- Isolation, mitigation strategy for applications and data, restricts access required for provision of autonomous solutions.
- The challenge of open loop systems, most anomaly based security systems, is the lack of the feedback loop required for automation.
- The challenge of logs, with inconsistencies in semantics and reliability, could introduce gaps and blind spots that could prevent automation. Monitoring of system calls for anomalies could provide better suited solutions for autonomous systems.
- Unidentified hosts are an automation challenge, because they lack a defined scope. Business and industrial applications give the host unique main purpose, which introduces complexities for management of unknown vulnerabilities.
- Anomaly detection and management is a manual process that closely relates to the Confidentiality, Integrity, and Availability of data, applications and people. Anomaly in application behaviour has been identified as an unknown change and not as an anomalous string in a database.
- Review of anomalies and ML research in cybersecurity
- Review of entropy based anomaly detection in cybersecurity.
- Cybersecurity Triple Triad as an addition to the CIA triad for identifying source of cyber anomalies and presenting the challenge of application integrity.
- Current application and host relationships when compared to biological symbiotic relationships.

The literature review identifies number of automation challenges, but the anomaly management as the most difficult, the literature gap, and the focus of this research. It appears when anomalies get detected by IDS and learning systems, the process stops in order for the SME to determine if the unknown change was malicious or not. The major source of the anomalies was identified as the business and industrial applications driving exponential technologies. The integrity of running applications, monitoring, verifying and managing anomalies autonomously, is the industry gap.

In Chapter 3, the main contributions are:

- The AutoWEKA (Kotthoff, Thornton, Hoos, Hutter, & Leyton-Brown, 2017)

which is part of the AutoML framework was tested and used to automatically select best learners and hyperparameters for the most commonly used cybersecurity dataset NSL-KDD (Tavallaee, Bagheri, Lu, & Ghorbani, 2009).

- AutoWEKA results were compared to learners such as: Naive Bayes (NB), BayesNet, Decision Tree J48, and two classifiers developed at Charles Sturt University, SysFor (Islam & Giggins, 2011) and CSForest (Siers & Islam, 2015) that are available in Weka (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009).
- The level of automation of AutoWeka is evaluated, and proposal was conducted for an improved solution.

In Chapter 4, the main contributions are:

- AID was proposed, to monitor diversity changes of system calls and to verify the applications' running integrity with real time data collection and graph-based visualisation.
- New application performance engineering techniques, dynamic tracing was used, instead of the traditional, offline, strace tool, to collect information from system calls.
- Bftrace (Robertson, 2019) that uses the extended Berkeley Packet Filter (eBPF) (Kozina, 2020) was used to collect and monitor system calls in real time using awk and quick one-liner scripts.
- Prometheus (Prometheus, 2019) and Grafana (Grafana, 2019) are used to present the entropy and the diversity index results of the real time system calls tracing.

In Chapter 5, the main contributions are:

- Continuous sampling, for real time entropy based measurements were taken of the exploited applications.
- Monitoring for unknown changes to AID, and to discover mIoX, caused by a hypothetical cyber parasite that propagates via web servers to targeted host web clients.
- Risk threat profiles were applied as used in (Bernaschi, Gabrielli, & Mancini, 2000) as part of a risk assessment using AID.

In Chapter 6, the main contributions are:

- Inspired by research undertaken in ecological approach, to evaluate cybersecurity by measuring software diversity, lead to the creation of the novel Cyber Diversity Index (CDI).
- Cyber Diversity Index is a self-populating index that contains system's applications' integrity diversification measurements, that are based on the diversity indexes.
- Usage of CDI in synthesised cyber ecology environment.

In Chapter 7, the main contributions are:

- Future research, a combined solution using CDI, Reinforcement Learning, and Moving Target Defences (MTD) are proposed, for the theoretical Autonomous Anomaly Management solution.

1.7 Research significance

- This research presents a comprehensive, but non-exhaustive list of automation challenges facing cybersecurity and learning systems.
- Verifying running integrity of dynamic business and industrial applications, have been identified with the highest significance for cybersecurity.
- A novel solution was developed to monitor unknown change, in running applications, based on the application's entropy and diversity index, Application Integrity Diversification (AID) behavioural profiles are created.
- This research presents a novel Cyber Diversity Index, a self- populating Index that address the challenges in this research, and also provides the link between automation, cybersecurity and learning systems.
- Novel, micro Indicators of Exploitation (mIoX) are presented, for detecting anomalies related to diversification, entropy, black-box analysis, exploitation and continuous sampling or real time analysis.
- Cybersecurity Triple Triad was presented to identify the source of the anomalies and also to present the importance and subjective nature of system integrity.
- Predefined concepts of anomaly detection are challenged, research looking for anomalies as human-readable string in a database or packets, are replaced with “unknown change” as the anomaly.

1.7.1 Practical usage

- AID, could assist with collection of local cyber intelligence.
- AID, could assist with provision of application integrity assurance, during application whitelisting and patching cycles (ASD, 2019).
- Real-time monitoring solution for critical business application.
- Application entropy based life-cycle profiling.
- Detecting rogue application or good apps behaving badly.
- CDI could assist with detection of Shadow IT.

1.8 Thesis Structure

This research will initially take an organisational approach, to identify the cybersecurity automation challenges, and then drill down to the area of significant concern and urgency, anomaly management. The main source of the anomalies has been identified as the business applications that drive emerging technologies. Entropy based anomaly detection has been selected as the quantitative measure for the experiments, to identify the application's behavioural profile and cyber diversity footprint.
Future research has been proposed by up-scaling this research to a larger diverse environment for the implementation of the Cyber Diversity Index in a cyber ecology environment.

The preliminaries, the hypothesis and the main research question and contribution are presented in chapter 1; the background and literature are reviewed in chapter 2; Automation in ML with cybersecurity dataset is covered in chapter 3; application integrity diversification (AID) is described in chapter 4; micro Indicators of Exploitation (mIoX) are presented in chapter 5 ; cyber diversity index is discussed in chapter 6; future research, discussion and thesis conclusion takes place in chapter 7.

Chapter 2

2. Background and literature review

2.1 Introduction

The background and literature review provides an overview of current automation challenges facing learning systems in cybersecurity. Review is undertaken of research that has been identified as important in joining or already assisting the interception of three different domains: automation, learning systems and cybersecurity, as the leading domain, towards a Secure Intelligent Automation (SIA). A comprehensive list is provided of automation challenges and current related research and solutions. The main focus of this review has been the integrity of business applications driving emerging technologies, which have been identified as the biggest threat to cybersecurity, and managing anomalies generated by these, the most challenging for automation (Donevski, & Zia, 2018).

2.2 Automation and Industry 4.0

Industrie 4.0 (2017) or Industry 4.0 is a strategy coined by the German government, where the physical and digital worlds are connected together using advanced automation, machine learning, sensor technology and other exponential technologies. “In the future businesses will establish global networks that incorporate their machinery, warehousing systems and production facilities. This will only be possible if a single set of common international standards are developed. Industry 4.0 connects and merges production with information and communications technology allowing components and machines to autonomously manage production in a flexible, efficient, and resource-saving manner “ (Standards, 2017, p. 3).

Automation is not new to the manufacturing industry, business and industrial processes

have been automated for many years now. There is a new demand in automation for more efficient and advanced manufacturing, known as smart factories. This increases the usage of new technologies, that connect systems and collects digital data. ML and AI are used to analyse data, collected from new creative sensors, resulting with innovative business and manufacturing intelligence.

There are levels of autonomy that correlate to the degree of the autonomous action. Level 0 has no autonomy, and the industrial process is fully managed by humans and Level 5 fully autonomous process where there is no human interaction, such as the lights out manufacturing or dark factories. “In order to attain a specific level of autonomy, intelligence is needed to develop the system further. Since intelligence is based on knowledge gained through experience, AI is well-suited to providing such capabilities. AI is therefore a technological means of attaining a certain level of autonomy” (DTMG, 2019, p.12)

Using automation to merge different technologies will increase the companies' attack-surface area and potentially introduce blind spots, where cyber anomalies might have been detected during manual human processes are now missed. Automation will amplify the impact of the errors and the severity of failures due to hidden, undiscovered vulnerabilities, problem known, the automation paradox (Greengard, 2020).

2.3 Automation in cybersecurity

Research in Autonomous Computing (AC) looks for automation solutions in system maintenance with development of mechanisms and techniques for system self-healing and automatic software patch generation when exposed to zero-day attacks (Gaudin, Vassev, Hinchey, Nixon, Pagano, Garcia, & Narayan, 2010). Demand for automation and solutions for fragmentation issues are one of the reasons for the development of new technologies such as Security Orchestration, Automation and Response (SOAR) (Engelbrecht, 2017) or Security Operations and Analytics Platform Architecture (SOAPA) (Oltsik, & Cahill, 2017) in addition to the traditional Security Information Event Management (SIEM) used in Security Operation Centres (SOC).

The Department of Homeland Security (CISA, 2020) has sponsored a number of different techniques to automate information sharing such as: Trusted Automated eXchange of Indicator Information (TAXII™), Structured Threat Information

eXpression (STIX™) and Cyber Observable eXpression (CybOX™).

The Security Content Automation Protocol (SCAP) are group of standards that enable the automation of vulnerability management and policy compliance (NIST, 2019). OpenSCAP (OpenSCAP, 2019) is the practical implementation of the SCAP model, currently supported by most major Linux distributions. OpenSCAP supports hardening guides such as Security Technical Implementation Guides (STIG) (DISA, 2019) provided by the Defense Information Systems Agency (DISA) and hardening images from the Center for Internet Security (CIS) (CIS, 2019). The SCAP components were created and are maintained by several entities, including NIST, MITRE Corporation, National Security Agency (NSA), and the Forum of Incident Response and Security Teams (FIRST) (NIST-SCAP, 2019).

The SCAP 1.3 components are:

- Extensible Configuration Checklist Description Format (XCCDF)
- Open Vulnerability and Assessment Language (OVAL)
- Open Checklist Interactive Language (OCIL)
- Common Configuration Enumeration (CCE)
- Common Platform Enumeration (CPE)
- Software Identification (SWID) Tags
- Common Vulnerabilities and Exposures (CVE)
- Common Vulnerability Scoring System (CVSS)
- Common Configuration Scoring System (CCSS)
- Asset Identification (AID)
- Asset Reporting Format (ARF)
- Trust Model for Security Automation Data (TMSAD)

Robotic Process Automation (RPA) enables process automation by using a computer user interface in the same way a human would. RPA aims to automate processes by reducing the burden of repetitive and simple tasks (van der Aalst, Bichler, & Heinzl, 2018).

Breach and Attack Simulation (BAS) is a cybersecurity technology that can automatically perform penetration testing and spot system vulnerabilities (Harvey, 2019).

Baah, Hobson, Okhravi, Roberts, Streilein, and Yuditskaya (2016) identify several gaps

in Cyber Defense Automation (CDA) which prevents automation towards building self-healing or self-immunising systems. They identified seven components: attack/vulnerability detection, attack/vulnerability analysis, impact blocking, recovery, vulnerability patching, system cleansing, and an optional active response component. They report that there are no known technical effective defence systems that perform treatment and classification. There are high level of false positives related to automated vulnerability discovery and remediation techniques. Vulnerability triage is a highly manual task. Blocking systems are still by-passable and not fully automated and automated blocking actions are unscalable. Automated patching for buggy applications is unscalable and there are no controls in place to verify the impact of the patch.

2.4 Automation challenges in cybersecurity

Fig. 1 illustrates a non-exhaustive list of automation challenges with anomaly management in cybersecurity, which are briefly explained in this section.

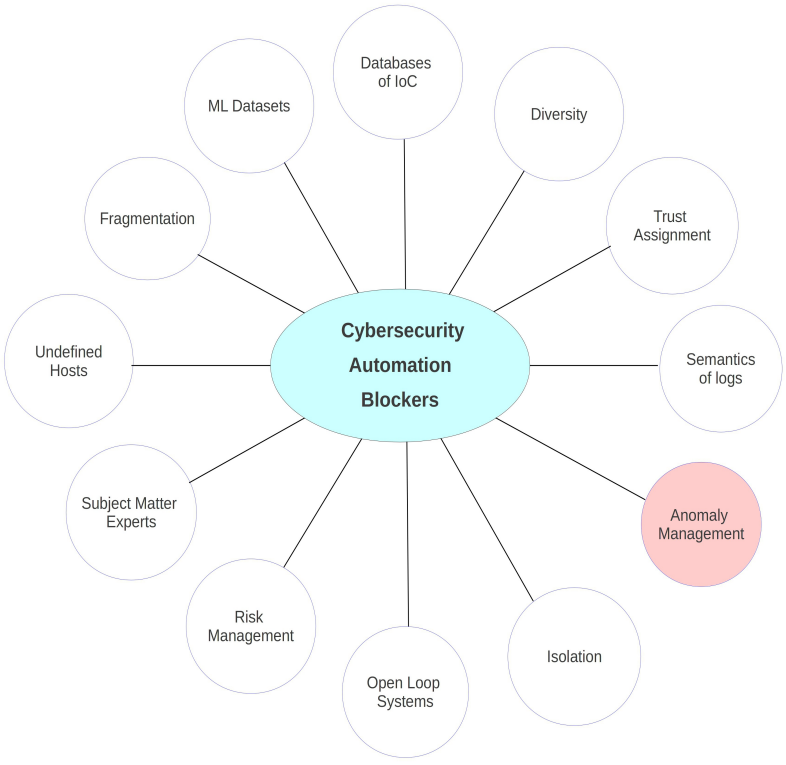

FIGURE 1. Cybersecurity Automation Challenges.

This research identifies a list of cybersecurity automation blockers (Donevski, & Zia, 2018) as seen in Fig.1, and attempts to develop a solution that either exposes, addresses

or prevents these challenges. While this research exposes a complex problem, the solution that is provided could be considered as a modest base for future research. Entropic anomaly solution is developed for monitoring system behavioural profiling, that enable this solution not to use traditional blockers, such as databases of IoC, ML datasets and system logs. Additionally, it provides a good dynamic base for future closed-loop systems, and possibly autonomous anomaly management, that could use dynamic trust assignment, based on the system's own risk assessment.

2.4.1 Subject Matter Experts

Global shortage for security professionals has been reported by many security vendors, including Michael Brown, the CEO at Symantec: “The demand for the cybersecurity workforce is expected to rise to 6 million globally by 2019, with a projected shortfall of 1.5 million” (Morgan, 2015). The Cisco 2014 Annual Security Report also warned, the worldwide shortage of information security professionals is at 1 million openings, even as cyber attacks and data breaches increase each year (Morgan, 2015). In more recent reports, these figures have escalated, where ISACA estimates a global shortage of two million cybersecurity professionals by 2019 and Cybersecurity Ventures predicts 3.5 million cybersecurity job openings by 2021 (Morgan, 2019). Demand has also been reported for data scientists, where IBM predicts demand for data scientists will increase by 28%, with annual 700,000 job openings by 2020 (Columbus, 2018).

Additionally, cybercrime is on the increase. McAfee and the Center for Strategic and International Studies (CSIS) reported an annual estimate that cybercrime may cost the world almost $600 billion. Australian government estimating associated losses of more than $15 million in 2016, and according to the United Arab Emirates (UAE) Cyber Security Centre, the UAE is the second most targeted country in the world for cyber attacks and high cost of cybercrime which is estimated at $1.4 billion per year (Lewis, 2018).

The development of more efficient ML models could increase the detection of anomalies in different domains and trigger a demand for polymatic SME or knowledge workers (Drucker, 2011), have expertise in multiple subjects or domains such as cybersecurity, ML, data science, Internet of Things (IoT), 5G, blockchain, cloud and

quantum computing. It appears there will be not enough SME to detect, understand and manage these new anomalies. From a human management and resource aspect it is crucial to understand this cybersecurity challenge, and to place the right people with the right skills in the right jobs, and not only to focus on organisational quotas and empty seats (Graeber, 2018) and C-level executive positions (Guardian, 2019).

While automation is needed to solve the shortage of SME, there are not enough experts in cybersecurity and data science required to work collectively, on the automation challenges. This challenge could escalate in the future and further increase the demand for automation and autonomous anomaly management solutions.

2.4.2 Fragmentation

The Defense in Depth model is the most popular security model that has been accepted in the cybersecurity domain. This is a layered solution that comprises different security controls, processes, standards, policies, etc. The main challenge is fragmentation, where security controls and people respond independently to alerts generated from anomalies to different teams with different skills and different vendors with different technologies. Each typical security control contains an independent database of Indicators of Compromise (IoC) (DeCianno, 2017) that require to be managed.

Most organisations would have multiple independent and incoherent systems with IoC databases across their environment. Automation and cohesion could be achieved by using an Application Programming Interface (API) between these. However, these systems could still be prone to system drift and corrosion due to degradation of maintenance and lack of skilled SME. Keeping these databases up to date is crucial to their effectiveness, thus collective cyber threat intelligence sharing of the latest cyber attacks provides a good solution to tackling some of these problems (US-CERT, 2018).

The cyber threat intelligence is normally the result of a thorough digital forensic investigations and in the commercial domain this could be seen as an intellectual property and by not sharing this information could give company a competitive advantage, especially information on the source of the anomalies that triggered the investigation.

Takahashi and Kadobayashi (2015) report, due to lack of information exchanges among organisations is the reason why some successful cyber attacks go undetected. Better

corporation and standardisation of this information is required. Additionally, they report that automation is needed, “Human operators often handle the operations manually since cybersecurity information in each organisation is often not well-structured” (Takahashi, & Kadobayashi, 2015, p. 2). They propose an ontology framework a set of concepts in cybersecurity that will shows their properties and the relations between them. This information will be machine-readable, streamlined and automated. During the research they have collaborated with nine cybersecurity organisations, including three different Security Operation Centers (SOC) in the USA, Japan and South Korea that makes this research more practically viable.

While collaboration and cyber intelligence sharing is important, it is not enough when trying to defend against targeted zero-day attacks. Custom based and in-house capabilities such as anomaly based IDS, spam with Bayesian filters, ML infused AV rules and other technologies that provide the capability to gather local threat intelligence provide more effective, focused and relevant defence.

To understand why detecting and defending against zero-day attacks is so difficult we need to analyse their creation and execution process. For example, the attacker will need to discover unreported and exploitable vulnerability by performing fuzzing on the targeted application that could be used by the victim, “fuzzing is a process of intentionally sending invalid data to a product in the hope of triggering an error condition or fault” (Sutton, Greene, & Amini, 2007, p. xix). Next stage is to create an exploit for delivery to the specific targeted software/systems in order to gain control over that system.

The defender could try to look for the unknown system vulnerabilities, but managing known vulnerabilities and reducing the cyber surface attack area seems to be the more effective option. To reduce the exposure to zero-day attacks the defender could analyze the cyber kill chain (Hutchins, Cloppert, & Amin, 2014) and identify the break points at each stage to defend against any cyber attacks in general.

The cyber kill chain is composed of seven stages, as seen in Table 1, that require to be successfully executed, to achieve a successful cyber attack. The defender's aim is to block all known Indicators of Compromise (IoC) in each stage and to “burn” the attacker's identifiable tools, techniques and processes (TTPs). Systems under attack at different stages of the kill chain will exhibit different anomalies and this requires

manual response and analysis to anomalies from cybersecurity professionals with different skills at different layers of their mitigation strategy.

TABLE I. Lockheed Martin - Cyber Kill Chain

1	Reconnaissance	Harvesting email addresses, conference information, etc.
2	Weaponization	Coupling exploit with backdoor into deliverable payload.
3	Delivery	Delivering weaponized bundle to the victim via, web, USB, etc.
4	Exploitation	Exploiting a vulnerability to execute code on victim's system.
5	Installation	Installing malware on the asset.
6	Command and Control	Command channel for remote manipulation of victim.
7	Actions on Objectives	With 'Hands on Keyboard' access, intruders accomplish their original goals.

Fragmentation issues are also caused by the rigid nature of security controls which use isolation and blocking mechanisms. It appears, one of the first actions when responding to detection of anomalies as an incident responder or as a system designer is to apply hard blocks. The strength of the resistance of the defensive security controls should efficiently match the force of the attack. For example, by not blocking IP addresses that are performing a reconnaissance scans on public facing systems and choosing another softer intelligence gathering defensive mechanism might be a better long term solution. Fragmentation lacks connectivity, that causes a major challenge for creation of cohesive organisational processes that are needed for automation. Additionally, each defensive layer has no awareness of the other layers and their security actions. Furthermore, this applies to business applications that have no self-awareness or awareness of other applications and the impact of their actions or functions have on the host.

Reduction of layers or removal of security controls and developing a Leaner Defense in Depth (LDD), that might be required for cybersecurity learning systems, has never been

done before. In addition to LDD, all security controls require continuous resources to monitor and block access, thus defensive security systems are not used only as needed.

The focus of this research is not on the organizational fragmentation as such, but rather on the high impact stage four of the cyber kill chain, the exploitation stage. **B**y developing an automated solution for anomaly management on local level, that also gathers distributed cyber threat intelligence could assist with solving some fragmentation challenges.

2.4.3 Databases of IoC

One common weakness that current security systems have is that they use flat static rules, signatures, blacklists and databases containing IoCs to protect systems and data. They operate on an Aristotelian theory that recognizes statements as only true or 1, and false or 0 (Kosko, 1994). For example, if an event such as unauthorized access, malicious file or suspicious network activity was identified as malicious, the security system will mark this event as true or malicious and block it, or otherwise mark it as false and allow it through.

Kosko goes further to note, "Western culture now sees binary precision as part of the scientific method. The digital revolution seems to digitized our minds" (Kosko, 1994, p. 23). The principle of true or false, unfortunately does not work well, against attacks not seen previously, attackers could potentially use "legitimate" cyber activity, bypass security controls and launch a successful cyber attack (Christiansen, 2010). This security issue introduces uncertainty as to the effectiveness of the current security controls. The lack of visibility and lack of information they provide to the Indicators of Attacks (IoA) (DeCianno, 2017) targeting unreported vulnerabilities is a problem that makes the current controls inadequate to deal with sophisticated zero-day attacks.

The effectiveness of cybersecurity controls, are highly dependent on their up to date databases containing the latest IoC that are shared, and manually managed by SME.

Cybersecurity analysts are perpetually performing manual data entries of IoC in vendor's or in-house databases. The major challenge is keeping these databases up to date.

This research attempts to provide an automated solution for a self-populating database which is necessary for autonomous cybersecurity systems. This solution or index will contain measurable dynamic entropic fingerprints of the applications' run-time

behavioural profile. The database will be populated by the application's own activity during normal operations and when under a cyber attack.

2.4.4 ML Datasets

The slow and expensive development of ML datasets, used in cybersecurity research appear to be inadequate (Buczak & Guven, 2016), when combating the latest attacks seen in the cyber operational environments. Wissner-Gross (2017) identified a possible reason as to "why the AI revolution took so long?" (Wissner-Gross, 2017). He explains, this could potentially be due to inadequate data and not due to low superiority of ML algorithms. Many research papers that deal with cyber attacks are using the same old datasets, and are under the assumption that these datasets are trustworthy, and correctly represent the current problems. Most ML datasets used in cybersecurity research lack diversity, and do not contain anomalies and cyber attacks that are now seen in the IT/OT environments relating to people, systems, data and processes using exponential technologies such as IoT, Cyber Physical Systems (CPS), cloud computing, containers, and 5G.

The creation of ML datasets is slow, manual and expensive, that prevents automation in cybersecurity. Real time solution and better suitable learning system that does not rely on datasets are required and proposed in this research for autonomous anomaly management.

2.4.5 Risk management

Managing risk when dealing with anomalies in production environments is a manual process performed by different teams focusing on different business demands. Risk assessment performed by system administrators in comparison to cybersecurity analysts might not have the same security value. For example, an assessment for usage of Open Source Software or anomalies generated by these, may provide higher level of security assurance, when performed by cybersecurity analysts, by additionally determining vendor reputation and establishing a trusted source.

Identifying anomalies in compliance and auditing during risk management is also a manual process covered by number of standards and guidelines such as: International Organization for Standardization (ISO) 3001:2018 Risk management (ISO 3001, 2018), National Institute of Standards and Technology (NIST) SP 800-30 Rev. 1 (NIST-800-30,

2018) and ISO/IEC 27000 family - Information security management systems (ISO-IEC 27000, 2018), a systematic approach to managing sensitive company information, people, processes and IT systems by applying a risk management process.

Risk assessment, is performed by first identifying the tangible, and intangible assets that the company is trying to protect. The tangible assets are physical in nature, such as property, physical systems, equipment, etc. Intangible assets could include information such as software code, company's customer database or intellectual and proprietary information. Any damage, destruction or unauthorised access to the company's assets accidentally or intentionally by an attacker are threats that are risk assessed against. Vulnerability is a weakness identified in systems, processes and people that manage assets that can be exploited by the attacker in order to gain access to the assets.

Risk assessment and management of anomalies generated in the System Development Life Cycle (SDLC) are manual processes and rely heavily on use cases, that could be vulnerable to blind spots or process corrosion when assessing risk from anomalies generated by zero-day attacks.

A comprehensive risk assessment might have been performed for unpatched software, non-technical C-level executives and neglecting cybersecurity, but not many security assessors could have predicted the damage that Equifax suffered, $1.35 billion in costs as a result of their data breach (White, 2019). While, C-level executives are not expected to be technical experts, the organisational failure in this case cost them their jobs, either because the cyber threat was not truly understood (Wroe, 2019) or the risk appetite was too high.

There is no “risk free” business environment or security control that can provide that. “The most effective security solution is that which provides the best level (that which is optimised) for “the least cost” (Wright & Zia, 2011, p. 208).

Risk assessment is a complex challenge and a manual process that prevents automation in cybersecurity. Automated risk assessment for anomalies triggered by software vulnerabilities in exponential technologies by self-mitigating and autonomous systems would have to match the organisational unique business risk appetite.

2.4.6 Trust assignment

Establishing and maintaining trust is a crucial, but manual process in the cybersecurity

domain, which is needed for access control, systems integrity, software assurance and development of scientific data or datasets for further ML research.

Bergstra and Burgess (2018) explore, how trust relates to an emerging technology, such as blockchain using Burgess's Promise Theory (Burgess, 2005). In this theory every autonomous agent follows an atomic policy that consists of promises with regard to configuration and operational states as part of its Service Level Agreement (SLA).

They report that if every action the agent performs has been verified than this is a zero trust model and if no verification has been done than this is a complete trust model. The blockchain technology has been called trust-free or trust-less technology.

Taking this into consideration, they raise a concern as to the "blockchain network is self-validating in this limited sense. Nevertheless, the idea that no trust is needed (which paradoxically implies that one should trust it completely) is a bold and quite misleading assertion. A more pertinent question is to ask which promises we are trusting" (Bergstra & Burgess, 2018, p. 14).

There is uncertainty, whether agents can be considered trustworthy. The 51% attack on blockchain, raises these uncertainties further, where "group of miners controlling more than 50% of the network's mining hashrate, or computing power could prevent new transactions from gaining confirmations and allowing them to halt payments between some or all users. They would also be able to reverse transactions that were completed while they were in control of the network, meaning they could double-spend coins" (Frankenfield, 2019, para 1-2).

The blockchain requires more than 50% of miners to be "honest" or trustworthy. To detect these type of attacks you would have to monitor and look for anomalies relating to trust, like in the case of the Coinbase cybersecurity incident response, where the attacker over 3 days managed to steal $1.1M (Jenkinson, 2019).

Weimer, Forrest, Kim, Goues, and Hurley (2016) propose a trust model for an automated software repair techniques, where systems are required not only to be resilient, but also trusted. They propose the automated program repair to be augmented with assessments, that are going to provide the trust needed for post-repair systems. The resilient systems would have the capability to recover from errors, attacks or environmental changes. For the augment resilient program repair, they consider three oracles, based on external execution signals observed at run-time; on behaviour in similar code fragments, and on inferred and validated program invariants (Weimer, Forrest, Kim, Goues, & Hurley, 2016, p. 1).

Trust in self-repairing or self-patching in systems is a complex problem, the process starts with software that already has vulnerabilities, and possibly is less trustworthy than a rugged software, where software development, is more than just secure coding, but also applies to the culture and secure mind that adds towards the resilient software (Rugged, 2019).

2.4.6.1 Zero trust model

Eidle, Ni, DeCusatis and Sager (2017) propose an automated solution to the Zero Trust Model (ZTM, 2019) originally introduced by Forrester Research and National Institute of Standards and Technology (NIST). Their test bed contained firewalls, gateways and other network devices and was modelled towards a cloud data center. The Zero Trust Model does not assume trust in any way, there are no trusted zones, access controls are strictly enforced and everything is logged. Continual monitoring has been provided with a feedback loop undertaking the following steps: Observe, Orient, Decide and Act (OODA). One way to stop wastefulness when it comes to using system logs and big data is to use the data effectively and collect only small fractions of useful information has been proposed in their test bed. Their environment contains a dynamic access control with eight trust levels. Eidle, Ni, DeCusatis and Sager (2017) report the following results from their experiment "By preventing unauthorised scanning and reconnaissance, Transport Access Control (TAC) disrupts the attackers kill chain, blocks both known and unknown attack vectors, and stops lateral attack spreading within a data center. In our testing, attempted DDoS attacks are blocked immediately" (Eidle, Ni, DeCusatis, & Sager, 2017, p. 3). In this dynamic solution Eidle, Ni, DeCusatis and Sager (2017) have demonstrated an automated solution that solves fragmentation issues with different devices and automatic response without the need to wait for analysts to respond to alerts.

2.4.6.2 Adversarial Machine Learning

One field of ML research that questions trust and focuses on security is Adversarial Machine Learning (AML), where researchers conduct experiments to deceive ML models to produce bogus results. One such research conducted by Srndic and Laskov

(2014) was to evade learning-based classifiers for detection of malicious code in pdf files. An online system named PDFrate (Smutz, & Stavrou, 2012) was used in the test case and a framework called MIMICUS that provides the means to modify malicious files and make them appear benign to the PDFrate system by implementing the Mimicry attack. Its idea is to transform a malicious sample in such a way that it mimics a chosen benign sample as much as possible, making the resulting mimicry sample harder to detect. The research shows a successful implementation of the MIMICUS framework and excellent results in bypassing and tricking the PDFrate classifiers (Smutz, & Stavrou, 2012). They also noted the increased usage of learning-based classifiers for detection of malicious data in various research papers and security controls. They further note that the reason why this attack was successful is that the learning-based classifiers in this research were deployed on-line, thus giving the attacker the opportunity to evade this security by tweaking and manipulating the data until passes all tests. Their future work will attempt to provide solutions for construction of attack resilient features (Srndic, & Laskov, 2014).

Lowd and Meek (2005) introduce a concept, where the task of the attacker is to learn sufficient information about a classifier, to construct a successful adversarial attack and to do this they proposed the Adversarial Classifier Reverse Engineering (ACRE) learning, which utilises algorithms for reverse engineering of linear classifiers in spam filtering systems. This research not only that it introduced the first query-based-black-box attack on classifiers but also defeated the spam filtering system in the process (Lowd & Meek, 2005).

Miller, et al. (2014) propose a solution for security oriented Active Learning, an area in ML that deals with actively engaging a human labeler, for additional information required for the original training set. Active learning is particularly useful in cybersecurity where many unlabelled examples and labelling data manually is expensive. Miller et al. point out that the accuracy of the ML models depends on the quality and integrity of the training and test data, if test data gets polluted a poorly performing model might get produced, unlike other domain, the cybersecurity application could suffer an adversarial drift. Miller et al. develop the Security-oriented Active Learning Testbed (SALT) a software framework, for research and implementation of active learning applications in adversarial contexts. This framework will allow experiments to be performed in a monitored environment where training data will be polluted and the adversarial drift will be analysed. This research provides an

excellent software framework for security researches to study the adversarial drift given a different labelling experiments in a safe and monitored environment. “Involving humans in the learning process is always expensive. Humans, even in crowdsourcing scenarios, are a limited resource. Humans have limited capacity. Humans make errors, sometimes maliciously” (Miller, et al, 2014, p. 10).

Corona, Biggio and Maiorca (2016) in their research, develop AdversariaLib, the first open-source python library for security evaluation of ML algorithms. The library is composed of three different modules: advlib, implements the attack algorithms and functions for the evaluation of classifiers under attack; prlib, implements standard functions for handling datasets, defining distances between samples in feature space; and util, contains library functions for storing datasets, classifiers, results, logging and progress of experiments (Corona, Biggio, & Maiorca, 2016).

The R programming language is one of the more popular languages used by data scientists (Kdnuggets, 2019), Ooms (2013) addresses a security concern in weaknesses with R and proposes straightening security by using the RappArmor package (Ubuntu, 2020). He points out that the original prime use for the R language was a single statistician using R on a local machine with an interactive console, however with the growth of cloud computing, introduces complexities with regard to the execution environment being unrestricted. He further warns of possibilities of data scientists running a not trusted code, perhaps received from a spear phishing email. “Suppose we found an R package in our email or on the Internet that looks interesting, but we are not quite sure who the author is, and if the package does not contain any malicious code. The package is too large for us to inspect all the code manually, and furthermore it contains a library in a foreign language (e.g. C++, Fortran) for which we lack the knowledge and insight to really understand its implications” (Ooms, 2013, p. 3).

Demand has increased for R to be integrated or be embedded with other languages, products, services and open source software and concerns are raised, to introducing vulnerabilities during this process into the products. He proposes RAppArmor package for creating security policies for R, using AppArmour to restrict access control to files, directories, services, and accounts (Ooms, 2013,).

While the focus of AML research is on the adversarial impact on ML classifiers and data integrity. Providing mitigation using current security solutions to protect classifiers and data will only further obstruct future automation. While zero trust model is effective it lacks the agility that is required for automation solutions. More research in provision

of assurance of business application integrity that produces and manages data for autonomous dynamic trust assessment is needed.

2.4.7 Diversity

A report by Gartner (2016) provide an insight to the increase of new technologies as more than 2000 technologies are expected to be introduced in the next 10 years. There will be an inevitable increase of new anomalies, unknown vulnerabilities, and also an increase of the companies' surface attack area. New emerging technologies could contribute towards beneficial diversity and from an AIS perspective, the exposure to new technologies will increase the systems' cyber resilience (Forrest, Somayaji, & Ackley, 1997). Diversifying technologies prevents having systems with the same vulnerability and makes it harder for an attacker to compromise many systems with the same exploit.

2.4.7.1 Software Diversity

Diversifying systems and software with randomisation, increases the system resilience by eliminating the predictability of static processes, that the attacker can exploit. Baudry and Monperrus (2015) in their comprehensive survey on software diversity identified three main directions in this domain: goals with fault-tolerance, security, and software engineering; means with managed or automated diversity; and analytical studies with quantification of diversity and its impact. Their survey was the first paper at the time to present a holistic research about randomisation used to diversify design and data for fault tolerance and also cybersecurity. They report the broad research that has been undertaken in this area from diversifying program execution paths, to diversification of software code, to diversification of open-source software development to automated diversity. Automated software diversity tries to remove the human in the loop, of normally manual processes, when introducing diversity in software (Baudry, & Monperrus, 2015).

Larsen, Homescu, Brunthaler, and Franz (2014) in their paper on automated software diversity try to answer what, when and where to diversify and how to evaluate security and resource overheads. They provide detailed information on taxonomy of an attack and the taxonomy of defensive controls that use diversity. The main motivation for their

research was the problem with the static or homogeneous online software distribution. The attacker can easily download the same software that the victim has and look for exploitable vulnerabilities. By introducing artificial diversity to compiled software, every download will be randomised and unique, thus making this process more difficult for the attacker to discover a single exploit that works on multiple systems. Larsen, Homescu, Brunthaler, and Franz note that to verify effectiveness of randomness or diversity will depend on the level of its entropy. Additionally, software diversity could be used in software patches or updates, where due to current homogeneous processes the attacker can download, reverse engineer these updates and discover exploitable vulnerability (Larsen, Homescu, Brunthaler, & Franz, 2014).

Software diversity tool, called XIFER developed by Davi, Dmitrienko, Nürnberger and Sadeghi (2012) aims to improve the cybersecurity on mobile devices against run-time attacks, in particular code-reuse attacks. Their tool requires no source code and induces an overhead of only 1%. XIFER applies diversity to the binary at run-time over the entire memory for each invocation by using its own binary rewriter to transform the control-flow graph of the program. Small code units, the basic blocks (BBLs) are being split and injected with nop instructions into the application. The BBLs are then permuted in memory by using the Pseudo Random Number Generator (PRNG) that introduces a higher randomized memory layout and also increases the randomization entropy, but still retaining the semantics of the initial program (Davi, Dmitrienko, Nürnberger, & Sadeghi, 2012).

DIVERSIFY is an EU funded project undertaken by Baudry, Monperrus, Mony, Chauvel, Fleurey, and Clarke (2014) aims to use spontaneous software diversification to increase the systems' adaptive capacity to unpredictable changes in distributed, heterogeneous and large-scale environments. The project reflects the bipartite ecological relationships to investigate the opportunities for automatic diversification of the current mono-culture software in client-server relationships. DIVERSIFY aims to provide the mechanisms in collaborative adaptive systems (CAS) to defend against attacks known as Blow-one Blow-everything (BOBE), where all servers run the same Operating System (OS) and the attacker needs to discover a single exploit to crash one server and potentially crash all servers (Baudry, et al., 2014).

Neti, Somayaji and Locasto (2012) in their research attempt to provide insight into better understanding and importance of choosing host diversity. They use Capture the Flag cyber game to offer a choice to the players to select and switch between different vulnerable services. The players are given a strategic options and the diversity is measured by using Renyi entropy with bipartite graph. The findings of their research addresses the significance to reduce the overlap of vulnerabilities, and to have an effective diversity defense strategy. Additionally, they hope more research is undertaken towards a successful diversity framework, where systems need to be unique in behaviour and software diversity needs to be practical and cost effective (Neti, Somayaji, & Locasto, 2012).

Anckaert, Sutter, and Bosschere (2004) attempt to solve the problem of software piracy by using diversity to produce a unique software install or instance and to tailor updates that fix bugs, provide security patches, compatibility and new functionality only for the installed copy. There are two types of software piracy that the research is trying to prevent. Softlifting refers to piracy where one installation is legal and the same software license is used to be illegally installed on additional systems. Hard disk loading is the unauthorized installation of software on hard disks that are sold as incentive for purchasing new personal computers. Third software piracy type that is difficult to prevent and is out of scope in this research is mischanneling, where academic license is used for commercial purposes. The diversification software updates play a crucial aspect in this research because it is assumed that the pirate providing the software and the end user are not the same person. By using software ageing techniques and diversification the pirated copy would not get updates and eventually decrease in value (Anckaert, Sutter, & Bosschere, 2004).

Wang, Duan, and Simmons (2016) use an ecological approach to evaluate system security by measuring software diversity. Shannon Wiener Index was used as the quantitative method in their research. Various operating systems were tested, desktops, tablets, supercomputers, mobile devices, and Internet servers. They report, most systems measured low diversity levels, which could mean lower cybersecurity resilience, except for Internet servers which also showed an increase during a two-year period. The following results were reported:

- Desktop and laptop diversity index, 4.2093, lower than the optimal value of 8.0;
- Web clients diversity index, 3.0526, lower than the optimal value of 6.0;

- Global tablets' diversity index, 2.2056, lower than the optimal value of 6.0;
- Mobile devices' diversity index, 3.1, lower than the optimal value of 8.0;
- Internet Servers measured high level of diversity 3.1 out of 4.0.

A troubling figure of low diversification in supercomputers was reported, where Linux is the predominating operating system, recording “a very low diversity index, approximately 1.2 out of 5.0, which indicates that the supercomputers consisting of almost homogeneous servers might have fair weak resistance to some security threats such as virus and worms spreading through networks” (Wang, Duan, & Simmons, 2016, p. 98).

System diversification increases resilience by introducing uncertainty to the attacker. Further, diversity of data and technologies are absolutely crucial for the cyber ontogeny of a learning systems. However, diversification also increases costs, maintenance overhead, requirement for new SMEs and funds for further training for the defender. Diversification could also introduce incompatibility issues that prevents automation. Diversification by using randomisation prevents automation because automation requires consistent processes in order to automate.

2.4.8 Isolation

Many security controls and technologies use isolation and tagging as a defensive mechanism, where systems, traffic and data are labeled and separated with firewalls, Virtual Local Area Networks (VLAN), sandboxes, containers, Multi-Level Security (MLS) and Demilitarised Zones (DMZ). While these mechanisms have provided a good level of protection, they are manually managed and lack agility. Future dynamic cybersecurity learning systems require diversity of data, and isolation could prevent merging of diversities, where beneficial anomalies are not going to be generated towards the system’s self-learning and development of a stronger and more cyber resilient AIS.

The anomaly management in a traditional centralised production environment has a holistic organisational model, where SIEM, NIDS and client-server architecture are used. New technologies that drive the distributed model, such as micro-services, containers, distributed applications, blockchain technology further increase this isolation, and will require some level of autonomous anomaly management in the future.

The major challenge with isolation of applications and data is the restriction of access required for provision of autonomous solutions.

2.4.9 Open Loop Systems

The problem with many proposed and production security systems is that they are open-loop systems, they detect anomalies of possible cyber attacks and create alerts for further investigation by cybersecurity analysts. Closed-loop systems provide an additional feedback loop that regulates the input in order to provide the expected output. The feedback loop in cybersecurity systems could automate anomaly management and take the pressure off the cybersecurity analysts. Most anomalies in production environments are managed manually by cybersecurity analysts. It appears the same issue applies in the data science domain, man in the loop or analyst in the loop is required for provision of assurance (Dietterich, 2018), and detecting any unwanted ML model bias or also known as overfitting. A development of a dynamic system called pH closes the open loop system concept by monitoring processes at the system-call level and automatically responds to anomalous behaviour by either slowing or aborting system calls (Somayaji, & Forrest, 2000).

The challenge of open loop systems, is the lack of the feedback loop, required for automation. The lack of good understanding of cyber anomalies perhaps is another reason security controls stay as open loop systems. Implementation of innovative closed-loop systems that utilise M2M and business intelligence are needed in the cybersecurity domain to provision a true automation.

2.4.10 Semantics of logs

In a traditional production environment, the SIEM system ingests logs from Intrusion Detection Systems (IDS), firewalls, Anti-Virus (AV), servers and desktops to detect possible cyber attacks. The problem with this scenario is that logs can be unreliable, taking into consideration that attacks can reside entirely in memory and don't leave any evidence on disk (Irani, & Weippl, 2009). Additionally, logs that are generated by an application only interpret the events and might not provide a full semantic picture of the event (Creech, & Hu, 2014)**.** Furthermore, the integrity of logs generated by the

vulnerable application that has been exploited could also be questionable.

A ground-breaking research by Forrest, Hofmeyr, Somayaji and Longstaff (1996) set the path for many research papers focusing on system calls instead of system logs, when looking for anomalies that could lead to detection of cyber attacks. The challenge appears to be in identifying the true positive sequence of system calls that show the malicious intent. This research uses the human immune system model to identify anomalies, or separating the normal environment or system "sense of self", form the foreign bodies or malware. The paper outlined a novel way of detecting anomalies that still creates interests in the research domain. The downside reported, was the resource intensity at the time and the recommendation of a short range of system calls.

2.4.10.1 System calls

Bernaschi, Gabrielli, and Mancini (2000) propose a cost effective mechanism, to control the invocation of critical system calls, and provide operating system enhancements, to protect from buffer overflow attacks by managing system calls. They propose a mitigation solution, in which system calls are interrupted before the malicious action occurs and designed a reference monitor model for system calls linked to an access control database that mitigates particularly system calls that could cause damage. They implemented a risk assessment of system calls and grouped them with threat levels, 1 and 2 are treated with higher priority than system calls rated 3 and 4 (Bernaschi, Gabrielli, & Mancini, 2000).

Creech and Hu (2017) research focuses on designing a host-based anomaly intrusion detection systems that uses system calls and report the challenge associated with high false positive alarm rates. Their paper introduces a new methodology by using discontiguous system call patterns, in an attempt to increase detection rates whilst reducing false alarm rates. The reason for adapting and perusing with the anomaly based approach, is that in theory no prior knowledge is necessary of the attack to detect a new zero-day attack. This gives the anomaly based systems advantage over the signature-based systems which have a crucial weakness of no capability to detect a previously unseen attack. The ADFA Linux Dataset (ADFA-LD) (Creech & Hu, 2017) was created as part of their research using a modern, at the time, operating system and contemporary hacking methods, which has since, spawned the creation of additional new dataset such

as Australian Defence Force Academy Windows Data Set (ADFA-WD) with a Stealth Attacks Addendum (ADFA-WD: SAA) (Waqas, Creech, Xie, & Hu, 2016).
The results of their experiments which compared the proposed Extreme Learning Machine (ELM) detection engine using the semantic feature outperformed other algorithms and this semantic algorithm was also suggested that in theory would have natural resilience to mimicry attacks or attacks that bypass security controls by mimicking legitimate operation such as sequence of system calls. Their proposal for future research contains: improving the transference process to different systems, reducing the ML training overhead and enhancing resilience to mimicry attacks (Creech & Hu, 2017).
Furthermore, inline with Creech and Hu (2017) research and motivated by deep RNN-based language modelling, Gyuwan, Hayoon, Jangho, Yunheung and Sungroh (2017) propose an application of neural language modelling to host-based intrusion detection where they consider system call sequences as a language used for communication between users (or programs) and the system. In their view, system calls and their sequences correspond to words and sentences in natural languages, respectively. Based on this system-call language model, they proposed improvement to detection of anomalous system call sequences (Gyuwan, Hayoon, Jangho, Yunheung, & Sungroh, 2017).

Yoon, Mohan, Choi, Christodorescu and Sha (2017) report that while there is a lot of research related to anomaly based Host Intrusion Detection Systems (HIDS) and system calls, none of them fully utilise the intrinsic properties of the smart embedded devices and IoT. They present an anomaly detection mechanism for embedded systems using a System Call Frequency Distribution (SCFD). In their research they use cluster analysis to learn the normal execution contexts of an open-source webcam application system calls and monitor for at run-time abnormal executions. Yoon, Mohan, Choi, Christodorescu and Sha (2017) report that their method can detect anomalous executions.

There is increased interest in the DevOps environment for container technologies, due to easier application deployment and lower operational dependencies. Abed, Charles and Levy (2015) raised a concern in their research, the possible vulnerabilities associated with this technology, container compromise, and report the lack of research

in this area. Linux containers share the same host kernel, security concerns were addressed from the aspect of the source of an attack: originating from outside the host, from another container and when a container attacks the host kernel. To target these attacks they proposed a real time host-based intrusion detection system to detect attacks against applications within Linux containers that passively monitors system calls between the container processes and the host kernel. The containers could be running in a standalone or in a cloud multi-tenancy environment. They use strace to generate a syscall-list file that holds a list of distinct system calls that is used to create a syscall-index lookup table. The result of their experiment returned 100 % detection rate (Abed, Clancy, & Levy, 2015). They are also reporting that there were no container datasets available at that time, and they had to create their own dataset in order to trace the container behaviour. Container profiling techniques are being utilised in production environments by using whitelisting of application system calls (Wang, 2018).

2.4.10.2 System call research traps

Systrace is a tool that was created as a result from research by Provos (2003) addressed the issue with malicious system calls, by using a system call interposition technique with a reference monitor, where only user-specified profiles of good system calls were allowed to be made by the application.

Reference monitor is a secure system design concept that defines a set of requirements on a reference validation mechanism, which enforces an access control policy over subjects (Jaeger, 2011), first proposed by Anderson (1972). The reference monitor mechanism has three design requirements: complete mediation, tamper-proof and verifiable. The mechanism needs to be small enough to be easily audited for assurance (Jaeger, 2011).

System call interposition technique has been used by many researchers experimenting with system calls and intrusion detection. This is a kernel extension that uses security policies without modifying the underlying code. Watson (2007) in his research successfully manages to exploit race conditions such as Time-of-check-to-time-of-use (TOCTTOU), Time-of-audit-to-time-of-use (TOATTOU) and Time-of-replacement-to-time-of-use (TORTTOU) vulnerabilities with the later two, first tested in his research. System call interposition combined with multiprocessing and threading could be open to concurrency vulnerabilities, leading to privilege escalation and audit bypass. Watson

further suggest that, while system call interposition was originally developed to stop OS modifications, this might have allowed more opportunities for race conditions (Watson, 2007).

In his research Garfinkel (2003) proposes Janus, a kernel module that provides a mechanism for secure system call interposition. Garfinkel also provides an inside to the traps and pitfalls related to designing such a system and that race conditions are possibly unavoidable between the OS and the monitor. Additionally, he points out that "the problem of secure system call tracing is similar to that of secure system call interposition. In the system call tracing case, the "viewer" is only interested in what calls an application makes, and not in modifying them or denying them. However, the same problems that can allow a system call interposition-based sandbox to be circumvented can also be used to evade a system call tracing-based IDS" (Garfinkel, 2003, p. 2).

Linux Security Module (LSM) (Kernel, 2019) is a framework, enabling the support of multiple reference validation mechanisms. Security policies are provided by Mandatory Access Control (MAC) extensions. There are large number of MAC extensions, that have been developed to use this framework such as SELinux (NSA, 2019), AppArmor (Ubuntu, 2019), Smack (Tizen, 2019), and TOMOYO (Tomoyo, 2019) to solve the referencing monitor problems. Jaeger (2011) reports that the main problem is verifying the reference monitor design requirements, where each requirement has its own unique challenge, for example, tamper-proofing has a problem with Trusted Computing Base (TCB), too large to determine whether tampering is been prevented (Jaeger, 2011).

The challenge with log-based security controls is unreliability, where gaps and blind spots in logs could prevent automation. Monitoring system calls for anomalies, could be a more reliable and a better option when developing autonomous distributed systems.

2.4.11 Undefined Hosts

Research undertaken in intrusion detection separates two distinct systems: Network Intrusion Detection Systems (NIDS) and Host Intrusion Detection Systems (HIDS). NIDS monitors network traffic and detects anomalies from data in transit, and HIDS look for anomalies generated on the hosts or the endpoints.

The problem with HIDS research is the lack of coverage and clear understanding of the scope of host systems. HIDS research has provided limited solutions to an incredibly fast expanding business and industrial environments. The host-based research has been underestimated in the industry, where a host could potentially be any physical, virtual, augmented and bio-mechanical system, hosting applications, regardless of its function as to workstation, desktop or server with or without an Operating System (OS) (Kantee, 2015).

The source of cyber anomalies could originate from any software-driven systems with no local HIDS capability, such as printers, routers, switches, firewalls, embedded systems, IoT, CPS, Internet of Medical Things (IoMT) (NIST-IoMT, 2019), manufacturing robotics and Supervisory Control And Data Acquisition (SCADA) systems. The one important thing that differentiate these hosts is their unique software and applications that defines and drives their main core business function.

2.4.11.1 Business and Industrial applications

"Software is eating the world" (Andreessen, 2019). Since the birth of software and the execution of the first stored-program (History-Computer, 2019), the importance of software has increased, and it has now become an integral component of the everyday business world. Some, once known traditional businesses, are now converting into exclusively software companies using data analytics. However, insecure, malicious and vulnerable software is possibly the biggest problem for cybersecurity, hence the emphasis given to the cybersecurity domain for software vulnerability management.

In line with Forrest, Hofmeyr, Somayaji and Longstaff (1996) research and the concept of 'sense of self', it would be difficult to focus only on the static nature of good and bad applications (malware) and not to look at the dynamic relationships and the trust levels, between these and the host or kernel. As well as malware, 'good' applications and programs with a high level of privilege or many vulnerabilities can cause just as much or more damage to the host if exploited.

Looking at the biological meaning of host (Brenne, 2019), it is clear that the host has many symbiotic relationships. Comparing these relationships with the current application relationship development in DevOps environments, the following could be concluded:

- Mutualism is a relationship that benefits both the host and the applications.
- Commensalism is a relationship that benefits the application while the host or kernel is neither helped nor harmed.
- Parasitism is a relationship where the organism (the application) benefits, while the other (the host) suffers.

While research in cyber ecology by Jorgensen, Rossignol, Takikawa and Upper (2001) identified unique cyber parasites as malicious and possibility of sentinel cyber parasites that are not. This unfortunately, could apply on a much larger scale, where most legitimate business applications developed today, could have some vulnerabilities, that would make them have a parasitic relationship with their host.

In nature, these symbiotic-relationships retain a delicate balance and the same could apply to cybersecurity. Dynamic relationships between the applications and the host could change over their life-cycle and so does the trust level the host has for these applications.

By monitoring application's behavioural profile, and its relationship with its environment, the trust could be dynamically adjusted. For example, the host could autonomously lower the trust level to a legitimate application that has gone rouge, by a previously undiscovered vulnerability that makes the host unstable, vulnerability that is or has been exploited, patch that has been compromised, or introduces a new vulnerability, change of DevOps process, where the third party vendor no longer practises secure coding and open source software supply patching chain, has been compromised.

Considering the symbiotic relationship concept the most beneficial mutualism where the host provides benefits to the applications and all applications provide benefits to the host, but until then the concept of, trust but verify, needs to apply where the host continually verifies the integrity of the applications.

Current business applications have no self-awareness, of their role, and the impact of their actions on their host. Badly automated, configured or coded applications, introduce vulnerability, and can potentially take their host down, like in the recent Boeing 737 MAX case (Boeing, 2020). Manoeuvring Characteristics Augmentation System (MCAS), a flight control software, was reported to be the possible cause, of two air-plane crashes that resulted to large number of human casualties. Due to an error with a sensor that was providing incorrect data, the MCAS software over

compensated and caused the air-plane to crash.

Part of an effective anomaly management plan, to identify the source of anomalies. Fig. 2. shows different technologies of interest related to the source of future cyber anomalies, especially from an integrity perspective. One common artefact that all software driven technologies have is application bugs, which are going to generate anomalies either accidentally or purposely by an attacker.

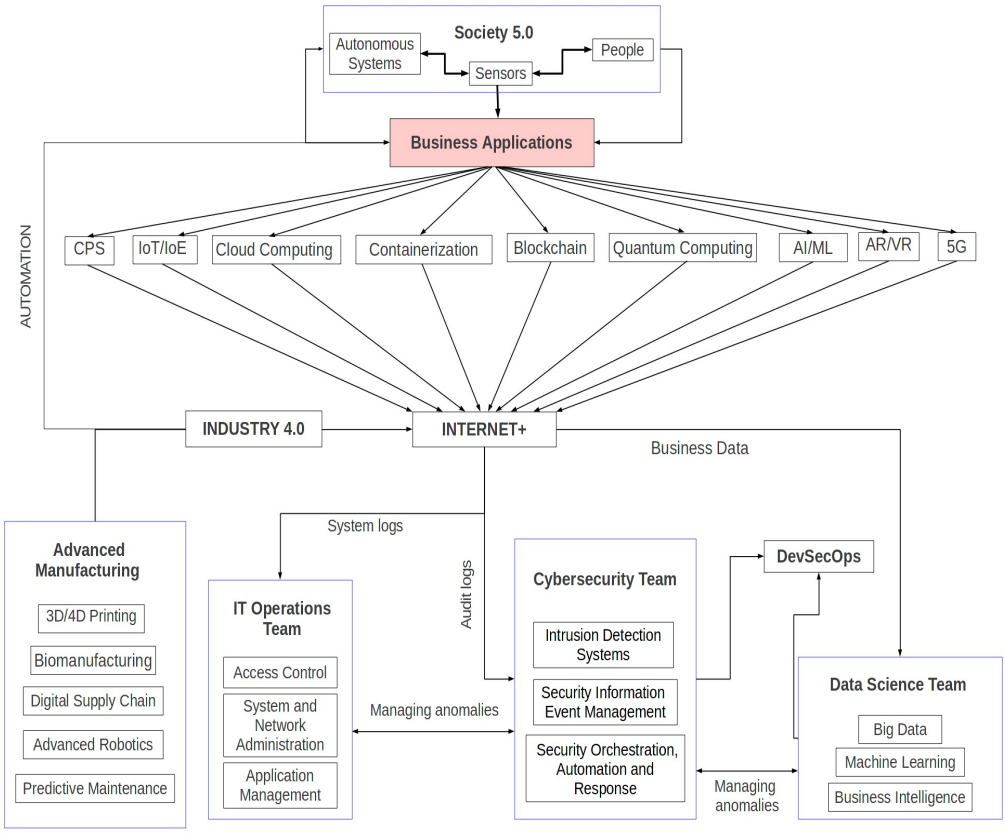

FIGURE 2. Business applications, the source of future cyber anomalies.

According to a recent research study by Deloitte "86% of the top 100 companies in Research and Development spending worldwide are from the manufacturing industry" (Columbus, 2019). Industry 4.0 (DTMG, 2019), and the Industrial Internet (II), introduces advanced manufacturing, automation in digital supply chains, predictive maintenance, computer-integrated manufacturing, and advanced data analytics using ML for Manufacturing Intelligence (MI).

Managing anomalies in these complex digital ecosystems from a cybersecurity perspective is going to be a challenge, monitoring and management of continual

availability of systems and sensors, only partially provides assurance of a system's operational integrity.

The German software company SAP with Enterprise Resource Planning (ERP) has large coverage in the manufacturing industry, issued security guidance in 2009 and 2013 to their clients to fix critical vulnerabilities. Security firm Onapsis (2019), report that large number of SAP systems are still not properly protected. The anomalies that have been generated by these vulnerabilities since 2013, during an attack or normal operations would have either been ignored or gone undetected. Anomaly management is about development of sensors capable of detecting anomalies of interest and also support innovations for the detection of future anomalies.

Each emerging technology could potentially generate its own unique anomalies and inconsistencies. For example, anomalies might only be present in the context of the business's mixed or Augmented Reality (AR) and Virtual Reality (VR) data streams and not in the application logs. Further, anomalies are managed manually, by three separate operational teams, as seen in Fig. 2, that are experiencing fragmentation issues that introduce blind spots, possible increase of the Mean Time To Detect (MTTD), and the Mean Time To Repair or Respond (MTTR) to alerts generated by cyber anomalies.

Every sensor, system, business application, exponential technology, user and data in Society 5.0 (2019) will provide new opportunities for an attacker to deliver a zero-day attack. Are production environments ready to deal with the amount of cyber anomalies that are going to be generated on the Internet+ (Schneier, 2018), and Internet Plus (PRC, 2019) or are these anomalies going to germinate totally undetected?

System integrity verification starts at the beginning of the technology life cycle, like Microsoft's Security Development Life Cycle (SDL) (Howard, & Lipner, 2006), and in the continual review and verification of web application through a Secure System Development Life Cycle (S-SDLC) (OWASP, 2019). Security should be embedded as in DevSecOps, in which security is automated (Red Hat, 2019).

It is difficult to predict future vulnerabilities, threats or anomalies that these diverse exponential technologies, and their merging might introduce. The dynamic and converging production IT and OT environments, with automation and learning systems could provide solutions to application integrity verification and autonomous anomaly management solutions.

2.4.11.2 Business and Industrial applications

The Top 4 mitigation strategies recommended by the Australian Signals Directorate (ASD) have been reported to mitigate at least 85% of the intrusion techniques that the Australian Cyber Security Centre responds to (ASD, 2019).

- Application whitelisting.
- Patch applications.
- Patch the operating system.
- Minimise administrative privileges.

Great emphasis has been given in the cybersecurity domain to software vulnerability management, which is triggered by poor secure coding practices (Chess, & West, 2007), inadequate systems hardening practices as recommended by ASD (2019), and lack of Software Security Assurance (SSA) (Goertzel, Winograd, McKinley, Lyndon, Colon, McGibbon, Fedchak, and Vienneau, 2018). Not having visibility to anomalies generated by exploitation attempts, made by the attacker before the vulnerability was patched is also a challenge. The application whitelisting effectiveness as the top mitigation strategy prevents any execution of unknown or unauthorized code. However, what seems to be challenging in production environments is the management of application whitelisting databases. Risk assessment of new applications' integrity and software assurance is a challenge, and a manual process. While there are products that provide some threat evaluation for applications, lack of visibility of unknown exploitable vulnerabilities in business applications is still a challenge. Legitimate business application that turn rogue or have exploitable vulnerability, pose more risk than corrupted malware that have been correctly identified by an AV product.

2.4.11.3 Vulnerability management

Eliminating or effectively managing known system vulnerabilities is not a small task, managing unknown application vulnerabilities that are not visible to the defenders is extremely difficult, especially while these vulnerabilities are being actively exploited.
While there are many products on the market that provide static or dynamic capabilities to detect vulnerable code for software developers, there is limited assurance provided by the open source community, that these tools are used correctly and thoroughly.

Software patching is possibly one of the most important mitigation available, however this process can be expensive and labour-intensive, especially within a large enterprise that use various technologies and experience Shadow IT (Cisco, 2018)

“Most vulnerabilities are never actually exploited” (Bullough, Yanchenko, Smith, and Zipkin, 2017, p. 1). An enormous amount of resources have been given to vulnerability management. Bullough, Yanchenko, Smith, and Zipkin (2017) attempt to prioritise the remediation of vulnerabilities that are likely to be exploited by using ML and the National Vulnerability Database (NVD) produced by the National Institute of Standards and Technology (NIST).

Further, they report that “while models based on the open source database information examined in this study led to poor prediction performance, more accurate predictions might be possible by exploiting better sources of data. We conclude that models of software vulnerability exploitation based on NVD database entries and social media alone are unlikely to have enough predictive power to be useful in practice” (Bullough, Yanchenko, Smith, and Zipkin, 2017, p. 8).

There are always going to be some system vulnerabilities, and regular or ad-hoc vulnerability assessments are performed to determine a current system's or IT/OT environmental vulnerability level or state, in case some critical patches have been missed or not applied by accident. While protecting systems that are inside and within the IT/OT environment, a devastating mistake can be made where assessments are not performed on the outside, the publicly facing IT/OT surface attack area. The defenders have limited visibility of this area being on the inside and unfortunately the discovered vulnerabilities are obvious to the attackers.

Developing zero-day exploits is difficult, but can be profitable for researchers, where exploits are sold to various bug bounty sites such as Zerodium (2019). This could be one of the reason why this area of security attracts lot of interest with researchers. Research paper by Ding, Wei, Xue, Zhang, Zhang, and Han (2017) focus on exploits by providing a unique tool called SeismoMeter to identify and classify exploits similarly to classification applied for malware by various AV vendors. “No easy methods exist to classify these exploits into meaningful categories and to accelerate diagnosis as well as detailed analysis. SeismoMeter recognises both control-flow-hijacking, and data-only attacks by combining approximate control-flow integrity, fast dynamic taint analysis and API sand-boxing schemes. Once it detects an exploit incident, SeismoMeter

generates a succinct data representation, called an exploit skeleton, to characterise the captured exploit. SeismoMeter then classifies the captured exploits into different exploit families by performing distance computing on the extracted skeletons" (Ding, Wei, Xue, Zhang, Zhang, & Han, 2017, p. 1). The researchers are reporting that proposed solution can be implemented as a practical solution in a real world honeynet environment. SeismoMeter characterises exploits based on the exploit routine instead of vulnerability-specific characteristics. Therefore, SeismoMeter can recognise different exploits that target the same vulnerability. In addition, SeismoMeter can defeat network-level obfuscation and encryption because the classification process does not rely on network data. Furthermore, Ding et al. (2017) report to the best of their knowledge, that their study is the first to classify exploits into representative categories based on target victim program's execution trace.

To create an exploit, the attacker needs to identify an exploitable vulnerability and for this the attacker needs a fuzzer. Li, Ji, Lv, Chen, Chen, Gu, & Wu (2019) propose a mutation-based evolutionary fuzzing system that uses DL, called V-Fuzz. This system is composed of two components: a neural network-based vulnerability prediction model and a vulnerability-oriented evolutionary fuzzer. "Most of the state-of-the-art fuzzers e.g., coverage-based fuzzers mainly focus on how to improve the code coverage. Intuitively, fuzzers with a higher code coverage can potentially find more bugs. Nevertheless, it is not appropriate to treat all codes of the program as equal" (Li, et al., 2019, p. 1). The uniqueness about V-Fuzz is that it performs a vulnerability assessment as to determine which part of the software code is more likely to be vulnerable and it improves efficiency. Li, et al. (2019) further report that program crashes during fuzzing are important to estimate the fuzzer's performance, however not necessarily that all crashes are linked to a vulnerability. In comparison to three other state-of-the-art fuzzers, American fuzzy lop (Zalewski, 2019), AFLfast (Böhme, 2019), and VUzzer (Rawat, et al., 2017). V-Fuzz identified the most unique crashes within 24 hours. Li, et al. leveraged DL over ML because the lack of SME required for feature engineering, pattern detection is not accurate enough and ML is too specific while DL can detect multiple vulnerabilities simultaneously. Furthermore, Li, et al. report there has been no DL approach to prediction of vulnerabilities in binaries to the best of their knowledge. Their research with V-Fuzz discovered 10 CVEs, which 3 of them were unknown (Li, et al., 2019).

Vulnerability management is important mitigation strategy and automation in regards to

patching has improved over the years. Application whitelisitng prevents unauthorised code to be executed and potentially stops exploitation of unknown vulnerabilities, it's database needs to managed manually. Better autonomous controls are needed for run-time applications. One way to reduce vulnerabilities and lower the burden of vulnerability management is to practice secure coding.

2.4.11.4 Secure coding

- Secure coding is the practice that takes security into considerations to develop software free from defects, bugs, vulnerabilities and be resilient to cyber attacks.
- Secure coding standards are set of rules and recommendations that ensures the security of software development (Chess, & West, 2007).

2.4.11.5 Software assurance

- Software assurance (SwA) is the level of confidence that the developed software is free from vulnerabilities, not only during coding, but also design, supply chain, processes and procedures are safe and secure.
- Software assurance provides trustworthiness, that no vulnerabilities exist, predictable execution of functions and conformance to requirements, standards and procedures (Goertzel, Winograd, McKinley, Lyndon, Colon, McGibbon, Fedchak, & Vienneau, 2018).

While secure coding and software assurance provide increased software or application trustworthiness, the application run-time environments of exponential technologies continue to be exposed to threats, that are being mitigated with limited and static isolation defences, such as sand-boxing. One technique that provides integrity verification of software at run-time is software self-cheksumming that is used as an anti-tampering mechanism for protecting against software piracy. “The idea is to compute a hash value from the instructions of the program (or something closely related to those instructions) and ensure that the program continues to function correctly if and only if the computed hash has the expected value. This can be used to protect software against piracy, since any attempt to tamper with the code, e.g., to disable or remove a license check, will be detected during checksumming” (Qiu, Yadegari, Johannesmeyer, Debray, & Su, 2015).

Static and dynamic self-checksumming detect changes during program execution. Static

self-checksumming can not detect changes to the memory image, this can be detected by dynamic self-checksumming that periodically checks the memory image. Qiu, Yadegari, Johannesmeyer, Debray, and Su (2015) in their research describe a dynamic information-flow-based attack that aims to identify and bypass dynamic self-checksumming behavior in software. They developed a proto-type tool to detect self-checksumming that is used with an Intel Pin Tool (Levi, 2012) for execution tracing. Two groups of programs were selected for the experiments, the first group contained open source programs with most likely no self-checksumming defenses and the second group contained programs developed by the researches containing advanced self-checksumming schemes. The first group was used as a baseline to make sure there are no false positives generated by the prototype tool. From their experiments and future work proposal they identified two major disadvantages of the dynamic analysis based approach. The first disadvantage is the limited code coverage and second is the large trace files that can increase storage and processing costs. This research raises the importance of understanding the normal behavioral profile of an application during run-time and any tampering or unknown change to be identified as an anomaly.

2.4.11.6 Application behavioural profiling

Vaas and Happa (2017) conducted an application behaviour profiling, by detecting disguised processes that pretended to be part of the OS. They collect an application fingerprint using psutil (Rodola, 2019) to take virtual memory snapshots of all processes including resource usage with sampling interval of every 60 seconds. The acquired data was taken from a single running system over several days, then used to create a model for anomaly detection. They report that it is possible to distinguish processes via memory fingerprints (Vaas & Happa, 2017).

Das, Joshi and Finin (2017) used system calls in app behavioural analysis and reported not so promising results. The research conducted profiling on 534 mobile apps for Android. The tool strace was used to acquire system calls information of the interaction between the apps and the kernel. User activity with the apps was generated using a tool called, Monkey (2019). The data was further analysed in Weka (Hall, Frank, Holmes, Pfahringer, Reutemann & Witten, 2009), against four different classifiers: Support Vector Machine (SVM), NaiveBayes (NB), Decision tree (J48) and Multilayer Perceptron (MLP). They report that MLP performed marginally better than the others.

Das, Joshi and Finin encountered many challenges with the Android emulator but still managed to collect data for analysis. Unfortunately they report the following: “given the presence of complex app functionality, system calls were not sufficient to be used as the only features in classification of app behaviour” (Das, Joshi & Finin, 2017, p. 1). They go to report an interesting trend with one mobile app that appears to be dominating the app market by providing: “multi-functional apps, sometimes called “super apps” where apps are trying to become the “only” app on your phone by providing a multitude of functionality” (Das, Joshi & Finin, 2017, p. 5). Story originally reported by Wang (2019). While a single app per device might reduce the need for comparing different applications and reducing malware. Application behavioural profiling is still going to be necessary to provide assurance for the application operational integrity. Diversity drives competition, creativity and innovation and this might be a reason as to why this trend might not last.

2.4.11.7 From HIDS to AIS

While a lot of research has been conducted on anomaly based HIDS and NIDS, another system that has generated interest and also detects intrusions are Artificial Immune Systems (AIS) (Forrest, Hofmeyr, Somayaji, & Longstaff, 1996). These are self-monitoring, self-aware and self-healing systems that are based on the biological immune systems and are able to differentiate between sense of self and non-self. They use algorithms, such as Negative Selection Algorithm (NSA) and Clonal Selection Algorithm (CSA) to detect intrusions. The AIS research has identified number of major challenges in cybersecurity systems that still apply today. Systems lack diversity, autonomy, robustness and adaptability (Hofmeyr, & Forrest, 2000).

The challenge with unidentified hosts is the large scale of distinct systems that possibly belong to this group, but are not considered in host-based research or security mitigation. What makes hosts unique might be their hardware, but it is their software that defines their main business purpose. The uncertainty or not clearly defining the current and future host scope only increases the automation challenging in this field.

While the automation challenges identified in this section appear to prevent automation in cybersecurity in their own unique way they all are closely related to anomaly management.

2.5 Anomaly Management

This research identifies anomaly management as the most challenging automation component, not only that this is mostly a manual process that requires subject matter experts, but also most anomaly related research today focuses on detection and ML or data science, rather on the management of the anomaly or the unknown change. Furthermore, this challenge deeply relates to the other components and amplifies their impact. For example, the more fragmented, isolated and diverse an IT/OT environment is, the more difficult is to manage anomalies. The more undefined hosts and open-loop systems an environment has, the more difficult is to manage anomalies. The higher the risk the lower the trust, any unknown changes or anomalies that can tip this balance are managed manually by subject matter experts that are in high demand, a trend and challenge that appears it will only increase in the future.

2.5.1 Cybersecurity Triple Triad

Challenges that prevent automation in cybersecurity, especially anomaly management are heavily influenced by the Confidentiality, Integrity and Availability (CIA) triad. The CIA triad is a security model used for the creation of security policies in the information security domain. This model applies to people, systems, data and processes, or the relationships between them. Anomalies generated by zero-day attacks could be distributed across all of these, but gaining fully effective anomaly based control that provides absolute security (Wright, & Zia, 2011), is an impossible task. In production environments only a fraction of these anomalies are detected, because business processes are inadequately translated or digitised for intrusion inspection by the IDS.

Interruption to the availability of business systems or data caused by a Distributed Denial of Service (DDoS) attack or ransomware is obvious, that is the attacker's intention. Incident response is critical to the interruption of availability of business services, but mitigation of DDoS attacks require investigation of the integrity of data in transit and possibly filter, block or scrub out excessive traffic caused by the attacker. Anomalies related to the integrity of systems, data, people and processes are difficult to detect and manage, this is because the attacker's intent is to not be discovered. Continual system integrity verification could prevent availability interruption and save the application from bringing the host and its core function down, but this is challenging.

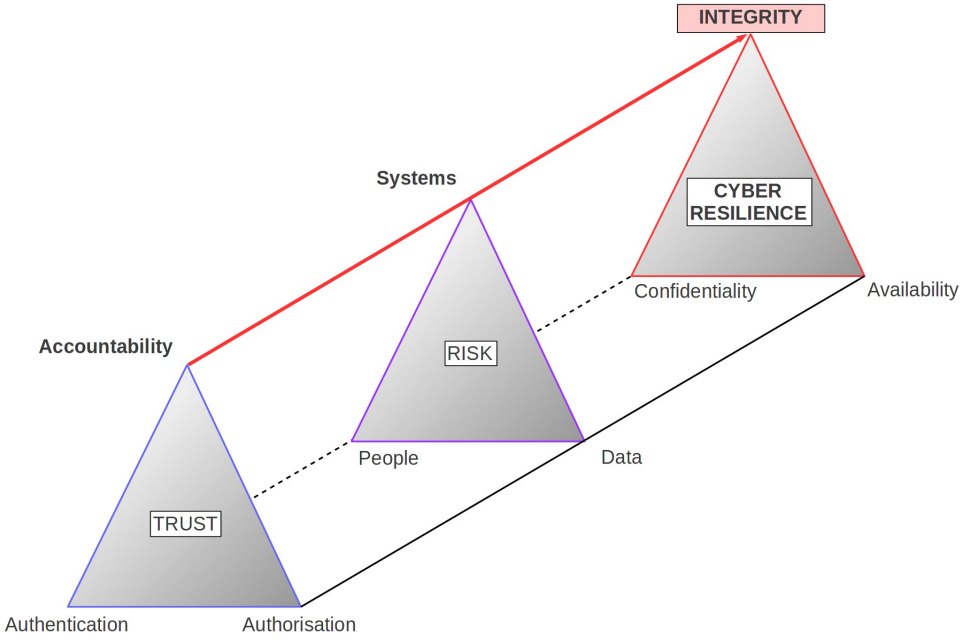

FIGURE 3. Cybersecurity Triple Triad.

Fig. 3. shows a cybersecurity triad with concepts and processes that could be taken into consideration when implementing automation in cybersecurity. In addition, the dynamic relationships between these could also be the source of anomalies. Integrity does not only apply to data; it also applies to people and systems, such as the integrity of software that produces and handles business data.

Authorisation, Authentication and Accountability (AAA) once normally applied to people, but now with the increase of autonomous systems and M2M interactions, AAA could be applicable to machines and an increase in anomalies could be expected by this trend. Furthermore, higher importance could be given to anomaly management relating to attribution and non-repudiation which directly affects the accountability and the level of trust given to a machine, person or organisation.

Table II. show the trust level and corresponding action. While actions related to availability and confidentiality appear to be straightforward, the integrity could be subjective.

TABLE II. CIA Actions

Trust and Actions	Permit	Deny
Availability	ON	OFF
Confidentiality	IN	OUT
Integrity	GOOD	BAD

“An operating system is said to have system integrity when it is designed, implemented and maintained to protect itself against unauthorised access, and does so to the extent that security controls specified for that system cannot be compromised. A multilevel-secure trusted computing base ensures system integrity. The trusted computing base has the ability to protect itself against unauthorised user access (IBM-Integrity, 2020).

Integrity appears to be subjective for software, or operating systems, because integrity is assumed until a vulnerability is discovered, then the system has no, or partial integrity, until patched, then the system has integrity, until another discovered vulnerability, and patch cycle. In this case, if the vulnerability was never discovered, but still there, the system never had true integrity in the first place. System integrity, perhaps is not about protecting one self or system that cannot be compromised, system that can self-patch or self-heal but is about system having a self-control of its own core purpose.

2.5.2 Anomaly, is the unknown change

“Everything changes” (Heraclitus, 2019).

If the known change is the base for the normal operating baseline, perhaps the unknown change is the anomaly that needs to be managed.

The introduction of new systems, applications and new technology in production environments following Information Technology Infrastructure Library (ITIL) change management practices, require a request submission for approval. The manual change management process is undertaken for this request, the Change Advisory Board (CAB) performs a review, risk assessment and approves or rejects the change (AXELOS, 2019). Every approved known change goes towards the new normal operational baseline.

An important point for data science research is that, change management is the manual process, that monitors the change of the normal IT/OT baseline. However, any unknown change that has not been approved, such as Shadow IT (Cisco, 2018), is potentially an anomaly that requires to be manually risk assessed and investigated, which could also

lead to a malicious event, such as data breach.

Authorised changes could also lead to an anomaly, straight after the change or sometime later. The following production teams manage anomalies generated by changes manually:

- IT Operations monitors systems normality.
- Data scientists provide business intelligence.
- Cybersecurity preform incident response.

Anomalies generated by a cyber attack can be temporary or permanent:

- Temporary anomalies are glitches that are accidentally generated by the attacker or by-products of the attack.
- Permanent anomalies like rootkits (Blunden, 2012), such as the Unified Extensible Firmware Interface (UEFI) rootkit (ESET, 2019), can be disastrous because they change the normal business baseline without being detected.

2.5.3 Anomalies and ML

It is evident from the results in Fig. 4., queries performed on Google Scholar that there is an increase in research interest relating to cybersecurity and ML for period between 2014 and 2018.

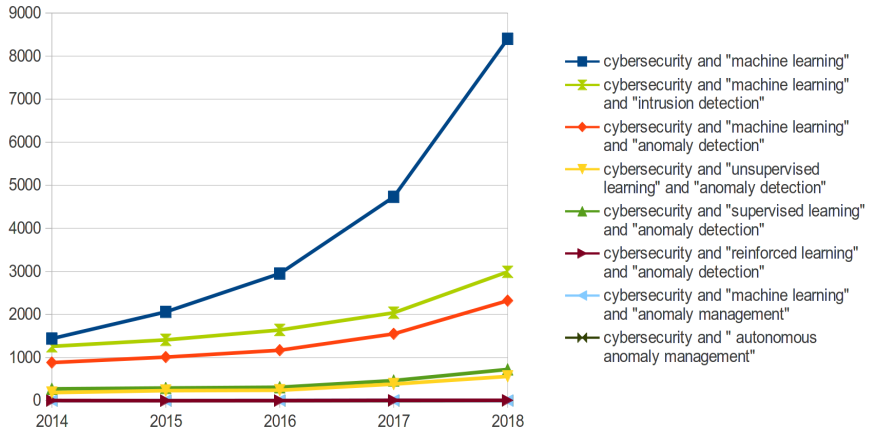

FIGURE 4. ML popularity in cybersecurity.

It appears the term intrusion detection, generates more interest than anomaly detection. The term supervised learning is slightly more popular for anomalies, than unsupervised learning. Results from the queries in 2018, Table III, show low interest in reinforcement learning, cybersecurity and anomalies and only one paper on cybersecurity and autonomous anomaly management (Donevski, & Zia, 2018).

TABLE III. Research interest ML cybersecurity

Queries dated: 31 March 2019	Results
Cybersecurity and “reinforced learning” and “anomaly detection”	7
Cybersecurity and “machine learning” and “anomaly management”	2
Cybersecurity and “autonomous anomaly management”	1

The increased interest in cybersecurity and ML research, matches findings from industry reports the increased interest in AI and ML cybersecurity tools for business. Where ” 71 percent of businesses surveyed in the United States plan to use more artificial intelligence and machine learning (AI/ML) in their cybersecurity tools this year, although over half (58%) aren’t sure what that technology really does” (Webroot, 2017).

2.5.4 Anomaly detection and classification

ML has three learning groups: supervised learning, unsupervised learning and reinforcement learning.

- In supervised learning the data contains the input and the expected output. The data needs to be labelled and classified, such as “normal” or “anomaly”. Because the data contains this information the learning is said to be task-driven or supervised.
- In unsupervised learning the data contains only the input with no expected output. Unsupervised learning is data driven and the algorithms can find patterns and structures in data and order these in groups or clusters by similarity.
- Reinforcement learning has an agent based learning concept, where the agent needs to learn from interaction with its environment. The agent performs an action, that is interpreted into a reward and a state given back to the agent as a feedback.

Buczak and Guven (2016) describe, in their survey, the emerging ML and Data Mining (DM) analytic methods in support for cybersecurity. Systems used to detect cyber intrusions as reported at the minimum are composed of firewall, AV and an IDS which support three main analytics methods: misuse-based or signature-based, anomaly based, and hybrid. They address the complexities of using ML algorithms and DM approaches

to resolve cybersecurity challenges. In their survey they attempted to determine the effectiveness of current and proposed ML algorithms against broad range of network based cyber attacks using mainly the Sponsored by Defense Advanced Research Projects Agency (DARPA), Knowledge Discovery in Databases (KDD) dataset (Tavallaee, Bagheri, Lu, & Ghorbani, 2009), and NetFlow generated data. Buczak and Guven report the issue with public datasets is "the fact that so many papers use the DARPA and KDD data sets is related to how difficult and time-consuming it is to obtain a representative data set. Once such a data set is available, researchers tend to reuse it" (Buczak & Guven, 2016, p. 1171). It appears that supervised learning would be the preferable choice of ML methods if only data scientist had better datasets. "ML and DM methods cannot work without representative data, and it is difficult and time-consuming to obtain such data sets" (Buczak & Guven, 2016, p. 1173). They also report that normally ML models are created and don't change for a long time, however in cyber environment they have to be trained daily or after an intrusion, a time-consuming process that opens possibilities for future research in incremental learning.

All solutions that were analysed were hybrid and Buczak and Guven (2016) didn't come across any only anomaly based systems. In hybrid systems the anomaly detection module classifies the abnormal traffic and the anomaly detector is based on a clustering method. For anomaly detectors they recommend density-based clustering DBSCAN or one-class SVM. Unfortunately, Buczak and Guven (2016) were unable to provide recommendation for the best and most effective method for cyber applications. This was caused by complexities in the research where datasets, algorithms and results were difficult to compare. It appears different methods perform differently with different cyber attacks and different datasets.

Kalinichenko, Shanin, and Taraban (2014) in their survey identified the best methods for anomaly detection. Anomalies are deviant or outlier events that are presented in Metric Data, Evolving Data and Multi-structured Data. Metric Data is the most common representation, where every object in a dataset has a certain set of attributes. Metric Data oriented methods represent the objects in regular and irregular points depending on the distance between each other. Clustering methods use the concept of distance, such as K-nearest neighbours, where the farthest object from its neighbours is more likely to be an anomaly. Evolving Data methods determine an outlier in data presented in sequences.

The deviation is measured by looking at specific positions in the whole sequence by using distance-based, frequency-based and Hidden Markov Model. Anomaly detection in ML can be seen as a classification problem, where it is assumed that the data classes have an inner predictable structure and have the class anomalies. The outlier class modelling can be unproductive because the only prediction that they have is they do not resemble normal data. The anomaly detection has a problem with construction of reliable training datasets for supervised methods due to the rarity of the outliers which could cause high false positive alarm rates (Kalinichenko, Shanin, and Taraban, 2014).

Xin, Kong, Liu, Chen, Li, Zhu, Gao, Hou, and Wang (2018) compare ML and DL methods and attempt to identify which is the better method for cybersecurity research. They report that the main difference between ML and DL is their data dependency where DL does not perform well with small data volumes. In ML the feature processing is time-consuming where the characteristics of an application must be determined by an expert and the performance of the ML algorithm will depend on the accuracy of the feature. All ML methods include the following four steps: feature engineering, choice of ML algorithm, training and evaluation of the model, and use the trained model to classify unknown data. The literature review only focused on three years prior to the research and unfortunately the authors were unable to define which method was better for cybersecurity. Additionally, they report the difficulties of obtaining a good representative dataset for their research (Xin, et al., 2018).

Kolosnjaji, Zarras, Webster, and Eckert (2016) found DL to be more beneficial in malware classification than the classic ML classifiers such as Hidden Markov Models and Support Vector Machines (SVM). They selected system calls as the source for their datasets. “The most important traceable events for determining malware behaviour are system calls. In order to execute malicious actions, malware needs to use the services from the operating system” (Kolosnjaji, Zarras, Webster, & Eckert, 2016, p. 2). By performing behavioural analysis on the malware using system calls they can focus on what the malware does and classify faster than performing static analysis and fighting obfuscation techniques. This approach has its weaknesses and can be prone to blind spots nevertheless, however the authors do acknowledge the importance of static analysis and propose a joint solution in the future.
Kolosnjaji, Zarras, Webster, and Eckert for their research construct deep neural

networks and apply them to analyse system call sequences, where they combine convolutional and recurrent approaches to DL in order to optimise malware classification. Malware gets executed in a safe environment using malware zoo (Malware-Zoo, 2017) and Cuckoo sandbox (Cuckoo, 2017) and every system call the malware makes to the kernel gets used for characterising the malware sample, where every sample contains, hopefully unique, sequence of calls used for the behavioural profiling of that malware. The authors also acknowledge a limitation in their experiment as to introduction of noise in their environment, which potentially could get false readings in the system call sequences, postponed for future research. They report "Our neural network outperforms not only other simpler neural architectures, but also previously widely-used Hidden Markov Models and Support Vector Machines" (Kolosnjaji, Zarras, Webster, & Eckert, 2016, p. 11).

Haq, Onik, Hridoy, Shah, Rafni and Farid (2015) conducted a survey related to 49 studies between 2009 and 2014 focusing on the best or most common used ML classifier for intrusion detection research. They report the following: from a single based classifier the Support Vector Machine (SVM) is by far the most commonly used and the KDD Cup 1999 dataset was most popular, however due to errors and duplicates in it was the reason for creation of the new updated version, NSL-KDD (Tavallaee, Bagheri, Lu, & Ghorbani, 2009).

Xiao, Wan, Lu, Zhang and Wu (2018) use ML to provide security to IoT and identified various attacks such as DDoS, jamming, spoofing, malware and eavesdropping are used to compromise a network of IoT systems. They report that various ML techniques such as unsupervised, supervised and RL have been applied to improve the network security by using anti-jamming and secure offloading. Furthermore, they report their observation on a problem related to RL based security schemes, the agent assumes an accurate state and has no assurance or verification of this state, which follows with reward for each action in time. Computational overhead when dealing with some ML-based security schemes was also an issue. They also report that "supervised and unsupervised learning sometimes fails to detect the attacks due to oversampling, insufficient training data and bad feature extraction" (Xiao, Wan, Lu, Zhang & Wu, 2018).

Mazel, Owezarski and Labit (2010) propose a two-step automated mechanism for detecting zero-day attacks, where the IDS with assistance of the proposed anomaly detection system (ADS) detects an anomaly in network traffic. This anomaly in the research is caused by a Distributed Denial of Service (DDoS) attack, the traffic is then been analysed and classified as an attack with the help of ML algorithms.
Yuan, Lu and Xue (2016) use a novel approach to develop a tool in identifying malware on android devices, called DroidDetector by using DL and the researchers report that it can achieve 96.76% malware detection accuracy which is much higher than the traditional signature based AV products.

A framework called CANTINA+ by Xiang, Hong, Rose, and Cranor (2011) uses ML for identifying phishing websites, these are fake websites used by criminals created to look the same as a legitimate website that require user information such as bank websites. The researchers achieved over 92% true positive detection and demonstrated to be a competitive anti-phishing solution.
Bilge, Sen, Balzarotti, Kirda and Kruegel (2014) in their research develop a system named EXPOSURE, that uses ML to passively analyse DNS traffic, detects, and reports malicious domains. The researchers reported good results against different DNS techniques used by criminals, including the Domain Generation Algorithm (DGA) attack, used to generated malicious domain names automatically.

A survey conducted by Kaur and Singh (2014) outlined the research efforts in relation to detection of zero-day attacks or zero-day malware in form of polymorphic worms or worms that change their layout after each infection. To deal with this challenge two different detection techniques are utilised, network-based systems that detect worms at the network level and host-based systems that detect worms at the system level. Their survey focused on the network hybrid signature-based systems. They initially report limited success to some attacks by introduction of dynamic solutions that look for anomalies and patterns, such in the case of Bayes signatures “which do not look for exact matching, they are resilient to red herring attacks, where a worm initially includes some garbage tokens in its payload so that these tokens are incorporated into a signature. Then after some time, the worm stops including such garbage tokens. Such an attack render the other two types of signatures useless in most cases” (Kaur and Singh, 2014, p. 1524).

Duessel, Gehl, Flegel, Dietrich and Meier (2017) propose detection of zero-day attacks by focusing on context-aware anomalies at the application layer. The focus of this research is the detection of anomalies triggered by attacks on network protocols such as Hypertext Transfer Protocol (HTTP) and Remote Procedure Call (RPC). Experiments were undertaken off-line using datasets for these two protocols and additionally using protocol analyzer, to extract and parse application layer messages. Attacks were generated using the Metasploit tool (Rapid7, 2017), and information provided in common security forums, with focus on SQL injections and buffer overflows. They present a new data representation, called cn-grams, that allows integrating syntactic and sequential features of payloads in a unified feature space and provides the basis for context-aware detection of network intrusions. The results of their experiments on both text-based and binary application-layer protocols demonstrates superior accuracy on the detection of various types of attacks over regular anomaly detection methods. The detection accuracy was boosted from 44% using unsupervised anomaly detection with plain sequential features to 80% using the proposed combined features. Duessel, Gehl, Flegel, Dietrich and Meier (2017) reported the importance of managing risk presented by vulnerable third party applications, which requires isolation of applications in virtual, container, or sandbox environments to reduce the risk of propagating.

Graham, Liang, Gruenwald and Grant (2017) report, while there has been great advancement with automation of discovery using algorithms, analysts are still required to manually configure parameters to keep these solutions accurate. While this is known in the data science as the human “in-the-loop”, they propose another concept as Human “over-the-loop” Analytics (HOLA) (Graham, Liang, Gruenwald, & Grant, 2017). This new concept could automate, this normally manual process. The proposed human-interactive algorithms will be based on one of three architectures: interactive, anytime, and interruptible. The proposed architecture for this solution was inspired by the system call interface and the division of the user/kernel spaces. Their goal is to have a collaborative system that acts like the kernel where multiple users can experiment on single dataset spontaneously which will prevent them from running algorithms one at the time.

2.5.5 Reinforcement Learning (RL)

RL has two model categories: Model-Based and Model-Free RL. Model-based RL is used when the agent has knowledge about its environment and state transitions. Model-free RL is used when the agent has no knowledge about its environment.

RL algorithms can also be active and passive depending on how the agent is learning from its environment. In passive RL the agent learns while having a fixed policy and ,active RL, the agent learns of which action is best in a given state. The agent can have any combinations for both such as Model-Free RL with Active RL. Two very important approaches in RL is the concept of exploration and exploitation (Ravishankar, & Vijayakumar, 2017).

Dayan and Niv (2008) in their research, with paper titled "Reinforcement learning: The Good, The Bad and The Ugly" point out some challenges in RL.'The Bad': relates to the balance between exploitation and exploration and the uncertainty of the exploration. Most work in RL focuses on exploitation or using past experience to optimise outcomes. More ambitious agents might also seek exploration, taking the potential benefits of learning its environment, new rewards and punishments, however there is risk associated with the sensitivity of choice.

Anderson, Kharkar, Filar, Evans, and Roth (2018) present a RL solution for bypassing ML models based on static features. Their focus is on static Windows PE malware evasions and their RL environment has been released as open source on OpenAI gym (OpenAI, 2017) for other researchers for further research. In their experiment they train a RL agent to learn an action policy where "the actions space A consists of a set of modifications to the PE file that (a) do not break the PE file format, and (b) do not alter the intended functionality of the malware sample" (Anderson, Kharkar, Filar, Evans, & Roth, 2018, p. 5). The RL environment is set as, one malware sample per game, and the agent can perform 10 PE modifications before declaring failure. Every time the modification to the PE file gets marked as malicious by the anti malware engine reward of 0 is fed back to the agent, if the modification passes as benign the agent gets reward R which is 10 in their environment. While they report a modest evasion results, this is a novel approach in malware evasion that could benefit other researchers in the future. While this technique would normally be used by an attacker, the RL environments and game theory is also suited for defensive mitigation strategies. The closing loop and provision of feedback is what security controls are lacking today.

Böttinger, Godefroid, and Singh (2018) formalise fuzzing as a reinforcement learning problem, where the fuzzing is modelled as a learning process with a feedback loop. The initial input is a Portable Document Format (PDF), where the fuzzer generates a new mutated input file represented as binary strings for the targeted program, pdftotext, for each learning cycle. Their research contains three main streams: fuzzing, grammar reconstruction, and deep Q-learning. Böttinger, Godefroid, and Singh select the grammar-based fuzzing over the blackbox and whitebox fuzzing, due to its proven effectiveness with applications that have complex structured input formats. Further, they report to the best of their knowledge their solution is novel because it uses character-based language models to learn a generative model of fuzzing inputs with reinforcement learning. Furthermore, they present empirical evidence that reinforcement fuzzing can outperform baseline random fuzzing (Böttinger, Godefroid, & Singh, 2018).

2.5.6 Entropy based anomaly detection

Entropy, a measure of uncertainty in a given system, has been identified by number of research papers as the solution for identifying anomalies, information entropy was first introduced by Shannon (1948). Entropy is a concept in information theory that measures the impurity of a collection of data items.

Lee and Xiang (2001) report, that much research in anomaly detection focuses on a specific method for a certain environment and does not attempt to solve the fundamental problems in this domain. In their research they propose information theoretic measures: conditional entropy, relative conditional entropy, information gain, and information cost for anomaly detection. Lee and Xiang report that entropy can be used to measure the regularity of a dataset and conditional entropy can be used to measure the regularity on sequential dependencies with good results.

Acker (2015) proposes a solution to minimise the number of features required in a set of intrusion detection dataset by using different types of entropy calculations. Acker runs his experiments against the KDD Cup 99 dataset using the Weka ML tool. Feature selection is a process of identifying relevant features from a set of data and the reason for feature selection is to reduce the size of the dataset and make it more efficient. Acker uses the C4.5 classification tree that uses entropy, originally developed by Quinlan (1993). C4.5 has been implemented in the WEKA analysis package as the J48 module.

Iwamoto and Wasaki (2015) develop a tool for detection and extraction of embedded malicious shellcodes in portable Microsoft documents using entropy. They report good results from 88 malware sample that were being analysed, resulted to extraction of 51 shellcodes. The downside reported was that this solution doesn't work on malicious documents without shellcode and Return-Oriented Programming (ROP) techniques (Shacham, 2007).

Eskandari, Khreich, Murtaza, Hamou-Lhadj, and Couture (2013) have identified that most anomaly detection research that monitors for deviation by observing system calls have not taken into consideration memory security controls such as Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP) techniques.

ASLR introduces randomization or diversity to memory addresses used by the target process and removing the ability for the attacker to find memory addresses statically (Sood, & Enbody, 2014). The higher the entropy the more effective the ASLR is against memory attacks, such as ASLR on 64-bit systems are more effective in comparison to ASLR on 32-bit systems (Ganz, & Peisert, 2017).

DEP can be hardware and software control that prevents the attacker from executing code from a non-executable memory location (Sood, & Enbody, 2014). ROP is the technique that is used by attackers to bypass ASLR and DEP (Eskandari, Khreich, Murtaza, Hamou-Lhadj, & Couture, 2013).

Li, hu, Fu, Chen, Zhu, and Liu (2018) introduce ROPNN that uses Deep Neural Networks to detect ROP payloads in HTTP requests, PDF files, and images. They propose to use convolutional neural network (ConvNet) as their classifier as it is good at capturing spatial structure and local dependencies. To the best of their knowledge, they report that ROPNN is the first intrusion detection method that applies Deep Learning (DL) to mitigate the threat of ROP attacks. ROPNN was designed to run on Linux and was tested on nginx (2020) and three other applications. Originally Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP) were tested because they are widely used classifiers. However, their results showed that SVM suffers 22.2% and MLP suffers 42.4% from false positive rates. Furthermore, subsequent evaluation results with ROPNN, they achieved high detection rate (98.3%) and low false positive rate (0.01%) (Li, hu, Fu, Chen, Zhu, & Liu, 2018).

Mannaert, De Bruyn and Verelst (2012) explore how entropy can be used for software maintenance and evolvability. They report that the more complex the software system is the higher the entropy. The entropy also potentially increases with the age of the software. They define two states macrostates and microstates, where a macrostate is the general, observable state of the system, such as logging, and microstates are instructions and data registers in the processor.
Furthermore, they report “it seems more promising to apply the concept of entropy to the run-time analysis of a software system, than to the compile-time analysis of the source code” (Mannaert, De Bruyn & Verelst, 2012, p. 95).

Having a good and continual understanding of the business normal IT/OT environment is critical to effective change and anomaly management. Identifying the system normality is important to both the attackers and the defenders. The attackers need to know the system normality before looking for exploitable vulnerabilities, where anomalies are purposely generated outside of that expected normality. Defenders also need to know the system normality when looking for anomalies and IoC. Request for Comment (RFC) (IETF, 2019), are memos of standards from the Internet Engineering Task Force (IETF) that provide good reference point for Internet related technologies and their normal operational specifications.

Zhang and Veitch (2011) examine entropy based anomaly detection research and illustrated blind spots, by presenting their camouflage technique to evade such defence mechanisms. The behaviour of Shannon entropy, they point out, it suffers from number of weaknesses when used to defend against network-based cyber attacks and that previous successful entropy based anomaly research didn't address two assumptions that are challenged in their research.
The entropy detects only significant changes in distribution and attackers, have not applied any strategies to evade entropy based anomaly detection. Zhang and Veitch note that entropy is blind to order, detects only heavy hitters and provides only a coarse summary of data. They additionally present quantitative research for behaviour of entropy as a function of the underlying distribution shape, where the compared shapes have similar and close probability ranges.

2.6 Cyber Ecology

Tremendous amount of research has been conducted, where solutions for cyber problems are found by looking at the nature's own way of solving problems. One such research field is Cyber Ecology, where biology, epidemiology, and ecology are applied to cybersecurity problems.

Jorgensen, Rossignol, Takikawa and Upper (2001) in their research are using ecology for cybersecurity solutions for provision of information assurance. They are analysing different ecological environments and trying to identify how these environments might apply to cyber environments. For example, different ecological models might apply differently to a single unit, organism or group of organisms. An installed program or an application's life-cycle might be seen as the life-cycle of an individual. The person is born, the application is installed. The person dies, the application is removed. The application execution cycle, could also be compared to an individual where, the application is executed, the person is born. The application execution stops, the person is dead. Cyber parasites are also compared to viruses and worms that do not inflict damage. Non-intimate cyber parasites can live in close proximity to the host without causing major harm.

Further research by Jorgensen and Rossignol (2003) present Cyber Ecology as a cross-disciplinary synthesis of these fields, toward development of information assurance for more resilient cybersecurity solutions. They define the scope of Cyber ecology, classification of malware and insider threat, by using ecological concepts and epidemiological applications. Information assurance, the ability to process information with desired results, maintain system, and be able to recover from failures was compared to system's health that has a vigor relating to performance and resilience that relates to recovery. Monitoring of system health has been proposed, by using sentinel species or cyber parasitism for indicators of health (IoH) in an ecosystem.

Research by Böhme (2018) attempts to solve fundamental challenges with software testing tools and processes. The limitations of fuzzing are being presented and addressed with a new framework called Software Testing and Analysis as Discovery of Species (STADS). This new framework applies ecological solutions to fuzzing problems such code coverage, time estimation and risk assessment. The main purpose for fuzzing

is to discover vulnerabilities, however as Böhme (2018) points out, it doesn't mean if the tool didn't detect any vulnerabilities, that there are no vulnerabilities in the software that has been tested. STADS framework provides a dynamic risk assessment of this problem, where security researchers can run a risk assessment report of probability of any detectable leftover vulnerabilities. An extrapolation is also used in STADS to predict the time and resources that are needed for detecting the next vulnerability during fuzzing.

2.7 Conclusion

This review of automation challenges in cybersecurity identified shortage of SMEs, not only in cybersecurity, but also in data science. In order to build new autonomous solutions, experts from both domains are needed, and this is a challenge that it appears will only escalate. Fragmented environments cause disruptions in connectivity, which is required for cohesiveness and automation of organisational processes. Managing databases of IoC is a manual process, that prevents automation. Self-populating databases are necessary for autonomous cybersecurity learning systems. The creation of trustworthy ML datasets, that are needed for models, used by learning systems, is a manual and an expensive process.

Risk management is a complex challenge that involves manual processes. Every user and company might have a different risk appetite, and this will require different autonomous solutions. Trust assignment or managing trust is mostly a manual process, that could follow the risk assessment process. While AML research questions results of ML solutions, their predictions, classifiers and data integrity. By trying to provide current security controls to protect classifiers and data integrity will only cause obstruction in development of autonomous solutions.

While system diversification increases resilience, by introducing uncertainty to the attacker, it also prevents automation, because automation needs predictable processes. Isolation as one of the mechanisms to defend against cyber attacks, prevent automation for learning systems because of reduced connectivity.
Open loop systems prevent automation, because they require a man in the loop to

manage anomalies. Closed loop systems are need for autonomous learning, they have a feedback loop that provides a reward or punishment for every action taken by the system or agent.

Log based security controls, rely heavily on constant feeds from systems, and application, mostly human-readable, logs. The challenge with semantics, and availability of logs, could present a problem for automation. Using system calls, could be a more reliable option. HIDS research, has limited scope and coverage of systems. This research provides limited solutions to fast expanding IT/OT environments. Managing anomalies in these complex digital environments is going to be a challenge. Software assurance and integrity of business applications, is possibly the biggest challenge for autonomous anomaly management, with cybersecurity learning systems,

This review also confirms previous report findings of Forrest, Hofmeyr and Somayaji where they note: “Although there has been extensive research into methods for intrusion detection using system calls, there has been much less work on how to respond to detected anomalies” (Forrest, Hofmeyr, & Somayaji, 2008, p. 8). While this report was produced almost 10 years ago it appears it is still much applicable. With the additional finding of, the demand for good quality datasets reflecting the real modern zero-day cyber attacks.

Review of entropy based anomaly detection research reports various success rates, but this could be a solution to provide the link between automation and cybersecurity learning systems.

The next chapters and experiments, work on the challenges noted in this review and take these into consideration when developing Cyber Diversity Index for autonomous anomaly management, where the focus is on beneficial anomalies from three different points of view business opportunities, system administration and cybersecurity. AutoML in Weka is tested and Diversity Indexes are used to compare different applications and measure and detect unknown system change.

Chapter 3

3. Automation and learning systems

Number of challenges and complexities were identified in the review that prevent automation in cybersecurity. Relevant research and industry solutions, have also been presented that could assist technologies and processes for automation. However, introducing ML to cybersecurity introduces an additional layer of manual processes that haven't been taken into consideration. Researchers are working on solutions to automate ML processes for various domains, including cybersecurity.

Great deal of research has been undertaken in determining the best algorithms for given datasets and data scientists are working on new and state of the art algorithms. Demand for automation and practical usage of ML in production environment by SME to use already established algorithms has also triggered development of new solutions. Automated Machine Learning (AutoML) is the process of automating the end-to-end process of applying ML to real-world problems.

In this chapter an attempt is made to answer the second research sub-question and to determine the level of automation ML provides in cybersecurity research and to the production environments. We know that ML improves the automation of discovery of anomalies, but does ML improve or hinders automation in cybersecurity at a general and more holistic level.

In this experiment AutoWEKA (Kotthoff, Thornton, Hoos, Hutter, & Leyton-Brown, 2017) is tested against number of classifiers including SysFor and CSForrest with the NSL-KDD cybersecurity dataset. This experiment adds new results of AutoWeka, Sysfor and CSForest to previous research by Devi and Abualkibash (2019) conducted to test classifiers with the NSL-KDD dataset.

3.1 Introduction

AutoWEKA (Kotthoff, Thornton, Hoos, Hutter, & Leyton-Brown, 2017) is part of the AutoML framework, an open-source machine learning platform designed to help

domain experts by automatically searching throughout WEKA's learning algorithms and their respective hyperparameter settings to maximise performance, using a state-of-the-art Bayesian optimisation method. AutoWEKA is tightly integrated with WEKA (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten (2009).

3.2 Dataset NSL-KDD description

NSL-KDD (Tavallaee, Bagheri, Lu, & Ghorbani, 2009) dataset has derived from the KDD Cup 99 dataset used in the KDD Cup Challenge. However, this dataset become widely discredited due to inadequately capturing the real problems and exposing a major challenge of the constant requirement for development of new ML datasets assisting the cybersecurity domain.

NSL-KDD is a dataset created to solve some of the inherent problems reported of the KDD Cup 99 dataset. This is a labelled dataset, binary class dataset divided instances labelled as normal or anomaly. This dataset has 42 attributes and 125973 instances, which 67343 are labelled as normal and 58630 are labelled as anomaly.

The NSL-KDD has number of improvements to the KDD'99 dataset such as, it does not include redundant or duplicate records. Data file contained in the NSL-KDD package provided:

- KDDTrain+.ARFF: The full NSL-KDD train set with binary labels in ARFF.
- KDDTrain+.TXT: The full NSL-KDD train set including attack-type labels.
- KDDTest+.ARFF: The full NSL-KDD test set with binary labels in ARFF.
- KDDTest+.TXT: The full NSL-KDD test set including attack-type labels.

Sample of attributes contained in the dataset as seen in Table IV shows data related to network traffic such as type of protocol, service, number of failed login attempts, root access, escalated privileges and the class of the event as to normal or anomaly :

TABLE IV. Sample of NSL-KDD Attributes

Sample of NSL-KDD Attributes	
Protocol type	{'tcp','udp', 'icmp'}
service	{'aol', 'auth', 'bgp', 'courier', 'csnet_ns', 'ctf', 'daytime', 'discard', 'domain', 'domain_u', 'echo',

	'eco_i', 'ecr_i', 'efs', 'exec', 'finger', 'ftp', 'ftp_data', 'gopher', 'harvest', 'hostnames', 'http', 'http_2784', 'http_443', 'http_8001', 'imap4', 'IRC', 'iso_tsap', 'klogin', 'kshell', 'ldap', 'link', 'login', 'mtp', 'name', 'netbios_dgm', 'netbios_ns', 'netbios_ssn', 'netstat', 'nnsp', 'nntp', 'ntp_u', 'other', 'pm_dump', 'pop_2', 'pop_3', 'printer', 'private', 'red_i', 'remote_job', 'rje', 'shell', 'smtp', 'sql_net', 'ssh', 'sunrpc', 'supdup', 'systat', 'telnet', 'tftp_u', 'tim_i', 'time', 'urh_i', 'urp_i', 'uucp', 'uucp_path', 'vmnet', 'whois', 'X11', 'Z39_50'}
num_failed_logins	real
root_shell	real
su_attempted	real
class	{'normal', 'anomaly'}

3.3 Dataset Algorithms description

The following ML algorithms were used in the experiment against the dataset:

- **ZeroR** algorithm has been selected as the baseline for the dataset and the measure by which all algorithms can be compared. The ZeroR algorithm doesn't count towards the research results, it's only used as a baseline.
- **Naive Bayes** (NB) is a technique for constructing ML classifiers and are very popular with researchers and have been used successfully in many complex real-world situations, such as defeating email Spam. One particular good advantage classifier has, is that it requires a small number of training data in order to estimate the parameters necessary for classification. The NB algorithm originates from the NB theorem, where assumptions are made that all attributes are independent of each other.

- **Bayes Network**, name in Weka: weka classifiers bayes BayesNet

  “Bayes Network learning using various search algorithms and quality measures. Base class for a Bayes Network classifier. Provides data structures (network structure, conditional probability distributions, etc.) and facilities common to Bayes Network learning algorithms like K2 and B” (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009).
- **Decision Tree J48** is an implementation of the C4.8 algorithm that is an extension of the popular algorithm C4.5 developed by Ross Quinlan but applicable for Java, hence the name J48. This algorithm is packaged in the Weka classifiers as weka.classifiers.trees.J48. (Quinlan, 1993)
- **Decision Table,** name in Weka: weka.classifiers.rules.DecisionTable

  Class for building and using a simple decision table majority classifier. For more information see: Ron Kohavi: The Power of Decision Tables. In: 8th European Conference on Machine Learning, 174-189, 1995. (Kohavi, 1995).
- **CSForest**, name in Weka weka classifiers trees CSForest

  “Implementation of the cost sensitive decision forest algorithm CSForest, which was published in: Siers, M. J., & Islam, M. Z. (2015). Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem” (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009).
- **SysFor**, name in Weka: weka classifiers trees SysFor

  “Implementation of the decision forest algorithm SysFor, which was published in: Md Zahidul Islam and Helen Giggins: Knowledge Discovery through SysFor - a Systematically Developed Forest of Multiple Decision Trees In: Ninth Australasian Data Mining Conference, 195-204, 2011” (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009).
- **Auto-Weka** is a system designed to help users by automatically searching through the WEKA's learning algorithms and their respective hyper-parameter settings in order to maximize performance, using a state-of-the-art Bayesian optimization method. (Kotthoff, Thornton, Hoos, Hutter & Leyton-Brown, 2017)

The experiments with NSL-KDD dataset produced results shown in Table V, the CSForest classifier correctly classified 87.4157 % of instances, and SysFor correctly

classified 80.0169 % with the longest time of 1543.78 seconds taken to build the model. Both classifiers produced good results for NSL-KDD dataset from the seven tested. The Auto-Weka, automatically selected the Random Forest for the best model for this dataset with 99.4935 % correctly classified instances.

TABLE V. NSL-KDD Dataset with ML models

NSL-KDD Dataset				
Classifiers	Testing	Correctly Classified	Incorrectly Classified	Time taken to build model:
ZeroR	Test set supplied	43.0758%	56.9242 %	0.09 seconds
Naive Bayes	Test set supplied	76.1178 %	23.8822 %	0.47 seconds
Bayes Network	Test set supplied	74.4322 %	25.5678 %	6.13 seconds
J48	Test set supplied	81.5339 %	18.4661 %	18.06 seconds
Decision Table	Test set supplied	72.5958 %	27.4042 %	78.87 seconds
CSForest	Test set supplied	87.4157 %	12.5843 %	539.88 seconds
SysFor	Test set supplied	80.0169 %	19.9831 %	1543.78 seconds
Auto-Weka	Test set supplied	99.4935 %	0.5065 %	1069.99 seconds

3.4 ROC curve graphical plots for NSL-KDD

Receiver Operating Characteristic (ROC) curve, graphical plots were conducted on datasets NSL-KDD to illustrate the diagnostic ability of their binary classifier systems. The ROC curves were created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). The accuracy was measured by the area under the ROC, with area of 1 being the perfect accuracy.

- NSL-KDD with Naive Bayes

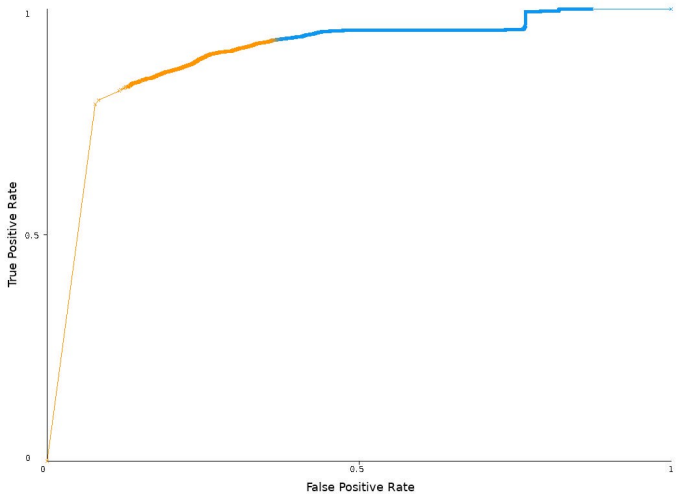

FIGURE 5. Area under the ROC for NSL-KDD with Naive Bayes

The Fig. 5. shows the area under the ROC plot, where the Weka classifier Naive Bayes was used on the NSDL-KDD dataset. The Plot Area under ROC = 0.9171, shows anomaly accuracy of this ML model for this dataset.

- NSL-KDD with Bayes Network

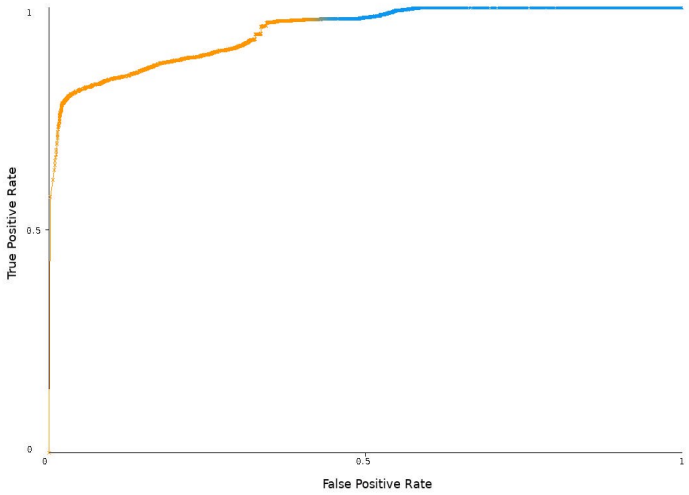

FIGURE 6. Area under the ROC for NSL-KDD with Bayes Network

The Fig. 6 shows the area under the ROC plot, where the Weka classifier Bayes Network was used on the NSDL-KDD dataset. The Plot Area under ROC = 0.945, shows anomaly accuracy of this ML model for this dataset.

- NSL-KDD with J48

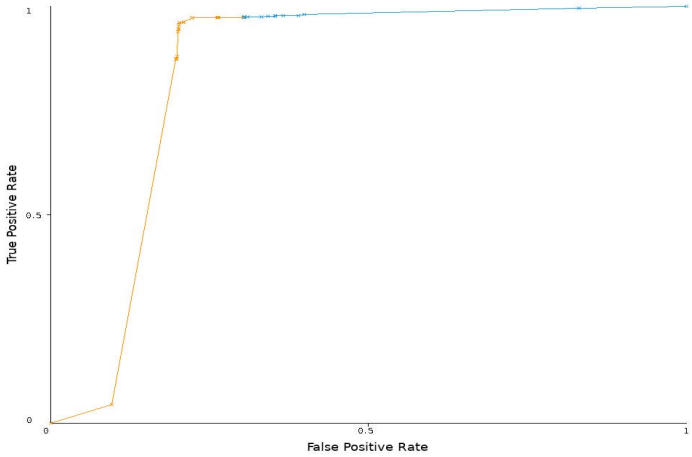

FIGURE 7. Area under the ROC for NSL-KDD with J48

The Fig. 7 shows the area under the ROC plot, where the Weka classifier J48s was used on the NSDL-KDD dataset. The Plot Area under ROC = 0.8401, shows that this

classifier has lower anomaly accuracy than the previous classifiers.

- NSL-KDD with Decision Table

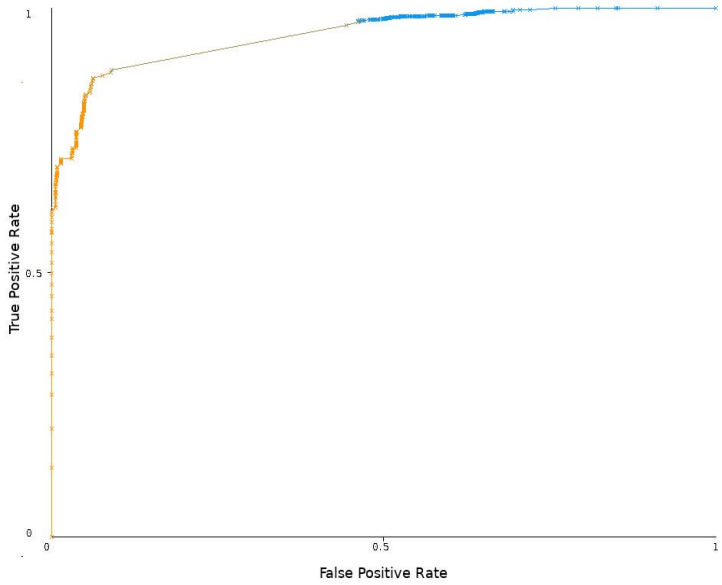

FIGURE 8. Area under the ROC for NSL-KDD with Decision Table

The Fig. 8 shows the area under the ROC plot, where the Weka classifier Decision Table was used on the NSDL-KDD dataset. The Plot Area under ROC = 0.9171, shows anomaly accuracy of this ML model for this dataset.

- NSL-KDD with CSForest

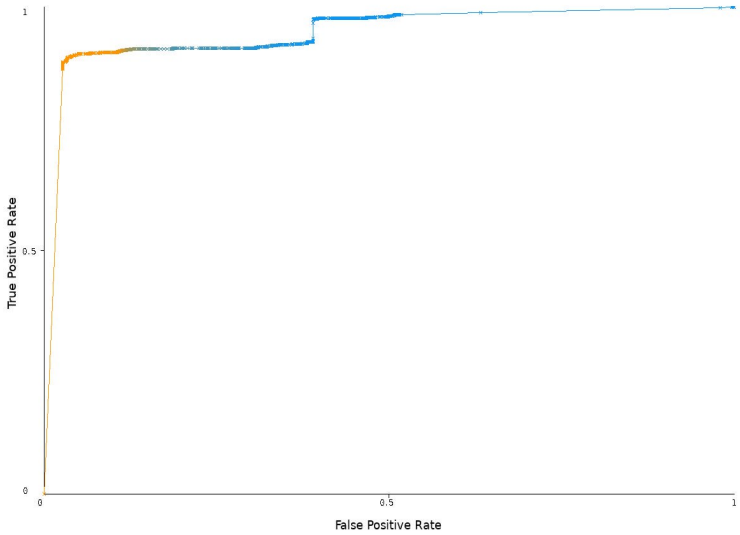

FIGURE 9. Area under the ROC for NSL-KDD with CSForest

The Fig. 9 shows the area under the ROC plot, where the Weka classifier CSForest was used on the NSDL-KDD dataset. The Plot Area under ROC = 0.947, shows good

anomaly accuracy of this ML model for this dataset.

- NSL-KDD with SysFor

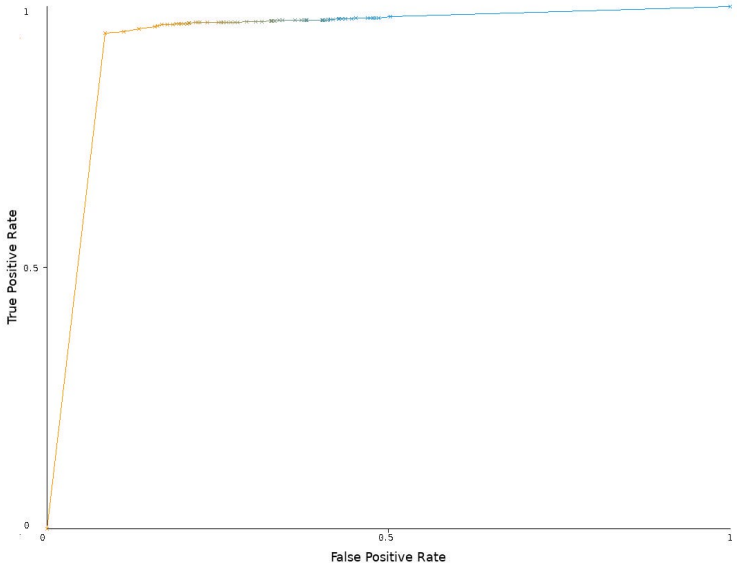

FIGURE 10. Area under the ROC for NSL-KDD with SysFor

The Fig. 10 shows the area under the ROC plot, where the Weka classifier SysFor was used on the NSDL-KDD dataset. The Plot Area under ROC = 0.937, shows anomaly accuracy of this ML model for this dataset.

- NSL-KDD with Auto-Weka

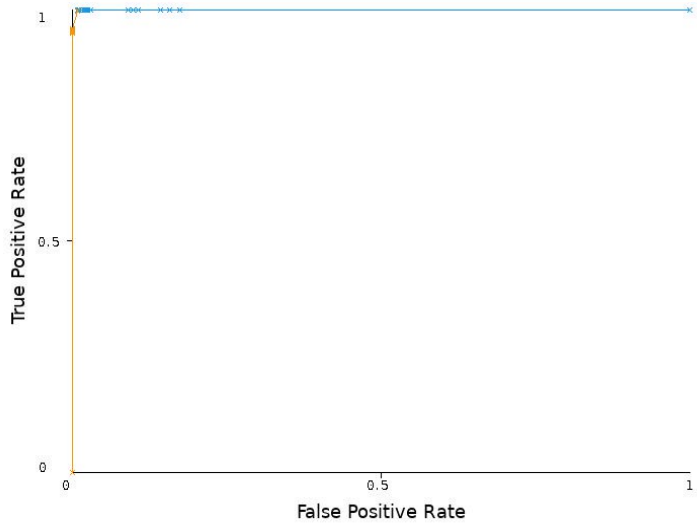

FIGURE 11. Area under the ROC for NSL-KDD with Auto-Weka

The Fig. 11 shows the area under the ROC plot, where the Auto-Weka classifier automatically selected Random Forest and used it on the NSDL-KDD dataset. The Plot Area under ROC = 0.9998, shows the best anomaly accuracy of this ML model for this

dataset in comparison to the other models.

3.5 SWOT and Confusion Matrix

There are four possible events that can be identified in cybersecurity:

- True positives (events correctly identified as threat), local intelligence.
- True negatives (events correctly identified as not a threat).
- False negatives (events incorrectly identified as benign) threats never seen before, possible mimicry and zero-day attacks.
- False positives (events incorrectly identified as threats but are benign)

TABLE VI. Security Events - SWOT

External	**Opportunities**	**Threats**
	True Positive	False Negative
Internal	**Strengths**	**Weaknesses**
	True Negatives	False Positive

Table VI. shows that the most harmful events are the false negatives. These events have been missed by all security controls and pose the highest risk. The second most harmful events are the false positives, these events are the most time-consuming, cause staff burnout and adds to the shortage of SMEs. The main challenge is how to eliminate false negatives and reduce false positives. The reconnaissance or the first stage of an attack is the stage that has in general the most false positives that get neglected due to shortage of resources. If more focus was placed in detecting IoA at this stage we could potentially reduce false positives and get closer to eliminating false negatives.

An example of a false negative would be if the attacker tweaks or mutates a malware in order to obfuscate the malicious code with the aim to bypass Anti Virus (AV) controls (Christiansen, 2010, p11). This would be considered as a false negative, the AV controls failed to detect the malware.

An example of a false positive would be if a software vendor uses compression to package their code, which could be detected as malicious by AV products because malware writers also use compression to hide their own code. The AV controls identify

this software as malware, yet this software doesn't contain any malicious code, this event would be considered as a false positive.

In this experiment some algorithms are more accurate and have lower numbers of false positives as seen in the confusion matrix for the NSL-KDD dataset in Fig. 12. This experiment uses supervised learning, that has traditionally less false positives than unsupervised learning. In production environments however unsupervised learning is more popular but this learner is prone to more anomalies and false positives, due to unknown changes and Shadow IT.

The exploitation stage, the stage analyzed in this research, could be prone to false negatives, thus the threat of identifying IoA at this stage is higher. Future research will be required as to getting broad visibility of IoA and not just mIoX but across all stages of a cyber attack. The mIoX will only be a small part of the list of IoA that would have large coverage across the cyber kill chain.

The SWOT of cybersecurity events corresponds to the Confusion matrix.

Classifier Used	**Confusion Matrix**			**Accuracy**	
Naive Bayes		Normal	Anomaly	Correct instances	Incorrect instances
	P	9041	670	17160	5348
	N	4714	8119		
Bayes Network	P	9449	262	16780	5764
	N	5502	7331		
J48	P	9448	263	18381	4163
	N	3900	8933		
Decision Table	P	9455	256	16366	6178
	N	5922	6911		
CSForest	P	8868	843	19707	2837
	N	1994	10839		
SysFor	P	9432	279	18039	4505
	N	4226	8607		
AutoWeka	P	67229	114	125335	638
	N	524	58106		

FIGURE 12. Confusion Matrix for NSL-KDD

Fig. 12 shows the confusion matrix for NSL-KDD and the accuracy of each selected classifier. The Decision Table was the least accurate classifier with 5922 (normal) and

256 (anomalies), or total 6178 instances were incorrectly classified with this model. CSForest accuracy shows 1994 (normal) and 843 (anomalies), or total 2837 instances were incorrectly classified with this model. The SysFor shows 4226 (normal) and 279 (anomalies), or total 4505 instances were incorrectly classified with this model. The most accurate classifier Auto-Weka (Random Forest) shows 524 (normal) and 114 (anomalies), or total of only 638 instances were incorrectly classified with this model.

In the experiment Auto-Weka uses a state-of-the-art Bayesian optimisation method that automatically selected Random Forest. This model correctly classified 99.5 instances and was the best classifier model for the NSL-KDD dataset, which matches research findings conducted by Devi and Abualkibash (2019). Random Forest is a classifier that consists of many different decision trees selected by using random subsets of training data in an attempt to reduce the over-fitting and high bias that decision trees could be prone to. The results from the experiment show that the other tree based models also performed better than the Bayesian models for this dataset.

CSForest (Siers, & Islam, 2015), as the second best classifier model for the NSL-KDD dataset was originally developed to identify software defect prediction using a cost sensitive decision forest and voting. The original structure was taken from a previous classifier model SysFor (Islam, & Giggins, 2011) that utilises J48, which is a C4.5 decision tree for Weka developed in Java. This classifier is based on information theory that uses information gain and entropy to split distribution on the best attribute.

3.6 Chapter conclusion

While Auto-Weka automatically identified the best suited classifier (Random Forest) for the NSL-KDD datasets, CSForest (Siers, & Islam, 2015), and SysFor (Islam, & Giggins, 2011) also showed surprisingly good results, considering they are not part of the selection of classifiers used by Auto-Weka.

3.6.1 Experimental result analysis.

The experimental data analyzed in this research show that the Naive Bayes was faster in all cases and SysFor (Islam, & Giggins, 2011) performed second best by correctly identifying dataset instances. The Auto-Weka classifier performed the best but was very slow in comparison to the other classifiers. This research experiment has identified a need for better quality datasets. While there are

many ML websites that do provide freely available datasets for testing, such OpenML (Vanschoren, Rijn, Bischl & Torgo, 2014), it is still very difficult for researchers to get a fit and good quality dataset for a given research problem that reflects a real problem such as zero day attack targeting new exponential technologies. Many research papers that relate to cyber attacks are using the same datasets and researchers are under the assumption that these datasets are trustworthy and correctly represent the real problem. Additionally, the cybersecurity domain is fast evolving and dynamic and it appears there are long delays between the discovered problem and the creation of new relevant datasets.

3.6.2 ML and better automation.

Alexander Wissner-Gross published an article, “Datasets over algorithms” (Wissner-Gross, 2016) identifying a possible reason as to “why the AI revolution took so long?” As Wissner-Gross (2016) explains, this could potentially be due to inadequate datasets and not due to low superiority ML algorithms. The latest AI breakthroughs, as he notes, used newer than 3 year old datasets in correlation with 18 year old algorithms.

The creation of datasets is a manual process that involves collaboration between data scientists and SMEs from a specific domain, in this case cybersecurity. While AutoML appears to be a viable solution, that could help cybersecurity experts identifying better suited classifiers for a freshly produced datasets, even higher level of automation is required in cybersecurity to deal with the fast evolving and escalating threat environment.

Santos, Castelo, Felix, Ono, Yu, Hong, Silva, Bertini, and Freire (2019) present a visual analytics framework, Visus, to combat the scarcity of data scientists problem, by assisting AutoML with model building and ML data processing pipelines. They address the limitation of AutoML, where the domain experts, have no ML knowledge, need easier and better interactive solutions that AutoML doesn't provide. Their framework supports exploratory data analysis (EDA), problem specification, model generation and selection, and confirmatory data analysis (CDA) (Santos et al., 2019). Additional components are included that

allows domain experts to search the Internet for relevant datasets as input data. Explanatory information and summary of models that were generated by AutoML, will provide the domain expert insight as to why this model was selected for the given dataset.

Creating datasets manually and using AutoML to select the best learners might not be the most efficient automation solution for the cybersecurity domain. A real time and continuous sampling as the source of data, for reinforcement learners appears to be more suitable for the cybersecurity domain. (Tuggener, Amirian, Rombach, Lörwald, Varlet, Westermann, & Stadelmann, 2019)

Khurana and Samulowitz (2019) address the limitation with automated machine learning, where data scientists that have many years of experience, sometimes in order to generate the best predictive models, they rely on an educated guess. They introduce a novel automated machine learning framework called APRL (Autonomous Predictive modeler via Reinforcement Learning). This system uses past experience by using reinforcement learning. The APRL is an RL agent that solves classification and regression problems in iterations, by selecting appropriate action, that are called data science explorations. OpenML (Vanschoren, Van Rijn, Bischl, & Torgo, 2014) was the primary source for the agent exploration, containing 62 datasets and 56 binary classification data problems. APRL when compared to AutoSklearn, consistently reported lower modeling error rates. (Khurana and Samulowitz, 2019)

RL DeepRL and Multi-agent RL (MARL) (Busoniu, Babuska, & Schutter, 2006) are not the most popular learners in the cybersecurity research, especially in anomaly detection, however the feedback loop provided in RL that is required for the self-regulating systems might be something that would have to be taken into consideration when automating cybersecurity.

The aim of this experiment is to provide answer to the second research sub-question, by testing AutoWeka, and to discover the level of automation ML provides to cybersecurity. Additionally, the experiment exposes the ML datasets as one of the automation challenges identified in the literature review. The

anomaly detection research focuses on detecting anomalies and correctly classifying instances by using classifier models against a given dataset. AutoWeka in this experiment did select the most accurate classifier model for NSL-KDD, however the aim of the experiment was also to expose the manual processes of processing datasets, finding accurate models, and also potentially managing false positives in production environments that will require manual intervention. The experiment shows that standard ML practices introduce manual processes and hinder automation, and AutoWeka only plays a small part towards autonomous cybersecurity solutions.

Chapter 4

4. Application Integrity Diversification

AID is a real time quantitative approach to define and monitor normal dynamic states for business and industrial applications, by monitoring diversity changes in real time for the verification of applications' running integrity. AID defines application behavioural profile and assists in providing assurance and trust dynamic assessment of the applications' main business and operation role. AID looks for unknown changes to the applications running integrity and main operating role, that could be triggered by exploitation of unknown vulnerabilities. AID is a combination of different diversity indexes and entropy based measurements taken from a continual sampling of data points, relating to system calls.

4.1 Introduction

This chapter explores a new way to define normality without the reliance of datasets or IDS rules and signatures. Black-box analysis is applied in the experiment and no knowledge is assumed of the attack or the vulnerability, when monitoring for anomalies. In order to detect unknown change or anomaly in this experiment, the change is measured.

Entropy is used to determine a change within a system, and entropic anomaly detection has been used in many research papers. By changing the volume, phase, mixing and composition of a system would lead to a change in entropy.
In this experiment entropy is used to measure system calls of an application during normal state and under an unknown change caused by zero-day attack in real time.

4.2 System Normality

4.2.1 Entropy based anomaly detection

An application state of system calls are observed during normal operation using

Shannon Entropy and two popular Diversity Indexes: Shannon-Wiener Index and Simpsons Index to determine normal baseline. Any deviation from this baseline could be an anomaly that needs to be further investigated.

4.2.2 Shannon Entropy

Shannon Entropy measures the system calls entropy, from a given sample where *n* is the total number of system calls in the sample, *i* is the total number of calls made by one specific system call, log2 is the binary log and p_i is the proportion of system calls that belong to *i* in a given sample.

$$H = -\sum_{i=1}^{n} p_i \log 2\, p_i$$

4.2.3 Biodiversity Index

Shannon-Wiener Index and Simpsons Index are the two most commonly used Diversity indexes in ecology. In this experiment they are used in AID to measure each application towards a larger Cyber Diversity Index that exist in a system or Cyber Ecology Environment (CEE).

4.2.3.1 Shannon Wiener Index

Shannon Wiener Index measures the diversity from a given sample where *s* is the total number of system calls in the sample, *i* is the total number of calls made by one specific system call, ln is the natural log and p_i is the proportion of system calls that belong to *i* in a given sample.

$$H = -\sum_{i=1}^{s} p_i \ln p_i$$

4.2.3.2 Simpson Index

Simpson's Index also measures the application's diversity from a given sample where *s* is the total number of system calls in the sample and p_i is the proportional abundance of each system call in a given sample. While Simpson's *D* value ranges from 0 to 1, 0 as infinite diversity and 1 as to low diversity the

inverse is normally applied (1/D), higher number matching higher diversity.

$$D=\sum_{i=1}^{k} p_i^2$$

4.2.3.3 Sørensen's Index

The Sørensen index, or Sørensen's coefficient, was developed by Thorvald Sørensen and identifies similarities between two bio communities from two given samples (Sørensen index, 2020).

$$CC=\frac{2c}{(S1+S2)}$$

The Sorenson's Coefficient result can be a range between 0 and 1, the closer the result to 1 the more similarities the bio communities have or in this case the applications.

It his experiment samples of system calls are taken from two different applications, a client browser Firefox and a Nagios a log server.

C in the experiment is the number of system calls the two applications have in common, *S1* is the total number of system calls found in application one or web client Firefox. *S2* is the total number of system calls found in application two or Nagios log server.

By comparing two different applications, that have different operational roles we could potentially identify any unique changes in AID that could warn us of the change of the integrity or the originally intended operational role of the measured application.

4.3 Real time data collection

Most research papers that create datasets of system calls use strace as the tracing tool to collect system calls activity. This tool appears to be outdated, and new technologies have been developed, that are more efficient, reliable, safer and provide better real-tine support for new technologies such a distributed applications.

4.3.1 Application performance engineering

The arrival of new technologies and the increased business demand for new and faster services introduces complexities that could cause performance issues and degradation of service availability. In distributed environments these issues can amplify quickly and bottlenecks can be difficult to trace. While bugs could cause performance issues, component coupling could also be a reason, performance is limited due to bottlenecks in the slowest component. Application performance engineering is the process of identifying performance issues early in the development cycle and fixing these before they amplify (Julien, 2018).

4.3.2 Dynamic tracing

Application performance issues can be resolved by applying application behavioural profiling or tracing. Tracing can use code instrumentation, where code is including in a program to display messages or log any failures and errors during application's run-time. Dynamic instrumentation or tracing is less intrusive and a more flexible when detecting performance issues, running software functions can be traced without application restart. Application profiles can be reconstructed by using traces, but traces can not be reconstructed by using profiles (Whitham, 2016).

Distributed tracing is used when tracing distributed applications and micro-services across different units, processes and hosts in to provide a timeline of an event during troubleshoot bottlenecks or performance issues (Apache-Incubating, 2020).

4.3.3 Extended Berkeley Packet Filter (eBPF)

An in-kernel virtual machine, extended Berkeley Packet Filter (eBPF) can be used to perform system tracing. Software developers can write code that executes in kernel space which normally would have required a new kernel module. A system call bpf is introduced in eBPF to load the program into a sandboxed environment (Kozina, 2020).

4.3.4 Bpftrace

Bpftrace is an open source tracer, developed by Robertson (2019), that uses eBPF to detect software performance issues in production. The bpftrace one-liner scripts provide a quick way to trace system calls and gain visibility to performance bottlenecks.

4.4 Application behavioural profiling with AID

Two systems are used in the experiments to generate relevant quantitative data to measure AID for the application behavioural profiling. Nagios XI 5.4.10 (Nagios-XI, 2019) was chosen, because it is used as a central server to ingest logs in production environments and could be part of many cybersecurity defence teams arsenal, which shows that even security tools can be compromised.
Firefox and Adobe Reader were chosen as the second and third client based applications for analysis and comparisons. Black box analysis was performed on the defending system, that had no knowledge of the attack and no knowledge of the vulnerability. No databases of IoC and no system or application logs were used to detect the attack.

The black-box analysis utilised enhanced Berkeley Packet Filter (eBPF) with bpftrace (Gregg, 2019) to collect information of system calls used by Nagios and Firefox during normal operational environment and when under cyber attack. An entropy based anomaly detection and Diversity Index were used to generate an application behavioural profiles. The measurements were further pushed to Prometheus (2019), and Grafana (2019) for real time visual continuous sampling.
The algorithm in Fig. 13 shows the pseudo-code for data ingestion, the input from bpftrace, containing information about system calls used by the targeted application, and while loop that ends with the last system call with ID 332. Information of the system calls, such as ID, name and the count of system calls made by the application are then uploaded to a web-based graphical dashboard for real time analysis.
This algorithm, is used in the first experiment, for all three applications to gather a basic information about the application's behavioural profile and the most system calls used by an application, such as the heavy hitters. Additionally, the results from this experiment will provide the input for the Sørensen's coefficient to identify similarities between two applications, Nagios and Firefox.

Algorithm 1: Normal application behavioural profile

Data: Continuous sampling of system calls trace from bpftrace
Result: Normal application behavioural profile

```
initialization of system call ID
x ← 1
while x < 332 do
    read current system call ID
    if system call ID exists then
        print system call name, ID and count
        go to next system call
    else
        pipe data to curl –data-binary @- http://server
    end
end
```

FIGURE 13. Algorithm 1 Normal application behavioural profile.

The printed output is then sent to compute the Shannon Entropy, Shannon Wiener and Simpson' Index of that string. The measurements are then piped to Prometheus (2019) and Grafana (2019) for graphical and real time visualisation.

Algorithm 2: Application behavioural profiling for AID

Result: Application behavioural profiling for AID
Input: Application system calls trace from bpftrace
Output: Entropy and Diversity Index measurements for AID

Calculate H, SW, and D
Check,
$x \leftarrow 1$
while $x < 332$ **do**
    read current system calls made by the targeted application
    if *system call ID exists* **then**
        print unique system call character
        compute the Shannon Entropy of system calls

$$H = -\sum_{i=1}^{x} p_i \, log_2 \, p_i$$

        compute the Shannon Wiener Index of system calls

$$SW = -\sum_{i=1}^{x} p_i \, ln \, p_i$$

        compute the Simpson Index of system calls

$$D = \sum_{i=1}^{x} p_i^2$$

        print H, D, SW
    else
        pipe data to curl –data-binary @- http://server
    end
end

FIGURE 14. Algorithm 2 Application Integrity Diversification.

The second algorithm in Fig. 14 shows the ingestion of data from bpftrace relating to system calls made by the targeted application. Each system call that is used by the

application is presented by unique character and printed as many times as counts made by the application.

Fig. 15 shows the average Shannon-Wiener Index for system calls made by Nagios in idle is 1.177 and the maximum measurement is 1.225

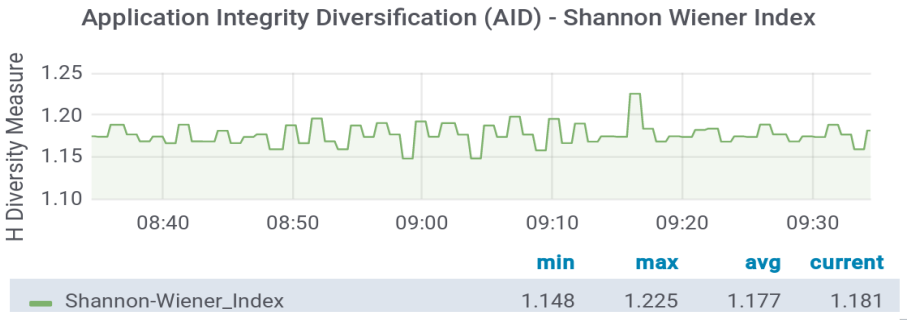

FIGURE 15. Shannon Wiener Index for Nagios in idle.

Fig. 16 shows the average Simpson's Index for system calls made by Nagios in idle is 6.5 and the maximum measurement is 8.101

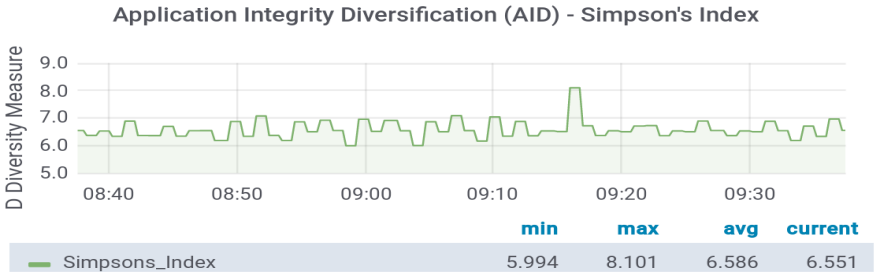

FIGURE 16. Simpson Index for Nagios in idle.

As seen in Fig. 17 the average Shannon Entropy for system calls made by Nagios in idle is 3.9080 bits/syscall and maximum measurement of 4.0700.

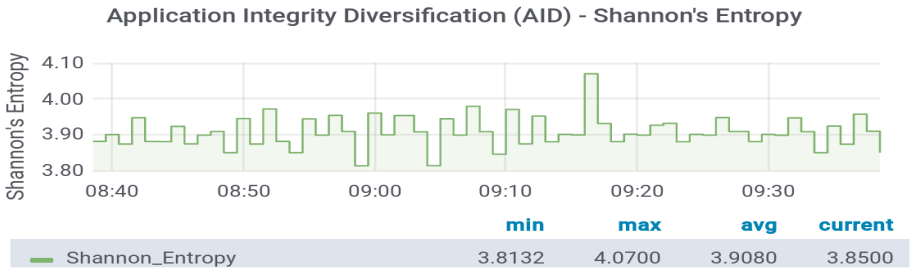

FIGURE 17. Shannon Entropy for Nagios in idle.

Results from Fig.15, Fig.16, and Fig.17 show all Diversity Indexes and the Shannon Entropy of the Nagios's system calls, have similar peaks and lows.

In the next experiment, same input has been applied to two very different hardware systems, Nagios_1 and Nagios_2 are running the same version of containerised Nagios XI 5.4.10 (Mavenquist, 2019). Fig. 18 shows an average Shannon Wiener Index of 1.20 for both systems, and it appears, Nagios 1 and Nagios 2 have similar measurements in idle state.

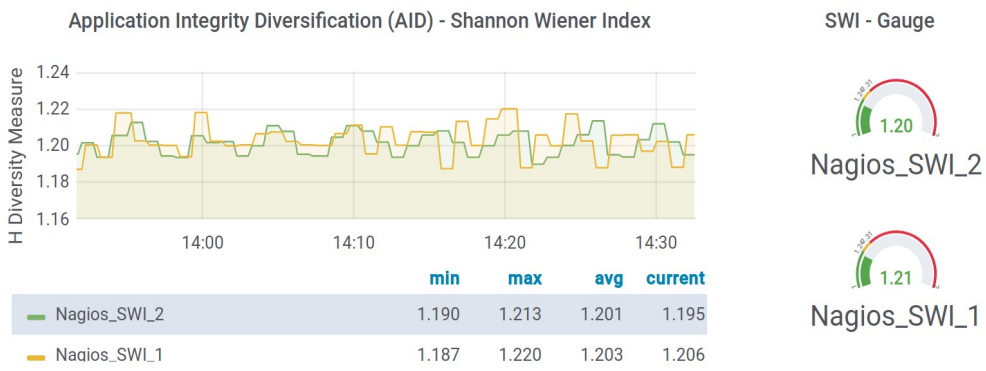

FIGURE 18. Shannon Wiener Index for Nagios 1 and 2 in idle.

Fig. 19 shows measurement of Simpson Index of 7.20 and 7.15 for Nagios 1 and Nagios 2, other measurements are also similar.

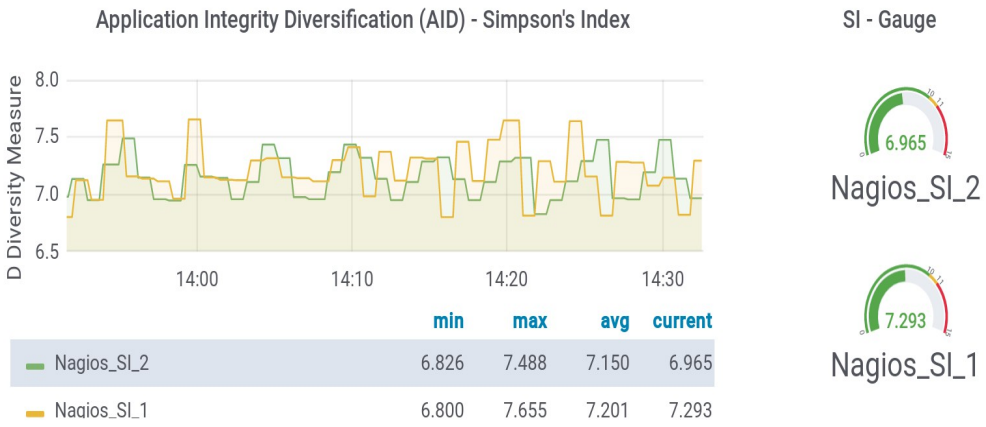

FIGURE 19. Simpson Index for Nagios 1 and Nagios 2 in idle.

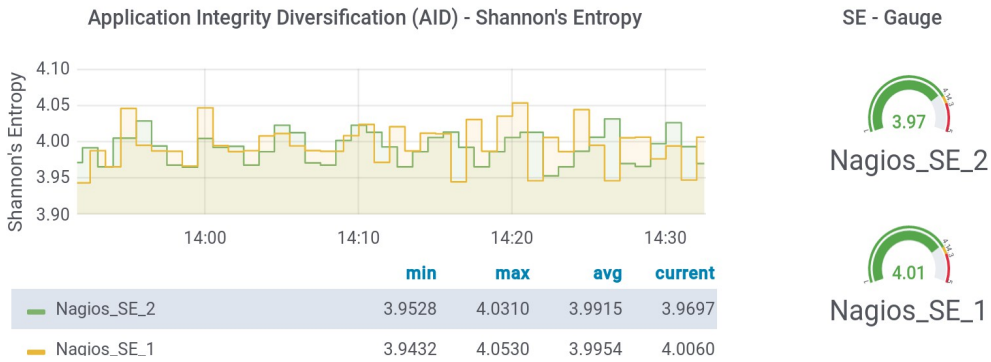

FIGURE 20. Shannon Entropy for Nagios_1/2 in idle.

Fig. 20 shows average measurement of Shannon Entropy of 3.9, and confirms the similar measurements as seen in, Fig. 19 and Fig. 18. These results show consistency and possible uniqueness, that Diversity Indexes, could be used for application behavioral profiling and defining a normal baseline. The additional SWI, SI and SE gauges show the current real time AID measurements of the system calls.

For the next experiment the system calls results from the algorithm 1 were used to determine the similarity between two communities or applications. Two system calls samples were taken from both Firefox and Nagios to compare their system call usage.

TABLE VII. Sorenson's Index for Firefox and Nagios

System call ID	System call threat group	System call name	Number of individual system calls	
			Firefox	Nagios
0	T3	read	2	5
1	T3	write	3	14
3	T3	close	7	2
4	T4	stat	2	12
5	T4	fstat	35	12
6	T4	lstat	6	1
7	T3	poll	5	20
8	T3	lseek	14	1

System call ID	System call threat	System call name	Number of individual system calls	
			Firefox	Nagios
9	T3	mmap	1	1
10	T3	mprotect	14	6
11	T3	munmap	1	1
12	T2	brk	1	5
13	T3	rt_sigaction	1	14
14	T3	rt_sigprocmask	4	374
15	T3	rt_sigreturn	72	1
16	T2	ioctl	2	2
17	T3	pread64	6	0
18	T3	pwrite64	1	0
20	T3	writev	308	0
21	T4	access	23	13
22	T4	pipe	3	2
23	T3	select	1	0
28	MS	madvise	1	0
29	MS	shmget	4	0
30	MS	shmat	4	0
31	MS	shmctl	4	0
33	T2	dup2	0	2
37	MS	alarm	0	4
39	T4	getpid	9	1
41	DT2	socket	4	1
42	MS	connect	5	1
44	MS	sendto	2	0
45	MS	recvfrom	23	0
46	MS	sendmsg	33	0
47	MS	recvmsg	13	0
48	MS	shutdown	3	0
49	MS	bind	1	0
51	MS	getsockname	4	0
52	MS	getpeername	8	0
53	MS	socketpair	1	0

System call ID	System call threat	System call name	Number of individual system calls	
			Firefox	Nagios
54	MS	setsockopt	1	0
55	MS	getsockopt	1	0
56	DT2	clone	1	1
59	T1	execve	0	4
60	T3	exit	1	0
61	T3	wait4	1	2
62	T2	kill	2	20
63	T4	uname	2	1
67	MS	shmdt	2	0
72	T3	fcntl	49	3
74	T3	fsync	1	1
75	T4	fdatasync	3	0
77	T2	ftruncate	8	0
78	T4	getdents	2	2
79	T4	getcwd	5	0
82	T1	rename	1	1
83	T1	mkdir	6	0
87	MS	unlink	1	0
88	T1	symlink	1	0
89	MS	readlink	9	0
90	T1	chmodd	1	0
91	T1	fchmod	0	1
95	T3	umask	2	1
97	T4	getrlimit	0	2
98	T4	getrusage	2	0
99	T4	sysinfo	1	0
102	MS	getuid	3	13
104	T4	getgid	2	13
107	T4	geteuid	5	13
108	T4	getegid	2	13
109	T3	setpgid	0	1
110	T4	getppid	0	1

System call ID	System call threat	System call name	Number of individual system calls	
			Firefox	Nagios
111	T4	getpgrp	0	1
117	T1	setresuid	38	0
118	T4	getresuid	3	0
119	T1	setresgid	38	0
120	T4	getresgid	3	0
131	MS	sigaltstack	1	0
132	T3	utime	1	0
137	T4	statfs	1	0
138	T4	fstatfs	1	0
143	T4	sched_getparam	1	0
144	T2	sched_setscheduler	1	0
145	T4	sched_getscheduler	1	0
146	T4	sched_get_priority_max	1	0
147	T4	sched_get_priority_min	1	0
157	T3	prctl	6	0
158	MS	arch_prctl	1	1
186	MS	gettid	1	0
187	MS	readahead	12	0
202	MS	futex	5	0
204	MS	sys_sched_getaffinity	1	0
213	MS	epoll_create	1	0
218	MS	set_tid_address	1	0
221	MS	fadvise64	67	0
229	T4	clock_getres	1	0
231	MS	exit_group	1	2
232	MS	epoll_wait	4	13
233	MS	epoll_ctl	1	4
234	MS	tgkill	84	0
254	MS	inotify_add_watch	1	0
257	MS	openat	3	0
273	MS	set_robust_list	1	0
285	MS	fallocate	1	0

System call ID	System call threat	System call name	Number of individual system calls	
			Firefox	Nagios
290	MS	eventfd2	7	0
293	MS	pipe2	1	2
294	MS	inotify_init1	1	0
302	MS	prlimit64	4	0
307	MS	sendmmsg	1	0
318	MS	getrandom	118	0
319	MS	memfd_create	2	0
System calls richness			103	47
Total number of system calls			1163	611
Common system calls			39	

From Table VII we can determine Firefox has system call richness or cardinality of 103, and Nagios has system calls richness of 47. Both applications have 39 system calls in common. From this, we compute the Sorenson's Coefficient as follows:

$$CC = \frac{2c}{(S1 + S2)}$$

$2c = 78$;

$S1 = 103$; $S2 = 47$;

$CC = 78/150 = 0.52$;

The closer the *CC* is to 1, the more similar the applications are. The result from the tested samples show a Sorenson's Coefficient of 0.52, and not a strong similarity.

The next experiment, tries to identify a possible similarity between two applications. Additional system calls were taken from Firefox version 74.0 and the previous sample of system calls from Firefox 11.0 were kept for comparison. Table VIII shows the summary of the samples and more detailed information has been added to Appendix C.

TABLE VIII. Sorenson's Index for Firefox v.11 and Firefox v.74

Application	Firefox version 11.0	Firefox version 74.0
System calls richness	103	92
In common system calls	80	

Total number of system calls	1163	66070
Application similarity	0.82	

$2c = 160$; S1 = 103 ; S2 = 92;

$CC = 160/195 = 0.82$;

From table XIII, the Sorenson's Coefficient measurement is 0.82, on system calls made by Firefox version 11.0 and Firefox version 74.0 browsers, The two running applications behavioral profiles are similar.

4.5 Chapter Conclusion

The quantitative results from the experiments show, different applications exhibit different behavioural profiles, and by comparing diversity indexes of their system calls usage, similarities can be identified.

The Shannon Entropy, Shannon-Wiener Index, Simpson Index and Sorenson's Index are used for monitoring of a normal baseline for application entropic behavioural profile. By monitoring this dynamic profile we can provide verification and assurance of a positive application integrity diversification.

The same containerised application that runs on two different hosts, exhibits similar behavioural profiles and the Indexes in AID, detected changes during different application loads of known changes on system calls.

The AID of applications, when placed under load exhibited reasonably stable behavioural profiles. The AID of Nagios server and Firefox client showed low behavioural similarity, while Firefox V11 and Firefox V74 showed high behavioural similarity. AID could provide application integrity assurance, between patching cycles, where the supply chain of application distribution, has be compromised and file hashes are inadequate.

The results from the experiment in this chapter shows that it is possible to measure a normal system state, by measuring the entropic state of its applications. These measurements are monitored in the application integrity diversification (AID) that provides the base for the normal behavioural profiling that is needed for anomaly detection. Further, this solution provides continuous real-time monitoring of the normal states that is required in order to manage the unknown changes or anomalies that were

triggered by zero-day attacks. Furthermore, this solution improves the visibility of the dynamic system normal states in comparison to normal states that are defined with ML datasets and databases of IoC that are manually created and managed.

In the next chapter the applications are going to be placed under cyber attacks and increase in AID is expected, caused by the expected increase in diversity and entropy. This a black-box analysis and the defending system has no knowledge of the attack or the vulnerability.

Chapter 5

5. Indicators of Exploitation

In continuation of IoC and IoA research, measurable, quantitative and non-human readable indicators are introduced in this research, called micro Indicators of Exploitation (mIoX). These indicators are measurable and relate to system calls, exploitation, black-box analysis and continual sampling. Entropic behavioural profiling (EBP), and diversity index solutions are used to measure applications diversification, in different environments, to detect the exploitation stage during zero-day attacks, and to gather local cybersecurity intelligence.

5.1 Introduction

In this experiment, a cause of a cyber attack is a hypothetical cyber parasite called, vegan beef in a can (VBIAC). This cyber parasite, adapts characteristics from the Toxoplasma gondii parasite. “T.gondii (or Toxo for short) infects a wide variety of mammals, but it only completes its life cycle in the guts of a cat. To get there, Toxo has ways of subverting the behaviour of dead-end hosts like mice. Its machinations are subtle, so subtle that it’s normally hard to tell an infected mouse from an uninfected one. But the difference becomes obvious when there’s cat pee in the air. Normal mice, even lab-born ones that have never met a cat, have an innate fear of cat smells. Those infected with Toxo do not. They (and their parasites) are more likely to end up in a cat” (Yong, 2013, para 2). The same way, Toxo uses mice to propagate, VBIAC uses web servers to the same. The end host for the Toxo is the cats' gut, while for VIBIAC is the web client's memory.

To replicate the exploitation and the impact of the cyber parasite, an attacking system will be using Metasploit (Rapid7, 2019). The propagation web server uses Ubuntu with containerised and unpatched version of Nagios XI (Nagios-XI-D, 2019) The end host,

the cat's gut, is an unpatched and exploitable Firefox (2020) web client. The experimental scenario is related to cyber attack that replicates the impact of a zero-day attacks, where the defending system, has no knowledge or information of the attack or the vulnerability. The focus of this experiment is the stage four or the exploitation stage from the Cyber Kill Chain (Hutchins, Cloppert, & Amin, 2014)

5.2 Cyber attack scenario

The purpose of this experiment, is to identify mIoX of an active zero-day exploitation. This experiment is divided in two parts, remote exploitation on Nagios XI server, and local exploitation on Firefox web browser client.
Nagios log server was selected as the propagating system for this cyber attack, because this application is used in many production environments as part of a cybersecurity strategy for collecting logs. The experiment shows, that no product or application, including cybersecurity products, are immune to vulnerabilities and exploitation.
A containerised version of Nagios server is used, to test a common misconception that container technologies provide strong and secure isolation controls. The browser exploitation framework (BeEF) (BeEF, 2020) is used to exploit Firefox browser client. The aim of the attack is to first “pop a shell” (Spring, 2019) and to take over the Nagios server. The second step is to redirect traffic from the Nagios server to a malicious landing page. The third step is to pop a shell (Spring, 2019) on the Firefox web clients and also take over the hosts, that visit the Nagios server. The experiment doesn't cover the second stage and it only focuses on the exploitation stages of the attack.

The term vegan in, the hypothetical cyber parasite, VBIAC, is used, to address the misconception that are containerised (can) applications provide secure environments, by using isolation mechanisms. Beef is used, in reference to the BeEF framework (BeEF, 2020). Popping a shell in a container could be dangerous, regardless of the usage of isolation mechanism, that some technologies provide, as seen in the experiment.
Fig. 21 shows the flow of the unknown change and the detection of the anomaly, that was triggered by the cyber parasite, during both the remote exploitation on Nagios and the exploitation of the Firefox client. The defender takes AID measures, and records the unknown changes. The controller takes measurements during the cyber exploitation, and looks for anomalies or unknown changes to the AID to detect spikes or peaks that

uncover mIoX.

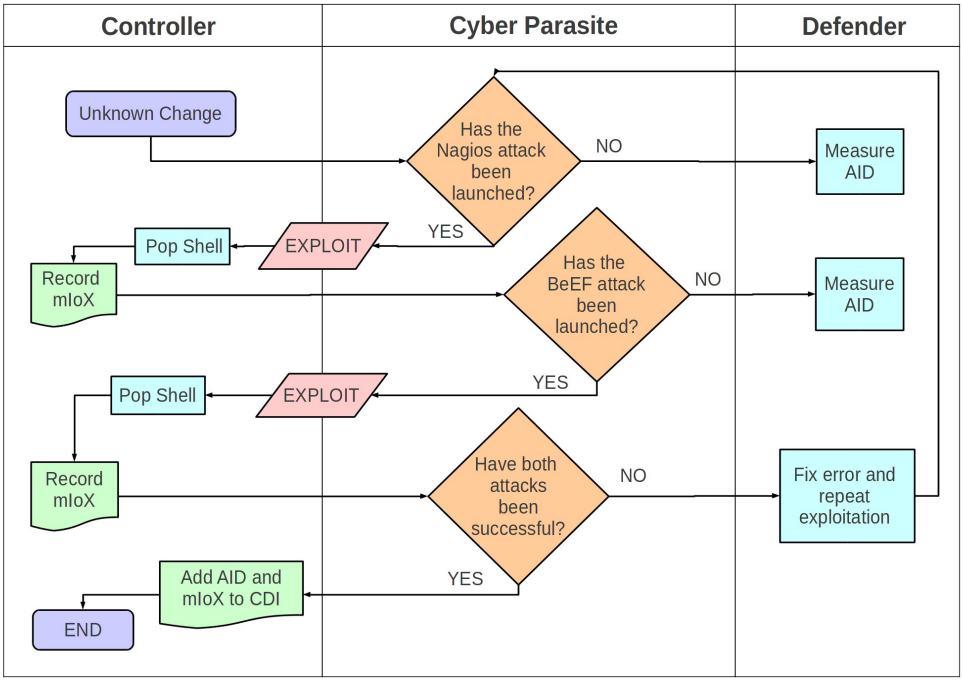

FIGURE 21. Vegan beef in a can attack.

5.2.1 Remote exploitation

Nagios XI Chained Remote Code Execution exploit, developed by Smith, Husted and Arave (Nagios-exploit, 2020) in Metasploit was chosen, because this is a reliable exploit, works every time, it is easy to run, the exploit pops a shell with root access, and it web application doesn't crash.

5.2.2 Local exploitation

The Browser Exploitation Framework (BeEF) project/tool has been utilised in the experiment to launch an attack on the exploitable Firefox 11.0 browser. The web client exploitation takes place, when the client lands on a watering hole managed by BeEF.

5.3 micro Indicators of Exploitation

When Nagios XI Chained Remote Code Execution exploit, was executed against

Nagios_S1, all three measurements showed increase of diversity, and higher entropy as seen in Fig. 22, Fig. 23 and Fig. 24.

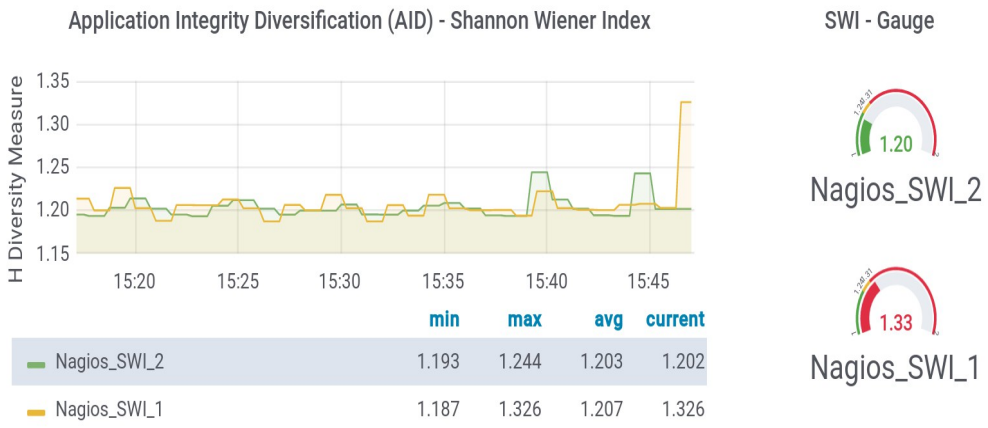

FIGURE 22. Shannon Wiener Index for Nagios 1 under attack.

As seen in Fig. 22 just after 15:45, mIoX was detected on Nagios-1 with magnitude of 1.33 on the Shannon Wiener Index. This mIoX was caused by an active exploitation, that resulted to the attacker taking over the Nagios service, by popping a shell inside the containerised service, and gaining root access. The SWI_1 gauge shows the peak of Shannon Wiener Index in real time, and red alert was activated.

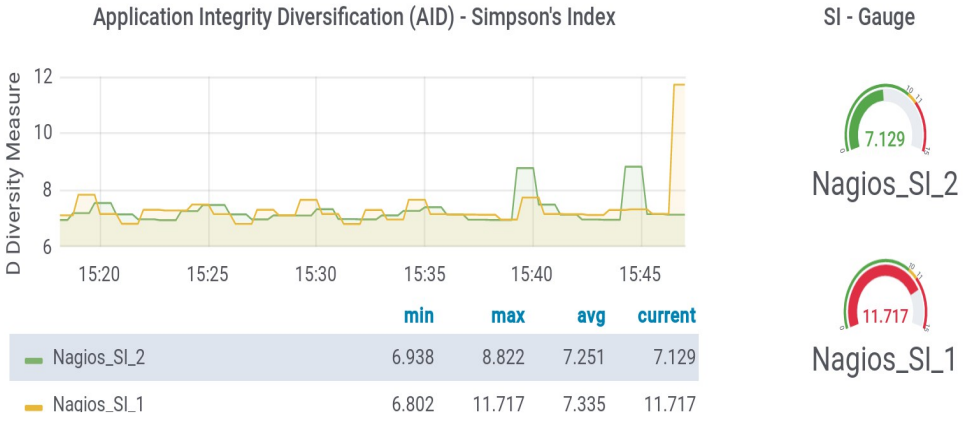

FIGURE 23. Simpson's Index for Nagios 1 under attack.

Fig. 23 also shows mIoX detection on Nagios-1, with magnitude of 11.717 on the Simpson Index. This was caused by the same exploitation, as previously mentioned. The SI_1 gauge shows the peak of Simpson Index in real time, and red alert was activated.

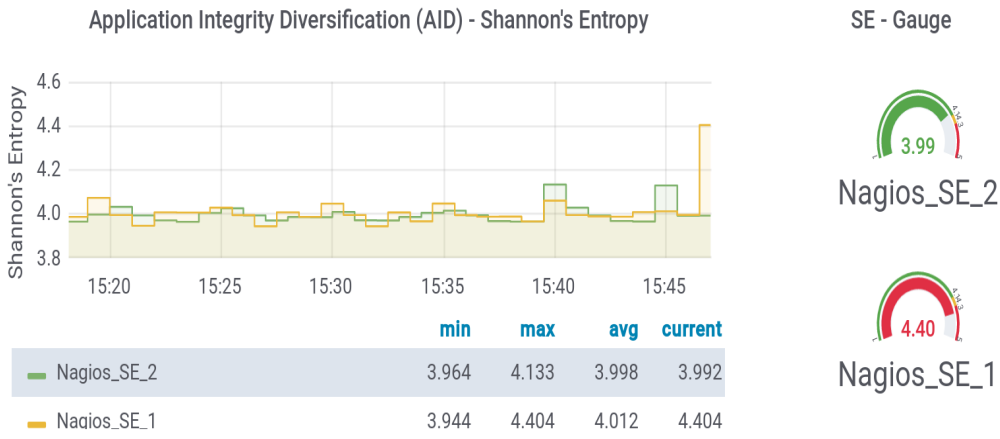

FIGURE 24. Shannon Entropy for Nagios 1 under attack.

Fig.24 confirms the findings from Fig. 22 and Fig. 23 and also shows mIoX detection on Nagios-1, with magnitude of 4.40 on the Shannon Entropy gauge. The mIoX as previously, was caused by the same cyber attack. The SE_1 gauge shows the peak of Shannon Entropy in real time, and red alert was activated like in the previous two cases.

5.4 Risk management

By performing risk assessment, system calls can be grouped by threat, as reported by Bernaschi, Gabrielli, and Mancini (2000). Table IX show the grouping of system calls used in the experiment, for both Nagios and Firefox. System calls in threat 1 group present higher threat than system calls in threat 2 group. Preforming risk assessment improves cybersecurity, and reduces number of system calls that are monitored, could also improve the effectiveness of the Diversity Indexes.

TABLE IX. B.G. & M. threat security profile

B.G. & M. threat security profile	
Threat 1	Threat 2
2 AID_sys_open	12 AID_sys_brk
59 AID_sys_execve	16 AID_sys_ioctl
82 AID_sys_rename	33 AID_sys_dup2
83 AID_sys_mkdir	57 AID_sys_fork
86 AID_sys_link	58 AID_sys_vfork
88 AID_sys_symlink	62 AID_sys_kill
90 AID_sys_chmod	73 AID_sys_flock

91 AID_sys_fchmod	76 AID_sys_truncate
92 AID_sys_chown	77 AID_sys_ftruncate
93 AID_sys_fchown	84 AID_sys_rmdir
94 AID_sys_lchown	85 AID_sys_creat
105 AID_sys_setuid	103 AID_sys_syslog
106 AID_sys_setgid	133 AID_sys_mknod
113 AID_sys_setreuid	141 AID_sys_setpriority
114 AID_sys_setregid	142 AID_sys_sched_setparam
116 AID_sys_setgroups	144_AID_sys_sched_setscheduler
117 AID_sys_setresuid	149 AID_sys_mlock
119 AID_sys_setresgid	151 AID_sys_mlockall
122 AID_sys_setfsuid	153 AID_sys_vhangup
123 AID_sys_setfsgid	154 AID_sys_modify_ldt
241 AID_sys_mq_unlink	159 AID_sys_adjtimex
	160 AID_sys_setrlimit
	170 AID_sys_sethostname
	171 AID_sys_setdomainname
	183 AID_sys_afs_syscall
	201 AID_sys_time

Table X shows Docker (2020) recommendations for grouping of system calls by level of threat, and used in the experiment for both Nagios and Firefox tests.

TABLE X. Docker Seccomp and B.G. & M. threat security profile

Docker Seccomp profile	Docker Seccomp and B.G. & M. profile
101 AID_sys_ptrace	56 AID_sys_clone
134 AID_sys_uselib	164 AID_sys_settimeofday
135 AID_sys_personality	165 AID_sys_mount
136 AID_sys_ustat	166 AID_sys_umount2
139 AID_sys_sysfs	167 AID_sys_swapon
155 AID_sys_pivot_root	168 AID_sys_swapoff
163 AID_sys_acct	169 AID_sys_reboot
175 AID_sys_init_module	156 AID_sys__sysctl
177 AID_sys_get_kernel_syms	172 AID_sys_iopl
178 AID_sys_query_module	173 AID_sys_ioperm
227 AID_sys_lock_settime	174 AID_sys_create_module
237 AID_sys_mbind	176 AID_sys_delete_module
238 AID_sys_set_mempolicy	179 AID_sys_quotactl
239 AID_sys_get_mempolicy	
248 AID_sys_add_key	
249 AID_sys_request_key	
272 AID_sys_unshare	

279 AID_sys_move_pages
298 AID_sys_perf_event_open
303_AID_sys_name_to_handle_at
304_AID_sys_sys_open_by_handle_at
305 AID_sys_sys_clock_adjtime
308 AID_sys_setns
310 AID_sys_process_vm_readv
311 AID_sys_process_vm_writev
312 AID_sys_kcmp
313 AID_sys_finit_module
320 AID_sys_kexec_file_load
321 AID_sys_bpf
323 AID_sys_userfaultfd

The Fig. 25 shows system calls that were marked as high threat levels 1 and 2, that were made by Nagios while running in idle. Fig. 26 shows the system calls made by Nagios during exploitation, when the system was attacked by the cyber parasite. The increased number of system calls, corresponds with the increased diversity indexes and AID of Nagios as expected.

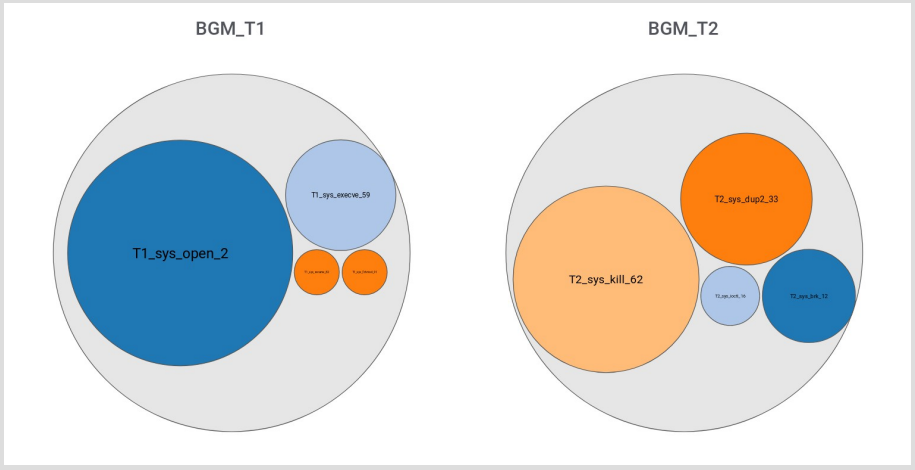

FIGURE 25. High threat system calls made by Nagios in idle.

Additional measurements were taken from the application's life-cycle, such as startup, exit, and admin tasks performed in Nagios, showed slight increase in diversity and entropy, but not as high, as reported by the cyber attack.

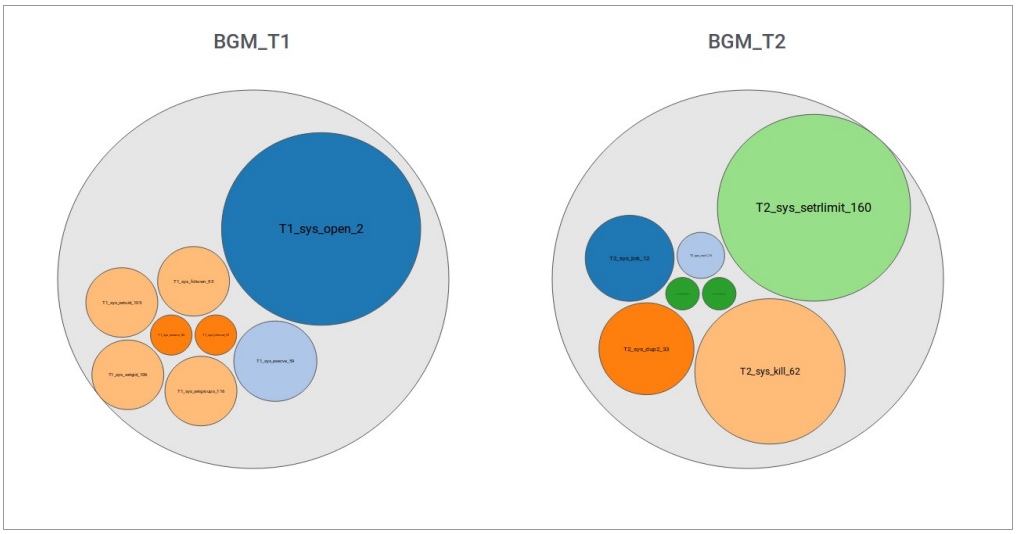

FIGURE 26. High threat system calls made by Nagios under attack.

Fig. 27 shows the algorithm for discovery of mIoX during exploitation of Firefox web client. Unlike algorithm 2, threat and risk assessment was implemented and the number of system calls were reduced to cover only groups T1 and T2 system calls.

Algorithm 3: Application behavioural profiling for AID

Result: Discovering mIoX from AID, by using reduced number of system calls
Input: System calls trace from bpftrace
Output: Entropy and Diversity Index measurements for AID by using risk and threat assesment

Calculate H, SW, and D
Check,
$x \leftarrow 1$
while $x < 332$ **do**
  read current system calls made by the targeted application
  if *system call ID exists* **then**
    print only unique system call character from a list of system calls with 1 and 2 threat rating
    compute the Shannon Entropy of system calls

$$H = -\sum_{i=1}^{x} p_i \, log_2 \, p_i$$

    compute the Shannon Wiener Index of system calls

$$SW = -\sum_{i=1}^{x} p_i \, ln \, p_i$$

    compute the Simpson Index of system calls

$$D = \sum_{i=1}^{x} p_i^2$$

    print H, D, SW
  else
    pipe data to curl –data-binary @- http://server
  end
end

FIGURE 27. Algorithm 3. AID, mIoX with focus on risk

The cyber attack was launched by the cyber parasite, when the victim with Firefox 11.0 web client accessed the compromised redirection page, hypothetical watering hole, (ACS, 2019) from the Nagios server and landed on the BeEF (BeEF, 2020) exploit page. The exploit was delivered to the client (Phreaklets, 2014), and remote access was gained by popping a shell via Metasploit (Rapid7, 2017).

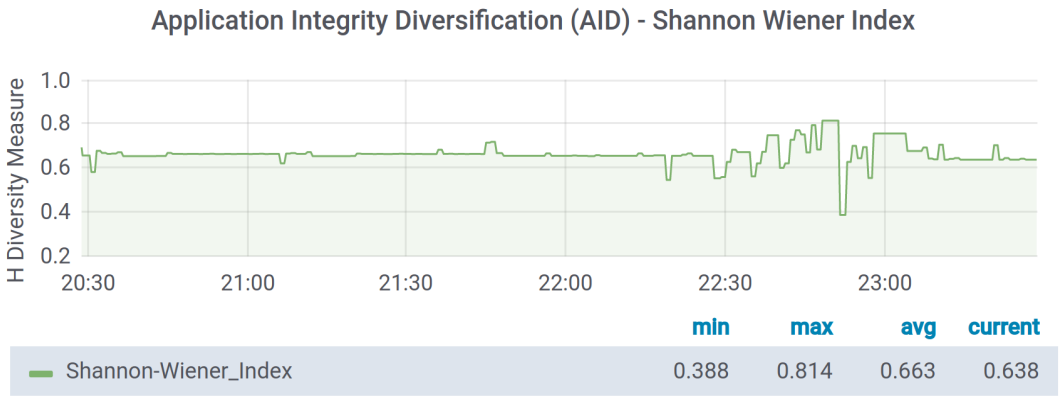

FIGURE 28. Shannon Wiener Index for Firefox under attack

Fig. 28 shows a possible mIoX, that was detected on the Firefox client with magnitude of 0.814 on the Shannon Wiener Index. This mIoX was caused by the active exploitation, that resulted to the cyber parasite taking over the web client, by popping a shell and gaining remote access.

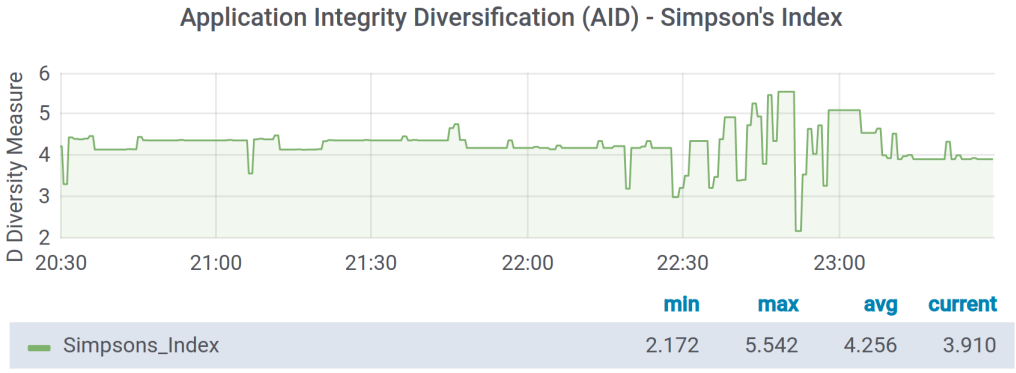

FIGURE 29. Simpson Index for Firefox under attack

As seen in Fig. 29 an increase in magnitude of 5.542 on the Simpson Index and possible mIoX, was also detected during the web client exploitation. Fig. 30 confirms the results from Fig.28 and Fig.29, shows increase of diversification and entropy of 2.705 on the Shannon Entropy graph. The slight dip in Fig.28, Fig.29 and Fig.30 after the maximum

reading, was due to an application reset.

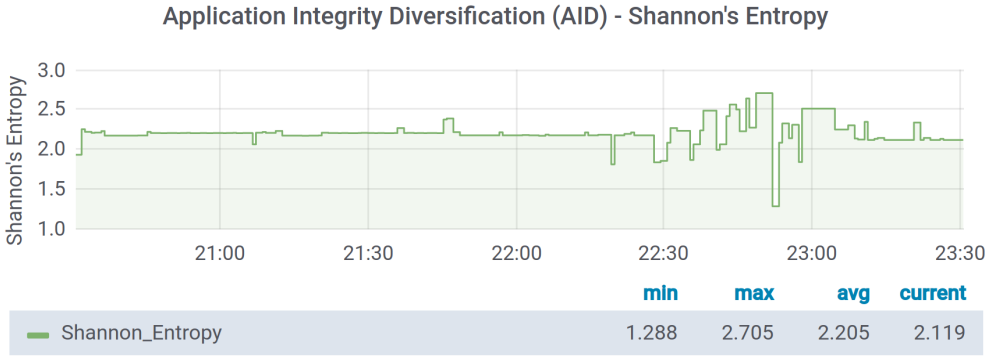

FIGURE 30. Shannon Entropy for Firefox under attack.

Fig.31 shows the systems calls that were made by Firefox during idle and browsing, with rating marked as high threat group 1, and high threat group 2. The number and the size of the bubbles represents the amount of system calls made by the application.

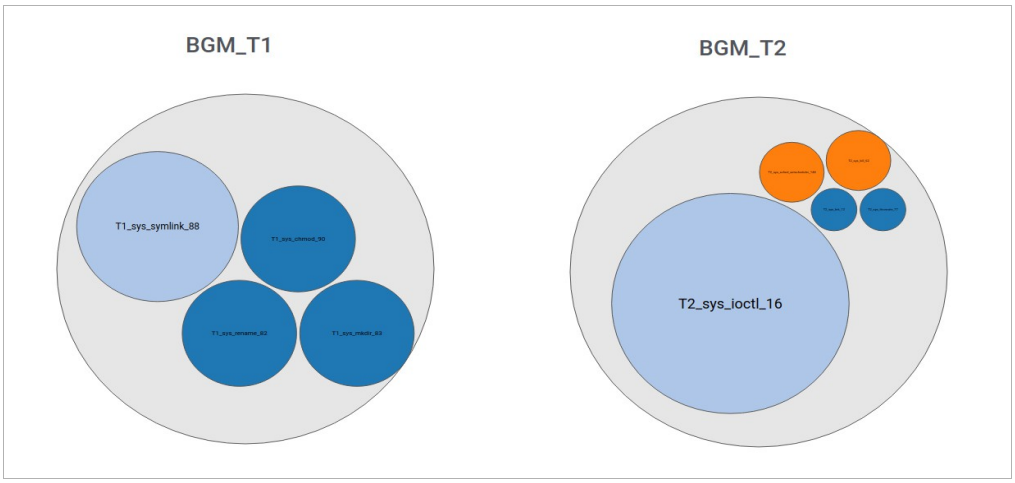

FIGURE 31. High threat system calls made by Firefox in idle.

Fig.32 shows the systems calls that were made by Firefox during the attack, with increase of system calls in both, high threat group 1, and high threat group 2. The number and the size of the bubbles have increased as expected, that shows increase of AID and the entropy of system calls made by the application.

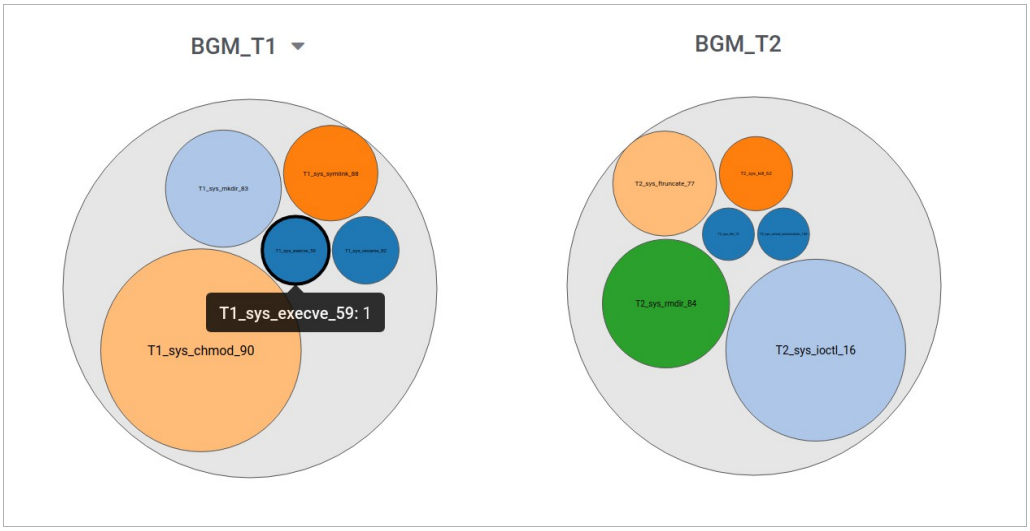

FIGURE 32. High threat system calls made by Firefox under attack.

While this attack appears to be stealthier than the previous Nagios experiment, one interesting system call made by the application was sys_execve_59, that most likely was called during of a non-consensual install of the malicious add-on.

5.5 Chapter conclusion and analysis results

The experiments in this chapter confirmed the hypothesis, the application integrity diversification increases, when the applications are placed under a cyber attack, that use exploitation. Peaks in Diversity Indexes were detected, that point to the discovery of mIoX. The Nagios exploitation was detected using AID, however admin tasks and applications at start up also reported slightly higher measurements. AID could provide visibility to diversification related to privilege escalation attacks and visibility of applications that should not have admin privileges.

AID application behavioural profiling is not just about establishing a behavioural baseline, for anomaly detection, but also to define the main operational role of the application, and to monitor for any diversification or skew from the original goal. Like in the experiment, where the Nagios main service role was to provide a containerised logs server, becomes a watering hole that serves malicious exploits. The unknown change of diversification, changed the behaviour of the application, and what is more dangerous, the business or operational role of that application.

Both exploitation and discovery of mIoX were detected on the web server and client,

without using any databases of IoC, datasets, rules and signatures. The AID system calls usage for Firefox, was reduced by using the threat and risk assessment classification to improve the discovery of mIoX.

The experiment in this chapter shows that it is possible to monitor an AID behavioural profile, establish dynamic normal states, and detect mIoX during zero-day attacks.

The novel mIoX are measurable and real-time indicators, unlike the popular IoC that are human readable indicators, and shared after the fact. MioX could assist with gathering local cyber intelligence by detecting exploited applications, rogue applications, and good applications behaving badly.

The experiment also confirms the research hypothesis, that the system entropy increases during a zero-day attack as predicted.

Chapter 6

6. Cyber Diversity Index

Cyber Diversity Index is a self-populating index, that contains AID measurements of diversity changes in an application's behavioural profile in real time. The quantitative measurements are taken from continuous system calls sampling.

6.1 Introduction

While only three applications were measured with AID, and compared in this research, AID in CDI could be further improved by future research taken to simple hosts, complex systems, IT/OT, and cyber ecological synthesised environments as proposed in the next chapter.

6.2 AID and CDI

Table XI shows, information and dynamic measurements from AID, that were collected during the experiments. The purpose of these measurements is to monitor the applications' Entropic Behavioural Profile (EBP) and diversification in real time, with continual sampling and self-populating capability.

The information provided by AID, will assist the CDI in monitoring changes throughout the application's life-cycle that includes, patching assurance and vulnerability management. All CDI measurements in Table XI, application version, cardinality of system calls, entropy, diversity indexes, mIoX and heavy hitters will change dynamically. How the application changes and diversifies over time, will be the baseline of the application normal operation profile. This dynamic change will be the application's behavioural profile, and not a predefined set of rules in a database and the assumption of a normal baseline.

TABLE XI. Cyber Diversity Index

Application		Nagios XI			Firefox			Adobe Reader		
Version		5.4.10.			11.0			9.1.3		
Cardinality		47			103			42		
AID	SWI (min|avg|max)	1.19	1.20	1.21	0.54	0.66	0.72	0.28	0.63	1.10
	SI (min|avg|max)	6.80	7.15	7.65	3.19	4.27	4.87	1.67	3.17	9.08
	SE (min|avg|max)	3.95	3.99	4.05	1.81	2.19	2.40	0.94	2.10	3.68
mIoX (SWI|SI|SE)		1.33	11.71	4.40	0.81	5.54	2.70	-	-	-
Heavy hitters		sys_rt_sigprocmask			sys_poll			sys_sched_get_priority_max		
		sys_write			sys_writev			sys_linkat		
		sys_close			sys_write			sys_getuid		
Threat 1 (Top 3)		sys_execve			sys_execve			sys_fchmod		
		sys_open			sys_mkdir			sys_setfsuid		
		sys_fchmod			sys_symlink			-		
Threat 2 (Top 3)		sys_kill			sys_ioctl			sys_swapoff		
		sys_brk			sys_ftruncate			sys_ftruncate		
		sys_ioctl			sys_setpriority			sys_afs_syscall		

6.3 Practical CDI

6.3.1 Software patching

From the results of the experiment in the previous chapter, seen in Table XV, similarities can be measured between two different applications, and two different versions. AID could help us with detection, of compromised patching distribution supply chains, by measuring the diversification of the running applications, during the patching cycles.

6.3.2 Application whitelisting

While this control provides good mitigation in production environments, it is a manual, static, rigid control, and difficult to manage Software assurance is provided via a whitelist of good hashes or approved applications. This control lacks agility as to mitigating risk, against good

applications going rouge, Trojanised applications, and applications that have been compromised. AID attempts to provide software assurance of running software, by looking at the application's operational diversification during its entire life-cycle.

6.3.3 Privilege escalations

AID appears to be more sensitive with administrative tasks, which could make AID in CDI, a good capability for detecting privileges escalation attacks. The application that runs with basic user rights has, lower entropy and diversification, then with administrative rights.

6.3.4 Local cyber intelligence

Local cyber intelligence, could be effective against targeted attacks, and CDI could provide a capability for local cyber intelligence for running applications, by detecting anomalies and unknown changes. Critical in-house applications could be good candidates for AID and CDI.

6.3.5 Shadow IT

More applications will increase the entropy or diversification, and the AID of a system or environment. The CDI could potentially be utilised to manage Shadow IT on a larger scale. The increase of diversification in an environment could be as a result to introduction of unknown and unauthorised applications or systems, like in Shadow IT.

6.4 CDI domain placement

While there is a slight misconception that ML could solve the automation challenges in cybersecurity, it appears both domains have several challenges in common. Fig. 33 shows the placement of CDI in relation to all three domain.

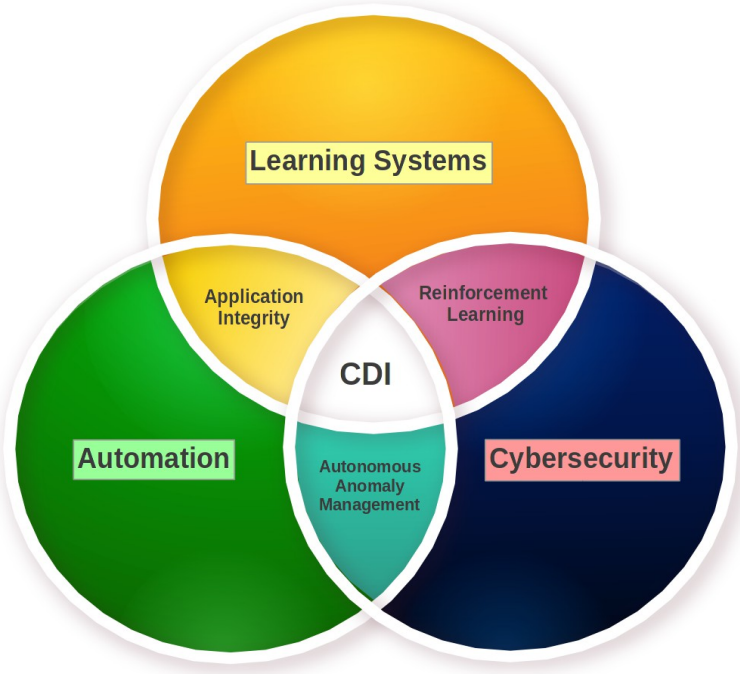

FIGURE 33. Cyber Diversity Index.

CDI driven by cybersecurity, could provide verification of application integrity, that drive exponential technologies using learning systems and automation.

This research reviewed unsupervised, supervised and reinforcement learning in ML and cybersecurity, the experiment in chapter 3 focuses on supervised learning due to the popularity for anomaly detection as seen in Fig 4. From the review and the experiment results, the reinforcement learning has been identified as the more suitable learner for CDI with AID that could provide the automation required for future autonomous anomaly management on the distributed hosts.

The research identifies many complex issues related to automation and cybersecurity that match and expand on the several gaps identified in research in Cyber Defense Automation (CDA) by Baah, Hobson, Okhravi, Roberts, Streilein, and Yuditskaya (2016). Autonomous anomaly management is identified as the most complex automation challenge towards anomaly-based research covering AIS (Forrest, Hofmeyr, Somayaji and Longstaff, 1996) and HIDS (Creech, & Hu, 2014) that use system calls

for application behavioural analysis (Das, Joshi and Finin, 2017). Further, entropic behavioural profiling for running applications is proposed in this research towards cybersecurity research in entropy (Mannaert, De Bruyn & Verelst, 2012), (Zhang & Veitch, 2011). Furthermore, CDI is proposed towards research in diversity index (Baudry & Monperrus, 2015) using Shannon Wiener Index (Wang, Duan, & Simmons, 2016) and cyberecology (Jorgensen & Rossignol, 2003) to define applications' entropic behavioural profiles and to sustain system integrity or self-control during unknown change.

6.5 Cyber ecological synthesis

Business and industries driven by the exuberant demand for digital data towards smart solutions. The accelerating advancements of bio-technologies, biosensors, 3D printing, green computing, advanced intelligent manufacturing and hyper interconnected systems will increase the need for merging or fusing of different physical environments from a digital perspective. While research conducted in cyber ecology (Jorgensen, Rossignol, Takikawa, Upper, 2001), applies ecological solutions to cyber problems, the merging of these environments, could be a new concept or trend, and that is the synthesis of these two different domains. The challenge, in the not so distant future, might be to determine the percentage of how cyber something is, in a cyber ecological synthesised environment.

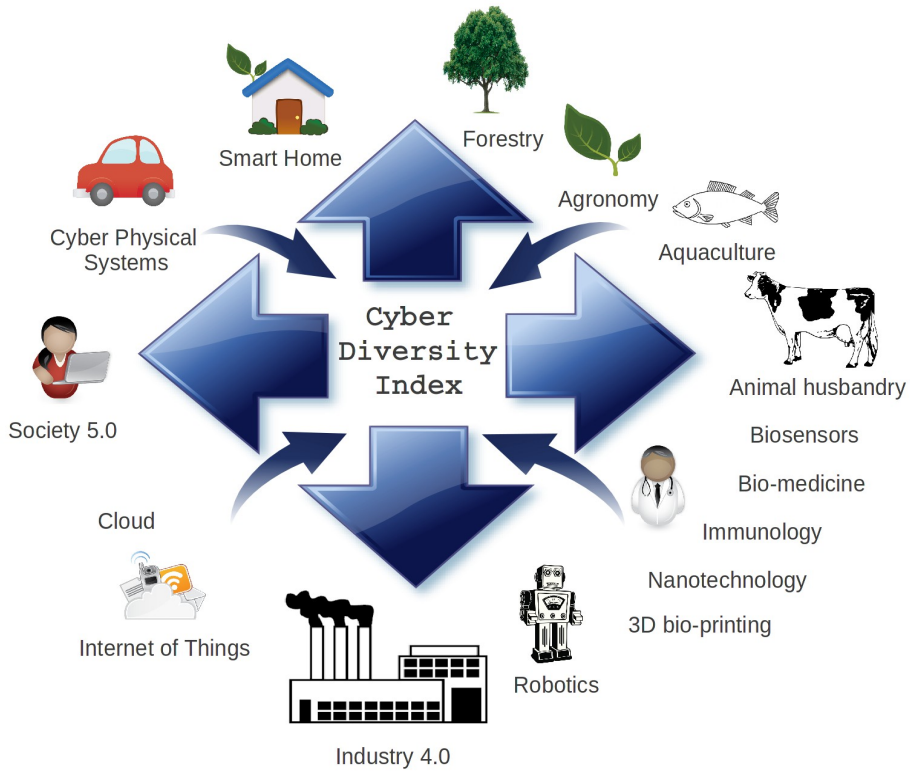

FIGURE 34. CDI and the cyber ecological synthesis.

Fig. 34 shows different technologies and industries that could merge at a digital level, due to increase of data points used from sensors developed in various domains. For example, diversity in cyber ecological synthesised environments could relate to CDI as follows:

- Devices with more applications could have higher CDI;
- People with more devices could have higher CDI;
- Companies with more systems could have higher CDI;
- Factories with more applications could have higher CDI;
- Environments with more cyber could have higher CDI;
- Robots will have higher AID and CDI than people;

6.6 Chapter conclusion

In this chapter, it was addressed the possible applications of CDI in practical, current and future scenarios. CDI attempts to provide a flexible solution to cybersecurity problems that are, or are going to, use automation and learning systems in the future. The CDI uses AID to provide assurance of dynamic applications, systems and environmental integrity. By providing continuous application integrity assurance, the CDI could be used in various industries using different exponential technologies. CDI is technology agnostic, which means, it aims to be applicable to all software driven technologies. CDI and AID could provide the base for future research in dynamic anomaly management solutions, utilising risk assessment and dynamic trust to manage unknown changes caused by zero-day attacks.

Chapter 7

7. Conclusion and future research

7.1 Introduction

The findings of a previous report by Forrest, Hofmeyr and Somayaji (2010), produced almost 10 years ago, appear to be still much applicable. A great deal of research has been conducted on anomaly detection and not enough on anomaly management and mitigation of cyber attacks. Unfortunately, the focus of research in automation and cybersecurity, has not been on complex issues, such as autonomous anomaly management, instead research in cybersecurity with learning systems still focuses on anomaly detection. Immune Systems and self-healing systems (Somayaji, & Forrest, 2000) are designed with a reactive approach to intrusions and taking little into account the management of unknown change or cyber anomaly as identified in this research. This research attempts to provide a flexible base with AID and CDI for future research toward an autonomous anomaly management solutions that use a combination of AID, with reinforcement learning and MTD.

7.2 Autonomous Anomaly Management

Every application, process and system could be a candidate for autonomous anomaly management. With the arrival of new business demands, exponential technologies, Internet+, Industry 4.0, and the increase of the global software footprint, it is inevitable that there will be an increase in anomalies, especially cyber anomalies related to the operational integrity of these technologies. Anomaly as defined in data science doesn't scale up well to cybersecurity, where cyber anomaly is the lack of knowledge relating to change in data, systems, people, and processes. Thus, cyber anomaly is not just a classification problem but also a change management problem. The aim of CDI is to

sustain the system's intrinsic purpose, with integrity being a self-control, during unknown changes.

With the increased popularity of distributed applications, the autonomous anomaly management could be implemented in a practical low risk distributed computing environment at the endpoints. By utilising reinforcement learning and the usage of the feedback loop, autonomous learning mitigation, could be applied to moving target defences and distributed deception platforms.

7.3 Autonomous Mitigation Strategies

7.3.1 Moving Target Defences (MTD)

Part of the anomaly management, when responding to zero-day attacks, could be to mitigate with Moving Target Defences (MTD), where the static nature of the security controls, the attacker's biggest advantage, is eliminated (Zhuang, Bardas, DeLoach, & Ou, 2015).

Researchers have identified the need to develop more dynamic security controls, to defend against advanced zero-day attacks. Implementation of decoying techniques could provide good mitigation strategy, such as decoying traffic of zero-day attacks to honeynets or isolated environments, using MTD or systems such as A2C that defends against malicious payload injection attacks. "A2C is based on the observation that payloads are highly fragile and thus any mutation would likely break their functionalities. Therefore, A2C mutates inputs from not trusted sources. Malicious payloads that reside in these inputs are hence mutated and broken" (Kwon, Saltaformaggio, Kim, Lee, Zhang, & Xu, 2017, p. 1).

7.3.2 Distributed Deception Platforms (DDP)

Almeshekah and Spafford (2016) use deception techniques, Distributed Deception Platforms (DDP), fake platforms to lure attackers, as part of their mitigation strategy for combating advanced cyber attacks. Deception techniques are: "Planned actions taken to mislead and/or confuse attackers and to thereby cause them to take (or not take) specific actions that aid computer-security defenses" (Almeshekah & Spafford, 2016, p. 2) which could be applied at each stage in the cyber kill chain.

Olagunju and Samu (2016) propose a hybrid system, a real time Intrusion Detection and Prevention System (IDPS) using honeypots, deception systems that are purposely made

vulnerable to entice attackers to execute attacks. The reason for incorporating honeypots in their solution is that the IDPS could generate many false positives, where these could be further analysed in a segmented environment. The honeynet system used in their experiments consists of a firewall, a router, a Bifrozt (2019), HonSSH (2019), Filebeat (2019), Elasticsearch (2019), Logstash (2019), Kibana (2019), Puppet (2019), four honeypots on virtual machines and a Puppet automation host. Results of their real online experiment performed over 10 days captured 498,471 attacks (Olagunju & Samu, 2016).

Hu, Hu, and Dolan-Gavitt (2018) introduce a defensive technique called chaff bugs. Their deceptive control does not eliminate bugs but it injects many non-exploitable bugs, chaff bugs, to programs nginx (2020), and libFLAC (2019). The intention is to waste the attacker's time when trying to build a working exploit.

“Bugs in software are both common and costly, these bugs can often be exploited by attackers to achieve arbitrary code execution. However, not all bugs are created equal. Depending on the exact nature of the bug and the run-time environment when it is triggered, a bug may not lead to a violation of the program’s security goals. In these cases we say that the bug is non-exploitable” (Hu, Hu, & Dolan-Gavitt, 2018, p. 1).

Hu, Hu, and Dolan-Gavitt report a couple of limitations in their research: it is assumed that the attacker has access only to the binaries and not to the source code, thus this control cannot protect open source software; and the chaff bugs are not indistinguishable from the real bugs, the artefacts, might warn the attackers to avoid and ignore the chaff bugs.

7.4 Automation and cybersecurity learning systems

Production environments that use anomaly detection with ML are using unsupervised learning that assume rarity of events. The learners are prone to false positives, and additional learning mechanisms are required (Tuggener, Amirian, Rombach, Lörwald, Varlet, Westermann, & Stadelmann, 2019). Supervised learning appears to be more popular with data scientists, and has less false positives but the creation of datasets is manual and expensive. Supervised learning in production requires new ML models to be generated regularly, due to changes in the normal operational baseline, such as implementation of new systems and technologies. Reinforcement learning appears to be better suited for development of autonomous distributed systems, that have mitigation

capability, provided by the feedback-loop that this learner has. The positive results from the AutoML experiments, shows that this automation solution that selects the best classifier and hyper-parameters can be used in production environments. However, a cybersecurity solution that provides an automation by using reinforcement learning for mitigation might be a more effective solution.

7.5 The research questions discussion

The Cyber Diversity Index and Application Integrity Diversification developed in this research show that it is possible to measure a system's entropic behavioural profile and detect anomalies caused by zero-day attacks, which addresses the main research question. Managing unknown changes to the application's behavioural profile could be possible by having a dynamic baseline and not relying on databases of IoC and ML datasets that need to be developed and managed manually, which addresses the first sub-question. Further, mitigation strategies could be implemented not by hard blocks but by deception techniques and MTD with risk assessment and dynamic trust. The research hypothesis is also confirmed in the experiments, that the AID and the system's entropic behavioural profile will increase during or after a zero-day attack. Reinforcement Learning appears to be more suitable for future cybersecurity research for this dynamic solution due to its feedback loop, which addresses the second sub-question.

7.6 Limitations of this research

While this research presented positive results, it is important to identify its limitations, or barriers for future research. The following limitations don't apply specifically to this research as such, but are relevant at a general level that targets a specific solution.

7.6.1 Automation paradox

It is important to understand the automation paradox from a cybersecurity perspective, because the impact of a problem that has not been identified earlier, in the stage of system or process development cycle, that is being automated will only increase in the future. This means the vulnerabilities that haven't been detected during the creation of the automation, will present a higher threat in the future (Boeing, 2020).

7.6.2 Entropy blind spots

While entropy based research, has provided positive results for anomaly detection, entropy has number of weakness, that need to be taken into account, before developing cybersecurity solutions for production environments. Entropy doesn't recognise the order of items it measures, and could only detect the heavy hitters or the items that make the most noise. This is the reason why in this research entropy was used in combination with other measurements, such as diversity and similarity indexes.

7.6.3 Risk assessment

Risk assessment is a difficult, even as a manual process, especially when attempting to mitigate against threats that have never been seen before, such as zero-day attacks. Different companies and users perceive risk differently, and have different level of risk appetite. Risk assessment that applies to cyber, might also trigger ethical dilemmas, and issues that could be difficult to solve with automation.

7.6.4 Resources intensity

This research was conducted on a small scale, and no apparent resource intensity was detected during the experiments. Applications were monitored, one at the time, and this could change if AID is used to monitor multiple applications on a single host. Up-scaling to large and complex IT/OT or cyber ecological environments might require additional resources that could test current technologies.

7.7 Conclusion

Automation challenges apply to both cybersecurity and learning systems. In this research, challenges were identified, and a novel solution was developed, a self populating index, CDI, that will contribute towards a joint, and a real time, automation solution for both domains.

This research identifies business applications, that support emerging technologies and specifically the anomalies they generate, with significant impact to cybersecurity. The focus of this research is on the development of CDI, and the theoretical autonomous anomaly management. The AID profiles that define the CDI also address, either

partially or fully, the many challenges that were identified in the literature review.

The results from the quantitative experiments, use Diversity Indexes and entropy to measure applications' behavioural profiles from data ingested from continuous sampling of system calls show modest and positive results.

The experiments in chapter 4 and 5 show that it is possible to monitor and establish a AID entropic behavioural profile, establish dynamic normal states, and detect mIoX during a zero-day attack. The experiments also confirms the research hypothesis that the system entropy increases during a zero-day attack. Experiment in chapter 3 shows that the most used supervised learner for anomaly detection is not necessary the most suitable for autonomous cybersecurity solution. The reinforcement learner is recommended for future autonomous anomaly management solutions.

AID could provide software assurance, for the application's integrity, that is needed for autonomous anomaly management. AID could also identify an application unique diversity footprint required for CDI. New non-string based indicators, called mIoX, are introduced in this research to narrow down on the vulnerability management problem. The Cybersecurity Triple Triad is introduced to expose the challenges of automation and the emerging future sources of new cyber anomalies.

References

- Abed, S. A., Clancy, C., and Levy, S. D. (2015). Intrusion detection system for applications using linux containers. In International Workshop on Security and Trust Management, pp. 123-135. Springer, Cham, 2015.
- Ablon, L., & Bogart, A. (2017). Zero Days, Thousands of Nights The Life and Times of Zero-Day Vulnerabilities and Their Exploits. RAND Corporation, Santa Monica, Calif. ISBN: 978-0-8330-9761-3, 2017
- ACS (2019). Watering Hole Attack. Australian Computer Society. Retrieved from https://ia.acs.org.au/article/2019/parliament-breached-by-watering-hole-attack.html
- Acker, F. (2015). Use of Entropy for Feature Selection with Intrusion Detection System Parameters. CEC Theses and Dissertations. Nova Southeastern University, Retrieved from http://nsuworks.nova.edu/cgi/viewcontent.cgi?article=1368&context=gscis_etd , 2015
- Almeshekah, H. M., & Spafford, H. E. (2016). Cyber Security Deception. Cyber Deception: Building the Scientific Foundation. 25-52. 10.1007/978-3-319-32699-3_2., 2016
- Anckaert, B., Sutter, D. B., & Bosschere, D. K. (2004). Software piracy prevention through diversity. In Proceedings of the 4th ACM workshop on Digital rights management (DRM '04). Association for Computing Machinery, New York, NY, USA, 63–71. DOI:https://doi.org/10.1145/1029146.1029157
- Anderson, J. (1973). Computer Security Technology Planning Study. ESD-TR-

73-51, US Air Force Electronic Systems Division (1973). Section 4.1.1, Retrieved from http://csrc.nist.gov/publications/history/ande72.pdf , 1973

- Anderson, S. H., Kharkar, A., Filar, B., Evans, D., & Roth, P. (2018). Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. arXiv preprint arXiv:1801.08917, 2018
- Andreessen, M.(2019). Why Software Is Eating The World? Retrieved from https://www.wsj.com/articles/SB10001424053111903480904576512250915629460
- Apache-Incubating (2020). Zipkin and Jaeger. Retrieved from https://developers.redhat.com/blog/2019/05/01/a-guide-to-the-open-source-distributed
- ASD (2019). Strategies to Mitigate Cyber Security Incidents - Mitigation Details. Retrieved from https://www.cyber.gov.au/publications/strategies-to-mitigate-cyber-security-incidents
- AXELOS (2019). What is ITIL? Retrieved from https://www.axelos.com/best-practice-solutions/itil/what-is-itil
- Baah, G., Hobson, T., Okhravi, H., Roberts, S., Streilein, W., & Yuditskaya, S. (2016). A Study of Gaps in Cyber Defense Automation. No. TR-1194. MIT Lincoln Laboratory Lexington United States, 2016
- Baudry, B., & Monperrus, M. (2015). The Multiple Facets of Software Diversity. ACM Computing Surveys. 48. 1-26. 10.1145/2807593.
- Baudry, B., Monperrus, M., Mony, C., Chauvel, F., Fleurey, F., & Clarke, S. (2014). DIVERSIFY: Ecology-inspired software evolution for diversity emergence. 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE),

Antwerp, 2014, pp. 395-398.

- BeEF (2020). The Browser Exploitation Framework Project. Retrieved from https://beefproject.com/
- Bergstra A. J., & Burgess, M. (2018). Blockchain Technology and its Applications A Promise Theory view-V0. 11. 2018
- Bernaschi, M., Gabrielli, E., & Mancini. V. L. (2000). Operating system enhancements to prevent the misuse of system calls. Proceedings of the 7th ACM conference on Computer and communications security. ACM, 2000.
- Bifrozt (2019). Honeynet. Retrieved from https://www.honeynet.org/node/1191
- Bilge, L., Sen, S., Balzarotti, D., Kirda, E., & Kruegel, C. (2014). Exposure: A Passive DNS Analysis Service to Detect and Report Malicious Domains. ACM Trans. Inf. Syst. Secur. 16, 4, Article 14, 2014
- Blunden, B. (2012). The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System, Jones & Bartlett Learning, ISBN-13: 978-1449626365, ISBN-10: 144962636X, 2012
- Boeing (2020). 737 Max software updates. Boeing. Retrieved from https://www.boeing.com/commercial/737max/737-max-software-updates.page
- Böhme M. (2018). STADS: Software Testing as Species Discovery. ACM Trans. Softw. Eng. Methodol. 27, 2, Article 7 (June 2018), 52 pages. DOI:https://doi.org/10.1145/3210309
- Böhme, M. (2019). AFLFast. Retrieved from https://github.com/mboehme/aflfast
- Böttinger, K., Godefroid, P., & Singh, R. (2018). Deep reinforcement fuzzing. In 2018 IEEE Security and Privacy Workshops (SPW) (pp. 116-122). IEEE.
- Brenne, L. (2019). What Is a Symbiotic Relationship? Retrieved from

https://sciencing.com/symbiotic-relationship-8794702.html

- Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications Surveys & Tutorials, Volume: 18, Issue: 2, Pages: 1153 - 1176, DOI:10.1109/COMST.2015.2494502. 2016
- Bullough, B.L., Yanchenko, A.K., Smith, C.L., & Zipkin, J.R. (2017). Predicting Exploitation of Disclosed Software Vulnerabilities Using Open-source Data. ArXiv, abs/1707.08015.
- Burgess, M. (2005). An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation. In: Schönwälder J., Serrat J. (eds) Ambient Networks. DSOM 2005. Lecture Notes in Computer Science, vol 3775. Springer, Berlin, Heidelberg
- Burgess, M., Haugerud, H., Straumsnes, S., and Reitan, T., (2002). Measuring system normality. ACM Trans. Computer Systems, 2002
- Busoniu, L., Babuska, R., and Schutter, D. B. (2006). Multi-agent reinforcement learning: A survey. In 2006 9th International Conference on Control, Automation, Robotics and Vision, pp. 1-6. IEEE, 2006.
- Chess, B. & West, J. (2007). Secure Programming with Static Analysis. Addison-Wesley Professional. ISBN-10: 0321424778, ISBN-13: 978-0321424778
- Christiansen, M. (2010). Bypassing Malware Defenses. SANS. Retrieved from https://www.sans.org/reading-room/whitepapers/malicious/bypassing-malware-defenses-33378
- CIS (2019). Cisecurity. Retrieved from https://www.cisecurity.org/
- CISA (2017). Information Sharing. DHA. Retrieved from https://www.us-cert.gov/Information-Sharing-Specifications-Cybersecurity

- Cisco (2018). Shadow IT. Retrieved from https://www.cisco.com/c/en/us/products/security/what-is-shadow-it.html
- Columbus, L. (2018). IBM Predicts Demand For Data Scientists Will Soar 28% By 2020. Retrieved from https://www.forbes.com/sites/louiscolumbus/2017/05/13/ibm-predicts-demand-for-data-scientists-will-soar-28-by-2020/#78fe65f07e3b , 2017
- Columbus, L. (2019). The Future Of Manufacturing Technologies. Retrieved from https://www.forbes.com/sites/louiscolumbus/2018/04/15/the-future-of-manufacturing-technologies-2018/#549b77922995
- Corona, I., Biggio, B., & Maiorca, D. (2016). AdversariaLib: An open-source library for the security evaluation of machine learning algorithms under attack," arXiv preprint arXiv:1611.04786, 2016
- Creech G., and Hu J., (2017). ADFA IDS Dataset, University of Arizona Artificial Intelligence Lab. AZSecure-data, Director Hsinchun Chen. Retrieved from http://www.azsecure-data.org/
- Creech, G., and Hu, J. (2014). A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns. IEEE Trans. Comput. 63, 4, 807-819
- Cuckoo (2017), Cuckoo. Retrieved from https://cuckoosandbox.org/ 20 May 2017,
- CVE-2018-8733 (2018). Mitre. Retrieved from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8733
- CVE-2018-8734 (2018). Mitre. Retrieved from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8734
- CVE-2018-8735 (2018). Mitre. Retrieved from https://cve.mitre.org/cgi-

bin/cvename.cgi?name=CVE-2018-8735

- CVE-2018-8736 (2018). Mitre. Retrieved from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8736
- Darktrace (2018). The Enterprise Immune System. Retrieved from https://www.darktrace.com/en/technology/
- Das, K. P., Joshi, A., & Finin, T. (2017). App behavioral analysis using system calls. 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2017.
- Davi, L., Dmitrienko, A., Nürnberger, S., & Sadeghi, A. (2012). XIFER: A Software Diversity Tool Against Code-Reuse Attacks. (2012).
- Dayan, P., & Niv, Y., (2008). Reinforcement learning: The Good, The Bad and The Ugly. Current opinion in neurobiology 18, no. 2, 2008
- DeCianno, J. (2017). Indicators of Attack vs. Indicators of Compromise. CrowdStrike. Retrieved from https://www.crowdstrike.com
- Denning, D. (1987). An Intrusion-Detection Model. IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, pp.222-232, 1987
- Devi R. R. & M. Abualkibash M., Intrusion Detection System Classification Using diffrent Machine Learning Algorithms on KDD-99 and NSL-KDD dataests - Review Paper. International Journal of Computer Science & Information Technology (IJCSIT). Vol 11, No 3, June 201
- Dietterich, T. (2018). Anomaly Detection: Algorithms, Explanations, Applications. Oregon State University. Retrieved from https://www.microsoft.com/en-us/research/video/anomaly-detection-algorithms-explanations-applications/ March 12, 2018.
- Ding, Y., Wei, T., Xue, H. Zhang, Y., Zhang C., & Han, X. (2017). Accurate and

efficient exploit capture and classification. Sci. China Inf. Sci. 60, 052110 (2017). https://doi.org/10.1007/s11432-016-5521-0

- DISA (2019). STIGS. DISA. Retrieved from https://iase.disa.mil/stigs/Pages/index.aspx
- Docker (2020). Seccomp security profiles for Docker. Docker. Retrieved from https://docs.docker.com/engine/security/seccomp/
- Domingos, P. (2018). The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. ISBN-10: 0465094279, ISBN-13: 978-0465094271, Basic Books; Reprint edition, 2018.
- Donevski, M., and Zia, T. (2018). A Survey of Anomaly and Automation from a Cybersecurity Perspective. 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, United Arab Emirates, 2018
- Drucker, F. P. (2011). The Essential Drucker: Selections from the Management Works of Peter F. Drucker. Butterworth-Heinemann Management, ISBN 0750650184, 9780750650182, 2011
- DTMG (2019). AI in Industry 4.0. Retrieved from https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/AI-in-Industrie4.0.html
- Duessel, P., Gehl, C., Flegel, U., Dietrich, S., & Meier, M. (2017). Detecting zero-day attacks using context-aware anomaly detection at the application-layer. International Journal of Information Security, pp 1–16, 2017
- Eidle, D., Ni, Y. S., DeCusatis, C., & Sager, A. (2017). Autonomic security for zero trust networks. In 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), pp. 288-293. IEEE,2017.
- Elasticsearch (2019). Elastic. Retrieved from https://www.elastic.co/

- Engelbrecht, S. (2017). The Evolution of SOAR Platforms Security Orchestration, Automation and Response (SOAR). Retrieved from https://www.securityweek.com/evolution-soar-platforms
- ESET (2019). Unified Extensible Firmware Interface (UEFI). Retrieved from https://www.welivesecurity.com/2018/09/27/lojax-first-uefi-rootkit-found-wild-courtesy-sednit-group/
- Eskandari, S., Khreich, W., Murtaza, S., Hamou-Lhadj, A., & Couture, M. (2013). Monitoring system calls for anomaly detection in modern operating systems. 2013 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2013. 19-20. 10.1109/ISSREW.2013.6688856.
- Filebeat (2019). Elastic. Retrieved from https://www.elastic.co/products/beats/filebeat
- Firefox (2020). Firefox. Mozila. Retrieved from https://www.mozilla.org/en-US/firefox/new/
- Forrest, S., Hofmeyr, A., S., Somayaji, A., and Longstaff, A., T. (1996). A Sense of Self for Unix Processes. In Proceedings of the 1996 IEEE Symposium on Security and Privacy (SP '96). IEEE Computer Society, Washington, DC, USA.
- Forrest, S., Hofmeyr, S., & Somayaji, A. (2008). The Evolution of System-Call Monitoring. In Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC '08). IEEE Computer Society, Washington, DC, USA, 418-430, 2008
- Forrest, S., Somayaji, A., & Ackley, D. H. (1997). Building diverse computer systems, Proceedings. The Sixth Workshop on Hot Topics in Operating Systems (Cat. No.97TB100133), Cape Cod, MA, USA, 1997, pp. 67-72

- Frankenfield, J. (2019). 51% Attack. Retrieved from https://www.investopedia.com/terms/1/51-attack.asp
- Ganz, J., & Peisert, S. (2017). ASLR: How Robust is the Randomness? 2017 IEEE Cybersecurity Development (SecDev). IEEE, 2017.
- Garfinkel, T. (2003). Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools. In NDSS, vol. 3, pp. 163-176, 2003.
- Gartner (2016). Gartner's 2016 Hype Cycle for Emerging Technologies. Identifies Three Key Trends That Organizations Must Track to Gain Competitive Advantage," STAMFORD, Retrieved from http://www.gartner.com/newsroom/id/3412017
- Gaudin, B., Vassev, E., Hinchey, M., Nixon, P., Pagano, D., Garcia, C. J., & Narayan, N. (2010). State-of-the-art in Self-Healing and Patch Generation. Lero, European Commission under the Seventh Framework Programme, 2010
- Goertzel, K. M., Winograd, T., McKinley, H. L. Lyndon, J. O., Colon, M., McGibbon, T., Fedchak, E., and Vienneau, R. (2018). Software Security Assurance: A State-of-Art Report (SAR). Information Assurance Technology Analysis Centre (IATAC) HERNDON VA, 2007, Retrieved from http://www.dtic.mil/dtic/tr/fulltext/u2/a472363.pdf
- Graeber, D. (2018). Bullshit Jobs. Simon & Schuster, ISBN: 978-1-5011-4331-1, 2018
- Grafana (2019). Grafana Labs. Retrieved from https://grafana.com/
- Graham, A., Liang, Y. A., Gruenwald, L., & Grant, C. (2017). Formalizing interruptible algorithms for human over-the-loop analytics. In 2017 IEEE International Conference on Big Data (Big Data), pp. 4378-4383. IEEE, 2017.
- Greengard, S. (2020). Making Automation Work , COMMUNICATIONS OF

THE ACM , DOI:10.1145/1610252.1610261, Retrieved from https://dl.acm.org/doi/pdf/10.1145/1610252.1610261?casa_token=f_viqkXw5VgAAAAA:6KsBviMdcTfWAhpCX5SkypFtGra_G9EeH-C8rst0SM2e_kGNw4vraa09ijOe_S4bBun1IXuPqrLvcw

- Gregg, B. (2019). An introduction to bpftrace for Linux. Retrieved from https://opensource.com/article/19/8/introduction-bpftrace
- Guardian (2019). Equifax. Retrieved from https://www.theguardian.com/technology/2017/sep/15/equifax-hack-susan-mauldin-david-webb
- Gyuwan, K., Hayoon, Y., Jangho, L., Yunheung, P., & Sungroh, Y. (2017). LSTM-Based System-Call Language Modeling and Robust Ensemble Method for Designing Host-Based Intrusion Detection Systems. ARXIV. eprint arXiv:1611.01726. Retrieved from https://arxiv.org/abs/1611.01726
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, H. I. (2009). The WEKA Data Mining Software: An Update. SIGKDD Explorations, Volume 11, Issue 1.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, H. I. (2009). The WEKA Data Mining Software: An Update. SIGKDD Explorations, Volume 11, Issue 1, 2009
- Haq, F. N., Onik, R. A., Hridoy, K. A. M., Rafni, M., Shah, M. F., & Farid, M. D.,(2015). Application of machine learning approaches in intrusion detection system: a survey. IJARAI-International Journal of Advanced Research in Artificial Intelligence, 4(3), pp.9-18, 2015
- Harvey, C. (2019). 11 Top Breach and Attack Simulation (BAS) Vendors. Retrieved from https://www.esecurityplanet.com/products/top-breach-and-

attack-simulation-bas-vendors.html

- Heraclitus (2019). Change. Retrieved from https://en.wikiquote.org/wiki/Heraclitus
- History-Computer (2019). Manchester Baby. Retrieved from https://history-computer.com/ModernComputer/Electronic/SSEM.html
- Hofmeyr, A. S., & Forrest. A. S. (2000). Architecture for an Artificial Immune System. Evol. Comput. 8, 4 (December 2000), 443–473. DOI:https://doi.org/10.1162/106365600568257
- HonSSH (2019). Github. Retrieved from https://github.com/tnich/honssh
- Howard, M., & Lipner, S. (2006). Microsoft Security Development Life Cycle (SDL). The Security Development Lifecycle. Microsoft Press, Redmond, WA, USA, 2006
- Hu, Hu, Y., & Dolan-Gavitt, B. (2018). Chaff Bugs: Deterring Attackers by Making Software Buggier. arXiv preprint arXiv:1808.00659, 2018
- Hutchins, M. E., Cloppert, J. M., & Amin, M. R. (2014). Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. Lockheed Martin Corporation. Retrieved from https://www.nationalcyberwatch.org/ncw-content/uploads/2016/03/LM-White-Paper-Intel-Driven-Defense-1-1.pdf , 2014
- IBM (2019). System calls. IBM. Retrieved from https://developer.ibm.com/tutorials/l-system-calls
- IBM-Integrity (2020). IBM, Retrieved from https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieaa800/iea3a8_System_integrity.htm
- IETF (2019). Request for Comments (RFC). Retrieved from

https://www.ietf.org/standards/rfcs/

- Industrie 4.0 (2019). DTM Germany. Retrieved from https://ec.europa.eu/growth/tools-databases/dem/monitor/sites/default/files/DTM_Industrie%204.0.pdf
- Irani, M. T., and Weippl, E. R. (2009). Automation of post-exploitation. International Journal of Web Information Systems. Vol. 5 Issue: 4, pp.518-536, Retrieved from https://doi.org/10.1108/17440080911006234, 2009
- Islam, M. Z., & Giggins, H. (2011). Knowledge Discovery through SysFor: A Systematically Developed Forest of Multiple Decision Trees. In V. Estivill-Castro, & S. Simoff (Eds.), 9th Australasian Data Mining Conference: AusDM 2011 (Vol. 121, pp. 195-204). (Conferences in Research and Practice in Information Technology Series; Vol. 121). Australian Computer Society Inc.
- ISO 3001 (2018). Risk management. Retrieved from https://www.iso.org/publication/PUB100426.html
- ISO-IEC 27000 (2018). Information security management systems. Retrieved from https://www.iso.org/isoiec-27001-information-security.html
- Iwamoto K., & Wasaki, K. (2015). The Method for Shellcode Extraction from Malicious Document File Using Entropy and Emulation. Journal of Information Processing. 56. 892-902.5, 2015
- Jaeger, T. (2011). Reference monitor. Encyclopedia of Cryptography and Security. pp.1038-1040, 2011
- Jenkinson, G. (2019). ETC 51 attack.Retrieved from https://cointelegraph.com/news/ethereum-classic-51-attack-the-reality-of-proof-of-work
- Jorgensen J. and Rossignol P. (2003) INFORMATION ASSURANCE CYBER

ECOLOGY IET, Incorporated - Sponsored by Defense Advanced Research Projects Agency (DARPA)

- Jorgensen, J., Rossignol, P., Takikawa, M., and Upper, D. (2001) "Cyber ecology: looking to ecology for insights into information assurance," Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01, Anaheim, CA, USA, 2001, pp. 287-296 vol.2. doi: 10.1109/DISCEX.2001.932180 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=932180&isnumber=20160
- Julien, J. (2018). Performance Engineering. The What, Why, and How's Explained. John Julien April 19, 2018 Developer Tips, Tricks & Resources Retrieved from https://stackify.com/performance-engineering/
- Kalinichenko, L., Shanin, I., & Taraban, I. (2014). Методы выявления аномалий: обзор (Methods for Anomaly Detection: a Survey). RCDL, 2014
- Kantee, A. (2015). The Rise and Fall of the Operating System, Usenix.org. ;login: issue: October 2015, Vol. 40, No. 5. Retrieved from: https://www.usenix.org/publications/login/oct15/kantee , 2015
- Kaur, R., & Singh, M.(2014). Survey on Zero-Day Polymorphic Worm Detection Techniques. IEEE Communications Surveys & Tutorials, Volume: 16, Issue: 3, Pages: 1520 – 1549, 2014
- Kdnuggets (2019). 9 Must have skills for DS. Retrieved from https://www.kdnuggets.com/2018/05/simplilearn-9-must-have-skills-data-scientist.html
- Kernel (2019). Linux Security Modules (LSM). Retrieved from https://www.kernel.org/doc/html/v4.15/admin-guide/LSM/index.html

- Kibana (2019). Elastic. Retrieved from https://www.elastic.co/
- Kohavi, R. (1995). The power of decision tables. In Proceedings of the 8th European Conference on Machine Learning (ECML'95). Springer-Verlag, Berlin, Heidelberg, 174–189. DOI:https://doi.org/10.1007/3-540-59286-5_57
- Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016). Deep learning for classification of malware system call sequences. In Australasian Joint Conference on Artificial Intelligence, pp.137-149. Springer, Cham, 2016.
- Kosko, B. (1994). Fuzzy Thinking: The New Science of Fuzzy Logic. Reprint Edition. New York. Hyperion; Reprint edition, 1994.
- Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. Journal of Machine Learning Research. 18. 1-5.
- Kozina, S. (2020). Introduction to eBPF in Red Hat Enterprise Linux 7. Red Hat. Retrieved from https://www.redhat.com/en/blog/introduction-ebpf-red-hat-enterprise-linux-7
- Kwon, Y., Saltaformaggio, B., Kim, L. I., Lee, K. H., Zhang, X., & Xu, D. (2017). A2C: Self Destructing Exploit Executions via Input Perturbation. Department of Computer Science, Purdue University, 2017
- Larsen, P., Homescu, A., Brunthaler S., & Franz, M. (2014). SoK: Automated Software Diversity. 2014 IEEE Symposium on Security and Privacy, San Jose, CA, 2014, pp. 276-291. doi: 10.1109/SP.2014.25
- Lee, W., & Xiang, D. (2001). Information-theoretic measures for anomaly detection. In Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001, pp. 130-143. IEEE, 2001.
- Levi. O., (2012). Intel Pin Tool. Intel. Retrieved from

https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

- Lewis, J. (2018). Economic Impact of Cybercrime - No Slowing Down, McAfee Center for Strategic and International Studies (CSIS), Retrieved from: https://www.mcafee.com/us/resources/reports/restricted/economic-impact-cybercrime.pdf
- Li, X., hu, Z., Fu, Y., Chen, P., Zhu, M. & Liu, P. (2018). ROPNN: Detection of ROP Payloads Using Deep Neural Networks.
- Li, Y., Ji, S., Lv, C., Chen, Y., Chen, J., Gu, Q., & Wu, C. (2019). V-Fuzz: Vulnerability-Oriented Evolutionary Fuzzing. arXiv preprint arXiv:1901.01142 (2019).
- libFLAC (2019). Xiph. Retrieved from Available: https://xiph.org/flac/
- Logstash (2019). Elastic. Retrieved from https://www.elastic.co/
- Lowd D., & Meek, C. (2005). Adversarial learning, In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD '05). ACM, New York, NY, USA, 641-647, 2005
- Malware-Zoo (2017). Malware-Zoo. Github. Retrieved from https://github.com/unixfreak0037/mwzoo 20 May 2017,
- Mannaert, H., Bruyn, D. P., & Verelst, J. (2012). Exploring entropy in software systems: Towards a precise definition and design rules. In Proceedings of the Seventh International Conference on Systems, ICONS, pp. 93-99, 2012.
- Mavenquist (2018). Docker. Retrieved from https://hub.docker.com/r/mavenquist/nagios-xi
- Mazel, J., Owezarski, P., & Labit, Y. (2010). 0day Anomaly Detection Made Possible Thanks to Machine Learning. 5th Conference on Network Architectures

and Information Systems Security (Menton). SAR-SSI 2010, May 18-21 2010

- Miller, B., Kantchelian, A., Afroz, S., Bachwani, R., Dauber, E., Huang, L., Tschantz, M.C., Joseph, A.D. & Tygar, J.D., (2014) . Adversarial active learning. In Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop (pp. 3-14).
- Monkey (2019). UI/Application Exerciser Monkey. Android, Retrieved from https://developer.android.com/studio/test/monkey
- Morgan, S. (2015). Cybersecurity Business Report. csoonline.com. Retrieved from http://www.csoonline.com/article/2953258/it-careers/cybersecurity-job-market-figures-2015-to-2019-indicate-severe-workforce-shortage.html.
- Morgan, S. (2019) Cybersecurity Jobs Report 2018-2021. Retrieved from https://cybersecurityventures.com/jobs/ , 2017
- Nagios-exploit (2020). Nagios XI 5.2.6-5.4.12 - Chained Remote Code Execution. Retrieved from https://www.rapid7.com/db/modules/exploit/linux/http/nagios_xi_chained_rce_2_electric_boogaloo
- Nagios-XI (2019). Nagios XI. Nagios. Retrieved from https://www.nagios.com/products/nagios-xi/
- Nagios-XI-D (2019). Docker. Retrieved from https://hub.docker.com/r/mavenquist/nagios-xi
- Neti, S., Somayaji, A., & Locasto, E. M. (2012). Software Diversity: Security, Entropy and Game Theory. 2012, HotSec.
- Nginx (2020). Nginx. Retrieved from https://www.nginx.com/
- NIST (2019). The Security Content Automation Protocol (SCAP). Retrieved from https://csrc.nist.gov/projects/security-content-automation-protocol/

- NIST-800-30 (2018). NIST P 800-30 Rev. 1, Guide for Conducting Risk Assessments. Joint Task Force Transformation Initiative. Retrieved from https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final
- NIST-IoMT (2019). Internet of Medical Things (IoMT) devices, NIST Special Publication 1800-8: Securing Wireless Infusion Pumps. Retrieved from https://www.nccoe.nist.gov/publication/1800-8/VolA/index.html.
- NIST-SCAP (2019). SCAP 1.3. NIST. Retrieved from https://csrc.nist.gov/Projects/Security-Content-Automation-Protocol/SCAP-Releases/SCAP-1-3
- NSA (2019). SELinux. Retrieved from https://www.nsa.gov/What-We-Do/Research/SELinux/
- Olagunju, A. O., & Samu, F. (2016). In Search of Effective Honeypot and Honeynet Systems for real time Intrusion Detection and Prevention. In Proceedings of the 5th Annual Conference on Research in Information Technology, ACM, New York, NY, USA, 41-46. 2016
- Oltsik, J., & Cahill, D. (2017). Industry Leaders Discuss SOAPA, Security Operations and Analytics Platform Architecture (SOAPA). Retrieved from https://www.esg-global.com/soapa
- Onapsis (2019). 10KBLAZE. Retrieved from https://www.onapsis.com/10kblaze
- Ooms, J. (2013). The Rapparmor package: Enforcing security policies in R using dynamic sandboxing on linux. arXiv preprint arXiv:1303.4808, 2013
- OpenAI (2017). OpenAI. Retrieved from https://gym.openai.com/
- OpenSCAP (2019). Open-Scap. Retrieved from https://www.open-scap.org/
- OWASP (0219). Secure Software Development Lifecycle Project(S-SDLC). Retrieved from

https://www.owasp.org/index.php/OWASP_Secure_Software_Development_Lifecycle_Project

- Oxford Dictionaries (2019). Cybersecurity. Retrieved from https://en.oxforddictionaries.com/definition/cybersecurity
- Paganini, P. (2019). 1-day exploits. Retrieved from https://securityaffairs.co/wordpress/3913/cyber-crime/1-day-exploitsbinary-diffing-patch-management-the-side-threats.html
- Phreaklets (2014). Using beef to pop shell. Retrieved from http://phreaklets.blogspot.com/2014/04/using-beef-metasploit-to-pop-shell-with.html
- PRC (2019). The State Council The People's Republic of China. Internet Plus. Retrieved from http://english.gov.cn/2016special/internetplu
- Prometheus (2019). Prometheus. Retrieved from https://prometheus.io/
- Provos, N. (2003). Improving host security with system call policies. In Proceedings of the 12th conference on USENIX Security Symposium - Volume 12 (SSYM'03), Vol. 12. USENIX Association, Berkeley, CA, USA, 18-18, 2003
- Puppet (2019). Puppet. Retrieved from https://puppet.com/
- Pushgateway (2019). Prometheus. Retrieved from https://github.com/prometheus/pushgateway
- Qiu, J., Yadegari, B., Johannesmeyer, B., Debray, S., & Su, X. (2015). Identifying and Understanding Self-Checksumming Defenses in Software. In Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY '15). Association for Computing Machinery, New York, NY, USA, 207–218. DOI:https://doi.org/10.1145/2699026.2699109 2015.
- Quinlan R. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann

Publishers, San Mateo, CA, 1993

- Rapid7 (2017). Metasploit. Rapid7, Retrieved from https://www.metasploit.com/
- Ravishankar, R. N., & Vijayakumar, V. M. (2017). Reinforcement Learning Algorithms: Survey and Classification. Indian Journal of Science and Technology. 10. 10.17485/ijst/2017/v10i1/109385, 2017
- Rawat, S., et al. (2017) VUzzer: Application-aware Evolutionary Fuzzing. NDSS. Vol. 17. 2017.
- Red Hat (2019). What is DevSecOps? Retrieved from https://www.redhat.com/en/topics/devops/what-is-devsecops
- Robertson, A. (2019). bpftrace. Github. Retrieved from https://github.com/iovisor/bpftrace
- Rodola, G., (2019). "Pypi," Psutil, Retrieved from https://pypi.org/project/psutil/
- Rugged (2019). Rugged Software. Retrieved from https://ruggedsoftware.org/
- Samuel, L. A. (1959). Some Studies in Machine Learning Using the Game of Checkers," in IBM Journal of Research and Development, vol. 3, no. 3, pp. 210-–229, July 1959. doi: 10.1147/rd.33.0210 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5392560&isnumber=5392559
- Santos, A., Castelo, S., Felix, C., Ono, P. J., Yu, B., Hong, R.S., Silva, T.C., Bertini, E., and Freire, J. (2019). Visus: An Interactive System for Automatic Machine Learning Model Building and Curation. In Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA'19). Association for Computing Machinery, New York, NY, USA, Article 6, 1–7. DOI:https://doi.org/10.1145/3328519.3329134
- Schneier, B. (2018). Click Here to Kill Everybody: Security and Survival in a

Hyper-connected World. W. W. Norton & Company, 2018

- Shacham, H. (2007). The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)," In ACM conference on Computer and communications security, pp. 552-561. 2007.
- Shannon, C.E. (1948), A Mathematical Theory of Communication. Bell System Technical Journal, 27: 379-423. doi:10.1002/j.1538-7305.1948.tb01338.x
- Siers, M. J., & Islam, M. Z. (2015). Cost-Sensitive Decision Forest: CSForest.
- Small, P. (2019). Defense in Depth: An Impractical Strategy for a Cyber World. SANS. Retrieved from https://www.sans.org/reading-room/whitepapers/warfare/paper/33896
- Smutz, C. & Stavrou, A. (2012). Malicious PDF Detection Using Metadata and Structural Features. in Proceedings of the 28th Annual Computer Security Applications Conference, New York, NY, USA, 2012
- Society 5.0 (2019). Society 5.0. Cabinet Office Government of Japan. Retrieved from https://www.gov-online.go.jp/cam/s5/eng/index.html
- Somayaji, A., & Forrest, S. (2000). Automated response using system-call delays. In Proceedings of the 9th conference on USENIX Security Symposium - Volume 9 (SSYM'00), Vol. 9. USENIX Association, Berkeley, CA, USA, 14-14. 2000
- Sood, K. A. & Enbody, R. (2014). Targeted Cyber Attacks Multi-staged Attacks Driven by Exploits and Malware. Syngress , ISBN-10: 0128006048, ISBN-13: 978-0128006047, 2014
- Sørensen index (2020). enacademic Retrieved from https://enacademic.com/dic.nsf/enwiki/8107894
- Spring T. (2019). Pop a Shell. Threat Post. Retrieved from

https://threatpost.com/researcher-exploits-microsofts-notepad-to-pop-a-shell/145242/

- Srndic, N., & Laskov, P. (2014). Practical Evasion of a Learning-Based Classifier: A Case Study. IEEE Symposium on Security and Privacy, Pages: 197 - 211, DOI: 10.1109/SP.2014.20. Retrieved from http://ieeexplore.ieee.org/document/6956565/ 2014
- Standards (2019). INDUSTRY 4.0: AN AUSTRALIAN PERSPECTIVE. Retrieved from https://www.standards.org.au/getmedia/29653164-cd4d-43f0-9afc-e8db58710f2e/Industry-4-0-Recommendations-Report.pdf
- Sutton, M., Greene, A., & Amini, P. (2007). Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley Professional; 1 edition. 0321446119, 2007
- Takahashi, T., & Kadobayashi, Y. (2015). Reference Ontology for Cybersecurity Operational Information. The Computer Journal, vol. 58, no. 10, pp. 2297-2312, Oct. 2015.
- Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 2009
- Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 2009.
- Tizen (2019). Smack. Retrieved from https://research.samsung.com/tizen
- Tomoyo (2019). Tomoyo Linux, Retrieved from http://tomoyo.osdn.jp/
- Tuggener, L., Amirian, M., Rombach, K., Lörwald, S., Varlet, A., Westermann, C., & Stadelmann, T. (2019). Automated machine learning in practice: state of

the art and recent results. In 2019 6th Swiss Conference on Data Science (SDS) (pp. 31-36). IEEE.

- Ubuntu (2019). AppRmour. Retrieved from https://wiki.ubuntu.com/AppArmor
- US-CERT (2018). Information Sharing Specifications for Cybersecurity. Retrieved from https://www.us-cert.gov/Information-Sharing-Specifications-Cybersecurity
- Vaas C., & Happa, J., (2017). Detecting disguised processes using application-behavior profiling. In 2017 IEEE International Symposium on Technologies for Homeland Security (HST), pp. 1-6. IEEE, 2017.
- van der Aalst, P. M. W., Bichler, M., & Heinzl, A. (2018). Robotic Process Automation. Bus Inf Syst Eng (2018) 60: 269. https://doi.org/10.1007/s12599-018-0542-4
- Vanschoren, J., Van Rijn, J. N., Bischl, B., & Torgo, L. (2014). OpenML: networked science in machine learning. ACM SIGKDD Explorations Newsletter, 15(2), 49-60.
- Wang, C. (2018). Protecting Containerized Applications with System Call Profiling and Whitelisting. Retrieved from https://www.twistlock.com/2016/12/06/protecting-containerized-apps/ , 2016
- Wang, Y. (2019). Tencent's 'super app' wechat is quietly taking over workplaces in china. Retrieved from https://www.forbes.com/sites/ywang/2016/08/19/tencents-super-app-wechat-is-quietly-taking-over-workplaces-in-china/
- Wang, Y., Duan, Q., & Simmons, D. (2016). Security Evaluation Using Software Diversity Measurement: An Ecological Approach. Int'l Conf. Software Eng. Research and Practice SERP'16

- Waqas, H., Creech, G., Xie, Y., and Hu, J. (2016). Windows based data sets for evaluation of robustness of host based intrusion detection systems (IDS) to zero-day and stealth attacks. Future Internet 8, no. 3 (2016): 29.
- Watson, M. N. R, (2007). Exploiting concurrency vulnerabilities in system call wrappers. In Proceedings of the first USENIX workshop on Offensive Technologies (WOOT '07). USENIX Association, Berkeley, CA, USA, 2007
- Webroot, (2017). Businesses recognise the need for AI & ML tools in cybersecurity. Retrieved from https://www.helpnetsecurity.com/2019/03/14/ai-ml-tools-cybersecurity/ Hal Lonas, CTO, Webroot. Help Net Security, 2019
- Weimer, W., Forrest, S., Kim, M., Goues, L. C., & Hurley, P. (2016). Trusted software repair for system resiliency. In 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), pp. 238-241. IEEE, 2016.
- White, S. (2019). Data breach bill for Equifax up to $1.4 billion. Retrieved from https://gdpr.report/news/2019/05/14/data-breach-bill-for-equifax-up-to-1-4-billion/
- Whitham, J. (2016). Software Engineer, Rapita Systems, York Retrieved from https://www.jwhitham.org/2016/02/profiling-versus-tracing.html
- Wiener, N. (1961). Cybernetics: Or Control and Communication in the Animal and the Machine. Paris, (Hermann & Cie) & Camb. Mass. (MIT Press) ISBN 978-0-262-73009-9; 2nd revised ed. 1961.
- Wissner-Gross, A. (2017). Datasets Over Algorithms. Edge. Retrieved from https://www.edge.org/response-detail/26587
- Wright, S. C. & Zia, A. T. (2011). Rationally Opting for the Insecure Alternative: Negative Externalities and the Selection of Security Controls. In: Herrero Á.,

Corchado E. (eds) Computational Intelligence in Security for Information Systems. Lecture Notes in Computer Science, vol 6694. Springer, Berlin, Heidelberg, 2011

- Wroe, D. (2019). The greatest threat we face: Cyber security tsar quits with a warning. Retrieved from https://www.smh.com.au/politics/federal/the-greatest-threat-we-face-cyber-security-tsar-quits-with-a-warning-20190503-p51jvh.html
- Xiang, G., Hong, J., Rose, C. P., & Cranor, L. (2011). CANTINA+: A Feature-Rich Machine Learning Framework for Detecting Phishing Web Sites. ACM Trans. Inf. Syst. Secur. 14, 2, Article 21, 2011
- Xiao, L., Wan, X., Lu, X., Zhang, Y., & Wu, D. (2018).IoT Security Techniques Based on Machine Learning: How Do IoT Devices Use AI to Enhance Security? in IEEE Signal Processing Magazine, vol. 35, no. 5, pp. 41-49, Sept. 2018.
- Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., & Wang, C. (2018). Machine learning and deep learning methods for cybersecurity. IEEE Access, 6, pp.35365-35381.
- Yong, E. (2013). Mind-Bending Parasite Permanently Quells Cat Fear in Mice. National Geographic. Retrieved from https://www.nationalgeographic.com/science/phenomena/2013/04/26/mind-bending-parasite-permanently-quells-cat-fear-in-mice/
- Yoon, M., Mohan, S., Choi, J., Christodorescu, M., & Sha, L. (2017). Learning execution contexts from system call distribution for anomaly detection in smart embedded system. In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, pp. 191-196. ACM, 2017.
- Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: android malware characterization and detection using deep learning. Tsinghua Science and

Technology, vol. 21, no. 1, pp. 114-123, 2016

- Zalewski, M. (2019). American fuzzy lop. Retrieved from http://lcamtuf.coredump.cx/afl/
- Zerodium (2019). 20 March 2019, Retrieved from https://zerodium.com/
- Zhang L., & Veitch, D. (2011). Learning entropy. In Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part I (NETWORKING'11). Springer-Verlag, Berlin, Heidelberg, 15–27. 2011
- Zhuang, R., Bardas, G. A., DeLoach, S. A., & Ou, X. (2015). A Theory of cyber attacks: A Step Towards Analyzing MTD Systems. In Proceedings of the Second ACM Workshop on Moving Target Defense. ACM, New York, NY, USA, 11-20, 2015
- ZTM (2019). Zero Trust Model. Retrieved from http://csrc.nist.gov/cyberframework/rfi_comments/040813_forrester_research.pdf

Appendix A

Nagios XI Vulnerabilities:

- CVE-2018-8733

  "Authentication bypass vulnerability in the core config manager in Nagios XI 5.2.x through 5.4.x before 5.4.13 allows an unauthenticated attacker to make configuration changes and leverage an authenticated SQL injection vulnerability" (CVE-2018-8733, 2018, para 2).

- CVE-2018-8734

  "SQL injection vulnerability in the core config manager in Nagios XI 5.2.x

through 5.4.x before 5.4.13 allows an attacker to execute arbitrary SQL commands via the selInfoKey1 parameter (CVE-2018-8734, 2018, para 2).

- CVE-2018-8735

  Remote command execution (RCE) vulnerability in Nagios XI 5.2.x through 5.4.x before 5.4.13 allows an attacker to execute arbitrary commands on the target system, aka OS command injection. (CVE-2018-8735, 2018, para 2)

- CVE-2018-8736

  A privilege escalation vulnerability in Nagios XI 5.2.x through 5.4.x before 5.4.13 allows an attacker to leverage an RCE vulnerability escalating to root (CVE-2018-8736, 2018, para 2)

Nagios XI exploitation

Nagios XI 5.2.6-5.4.12

Nagios XI Chained Remote Code Execution

Author(s) Cale Smith , Benny Husted , Jared Arave

- Description

  “This module exploits a few different vulnerabilities in Nagios XI 5.2.6-5.4.12 to gain remote root access. The steps are: 1. Issue a POST request to /nagiosql/admin/settings.php which sets the database user to root. 2. SQLi on /nagiosql/admin/helpedit.php allows us to enumerate API keys. 3. The API keys are then used to add an administrative user. 4. An authenticated session is established with the newly added user 5. Command Injection on /nagiosxi/backend/index.php allows us to execute the payload with nopasswd

sudo, giving us a root shell. 6. Remove the added admin user and reset the database user" (Nagios-exploit, 2020, para 1).

Appendix B

While system calls table change, these are the names and ID used in the experiment.

TABLE XII. Ubuntu 18.04 System calls table

Syscall ID	Syscall name
0	read
1	write
2	open
3	close
4	stat
5	fstat
6	lstat
7	poll
8	lseek
9	mmap
10	mprotect
11	munmap
12	brk
13	rt_sigaction
14	rt_sigprocmask
15	rt_sigreturn
16	ioctl
17	pread
18	pwrite
19	readv
20	writev
21	access
22	pipe

23	select
24	sched_yield
25	mremap
26	msync
27	mincore
28	madvise
29	shmget
30	shmat
31	shmctl
32	dup
33	dup2
34	pause
35	nanosleep
36	getitimer
37	alarm
38	setitimer
39	getpid
40	sendfile
41	socket
42	connect
43	accept
44	sendto
45	recvfrom
46	sendmsg
47	recvmsg
48	shutdown
49	bind
50	listen
51	getsockname
52	getpeername
53	socketpair
54	setsockopt
55	getsockopt
56	clone
57	fork
58	vfork

59	execve
60	exit
61	wait4
62	kill
63	uname
64	semget
65	semop
66	semctl
67	shmdt
68	msgget
69	msgsnd
70	msgrcv
71	msgctl
72	fcntl
73	flock
74	fsync
75	fdatasync
76	truncate
77	ftruncate
78	getdents
79	getcwd
80	chdir
81	fchdir
82	rename
83	mkdir
84	rmdir
85	creat
86	link
87	unlink
88	symlink
89	readlink
90	chmod
91	fchmod
92	chown
93	fchown
94	lchown

95	umask
96	gettimeofday
97	getrlimit
98	getrusage
99	sysinfo
100	times
101	ptrace
102	getuid
103	syslog
104	getgid
105	setuid
106	setgid
107	geteuid
108	getegid
109	setpgid
110	getppid
111	getpgrp
112	setsid
113	setreuid
114	setregid
115	getgroups
116	setgroups
117	setresuid
118	getresuid
119	setresgid
120	getresgid
121	getpgid
122	setfsuid
123	setfsgid
124	getsid
125	capget
126	capset
127	rt_sigpending
128	rt_sigtimedwait
129	rt_sigqueueinfo
130	rt_sigsuspend

131	sigaltstack
132	utime
133	mknod
134	uselib
135	personality
136	ustat
137	statfs
138	fstatfs
139	sysfs
140	getpriority
141	setpriority
142	sched_setparam
143	sched_getparam
144	sched_setscheduler
145	sched_getscheduler
146	sched_get_priority_max
147	sched_get_priority_min
148	sched_rr_get_interval
149	mlock
150	munlock
151	mlockall
152	munlockall
153	vhangup
154	modify_ldt
155	pivot_root
156	_sysctl
157	prctl
158	arch_prctl
159	adjtimex
160	setrlimit
161	chroot
162	sync
163	acct
164	settimeofday
165	mount
166	umount2

167	swapon
168	swapoff
169	reboot
170	sethostname
171	setdomainname
172	iopl
173	ioperm
174	create_module
175	init_module
176	delete_module
177	get_kernel_syms
178	query_module
179	quotactl
180	nfsservctl
181	getpmsg
182	putpmsg
183	afs_syscall
184	tuxcall
185	security
186	gettid
187	readahead
188	setxattr
189	lsetxattr
190	fsetxattr
191	getxattr
192	lgetxattr
193	fgetxattr
194	listxattr
195	llistxattr
196	flistxattr
197	removexattr
198	lremovexattr
199	fremovexattr
200	tkill
201	time
202	futex

203	sched_setaffinity
204	sched_getaffinity
205	set_thread_area
206	io_setup
207	io_destroy
208	io_getevents
209	io_submit
210	io_cancel
211	get_thread_area
212	lookup_dcookie
213	epoll_create
214	epoll_ctl_old
215	epoll_wait_old
216	remap_file_pages
217	getdents64
218	set_tid_address
219	restart_syscall
220	semtimedop
221	fadvise64
222	timer_create
223	timer_settime
224	timer_gettime
225	timer_getoverrun
226	timer_delete
227	clock_settime
228	clock_gettime
229	clock_getres
230	clock_nanosleep
231	exit_group
232	epoll_wait
233	epoll_ctl
234	tgkill
235	utimes
236	vserver
237	mbind
238	set_mempolicy

239	get_mempolicy
240	mq_open
241	mq_unlink
242	mq_timedsend
243	mq_timedreceive
244	mq_notify
245	mq_getsetattr
246	kexec_load
247	waitid
248	add_key
249	request_key
250	keyctl
251	ioprio_set
252	ioprio_get
253	inotify_init
254	inotify_add_watch
255	inotify_rm_watch
256	migrate_pages
257	openat
258	mkdirat
259	mknodat
260	fchownat
261	futimesat
262	newfstatat
263	unlinkat
264	renameat
265	linkat
266	symlinkat
267	readlinkat
268	fchmodat
269	faccessat
270	pselect6
271	ppoll
272	unshare
273	set_robust_list
274	get_robust_list

275	splice
276	tee
277	sync_file_range
278	vmsplice
279	move_pages
280	utimensat
281	epoll_pwait
282	signalfd
283	timerfd
284	eventfd
285	fallocate
286	timerfd_settime
287	timerfd_gettime
288	accept4
289	signalfd4
290	eventfd2
291	epoll_create1
292	dup3
293	pipe2
294	inotify_init1
295	preadv
296	pwritev
297	rt_tgsigqueueinfo
298	perf_event_open
299	recvmmsg
300	fanotify_init
301	fanotify_mark
302	prlimit64
303	name_to_handle_at
304	open_by_handle_at
305	clock_adjtime
306	syncfs
307	sendmmsg
308	setns
309	getcpu
310	process_vm_readv

311	process_vm_writev
312	kcmp
313	finit_module
314	sched_setattr
315	sched_getattr
316	renameat2
317	seccomp
318	getrandom
319	memfd_create
320	kexec_file_load
321	bpf
322	execveat
323	userfaultfd
324	membarrier
325	mlock2
326	copy_file_range
327	preadv2
328	pwritev2
329	pkey_mprotect
330	pkey_alloc
331	pkey_free
332	statx

Appendix C

Table XX shows the detailed system calls samples from the experiment in chapter 5, comparison of Firefox 11 and Firefox 74 with Sorenson's Index.

TABLE XIII. Firefox V11.0 and V74.00 comparisons

System call ID	System call threat group	System call name	Number of individual system calls	
			Firefox 11	Firefox 74.0
0	T3	read	2	1035
1	T3	write	3	348
3	T3	close	7	33
4	T4	stat	2	299
5	T4	fstat	35	200
6	T4	lstat	6	6
7	T3	poll	5	17827
8	T3	lseek	14	4
9	T3	mmap	1	40
10	T3	mprotect	14	86
11	T3	munmap	1	39
12	T2	brk	1	1
13	T3	rt_sigaction	1	49
14	T3	rt_sigprocmask	4	6
15	T3	rt_sigreturn	72	1
16	T2	ioctl	2	63
17	T3	pread64	6	22
18	T3	pwrite64	1	1
20	T3	writev	308	5945
21	T4	access	23	2899
22	T4	pipe	3	6
23	T3	select	1	**0**

28	MS	madvise	1	159
29	MS	shmget	4	**0**
30	MS	shmat	4	**0**
31	MS	shmctl	4	**0**
32	T3	dup	**0**	33
33	T2	dup2	**0**	4
39	T4	getpid	9	256
41	DT2	socket	4	2
42	MS	connect	5	2
44	MS	sendto	2	1
45	MS	recvfrom	23	3
46	MS	sendmsg	33	4
47	MS	recvmsg	13	32852
48	MS	shutdown	3	1
49	MS	bind	1	**0**
51	MS	getsockname	4	1
52	MS	getpeername	8	1
53	MS	socketpair	1	10
54	MS	setsockopt	1	1
55	MS	getsockopt	1	2
56	DT2	clone	1	11
59	T1	execve	**0**	3
60	T3	exit	1	3
61	T3	wait4	1	4
62	T2	kill	2	2
63	T4	uname	2	5
67	MS	shmdt	2	2
72	T3	fcntl	49	21
74	T3	fsync	1	**0**
75	T4	fdatasync	3	**0**
77	T2	ftruncate	8	33
78	T4	getdents	2	6
79	T4	getcwd	5	3
82	T1	rename	1	**0**
83	T1	mkdir	6	1
84	T2	rmdir	**0**	1
87	MS	unlink	1	33
88	T1	symlink	1	2
89	MS	readlink	9	1
90	T1	chmodd	1	**0**
95	T3	umask	2	12

98	T4	getrusage	2	**0**
99	T4	sysinfo	1	7
102	MS	getuid	3	185
104	T4	getgid	2	30
107	T4	geteuid	5	43
108	T4	getegid	2	32
110	T4	getppid	**0**	1
117	T1	setresuid	38	**0**
118	T4	getresuid	3	3
119	T1	setresgid	38	**0**
120	T4	getresgid	3	3
131	MS	sigaltstack	1	18
132	T3	utime	1	**0**
137	T4	statfs	1	12
138	T4	fstatfs	1	166
140	T4	getpriority	**0**	9
141	T2	setpriority	**0**	9
143	T4	sched_getparam	1	**0**
144	T2	sched_setscheduler	1	1
145	T4	sched_getscheduler	1	**0**
146	T4	sched_get_priority_max	1	**0**
147	T4	sched_get_priority_min	1	**0**
157	T3	prctl	6	11
158	MS	arch_prctl	1	3
179	DT2	quotactl	**0**	3
186	MS	gettid	1	4
187	MS	readahead	12	13
202	MS	futex	5	2979
204	MS	sys_sched_getaffinity	1	2
213	MS	epoll_create	1	**0**
217	MS	getdents64	**0**	3
218	MS	set_tid_address	1	2
221	MS	fadvise64	67	68
229	T4	clock_getres	1	1
231	MS	exit_group	1	2
232	MS	epoll_wait	4	**0**

233	MS	epoll_ctl	1	**0**
234	MS	tgkill	84	**0**
254	MS	inotify_add_watch	1	8
257	MS	openat	3	33
269	MS	faccessat	**0**	1
273	MS	set_robust_list	1	11
285	MS	fallocate	1	**0**
290	MS	eventfd2	7	7
293	MS	pipe2	1	1
294	MS	inotify_init1	1	1
302	MS	prlimit64	4	9
307	MS	sendmmsg	1	**0**
317	MS	seccomp	**0**	2
318	MS	getrandom	118	1
319	MS	memfd_create	2	1
System calls richness			103	92
Total number of system calls			1163	66070
Common system calls			80	
	2x80/103+92	160/195	0.82	

Appendix D

Install instructions for the experimental environment on Ubuntu 18.04 LTS

- **Installation of Prometheus 2.1.0**

```
$ wget https://s3-eu-west-1.amazonaws.com/deb.robustperception.io/41EFC99D.gpg | $ $ sudo apt-key add -
$ apt-get update -y
$ apt-get install prometheus prometheus-node-exporter prometheus-pushgateway
$ prometheus-alertmanager
$ systemctl start prometheus
$ systemctl enable prometheus
$ systemctl status prometheus
```

- **Installation of Grafana 6.2.5**

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install -y software-properties-common
$ sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"
$ wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
```

```
$ sudo apt-get update

$ sudo apt-get install grafana

$ sudo apt-get install -y apt-transport-https

$ service grafana-server start
```

- **Configuring Grafana server to start at boot time:**

```
$ sudo update-rc.d grafana-server defaults

$ systemctl daemon-reload

$ systemctl start grafana-server

$ systemctl status grafana-server
```

- **Installation of Pushgateway 0.9.1**

```
$ wget https://github.com/prometheus/pushgateway/releases/download/v0.9.1/pushgateway-0.9.1.linux-amd64.tar.gz

$ tar xvzf pushgateway-0.9.1.linux-amd64.tar.gz

$ cd pushgateway-0.9.1.linux-amd64/

$  ./pushgateway &
```

- **Installation of bpftrace**

```
$ sudoesnap install --devmode bpftrace
```

Test bpftrace.

```
$ bpftrace -e 'tracepoint:syscalls:sys_enter_* { @[probe] = count(); }'
```

- **Installation of containerised Nagios server**

```
$ docker pull mavenquist/nagios-xi

$ docker run -d -p 80:80 --name nagios-xi mavenquist/nagios-xi
```

- **Installation of Curl**

```
$ sudo apt update && apt upgrade

$ sudo apt install curl

$ sudo curl --version
```

- **Installation of Awk**

```
$ sudo apt-get update

$ sudo apt-get install gawk
```

Appendix E

The following scripts were used in the experiments from chapter 4 and chapter 5. Sample of out also included , following the scripts.

- **aid_9.sh**

  Script aid_9 produces output of information, system calls used only by Firefox. The information was ingested from bftrace, at 10 second iterations. The output contains the system call ID, name, the number of times a unique system call was used by Firefox. The output also contains the threat marking groups, that the system call belongs to. The output is then piped to Promethius and Pushgateways to be presented in Grafana.

```
#!/bin/bash
sudo timeout -s SIGINT 10 bpftrace -e 'tracepoint:raw syscalls:sys enter {@[args->id, comm] = count();}
interval:s:10 { print(@); clear(@); }' | awk '/ firefox/{gsub(/\@|\[|\,|\]|\:/,"");\
 if ($1=='0') print "AID T3 sys read{label=\"T3 sys read 0\"}",$3;
 else if ($1=='1') print "AID T3 sys write{label=\"T3 sys write 1\"}",$3;
 else if ($1=='2') print "AID T1 sys open{label=\"T1 sys open 2\"}",$3;
 else if ($1=='3') print "AID T3 sys close{label=\"T3 sys close 3\"}",$3;
 else if ($1=='4') print "AID T4 sys stat{label=\"T4 sys stat 4\"}",$3;
 else if ($1=='5') print "AID T4 sys fstat{label=\"T4 sys fstat 5\"}",$3;
 else if ($1=='6') print "AID T4 sys lstat{label=\"T4 sys lstat 6\"}",$3;
```

```
else if ($1=='7') print "AID T3 sys poll{label=\"T3 sys poll 7\"}",$3;
else if ($1=='8') print "AID T3 sys lseek{label=\"T3 sys lseek 8\"}",$3;
else if ($1=='9') print "AID T3 sys mmap{label=\"T3 sys mmap 9\"}",$3;
else if ($1=='10') print "AID T3 sys mprotect{label=\"T3 sys mprotect 10\"}",$3;
else if ($1=='11') print "AID T3 sys munmap{label=\"T3 sys munmap 11\"}",$3;
else if ($1=='12') print "AID T2 sys brk{label=\"T2 sys brk 12\"}",$3;
else if ($1=='13') print "AID T3 sys rt sigaction{label=\"T3 sys rt sigaction 13\"}",$3;
else if ($1=='14') print "AID T3 sys rt sigprocmask{label=\"T3 sys rt sigprocmask 14\"}",$3;
else if ($1=='15') print "AID T3 sys rt sigreturn{label=\"T3 sys rt sigreturn 15\"}",$3;
else if ($1=='16') print "AID T2 sys ioctl{label=\"T2 sys ioctl 16\"}",$3;
else if ($1=='17') print "AID T3 sys pread64{label=\"T3 sys pread64 17\"}",$3;
else if ($1=='18') print "AID T3 sys pwrite64{label=\"T3 sys pwrite64 18\"}",$3;
else if ($1=='19') print "AID T3 sys readv{label=\"T3 sys readv 19\"}",$3;
else if ($1=='20') print "AID T3 sys writev{label=\"T3 sys writev 20\"}",$3;
else if ($1=='21') print "AID T4 sys access{label=\"T4 sys access 21\"}",$3;
else if ($1=='22') print "AID T4 sys pipe{label=\"T4 sys pipe 22\"}",$3;
else if ($1=='23') print "AID T3 sys select{label=\"T3 sys select 23\"}",$3;
else if ($1=='24') print "AID T3 sys sched yield{label=\"T3 sys sched yield 24\"}",$3;
else if ($1=='25') print "AID T3 sys mremap{label=\"T3 sys mremap 25\"}",$3;
else if ($1=='26') print "AID T3 sys msync{label=\"T3 sys msync 26\"}",$3;
else if ($1=='27') print "AID MS sys mincore{label=\"MS sys mincore 27\"}",$3;
else if ($1=='28') print "AID MS sys madvise{label=\"MS sys madvise 28\"}",$3;
else if ($1=='29') print "AID MS sys shmget{label=\"MS sys shmget 29\"}",$3;
else if ($1=='30') print "AID MS sys shmat{label=\"MS sys shmat 30\"}",$3;
else if ($1=='31') print "AID MS sys shmctl{label=\"MS sys shmctl 31\"}",$3;
else if ($1=='32') print "AID T3 sys dup{label=\"T3 sys dup 32\"}",$3;
else if ($1=='33') print "AID T2 sys dup2{label=\"T2 sys dup2 33\"}",$3;
else if ($1=='34') print "AID T3 sys pause{label=\"T3 sys pause 34\"}",$3;
else if ($1=='35') print "AID T3 sys nanosleep{label=\"T3 sys nanosleep 35\"}",$3;
else if ($1=='36') print "AID T4 sys getitimer{label=\"T4 sys getitimer 36\"}",$3;
else if ($1=='37') print "AID MS sys alarm{label=\"MS sys alarm 37\"}",$3;
else if ($1=='38') print "AID T3 sys setitimer{label=\"T3 sys setitimer 38\"}",$3;
else if ($1=='39') print "AID T4 sys getpid{label=\"T4 sys getpid 39\"}",$3;
else if ($1=='40') print "AID T3 sys sendfile{label=\"T3 sys sendfile 40\"}",$3;
```

```
else if ($1=='41') print "AID D T2 sys socket{label=\"D T2 sys socket 41\"}",$3;
else if ($1=='42') print "AID MS sys connect{label=\"MS sys connect 42\"}",$3;
else if ($1=='43') print "AID MS sys accept{label=\"MS sys accept 43\"}",$3;
else if ($1=='44') print "AID MS sys sendto{label=\"MS sys sendto 44\"}",$3;
else if ($1=='45') print "AID MS sys recvfrom{label=\"MS sys recvfrom 45\"}",$3;
else if ($1=='46') print "AID MS sys sendmsg{label=\"MS sys sendmsg 46\"}",$3;
else if ($1=='47') print "AID MS sys recvmsg{label=\"MS sys recvmsg 47\"}",$3;
else if ($1=='48') print "AID MS sys shutdown{label=\"MS sys shutdown 48\"}",$3;
else if ($1=='49') print "AID MS sys bind{label=\"MS sys bind 49\"}",$3;
else if ($1=='50') print "AID MS sys listen{label=\"MS sys listen 50\"}",$3;
else if ($1=='51') print "AID MS sys getsockname{label=\"MS sys getsockname 51\"}",$3;
else if ($1=='52') print "AID MS sys getpeername{label=\"MS sys getpeername 52\"}",$3;
else if ($1=='53') print "AID MS sys socketpair{label=\"MS sys socketpair 53\"}",$3;
else if ($1=='54') print "AID MS sys setsockopt{label=\"MS sys setsockopt 54\"}",$3;
else if ($1=='55') print "AID MS sys getsockopt{label=\"MS sys getsockopt 55\"}",$3;
else if ($1=='56') print "AID D T2 sys clone{label=\"D T2 sys clone 56\"}",$3;
else if ($1=='57') print "AID T2 sys fork{label=\"T2 sys fork 57\"}",$3;
else if ($1=='58') print "AID T2 sys vfork{label=\"T2 sys vfork 58\"}",$3;
else if ($1=='59') print "AID T1 sys execve{label=\"T1 sys execve 59\"}",$3;
else if ($1=='60') print "AID T3 sys exit{label=\"T3 sys exit 60\"}",$3;
else if ($1=='61') print "AID T3 sys wait4{label=\"T3 sys wait4 61\"}",$3;
else if ($1=='62') print "AID T2 sys kill{label=\"T2 sys kill 62\"}",$3;
else if ($1=='63') print "AID T4 sys uname{label=\"T4 sys uname 63\"}",$3;
else if ($1=='64') print "AID MS sys semget{label=\"MS sys semget 64\"}",$3;
else if ($1=='65') print "AID MS sys semop{label=\"MS sys semop 65\"}",$3;
else if ($1=='66') print "AID MS sys semctl{label=\"MS sys semctl 66\"}",$3;
else if ($1=='67') print "AID MS sys shmdt{label=\"MS sys shmdt 67\"}",$3;
else if ($1=='68') print "AID MS sys msgget{label=\"MS sys msgget 68\"}",$3;
else if ($1=='69') print "AID MS sys msgsnd{label=\"MS sys msgsnd 69\"}",$3;
else if ($1=='70') print "AID MS sys msgrcv{label=\"MS sys msgrcv 70\"}",$3;
else if ($1=='71') print "AID MS sys msgctl{label=\"MS sys msgctl 71\"}",$3;
else if ($1=='72') print "AID T3 sys fcntl{label=\"T3 sys fcntl 72\"}",$3;
else if ($1=='73') print "AID T2 sys flock{label=\"T2 sys flock 73\"}",$3;
else if ($1=='74') print "AID T3 sys fsync{label=\"T3 sys fsync 74\"}",$3;
```

```
else if ($1=='75') print "AID T4 sys fdatasync{label=\"T4 sys fdatasync 75\"}",$3;
else if ($1=='76') print "AID T2 sys truncate{label=\"T2 sys truncate 76\"}",$3;
else if ($1=='77') print "AID T2 sys ftruncate{label=\"T2 sys ftruncate 77\"}",$3;
else if ($1=='78') print "AID T4 sys getdents{label=\"T4 sys getdents 78\"}",$3;
else if ($1=='79') print "AID T4 sys getcwd{label=\"T4 sys getcwd 79\"}",$3;
else if ($1=='80') print "AID T3 sys chdir{label=\"T3 sys chdir 80\"}",$3;
else if ($1=='81') print "AID T3 sys fchdir{label=\"T3 sys fchdir 81\"}",$3;
else if ($1=='82') print "AID T1 sys rename{label=\"T1 sys rename 82\"}",$3;
else if ($1=='83') print "AID T1 sys mkdir{label=\"T1 sys mkdir 83\"}",$3;
else if ($1=='84') print "AID T2 sys rmdir{label=\"T2 sys rmdir 84\"}",$3;
else if ($1=='85') print "AID T2 sys creat{label=\"T2 sys creat 85\"}",$3;
else if ($1=='86') print "AID T1 sys link{label=\"T1 sys link 86\"}",$3;
else if ($1=='87') print "AID MS sys unlink{label=\"MS sys unlink 87\"}",$3;
else if ($1=='88') print "AID T1 sys symlink{label=\"T1 sys symlink 88\"}",$3;
else if ($1=='89') print "AID MS sys readlink{label=\"MS sys readlink 89\"}",$3;
else if ($1=='90') print "AID T1 sys chmod{label=\"T1 sys chmod 90\"}",$3;
else if ($1=='91') print "AID T1 sys fchmod{label=\"T1 sys fchmod 91\"}",$3;
else if ($1=='92') print "AID T1 sys chown{label=\"T1 sys chown 92\"}",$3;
else if ($1=='93') print "AID T1 sys fchown{label=\"T1 sys fchown 93\"}",$3;
else if ($1=='94') print "AID T1 sys lchown{label=\"T1 sys lchown 94\"}",$3;
else if ($1=='95') print "AID T3 sys umask{label=\"T3 sys umask 95\"}",$3;
else if ($1=='96') print "AID T4 sys gettimeofday{label=\"T4 sys gettimeofday 96\"}",$3;
else if ($1=='97') print "AID T4 sys getrlimit{label=\"T4 sys getrlimit 97\"}",$3;
else if ($1=='98') print "AID T4 sys getrusage{label=\"T4 sys getrusage 98\"}",$3;
else if ($1=='99') print "AID T4 sys sysinfo{label=\"T4 sys sysinfo 99\"}",$3;
else if ($1=='100') print "AID T4 sys times{label=\"T4 sys times 100\"}",$3;
else if ($1=='101') print "AID D T3 sys ptrace{label=\"D T3 sys ptrace 101\"}",$3;
else if ($1=='102') print "AID MS sys getuid{label=\"MS sys getuid 102\"}",$3;
else if ($1=='103') print "AID T2 sys syslog{label=\"T2 sys syslog 103\"}",$3;
else if ($1=='104') print "AID T4 sys getgid{label=\"T4 sys getgid 104\"}",$3;
else if ($1=='105') print "AID T1 sys setuid{label=\"T1 sys setuid 105\"}",$3;
else if ($1=='106') print "AID T1 sys setgid{label=\"T1 sys setgid 106\"}",$3;
else if ($1=='107') print "AID T4 sys geteuid{label=\"T4 sys geteuid 107\"}",$3;
else if ($1=='108') print "AID T4 sys getegid{label=\"T4 sys getegid 108\"}",$3;
```

```
else if ($1=='109') print "AID T3 sys setpgid{label=\"T3 sys setpgid 109\"}",$3;
else if ($1=='110') print "AID T4 sys getppid{label=\"T4 sys getppid 110\"}",$3;
else if ($1=='111') print "AID T4 sys getpgrp{label=\"T4 sys getpgrp 111\"}",$3;
else if ($1=='112') print "AID T3 sys setsid{label=\"T3 sys setsid 112\"}",$3;
else if ($1=='113') print "AID T1 sys setreuid{label=\"T1 sys setreuid 113\"}",$3;
else if ($1=='114') print "AID T1 sys setregid{label=\"T1 sys setregid 114\"}",$3;
else if ($1=='115') print "AID T4 sys getgroups{label=\"T4 sys getgroups 115\"}",$3;
else if ($1=='116') print "AID T1 sys setgroups{label=\"T1 sys setgroups 116\"}",$3;
else if ($1=='117') print "AID T1 sys setresuid{label=\"T1 sys setresuid 117\"}",$3;
else if ($1=='118') print "AID T4 sys getresuid{label=\"T4 sys getresuid 118\"}",$3;
else if ($1=='119') print "AID T1 sys setresgid{label=\"T1 sys setresgid 119\"}",$3;
else if ($1=='120') print "AID T4 sys getresgid{label=\"T4 sys getresgid 120\"}",$3;
else if ($1=='121') print "AID T4 sys getpgid{label=\"T4 sys getpgid 121\"}",$3;
else if ($1=='122') print "AID T1 sys setfsuid{label=\"T1 sys setfsuid 122\"}",$3;
else if ($1=='123') print "AID T1 sys setfsgid{label=\"T1 sys setfsgid 123\"}",$3;
else if ($1=='124') print "AID T4 sys getsid{label=\"T4 sys getsid 124\"}",$3;
else if ($1=='125') print "AID T4 sys capget{label=\"T4 sys capget 125\"}",$3;
else if ($1=='126') print "AID T3 sys capset{label=\"T3 sys capset 126\"}",$3;
else if ($1=='127') print "AID T3 sys rt sigpending{label=\"T3 sys rt sigpending 127\"}",$3;
else if ($1=='128') print "AID T3 sys rt sigtimedwait{label=\"T3 sys rt sigtimedwait 128\"}",$3;
else if ($1=='129') print "AID T3 sys rt sigqueueinfo{label=\"T3 sys rt sigqueueinfo 129\"}",$3;
else if ($1=='130') print "AID T3 sys rt sigsuspend{label=\"T3 sys rt sigsuspend 130\"}",$3;
else if ($1=='131') print "AID MS sys sigaltstack{label=\"MS sys sigaltstack 131\"}",$3;
else if ($1=='132') print "AID T3 sys utime{label=\"T3 sys utime 132\"}",$3;
else if ($1=='133') print "AID T2 sys mknod{label=\"T2 sys mknod 133\"}",$3;
else if ($1=='134') print "AID D T3 sys uselib{label=\"D T3 sys uselib 134\"}",$3;
else if ($1=='135') print "AID D T3 sys personality{label=\"D T3 sys personality 135\"}",$3;
else if ($1=='136') print "AID D T4 sys ustat{label=\"D T4 sys ustat 136\"}",$3;
else if ($1=='137') print "AID T4 sys statfs{label=\"T4 sys statfs 137\"}",$3;
else if ($1=='138') print "AID T4 sys fstatfs{label=\"T4 sys fstatfs 138\"}",$3;
else if ($1=='139') print "AID D sys sysfs{label=\"D sys sysfs 139\"}",$3;
else if ($1=='140') print "AID T4 sys getpriority{label=\"T4 sys getpriority 140\"}",$3;
else if ($1=='141') print "AID T2 sys setpriority{label=\"T2 sys setpriority 141\"}",$3;
else if ($1=='142') print "AID T2 sys sched setparam{label=\"T2 sys sched setparam 142\"}",$3;
```

```
else if ($1=='143') print "AID T4 sys sched getparam{label=\"T4 sys sched getparam 143\"}",$3;
else if ($1=='144') print "AID T2 sys sched setscheduler{label=\"T2 sys sched setscheduler 144\"}",$3;
else if ($1=='145') print "AID T4 sys sched getscheduler{label=\"T4 sys sched getscheduler 145\"}",$3;
else if ($1=='146') print "AID T4 sys sched get priority max{label=\"T4 sys sched get priority max 146\"}",$3;
else if ($1=='147') print "AID T4 sys sched get priority min{label=\"T4 sys sched get priority min 147\"}",$3;
else if ($1=='148') print "AID T4 sys sched rr get interval{label=\"T4 sys sched rr get interval 148\"}",$3;
else if ($1=='149') print "AID T2 sys mlock{label=\"T2 sys mlock 149\"}",$3;
else if ($1=='150') print "AID T3 sys munlock{label=\"T3 sys munlock 150\"}",$3;
else if ($1=='151') print "AID T2 sys mlockall{label=\"T2 sys mlockall 151\"}",$3;
else if ($1=='152') print "AID T3 sys munlockall{label=\"T3 sys munlockall 152\"}",$3;
else if ($1=='153') print "AID T2 sys vhangup{label=\"T2 sys vhangup 153\"}",$3;
else if ($1=='154') print "AID T2 sys modify ldt{label=\"T2 sys modify ldt 154\"}",$3;
else if ($1=='155') print "AID D sys pivot root{label=\"D sys pivot root 155\"}",$3;
else if ($1=='156') print "AID MS sys  sysctl{label=\"D T2 sys  sysctl 156\"}",$3;
else if ($1=='157') print "AID T3 sys prctl{label=\"T3 sys prctl 157\"}",$3;
else if ($1=='158') print "AID MS sys arch prctl{label=\"MS sys arch prctl 158\"}",$3;
else if ($1=='159') print "AID T2 sys adjtimex{label=\"T2 sys adjtimex 159\"}",$3;
else if ($1=='160') print "AID T2 sys setrlimit{label=\"T2 sys setrlimit 160\"}",$3;
else if ($1=='161') print "AID T3 sys chroot{label=\"T3 sys chroot 161\"}",$3;
else if ($1=='162') print "AID T4 sys sync{label=\"T4 sys sync 162\"}",$3;
else if ($1=='163') print "AID D T4 sys acct{label=\"D T4 sys acct 163\"}",$3;
else if ($1=='164') print "AID D T2 sys settimeofday{label=\"D T2 sys settimeofday 164\"}",$3;
else if ($1=='165') print "AID D T1 sys mount{label=\"D T1 sys mount 165\"}",$3;
else if ($1=='166') print "AID D T2 sys umount2 {label=\"D T2 sys umount2 166\"}",$3;
else if ($1=='167') print "AID D T2 sys swapon{label=\"D T2 sys swapon 167\"}",$3;
else if ($1=='168') print "AID D T2 sys swapoff{label=\"D T2 sys swapoff 168\"}",$3;
else if ($1=='169') print "AID D T2 sys reboot{label=\"D T2 sys reboot 169\"}",$3;
else if ($1=='170') print "AID T2 sys sethostname{label=\"T2 sys sethostname 170\"}",$3;
else if ($1=='171') print "AID T2 sys setdomainname{label=\"T2 sys setdomainname 171\"}",$3;
else if ($1=='172') print "AID D T2 sys iopl{label=\"D T2 sys iopl 172\"}",$3;
else if ($1=='173') print "AID D T2 sys ioperm{label=\"D T2 sys ioperm 173\"}",$3;
else if ($1=='174') print "AID D T1 sys create module{label=\"D T1 sys create module 174\"}",$3;
else if ($1=='175') print "AID D T4 sys init module{label=\"D T4 sys init module 175\"}",$3;
else if ($1=='176') print "AID D T2 sys delete module{label=\"D T2 sys delete module 176\"}",$3;
```

```
else if ($1=='177') print "AID D sys get kernel syms{label=\"D sys get kernel syms 177\"}",$3;
else if ($1=='178') print "AID D T4 sys query module{label=\"D T4 sys query module 178\"}",$3;
else if ($1=='179') print "AID D T2 sys quotactl{label=\"D T2 sys quotactl 179\"}",$3;
else if ($1=='180') print "AID D T2 sys nfsservctl{label=\"D T2 sys nfsservctl 180\"}",$3;
else if ($1=='181') print "AID T4 sys getpmsg{label=\"T4 sys getpmsg 181\"}",$3;
else if ($1=='182') print "AID T3 sys putpmsg{label=\"T3 sys putpmsg 182\"}",$3;
else if ($1=='183') print "AID T2 sys afs syscall{label=\"T2 sys afs syscall 183\"}",$3;
else if ($1=='184') print "AID MS sys tuxcall{label=\"MS sys tuxcall 184\"}",$3;
else if ($1=='185') print "AID MS sys security{label=\"MS sys security 185\"}",$3;
else if ($1=='186') print "AID MS sys gettid{label=\"MS sys gettid 186\"}",$3;
else if ($1=='187') print "AID MS sys readahead{label=\"MS sys readahead 187\"}",$3;
else if ($1=='188') print "AID MS sys setxattr{label=\"MS sys setxattr 188\"}",$3;
else if ($1=='189') print "AID MS sys lsetxattr{label=\"MS sys lsetxattr 189\"}",$3;
else if ($1=='190') print "AID MS sys fsetxattr{label=\"MS sys fsetxattr 190\"}",$3;
else if ($1=='191') print "AID MS sys getxattr{label=\"MS sys getxattr 191\"}",$3;
else if ($1=='192') print "AID MS sys lgetxattr{label=\"MS sys lgetxattr 192\"}",$3;
else if ($1=='193') print "AID MS sys fgetxattr{label=\"MS sys fgetxattr 193\"}",$3;
else if ($1=='194') print "AID MS sys listxattr{label=\"MS sys listxattr 194\"}",$3;
else if ($1=='195') print "AID MS sys llistxattr{label=\"MS sys llistxattr 195\"}",$3;
else if ($1=='196') print "AID MS sys flistxattr{label=\"MS sys flistxattr 196\"}",$3;
else if ($1=='197') print "AID MS sys removexattr{label=\"MS sys removexattr 197\"}",$3;
else if ($1=='198') print "AID MS sys lremovexattr{label=\"MS sys lremovexattr 198\"}",$3;
else if ($1=='199') print "AID MS sys fremovexattr{label=\"MS sys fremovexattr 199\"}",$3;
else if ($1=='200') print "AID MS sys tkill{label=\"MS sys tkill 200\"}",$3;
else if ($1=='201') print "AID T2 sys time{label=\"T2 sys time 201\"}",$3;
else if ($1=='202') print "AID MS sys futex{label=\"MS sys futex 202\"}",$3;
else if ($1=='203') print "AID MS sys sched setaffinity{label=\"MS sys sched setaffinity 203\"}",$3;
else if ($1=='204') print "AID MS sys sys sched getaffinity{label=\"MS sys sched getaffinity 204\"}",$3;
else if ($1=='205') print "AID MS sys set thread area{label=\"MS sys set thread area 205\"}",$3;
else if ($1=='206') print "AID MS sys io setup{label=\"MS sys io setup 206\"}",$3;
else if ($1=='207') print "AID MS sys io destroy{label=\"MS sys io destroy 207\"}",$3;
else if ($1=='208') print "AID MS sys io getevents{label=\"MS sys io getevents 208\"}",$3;
else if ($1=='209') print "AID MS sys io submit{label=\"MS sys io submit 209\"}",$3;
else if ($1=='210') print "AID MS sys io cancel{label=\"MS sys io cancel 210\"}",$3;
```

```
else if ($1=='211') print "AID MS sys get thread area{label=\"MS sys get thread area 211\"}",$3;
else if ($1=='212') print "AID MS sys lookup dcookie{label=\"MS sys lookup dcookie 212\"}",$3;
else if ($1=='213') print "AID MS sys epoll create{label=\"MS sys epoll create 213\"}",$3;
else if ($1=='214') print "AID MS sys epoll ctl old{label=\"MS sys epoll ctl old 214\"}",$3;
else if ($1=='215') print "AID MS sys epoll wait old{label=\"MS sys epoll wait old 215\"}",$3;
else if ($1=='216') print "AID MS sys remap file pages{label=\"MS sys remap file pages 216\"}",$3;
else if ($1=='217') print "AID MS sys getdents64{label=\"MS sys getdents64 217\"}",$3;
else if ($1=='218') print "AID MS sys set tid address{label=\"MS sys set tid address 218\"}",$3;
else if ($1=='219') print "AID MS sys restart syscall{label=\"MS sys restart syscall 219\"}",$3;
else if ($1=='220') print "AID MS sys semtimedop{label=\"MS sys semtimedop 220\"}",$3;
else if ($1=='221') print "AID MS sys fadvise64{label=\"MS sys fadvise64 221\"}",$3;
else if ($1=='222') print "AID MS sys timer create{label=\"MS sys timer create 222\"}",$3;
else if ($1=='223') print "AID MS sys timer settime{label=\"MS sys timer settime 223\"}",$3;
else if ($1=='224') print "AID MS sys timer gettime{label=\"MS sys timer gettime 224\"}",$3;
else if ($1=='225') print "AID MS sys timer getoverrun{label=\"MS sys timer getoverrun 225\"}",$3;
else if ($1=='226') print "AID MS sys timer delete{label=\"MS sys timer delete 226\"}",$3;
else if ($1=='227') print "AID D sys lock settime{label=\"D sys clock settime 227\"}",$3;
else if ($1=='228') print "AID MS sys clock gettime{label=\"MS sys clock gettime 228\"}",$3;
else if ($1=='229') print "AID T4 sys clock getres{label=\"T4 sys clock getres 229\"}",$3;
else if ($1=='230') print "AID MS sys clock nanosleep{label=\"MS sys clock nanosleep 230\"}",$3;
else if ($1=='231') print "AID MS sys exit group{label=\"MS sys exit group 231\"}",$3;
else if ($1=='232') print "AID MS sys epoll wait{label=\"MS sys epoll wait 232\"}",$3;
else if ($1=='233') print "AID MS sys epoll ctl{label=\"MS sys epoll ctl 233\"}",$3;
else if ($1=='234') print "AID MS sys tgkill{label=\"MS sys tgkill 234\"}",$3;
else if ($1=='235') print "AID MS sys utimes{label=\"MS sys utimes 235\"}",$3;
else if ($1=='236') print "AID MS sys vserver{label=\"MS sys vserver 236\"}",$3;
else if ($1=='237') print "AID D sys mbind{label=\"D sys mbind 237\"}",$3;
else if ($1=='238') print "AID D sys set mempolicy{label=\"D sys set mempolicy 238\"}",$3;
else if ($1=='239') print "AID D sys get mempolicy{label=\"D sys get mempolicy 239\"}",$3;
else if ($1=='240') print "AID MS sys mq open{label=\"MS sys mq open 240\"}",$3;
else if ($1=='241') print "AID T1 sys mq unlink{label=\"T1 sys mq unlink 241\"}",$3;
else if ($1=='242') print "AID MS sys mq timedsend{label=\"MS sys mq timedsend 242\"}",$3;
else if ($1=='243') print "AID MS sys mq timedreceive{label=\"MS sys mq timedreceive 243\"}",$3;
else if ($1=='244') print "AID MS sys mq notify{label=\"MS sys mq notify 244\"}",$3;
```

```
else if ($1=='245') print "AID MS sys mq getsetattr{label=\"MS sys mq getsetattr 245\"}",$3;
else if ($1=='246') print "AID MS sys kexec load{label=\"MS sys kexec load 246\"}",$3;
else if ($1=='247') print "AID MS sys waitid{label=\"MS sys waitid 247\"}",$3;
else if ($1=='248') print "AID D sys add key{label=\"D sys add key 248\"}",$3;
else if ($1=='249') print "AID D sys request key{label=\"D sys request key 249\"}",$3;
else if ($1=='250') print "AID MS sys keyctl{label=\"MS sys keyctl 250\"}",$3;
else if ($1=='251') print "AID MS sys ioprio set{label=\"MS sys ioprio set 251\"}",$3;
else if ($1=='252') print "AID MS sys ioprio get{label=\"MS sys ioprio get 252\"}",$3;
else if ($1=='253') print "AID MS sys inotify init{label=\"MS sys inotify init 253\"}",$3;
else if ($1=='254') print "AID MS sys inotify add watch{label=\"MS sys inotify add watch 254\"}",$3;
else if ($1=='255') print "AID MS sys inotify rm watch{label=\"MS sys inotify rm watch 255\"}",$3;
else if ($1=='256') print "AID MS sys migrate pages{label=\"MS sys migrate pages 256\"}",$3;
else if ($1=='257') print "AID MS sys openat{label=\"MS sys openat 257\"}",$3;
else if ($1=='258') print "AID MS sys mkdirat{label=\"MS sys mkdirat 258\"}",$3;
else if ($1=='259') print "AID MS sys mknodat{label=\"MS sys mknodat 259\"}",$3;
else if ($1=='260') print "AID MS sys fchownat{label=\"MS sys fchownat 260\"}",$3;
else if ($1=='261') print "AID MS sys futimesat{label=\"MS sys futimesat 261\"}",$3;
else if ($1=='262') print "AID MS sys newfstatat{label=\"MS sys newfstatat 262\"}",$3;
else if ($1=='263') print "AID MS sys unlinkat{label=\"MS sys unlinkat 263\"}",$3;
else if ($1=='264') print "AID MS sys renameat{label=\"MS sys renameat 264\"}",$3;
else if ($1=='265') print "AID MS sys linkat{label=\"MS sys linkat 265\"}",$3;
else if ($1=='266') print "AID MS sys symlinkat{label=\"MS sys symlinkat 266\"}",$3;
else if ($1=='267') print "AID T4 sys readlinkat{label=\"T4 sys readlinkat 267\"}",$3;
else if ($1=='268') print "AID MS sys fchmodat{label=\"MS sys fchmodat 268\"}",$3;
else if ($1=='269') print "AID MS sys faccessat{label=\"MS sys faccessat 269\"}",$3;
else if ($1=='270') print "AID MS sys pselect6{label=\"MS sys pselect6 270\"}",$3;
else if ($1=='271') print "AID MS sys ppoll{label=\"MS sys ppoll 271\"}",$3;
else if ($1=='272') print "AID D sys unshare{label=\"D sys unshare 272\"}",$3;
else if ($1=='273') print "AID MS sys set robust list{label=\"MS sys set robust list 273\"}",$3;
else if ($1=='274') print "AID MS sys get robust list{label=\"MS sys get robust list 274\"}",$3;
else if ($1=='275') print "AID MS sys splice{label=\"MS sys splice 275\"}",$3;
else if ($1=='276') print "AID MS sys tee{label=\"MS sys tee 276\"}",$3;
else if ($1=='277') print "AID MS sys sync file range{label=\"MS sys sync file range 277\"}",$3;
else if ($1=='278') print "AID MS sys vmsplice{label=\"MS sys vmsplice 278\"}",$3;
```

```
else if ($1=='279') print "AID D sys move pages{label=\"D sys move pages 279\"}",$3;
else if ($1=='280') print "AID MS sys utimensat{label=\"MS sys utimensat 280\"}",$3;
else if ($1=='281') print "AID MS sys epoll pwait{label=\"MS sys epoll pwait 281\"}",$3;
else if ($1=='282') print "AID T3 sys signalfd{label=\"T3 sys signalfd 282\"}",$3;
else if ($1=='283') print "AID MS sys timerfd create{label=\"MS sys timerfd create 283\"}",$3;
else if ($1=='284') print "AID MS sys eventfd{label=\"MS sys eventfd 284\"}",$3;
else if ($1=='285') print "AID MS sys fallocate{label=\"MS sys fallocate 285\"}",$3;
else if ($1=='286') print "AID MS sys timerfd settime{label=\"MS sys timerfd settime 286\"}",$3;
else if ($1=='287') print "AID MS sys timerfd gettime{label=\"MS sys timerfd gettime 287\"}",$3;
else if ($1=='288') print "AID MS sys accept4{label=\"MS sys accept4 288\"}",$3;
else if ($1=='289') print "AID MS sys signalfd4{label=\"MS sys signalfd4 289\"}",$3;
else if ($1=='290') print "AID MS sys eventfd2{label=\"MS sys eventfd2 290\"}",$3;
else if ($1=='291') print "AID MS sys epoll create1{label=\"MS sys epoll create1 291\"}",$3;
else if ($1=='292') print "AID MS sys dup3{label=\"MS sys dup3 292\"}",$3;
else if ($1=='293') print "AID MS sys pipe2{label=\"MS sys pipe2 293\"}",$3;
else if ($1=='294') print "AID MS sys inotify init1{label=\"MS sys inotify init1 294\"}",$3;
else if ($1=='295') print "AID T3 sys preadv{label=\"T3 sys preadv 295\"}",$3;
else if ($1=='296') print "AID T3 sys pwritev{label=\"T3 sys pwritev 296\"}",$3;
else if ($1=='297') print "AID MS sys rt tgsigqueueinfo{label=\"MS sys rt tgsigqueueinfo 297\"}",$3;
else if ($1=='298') print "AID D sys perf event open{label=\"D sys perf event open 298\"}",$3;
else if ($1=='299') print "AID MS sys recvmmsg{label=\"MS sys recvmmsg 299\"}",$3;
else if ($1=='300') print "AID MS sys fanotify init{label=\"MS sys fanotify init 300\"}",$3;
else if ($1=='301') print "AID MS sys fanotify mark{label=\"MS sys fanotify mark 301\"}",$3;
else if ($1=='302') print "AID MS sys prlimit64{label=\"MS sys prlimit64 302\"}",$3;
else if ($1=='303') print "AID D sys name to handle at{label=\"D sys name to handle at 303\"}",$3;
else if ($1=='304') print "AID D sys sys open by handle at{label=\"D sys open by handle at 304\"}",$3;
else if ($1=='305') print "AID D sys sys clock adjtime{label=\"D sys clock adjtime 305\"}",$3;
else if ($1=='306') print "AID MS sys syncfs{label=\"MS sys syncfs 306\"}",$3;
else if ($1=='307') print "AID MS sys sendmmsg{label=\"MS sys sendmmsg 307\"}",$3;
else if ($1=='308') print "AID D sys setns{label=\"D sys setns 308\"}",$3;
else if ($1=='309') print "AID MS sys getcpu{label=\"MS sys getcpu 309\"}",$3;
else if ($1=='310') print "AID D sys process vm readv{label=\"D sys process vm readv 310\"}",$3;
else if ($1=='311') print "AID D sys process vm writev{label=\"D sys process vm writev 311\"}",$3;
else if ($1=='312') print "AID D sys kcmp{label=\"D sys kcmp 312\"}",$3;
```

```
else if ($1=='313') print "AID D sys finit module{label=\"D sys finit module 313\"}",$3;
else if ($1=='314') print "AID MS sys sched setattr{label=\"MS sys sched setattr 314\"}",$3;
else if ($1=='315') print "AID MS sys sched getattr{label=\"MS sys sched getattr 315\"}",$3;
else if ($1=='316') print "AID MS sys renameat2{label=\"MS sys renameat2 316\"}",$3;
else if ($1=='317') print "AID MS sys seccomp{label=\"MS sys seccomp 317\"}",$3;
else if ($1=='318') print "AID MS sys getrandom{label=\"MS sys getrandom 318\"}",$3;
else if ($1=='319') print "AID MS sys memfd create{label=\"MS sys memfd create 319\"}",$3;
else if ($1=='320') print "AID D sys kexec file load{label=\"D sys kexec file load 320\"}",$3;
else if ($1=='321') print "AID D sys bpf{label=\"D sys bpf 321\"}",$3;
else if ($1=='322') print "AID MS sys execveat{label=\"MS sys execveat 322\"}",$3;
else if ($1=='323') print "AID D sys userfaultfd{label=\"D sys userfaultfd 323\"}",$3;
else if ($1=='324') print "AID MS sys membarrier{label=\"MS sys membarrier 324\"}",$3;
else if ($1=='325') print "AID MS sys mlock2{label=\"MS sys mlock2 325\"}",$3;
else if ($1=='326') print "AID MS sys copy file range{label=\"MS sys copy file range 326\"}",$3;
else if ($1=='327') print "AID T3 sys preadv2{label=\"T3 sys preadv2 327\"}",$3;
else if ($1=='328') print "AID T3 sys pwritev2{label=\"T3 sys pwritev2 328\"}",$3;
else if ($1=='329') print "AID MS sys pkey mprotect{label=\"MS sys pkey mprotect 329\"}",$3;
else if ($1=='330') print "AID MS sys pkey alloc{label=\"MS sys pkey alloc 330\"}",$3;
else if ($1=='331') print "AID MS sys pkey free{label=\"MS sys pkey free 331\"}",$3;
else if ($1=='332') print "AID MS sys statx{label=\"MS sys statx 332\"}",$3;
else print $1,$3;}' | curl --data-binary @- http://localhost:9091/metrics/job/ADI job/instance/ADI F11 instance
exit
```

Sample output from script aid 9.sh

Firefox - normal idle

```
local@localhost:~$ while true; do ./aid 9.sh; sleep 1; done
```

```
AID T3 sys exit{label="T3 sys exit 60"} 1
AID MS sys madvise{label="MS sys madvise 28"} 1
AID MS sys futex{label="MS sys futex 202"} 1
AID T3 sys poll{label="T3 sys poll 7"} 3
AID MS sys recvmsg{label="MS sys recvmsg 47"} 8
AID T3 sys read{label="T3 sys read 0"} 51
AID T3 sys write{label="T3 sys write 1"} 75
AID T3 sys writev{label="T3 sys writev 20"} 341
AID MS sys futex{label="MS sys futex 202"} 372
AID T3 sys poll{label="T3 sys poll 7"} 906
AID MS sys recvmsg{label="MS sys recvmsg 47"} 2211
AID T3 sys writev{label="T3 sys writev 20"} 18
AID T3 sys read{label="T3 sys read 0"} 22
AID T3 sys write{label="T3 sys write 1"} 33
AID MS sys futex{label="MS sys futex 202"} 44
AID T3 sys poll{label="T3 sys poll 7"} 84
AID MS sys recvmsg{label="MS sys recvmsg 47"} 233
AID T4 sys getpid{label="T4 sys getpid 39"} 1
AID T1 sys rename{label="T1 sys rename 82"} 1
AID MS sys sys sched getaffinity{label="MS sys sched getaffinity 204"} 1
AID T3 sys fsync{label="T3 sys fsync 74"} 1
AID MS sys readlink{label="MS sys readlink 89"} 1
AID T4 sys statfs{label="T4 sys statfs 137"} 1
AID T3 sys mprotect{label="T3 sys mprotect 10"} 2
AID T4 sys getdents{label="T4 sys getdents 78"} 2
AID T4 sys sysinfo{label="T4 sys sysinfo 99"} 3
AID T3 sys exit{label="T3 sys exit 60"} 3
AID T4 sys uname{label="T4 sys uname 63"} 4
```

AID MS sys unlink{label="MS sys unlink 87"} 5

AID T2 sys sched setscheduler{label="T2 sys sched setscheduler 144"} 6

AID T4 sys sched get priority min{label="T4 sys sched get priority min 147"} 6

AID D T2 sys clone{label="D T2 sys clone 56"} 6

AID MS sys set robust list{label="MS sys set robust list 273"} 6

AID T4 sys sched get priority max{label="T4 sys sched get priority max 146"} 6

AID MS sys sendmmsg{label="MS sys sendmmsg 307"} 8

AID MS sys bind{label="MS sys bind 49"} 8

AID T4 sys fdatasync{label="T4 sys fdatasync 75"} 9

AID T2 sys ftruncate{label="T2 sys ftruncate 77"} 10

AID T4 sys lstat{label="T4 sys lstat 6"} 11

AID MS sys getsockopt{label="MS sys getsockopt 55"} 12

AID MS sys setsockopt{label="MS sys setsockopt 54"} 14

AID MS sys fallocate{label="MS sys fallocate 285"} 15

AID T1 sys mkdir{label="T1 sys mkdir 83"} 17

AID T2 sys ioctl{label="T2 sys ioctl 16"} 31

AID D T2 sys socket{label="D T2 sys socket 41"} 36

AID T4 sys access{label="T4 sys access 21"} 40

AID MS sys connect{label="MS sys connect 42"} 40

AID MS sys sendto{label="MS sys sendto 44"} 70

AID T4 sys fstat{label="T4 sys fstat 5"} 72

AID MS sys getsockname{label="MS sys getsockname 51"} 73

AID MS sys getpeername{label="MS sys getpeername 52"} 78

AID MS sys openat{label="MS sys openat 257"} 85

AID T3 sys close{label="T3 sys close 3"} 109

AID MS sys madvise{label="MS sys madvise 28"} 148

AID T3 sys munmap{label="T3 sys munmap 11"} 151

AID T4 sys stat{label="T4 sys stat 4"} 252

AID T3 sys mmap{label="T3 sys mmap 9"} 280

AID T3 sys lseek{label="T3 sys lseek 8"} 597

AID T3 sys fcntl{label="T3 sys fcntl 72"} 829

AID T3 sys writev{label="T3 sys writev 20"} 1167

AID MS sys recvfrom{label="MS sys recvfrom 45"} 2167

AID MS sys futex{label="MS sys futex 202"} 4991

AID T3 sys read{label="T3 sys read 0"} 47711

AID T3 sys poll{label="T3 sys poll 7"} 49763

AID T3 sys write{label="T3 sys write 1"} 69817

AID MS sys recvmsg{label="MS sys recvmsg 47"} 144408

AID MS sys sys sched getaffinity{label="MS sys sched getaffinity 204"} 1

AID T3 sys fsync{label="T3 sys fsync 74"} 1

AID T1 sys rename{label="T1 sys rename 82"} 1

AID T4 sys stat{label="T4 sys stat 4"} 1

AID T4 sys access{label="T4 sys access 21"} 1

AID T3 sys mmap{label="T3 sys mmap 9"} 2

AID T3 sys close{label="T3 sys close 3"} 2

AID MS sys openat{label="MS sys openat 257"} 3

AID T4 sys lstat{label="T4 sys lstat 6"} 5

AID T3 sys munmap{label="T3 sys munmap 11"} 66

AID T3 sys writev{label="T3 sys writev 20"} 85

AID MS sys madvise{label="MS sys madvise 28"} 115

AID T3 sys read{label="T3 sys read 0"} 206

AID T3 sys write{label="T3 sys write 1"} 310

AID MS sys futex{label="MS sys futex 202"} 487

AID T3 sys poll{label="T3 sys poll 7"} 584

AID MS sys recvmsg{label="MS sys recvmsg 47"} 1577

AID T3 sys writev{label="T3 sys writev 20"} 1

AID T3 sys read{label="T3 sys read 0"} 64

AID T3 sys write{label="T3 sys write 1"} 96

AID T3 sys poll{label="T3 sys poll 7"} 129

AID MS sys futex{label="MS sys futex 202"} 142

AID MS sys recvmsg{label="MS sys recvmsg 47"} 384

AID MS sys madvise{label="MS sys madvise 28"} 1

AID T3 sys exit{label="T3 sys exit 60"} 1

AID T3 sys writev{label="T3 sys writev 20"} 11

AID T3 sys read{label="T3 sys read 0"} 74

AID T3 sys write{label="T3 sys write 1"} 111

AID MS sys futex{label="MS sys futex 202"} 164

AID T3 sys poll{label="T3 sys poll 7"} 168

AID MS sys recvmsg{label="MS sys recvmsg 47"} 468

AID T3 sys writev{label="T3 sys writev 20"} 33

AID T3 sys read{label="T3 sys read 0"} 86

AID T3 sys write{label="T3 sys write 1"} 129

AID MS sys futex{label="MS sys futex 202"} 192

AID T3 sys poll{label="T3 sys poll 7"} 237

AID MS sys recvmsg{label="MS sys recvmsg 47"} 640

AID T3 sys writev{label="T3 sys writev 20"} 8

AID T3 sys read{label="T3 sys read 0"} 58

AID T3 sys write{label="T3 sys write 1"} 87

AID T3 sys poll{label="T3 sys poll 7"} 135

AID MS sys futex{label="MS sys futex 202"} 138

AID MS sys recvmsg{label="MS sys recvmsg 47"} 395

AID T4 sys sched get priority max{label="T4 sys sched get priority max 146"} 1

AID T4 sys sched get priority min{label="T4 sys sched get priority min 147"} 1

AID T2 sys ftruncate{label="T2 sys ftruncate 77"} 1

AID T2 sys sched setscheduler{label="T2 sys sched setscheduler 144"} 1

AID MS sys set robust list{label="MS sys set robust list 273"} 1

AID MS sys madvise{label="MS sys madvise 28"} 1

AID T4 sys fstat{label="T4 sys fstat 5"} 1

AID T3 sys munmap{label="T3 sys munmap 11"} 1

AID T3 sys exit{label="T3 sys exit 60"} 1

AID D T2 sys clone{label="D T2 sys clone 56"} 1

AID T1 sys mkdir{label="T1 sys mkdir 83"} 1

AID MS sys fallocate{label="MS sys fallocate 285"} 1

AID MS sys sendto{label="MS sys sendto 44"} 2

AID MS sys getsockname{label="MS sys getsockname 51"} 2

AID T4 sys access{label="T4 sys access 21"} 3

AID T3 sys close{label="T3 sys close 3"} 4

AID MS sys openat{label="MS sys openat 257"} 4

AID T3 sys fcntl{label="T3 sys fcntl 72"} 6

AID T3 sys mmap{label="T3 sys mmap 9"} 7

AID T4 sys stat{label="T4 sys stat 4"} 9

AID T3 sys lseek{label="T3 sys lseek 8"} 15

AID MS sys recvfrom{label="MS sys recvfrom 45"} 61

AID T3 sys write{label="T3 sys write 1"} 389

AID T3 sys read{label="T3 sys read 0"} 402

AID T3 sys writev{label="T3 sys writev 20"} 566

AID MS sys futex{label="MS sys futex 202"} 1027

AID T3 sys poll{label="T3 sys poll 7"} 1778

AID MS sys recvmsg{label="MS sys recvmsg 47"} 4491

AID T3 sys exit{label="T3 sys exit 60"} 1

AID MS sys madvise{label="MS sys madvise 28"} 1

AID MS sys fallocate{label="MS sys fallocate 285"} 2

AID MS sys unlink{label="MS sys unlink 87"} 2

AID T1 sys mkdir{label="T1 sys mkdir 83"} 3

AID T4 sys fdatasync{label="T4 sys fdatasync 75"} 4

AID T3 sys mprotect{label="T3 sys mprotect 10"} 4

AID MS sys bind{label="MS sys bind 49"} 5

AID MS sys sendmmsg{label="MS sys sendmmsg 307"} 5

AID T2 sys ftruncate{label="T2 sys ftruncate 77"} 5

AID T4 sys sched get priority max{label="T4 sys sched get priority max 146"} 6

AID MS sys set robust list{label="MS sys set robust list 273"} 6

AID T2 sys sched setscheduler{label="T2 sys sched setscheduler 144"} 6

AID D T2 sys clone{label="D T2 sys clone 56"} 6

AID T4 sys sched get priority min{label="T4 sys sched get priority min 147"} 6

AID T4 sys access{label="T4 sys access 21"} 9

AID T4 sys stat{label="T4 sys stat 4"} 11

AID T3 sys mmap{label="T3 sys mmap 9"} 14

AID MS sys openat{label="MS sys openat 257"} 15

AID T2 sys ioctl{label="T2 sys ioctl 16"} 17

AID T4 sys fstat{label="T4 sys fstat 5"} 19

AID MS sys setsockopt{label="MS sys setsockopt 54"} 20

AID MS sys getsockopt{label="MS sys getsockopt 55"} 20

AID T3 sys close{label="T3 sys close 3"} 30

AID D T2 sys socket{label="D T2 sys socket 41"} 35

AID MS sys connect{label="MS sys connect 42"} 62

AID MS sys sendto{label="MS sys sendto 44"} 67

Sample output from Prometheus to Grafana

```
# TYPE AID D T2 sys clone untyped
AID D T2 sys clone{instance="ADI F11 instance",job="ADI job",label="D T2 sys clone 56"} 1
# TYPE AID D T2 sys socket untyped
AID D T2 sys socket{instance="ADI F11 instance",job="ADI job",label="D T2 sys socket 41"} 4
# TYPE AID MS sys arch prctl untyped
AID MS sys arch prctl{instance="ADI F11 instance",job="ADI job",label="MS sys arch prctl 158"} 1
# TYPE AID MS sys bind untyped
AID MS sys bind{instance="ADI F11 instance",job="ADI job",label="MS sys bind 49"} 1
# TYPE AID MS sys connect untyped
AID MS sys connect{instance="ADI F11 instance",job="ADI job",label="MS sys connect 42"} 5
# TYPE AID MS sys epoll create untyped
AID MS sys epoll create{instance="ADI F11 instance",job="ADI job",label="MS sys epoll create 213"} 1
# TYPE AID MS sys epoll ctl untyped
AID MS sys epoll ctl{instance="ADI F11 instance",job="ADI job",label="MS sys epoll ctl 233"} 1
# TYPE AID MS sys epoll wait untyped
AID MS sys epoll wait{instance="ADI F11 instance",job="ADI job",label="MS sys epoll wait 232"} 4
# TYPE AID MS sys eventfd2 untyped
AID MS sys eventfd2{instance="ADI F11 instance",job="ADI job",label="MS sys eventfd2 290"} 7
# TYPE AID MS sys exit group untyped
AID MS sys exit group{instance="ADI F11 instance",job="ADI job",label="MS sys exit group 231"} 1
# TYPE AID MS sys fadvise64 untyped
AID MS sys fadvise64{instance="ADI F11 instance",job="ADI job",label="MS sys fadvise64 221"} 67
# TYPE AID MS sys fallocate untyped
AID MS sys fallocate{instance="ADI F11 instance",job="ADI job",label="MS sys fallocate 285"} 1
# TYPE AID MS sys futex untyped
AID MS sys futex{instance="ADI F11 instance",job="ADI job",label="MS sys futex 202"} 5
# TYPE AID MS sys getpeername untyped
AID MS sys getpeername{instance="ADI F11 instance",job="ADI job",label="MS sys getpeername 52"} 8
# TYPE AID MS sys getrandom untyped
AID MS sys getrandom{instance="ADI F11 instance",job="ADI job",label="MS sys getrandom 318"} 118
# TYPE AID MS sys getsockname untyped
```

AID MS sys getsockname{instance="ADI F11 instance",job="ADI job",label="MS sys getsockname 51"} 4

TYPE AID MS sys getsockopt untyped

AID MS sys getsockopt{instance="ADI F11 instance",job="ADI job",label="MS sys getsockopt 55"} 1

TYPE AID MS sys gettid untyped

AID MS sys gettid{instance="ADI F11 instance",job="ADI job",label="MS sys gettid 186"} 1

TYPE AID MS sys getuid untyped

AID MS sys getuid{instance="ADI F11 instance",job="ADI job",label="MS sys getuid 102"} 3

TYPE AID MS sys inotify add watch untyped

AID MS sys inotify add watch{instance="ADI F11 instance",job="ADI job",label="MS sys inotify add watch 254"}

1

TYPE AID MS sys inotify init1 untyped

AID MS sys inotify init1{instance="ADI F11 instance",job="ADI job",label="MS sys inotify init1 294"} 1

TYPE AID MS sys madvise untyped

AID MS sys madvise{instance="ADI F11 instance",job="ADI job",label="MS sys madvise 28"} 1

TYPE AID MS sys memfd create untyped

AID MS sys memfd create{instance="ADI F11 instance",job="ADI job",label="MS sys memfd create 319"} 2

TYPE AID MS sys openat untyped

AID MS sys openat{instance="ADI F11 instance",job="ADI job",label="MS sys openat 257"} 3

TYPE AID MS sys pipe2 untyped

AID MS sys pipe2{instance="ADI F11 instance",job="ADI job",label="MS sys pipe2 293"} 1

TYPE AID MS sys prlimit64 untyped

AID MS sys prlimit64{instance="ADI F11 instance",job="ADI job",label="MS sys prlimit64 302"} 4

TYPE AID MS sys readahead untyped

AID MS sys readahead{instance="ADI F11 instance",job="ADI job",label="MS sys readahead 187"} 12

TYPE AID MS sys readlink untyped

AID MS sys readlink{instance="ADI F11 instance",job="ADI job",label="MS sys readlink 89"} 9

TYPE AID MS sys recvfrom untyped

AID MS sys recvfrom{instance="ADI F11 instance",job="ADI job",label="MS sys recvfrom 45"} 23

TYPE AID MS sys recvmsg untyped

AID MS sys recvmsg{instance="ADI F11 instance",job="ADI job",label="MS sys recvmsg 47"} 13

TYPE AID MS sys sendmmsg untyped

AID MS sys sendmmsg{instance="ADI F11 instance",job="ADI job",label="MS sys sendmmsg 307"} 1

TYPE AID MS sys sendmsg untyped

AID MS sys sendmsg{instance="ADI F11 instance",job="ADI job",label="MS sys sendmsg 46"} 33

```
# TYPE AID MS sys sendto untyped
AID MS sys sendto{instance="ADI F11 instance",job="ADI job",label="MS sys sendto 44"} 2
# TYPE AID MS sys set robust list untyped
AID MS sys set robust list{instance="ADI F11 instance",job="ADI job",label="MS sys set robust list 273"} 1
# TYPE AID MS sys set tid address untyped
AID MS sys set tid address{instance="ADI F11 instance",job="ADI job",label="MS sys set tid address 218"} 1
# TYPE AID MS sys setsockopt untyped
AID MS sys setsockopt{instance="ADI F11 instance",job="ADI job",label="MS sys setsockopt 54"} 1
# TYPE AID MS sys shmat untyped
AID MS sys shmat{instance="ADI F11 instance",job="ADI job",label="MS sys shmat 30"} 4
# TYPE AID MS sys shmctl untyped
AID MS sys shmctl{instance="ADI F11 instance",job="ADI job",label="MS sys shmctl 31"} 4
# TYPE AID MS sys shmdt untyped
AID MS sys shmdt{instance="ADI F11 instance",job="ADI job",label="MS sys shmdt 67"} 2
# TYPE AID MS sys shmget untyped
AID MS sys shmget{instance="ADI F11 instance",job="ADI job",label="MS sys shmget 29"} 4
# TYPE AID MS sys shutdown untyped
AID MS sys shutdown{instance="ADI F11 instance",job="ADI job",label="MS sys shutdown 48"} 3
# TYPE AID MS sys sigaltstack untyped
AID MS sys sigaltstack{instance="ADI F11 instance",job="ADI job",label="MS sys sigaltstack 131"} 1
# TYPE AID MS sys socketpair untyped
AID MS sys socketpair{instance="ADI F11 instance",job="ADI job",label="MS sys socketpair 53"} 1
# TYPE AID MS sys sys sched getaffinity untyped
AID MS sys sys sched getaffinity{instance="ADI F11 instance",job="ADI job",label="MS sys sched getaffinity
204"} 1
# TYPE AID MS sys tgkill untyped
AID MS sys tgkill{instance="ADI F11 instance",job="ADI job",label="MS sys tgkill 234"} 84
# TYPE AID MS sys unlink untyped
AID MS sys unlink{instance="ADI F11 instance",job="ADI job",label="MS sys unlink 87"} 1
# TYPE AID T1 sys chmod untyped
AID T1 sys chmod{instance="ADI F11 instance",job="ADI job",label="T1 sys chmod 90"} 1
# TYPE AID T1 sys mkdir untyped
AID T1 sys mkdir{instance="ADI F11 instance",job="ADI job",label="T1 sys mkdir 83"} 6
# TYPE AID T1 sys rename untyped
```

- **aid nagios.sh**

  Script aid nagios produces output of information, system calls used only by Nagios. The information was ingested from bftrace, in 60 second iterations. The output contains a unique character per system call that was used by Nagios. Every time a system call is called by Nagios, the unique character is printed. The output is then piped to an awk script for calculating the Shannon entropy, Shannon Wiener Index and Simpson Index of the output. The mesuremnts with the title of Index are piped to Promethius and Pushgateways to be presented in Grafana.

aid nagios.sh

```
#!/bin/bash
sudo timeout -s SIGINT 60 bpftrace -e 'tracepoint:raw_syscalls:sys_enter* {@[args->id, comm] = count();} interval:s:60 {print(@); clear(@);}' | awk '/ nagios/{gsub(/\@|\[|\,|\]|\:/,""); for (c=0;c<$3;c++) if ($1=='0') printf "0"; \
 else if ($1=='1') printf "1";
 else if ($1=='2') printf "2";
 else if ($1=='3') printf "3";
 else if ($1=='4') printf "4";
 else if ($1=='5') printf "5";
 else if ($1=='6') printf "6";
 else if ($1=='7') printf "7";
 else if ($1=='8') printf "8";
 else if ($1=='9') printf "9";
 else if ($1=='10') printf "a";
 else if ($1=='11') printf "b";
 else if ($1=='12') printf "c";
 else if ($1=='13') printf "d";
 else if ($1=='14') printf "e";
 else if ($1=='15') printf "g";
 else if ($1=='16') printf "g";
 else if ($1=='17') printf "17";
 else if ($1=='18') printf "18";
 else if ($1=='19') printf "19";
 else if ($1=='20') printf "20";
 else if ($1=='21') printf "h";
 else if ($1=='22') printf "i";
 else if ($1=='23') printf "j";
```

```
else if ($1=='24') printf "24";
else if ($1=='25') printf "25";
else if ($1=='26') printf "26";
else if ($1=='27') printf "27";
else if ($1=='28') printf "28";
else if ($1=='29') printf "29";
else if ($1=='30') printf "30";
else if ($1=='31') printf "31";
else if ($1=='32') printf "32";
else if ($1=='33') printf "k";
else if ($1=='34') printf "34";
else if ($1=='35') printf "35";
else if ($1=='36') printf "36";
else if ($1=='37') printf "l";
else if ($1=='38') printf "38";
else if ($1=='39') printf "m";
else if ($1=='40') printf "40";
else if ($1=='41') printf "n";
else if ($1=='42') printf "o";
else if ($1=='43') printf "43";
else if ($1=='44') printf "q";
else if ($1=='45') printf "r";
else if ($1=='46') printf "46";
else if ($1=='47') printf "47";
else if ($1=='48') printf "s";
else if ($1=='49') printf "t";
else if ($1=='50') printf "u";
else if ($1=='51') printf "51";
else if ($1=='52') printf "52";
else if ($1=='53') printf "v";
```

```
else if ($1=='54') printf "w";
else if ($1=='55') printf "x";
else if ($1=='56') printf "y";
else if ($1=='57') printf "57T";
else if ($1=='58') printf "58T";
else if ($1=='59') printf "z";
else if ($1=='60') printf "60";
else if ($1=='61') printf "A";
else if ($1=='62') printf "B";
else if ($1=='63') printf "C";
else if ($1=='64') printf "64";
else if ($1=='65') printf "65";
else if ($1=='66') printf "66";
else if ($1=='67') printf "67";
else if ($1=='68') printf "68";
else if ($1=='69') printf "69";
else if ($1=='70') printf "70";
else if ($1=='71') printf "71";
else if ($1=='72') printf "D";
else if ($1=='73') printf "73T";
else if ($1=='74') printf "E";
else if ($1=='75') printf "75";
else if ($1=='76') printf "76T";
else if ($1=='77') printf "F";
else if ($1=='78') printf "G";
else if ($1=='79') printf "H";
else if ($1=='80') printf "I";
else if ($1=='81') printf "81";
else if ($1=='82') printf "J";
else if ($1=='83') printf "83T";
```

```
else if ($1=='84') printf "84T";
else if ($1=='85') printf "85T";
else if ($1=='86') printf "86T";
else if ($1=='87') printf "K";
else if ($1=='88') printf "88T";
else if ($1=='89') printf "89";
else if ($1=='90') printf "90";
else if ($1=='91') printf "L";
else if ($1=='92') printf "92T";
else if ($1=='93') printf "93T";
else if ($1=='94') printf "94T";
else if ($1=='95') printf "M";
else if ($1=='96') printf "N";
else if ($1=='97') printf "O";
else if ($1=='98') printf "98T";
else if ($1=='99') printf "99T";
else if ($1=='100') printf "100";
else if ($1=='101') printf "101D";
else if ($1=='102') printf "P";
else if ($1=='103') printf "103T";
else if ($1=='104') printf "Q";
else if ($1=='105') printf "R";
else if ($1=='106') printf "S";
else if ($1=='107') printf "T";
else if ($1=='108') printf "U";
else if ($1=='109') printf "V";
else if ($1=='110') printf "W";
else if ($1=='111') printf "X";
else if ($1=='112') printf "Y";
else if ($1=='113') printf "113T";
```

```
else if ($1=='114') printf "114T";
else if ($1=='115') printf "115";
else if ($1=='116') printf "Z";
else if ($1=='117') printf "117T";
else if ($1=='118') printf "118";
else if ($1=='119') printf "119T";
else if ($1=='120') printf "120";
else if ($1=='121') printf "121";
else if ($1=='122') printf "122T";
else if ($1=='123') printf "123T";
else if ($1=='124') printf "124";
else if ($1=='125') printf "125";
else if ($1=='126') printf "126";
else if ($1=='127') printf "127";
else if ($1=='128') printf "128";
else if ($1=='129') printf "129";
else if ($1=='130') printf "130";
else if ($1=='131') printf "131";
else if ($1=='132') printf "132";
else if ($1=='133') printf "-";
else if ($1=='134') printf "134D";
else if ($1=='135') printf "135D";
else if ($1=='136') printf "136D";
else if ($1=='137') printf "137";
else if ($1=='138') printf "138";
else if ($1=='139') printf "139D";
else if ($1=='140') printf "140";
else if ($1=='141') printf "141T";
else if ($1=='142') printf "142T";
else if ($1=='143') printf "143";
```

```
else if ($1=='144') printf "144T";
else if ($1=='145') printf "145";
else if ($1=='146') printf "146";
else if ($1=='147') printf "147";
else if ($1=='148') printf "148";
else if ($1=='149') printf "149T";
else if ($1=='150') printf "150";
else if ($1=='151') printf "151T";
else if ($1=='152') printf "152";
else if ($1=='153') printf "153T";
else if ($1=='154') printf "154T";
else if ($1=='155') printf "155D";
else if ($1=='156') printf "156T";
else if ($1=='157') printf "157";
else if ($1=='158') printf "+";
else if ($1=='159') printf "159T";
else if ($1=='160') printf "^";
else if ($1=='161') printf "161";
else if ($1=='162') printf "162";
else if ($1=='163') printf "163D";
else if ($1=='164') printf "164DT";
else if ($1=='165') printf "165DT";
else if ($1=='166') printf "166DT";
else if ($1=='167') printf "167DT";
else if ($1=='168') printf "168DT";
else if ($1=='169') printf "169DT";
else if ($1=='170') printf "170T";
else if ($1=='171') printf "171T";
else if ($1=='172') printf "172DT";
else if ($1=='173') printf "173DT";
```

```
else if ($1=='174') printf "174DT";
else if ($1=='175') printf "175D";
else if ($1=='176') printf "176DT";
else if ($1=='177') printf "177D";
else if ($1=='178') printf "178D";
else if ($1=='179') printf "179D";
else if ($1=='180') printf "180DT";
else if ($1=='181') printf "181";
else if ($1=='182') printf "182";
else if ($1=='183') printf "183T";
else if ($1=='184') printf "184";
else if ($1=='185') printf "185";
else if ($1=='186') printf "186";
else if ($1=='187') printf "187";
else if ($1=='188') printf "188";
else if ($1=='189') printf "189";
else if ($1=='190') printf "190";
else if ($1=='191') printf "191";
else if ($1=='192') printf "192";
else if ($1=='193') printf "193";
else if ($1=='194') printf "194";
else if ($1=='195') printf "195";
else if ($1=='196') printf "196";
else if ($1=='197') printf "197";
else if ($1=='198') printf "198";
else if ($1=='199') printf "199";
else if ($1=='200') printf "200";
else if ($1=='201') printf "201T";
else if ($1=='202') printf "@";
else if ($1=='203') printf "203";
```

```
else if ($1=='204') printf "204";
else if ($1=='205') printf "205";
else if ($1=='206') printf "206";
else if ($1=='207') printf "207";
else if ($1=='208') printf "208";
else if ($1=='209') printf "209";
else if ($1=='210') printf "210";
else if ($1=='211') printf "211";
else if ($1=='212') printf "212";
else if ($1=='213') printf "!";
else if ($1=='214') printf "214";
else if ($1=='215') printf "215";
else if ($1=='216') printf "216";
else if ($1=='217') printf "217";
else if ($1=='218') printf "218";
else if ($1=='219') printf "219";
else if ($1=='220') printf "220";
else if ($1=='221') printf "221";
else if ($1=='222') printf "222";
else if ($1=='223') printf "223";
else if ($1=='224') printf "224";
else if ($1=='225') printf "225";
else if ($1=='226') printf "226";
else if ($1=='227') printf "227D";
else if ($1=='228') printf "228";
else if ($1=='229') printf "229";
else if ($1=='230') printf "230";
else if ($1=='231') printf "=";
else if ($1=='232') printf " ";
else if ($1=='233') printf "%";
```

```
else if ($1=='234') printf "234";
else if ($1=='235') printf "235";
else if ($1=='236') printf "236";
else if ($1=='237') printf "237D";
else if ($1=='238') printf "238D";
else if ($1=='239') printf "239D";
else if ($1=='240') printf "240";
else if ($1=='241') printf "241T";
else if ($1=='242') printf "242";
else if ($1=='243') printf "243";
else if ($1=='244') printf "244";
else if ($1=='245') printf "245";
else if ($1=='246') printf "246";
else if ($1=='247') printf "247";
else if ($1=='248') printf "248D";
else if ($1=='249') printf "249D";
else if ($1=='250') printf "250";
else if ($1=='251') printf "251";
else if ($1=='252') printf "252";
else if ($1=='253') printf "253";
else if ($1=='254') printf "254";
else if ($1=='255') printf "255";
else if ($1=='256') printf "256";
else if ($1=='257') printf "257";
else if ($1=='258') printf "258";
else if ($1=='259') printf "259";
else if ($1=='260') printf "260";
else if ($1=='261') printf "261";
else if ($1=='262') printf "262";
else if ($1=='263') printf "263";
```

```
else if ($1=='264') printf "264";
else if ($1=='265') printf "265";
else if ($1=='266') printf "266";
else if ($1=='267') printf "267";
else if ($1=='268') printf "268";
else if ($1=='269') printf "269";
else if ($1=='270') printf "270";
else if ($1=='271') printf "271";
else if ($1=='272') printf "272D";
else if ($1=='273') printf "273";
else if ($1=='274') printf "274";
else if ($1=='275') printf "275";
else if ($1=='276') printf "276";
else if ($1=='277') printf "277";
else if ($1=='278') printf "278";
else if ($1=='279') printf "279D";
else if ($1=='280') printf "280";
else if ($1=='281') printf "281";
else if ($1=='282') printf "282";
else if ($1=='283') printf "283";
else if ($1=='284') printf "284";
else if ($1=='285') printf "285";
else if ($1=='286') printf "286";
else if ($1=='287') printf "287";
else if ($1=='288') printf "288";
else if ($1=='289') printf "289";
else if ($1=='290') printf "290";
else if ($1=='291') printf "291";
else if ($1=='292') printf "292";
else if ($1=='293') printf ".";
```

```
else if ($1=='294') printf "294";
else if ($1=='295') printf "295";
else if ($1=='296') printf "296";
else if ($1=='297') printf "297";
else if ($1=='298') printf "298D";
else if ($1=='299') printf "299";
else if ($1=='300') printf "300";
else if ($1=='301') printf "301";
else if ($1=='302') printf "302";
else if ($1=='303') printf "303D";
else if ($1=='304') printf "304D";
else if ($1=='305') printf "305D";
else if ($1=='306') printf "306";
else if ($1=='307') printf "307";
else if ($1=='308') printf "308D";
else if ($1=='309') printf "309";
else if ($1=='310') printf "310D";
else if ($1=='311') printf "311D";
else if ($1=='312') printf "312D";
else if ($1=='313') printf "313D";
else if ($1=='314') printf "314";
else if ($1=='315') printf "315";
else if ($1=='316') printf "316";
else if ($1=='317') printf "317";
else if ($1=='318') printf "318";
else if ($1=='319') printf "319";
else if ($1=='320') printf "320D";
else if ($1=='321') printf "321D";
else if ($1=='322') printf "322";
else if ($1=='323') printf "323D";
```

```
 else if ($1=='324') printf "324";
 else if ($1=='325') printf "325";
 else if ($1=='326') printf "326";
 else if ($1=='327') printf "327";
 else if ($1=='328') printf "328";
 else if ($1=='329') printf "329";
 else if ($1=='330') printf "330";
 else if ($1=='331') printf "331";
 else if ($1=='332') printf "332"; \
else print $1;}' | ./Sh index.awk | curl --data-binary @- http://localhost:9091/metrics/job/ADI job/instance/ADI N inst
exit
```

- **Sample output from aid nagios.sh**

local@localhost:~$ while true; do ./aid firefox n.sh; sleep 1; done

```
hKKKKK0000000000000000000000000000000000000777777777777777777
7777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777777777777444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444
4444444444444444qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
```

qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqq111KKKKKKK0000000000000000000000000000000000000000000000000007
77777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777
77774444444444444444444444444444444444444444444444444444444444444444444444444
44444444444444444444444444444444444444444444444444444444444444444444444444444
44444444444444444444444444444444444444444444444444444444444444444444444444444
44444444444444444444444444444444444444444444444444444444444444444444444444444
44444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444qqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq

- **aid index.awk**

  Script aid index.awk produces output of information, system calls used only by Nagios. The information was ingested from bftrace, in 60 second iterations. The output contains a calculation of Shannon entropy, Shannon Wiener Index and Simpson Index of the input of unique string made by the system calls. The measurements with the titles of Indexes are piped to Promethius and Pushgateways to be presented in Grafana.

```
#!/usr/bin/awk -f
# Original entropy awk script creators at Rosseta Stone
# Script was modified to compute additional Shannon Wiener Index and Simpson Index
for system call input strings
{
        for (i=1; i<= length($0); i++) {
                H[substr($0,i,1)]++;
                N++;
        }
}
END {
        for (i in H) {
                p = H[i]/N;
                E -= p * log(p);
```

```
            D -= p * (p);
        }
        print "AID Nagios SE{label=\"Nagios SE 1\"}", E/log(2) "\n" "AID Nagios SW{label=\"Nagios SWI 1\"}", E/log(10) "\n" "AID Nagios SW NL{label=\"SWI NL 1\"}", E "\n" "AID Nagios SI{label=\"Nagios SI 1\"}", -1/D;}
```

Sample output form aid index.awk

```
local@localhost:~$ while true; do ./aid firefox.sh; sleep 1; done
AID Firefox SE{label="Shannon Entropy"} 2.05952
AID Firefox SW{label="Shannon-Wiener Index"} 0.619976
AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.42755
AID Firefox SI{label="Simpsons Index"} 2.92804
AID Firefox SE{label="Shannon Entropy"} 1.95951
AID Firefox SW{label="Shannon-Wiener Index"} 0.589871
AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.35823
AID Firefox SI{label="Simpsons Index"} 2.86929
AID Firefox SE{label="Shannon Entropy"} 2.09962
AID Firefox SW{label="Shannon-Wiener Index"} 0.632049
AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.45535
AID Firefox SI{label="Simpsons Index"} 3.20786
AID Firefox SE{label="Shannon Entropy"} 1.97771
AID Firefox SW{label="Shannon-Wiener Index"} 0.595349
AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.37084
AID Firefox SI{label="Simpsons Index"} 3.08319
```

AID Firefox SE{label="Shannon Entropy"} 1.69302

AID Firefox SW{label="Shannon-Wiener Index"} 0.509649

AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.17351

AID Firefox SI{label="Simpsons Index"} 2.75949

AID Firefox SE{label="Shannon Entropy"} 1.88418

AID Firefox SW{label="Shannon-Wiener Index"} 0.567193

AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.30601

AID Firefox SI{label="Simpsons Index"} 2.67647

AID Firefox SE{label="Shannon Entropy"} 2.17944

AID Firefox SW{label="Shannon-Wiener Index"} 0.656078

AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.51068

AID Firefox SI{label="Simpsons Index"} 3.11901

AID Firefox SE{label="Shannon Entropy"} 2.36263

AID Firefox SW{label="Shannon-Wiener Index"} 0.711223

AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.63765

AID Firefox SI{label="Simpsons Index"} 3.5209

AID Firefox SE{label="Shannon Entropy"} 2.09517

AID Firefox SW{label="Shannon-Wiener Index"} 0.63071

AID Firefox SW NL{label="Shannon-Wiener Index NL"} 1.45226

AID Firefox SI{label="Simpsons Index"} 3.3508