Information Security 2015

Proceedings of the
13th Australasian Information Security Conference (AISC 2015)
Sydney, Australia, 27 - 30 January 2015

Ian Welch and Xun Yi, Eds.

Volume 161 in the Conferences in Research and Practice in Information Technology Series.
Published by the Australian Computer Society Inc.

Published in association with the ACM Digital Library.
# Table of Contents

**Proceedings of the 13th Australasian Information Security Conference (AISC 2015), Sydney, Australia, 27 - 30 January 2015**

- Preface .......................................................... vii
- Programme Committee ........................................ viii
- Organising Committee .......................................... ix
- Welcome from the Organising Committee .................. x
- CORE - Computing Research & Education .................. xi
- ACSW Conferences and the Australian Computer Science Communications ......................... xiii
- ACSW and AISC 2015 Sponsors ................................. xv

## Contributed Papers

- JMD: A Hybrid Approach for Detecting Java Malware .......................... 3  
  Adrian Herrera and Ben Cheney

- Hypervisor based security architecture to protect web applications .......... 15  
  Dilshan Jayarathna, Udaya Tupakula and Vijay Varadharajan

- Annex: A Middleware for Constructing High-Assurance Software Systems .... 25  
  Tristan Newby, Duncan Grove, Alex Murray, Chris Owen, Jim McCarthy and Chris North

- On the Effectiveness of Virtualisation Assisted View Comparison for Rootkit Detection .................... 35  
  Toby Richer, Grant Neale and Grant Osborne

- Hardware Trojans - A Systemic Threat ................................ 45  
  John Shield, Bradley Hopkins, Mark Beaumont and Chris North

## Contributed Posters

- Information Privacy Concerns of Real Estate Customers and Information Security in the Real Estate Industry: an Empirical Analysis .......................... 53  
  Deepa Mani, Alireza Heravi, Kim-Kwang Raymond Choo and Sameera Mubarak

- Correcting flaws in Mitchell’s analysis of EPBC .................................. 57  
  Binbin Di, Leonie Simpson, Harry Bartlett, Ed Dawson and Kenneth Wong

- Internet-wide Scanning Taxonomy and Framework ............................... 61  
  David Myers, Ernest Foo and Kenneth Radke

## Doctoral Symposium
Preface

The 13th annual meeting of the Australasian Information Security Conference (ACSW-AISC 2015) was held in Sydney, at the Parramatta campus of the University of Western Sydney (UWS), Australia, as part of the Australasian Computer Science Week, January 27-30, 2015. Originally, our conference was called the Australasian Information Security Workshop. In 2008, it was renamed the Australasian Information Security Conference. The main aim of the ACSW-AISC is to provide a venue for researchers to present their work on all aspects of information security, and to promote collaboration between academic and industrial researchers working in this area.

We received 13 submissions from both academia and industry with most authors being based in either Australian or New Zealand institutions. After a rigorous refereeing process, we accepted 5 papers (38%) for presentation at ACSW-AISC 2015. We also accepted 3 short student papers for presentation at the poster session and nominated an additional student paper for presentation at the Doctoral Consortium. We extend our thanks to all the authors for their quality submissions, and to all the members of the Program Committee and external referees for their expert reviews.

We used EasyChair to manage the submissions and reviews. We found this webservice very helpful and we thank its maintainers. Last but not least, we are grateful to the ACSW 2015 organizing committee for their hard work and invaluable support throughout the preparation of the conference.

Ian Welch
Victoria University of Wellington

Xun Yi
Royal Melbourne Institute of Technology

ACSW-AISC 2015 Programme Chairs
January 2015
Programme Committee

Chairs

Ian Welch, Victoria University of Wellington, New Zealand
Xun Yi, Royal Melbourne Institute of Technology, Australia

Members

Lynn Batten, Deakin University
Philip Branch, Swinburne University of Technology
Ljiljana Brankovic, University of Newcastle
Kim-Kwang Raymond Choo, University of South Australia
Sherman S. M. Chow, Chinese University of Hong Kong
Olivier De Vel, DSTO
Denise De Vries, Flinders University
Juan Gonzalez Nieto, QUT
Feng Hao, Newcastle University
Dong Seong Kim, University of Canterbury
Van Lam Le, Can Tho University
Paul Montague, DSTO
Shaoning Pang, Unitec Institute of Technology
Josef Pieprzyk, Queensland University of Technology
Giovanni Russello, The University of Auckland
Christian Seifert, Microsoft Cooperation
Ron Steinfeld, Monash University
Clark Thomborson, University of Auckland
Emiliano Tramontana, University of Catania
Giang Vu Thi Huong, Hanoi University of Science and Technology
Ian Welch, Victoria University of Wellington
Yang Xiang, Deakin University
Xun Yi, RMIT
Organising Committee

Chairs
Professor Athula Ginige, University of Western Sydney, Australia
Associate Professor Paul Kennedy, University of Technology, Sydney, Australia

Local Chair
Dr Bahman Javadi, University of Western Sydney, Australia

Publicity Chair
Dr Ante Prodan, University of Western Sydney, Australia

Publication Chair
Dr Laurence Park, University of Western Sydney, Australia

Finance Chair
Michael Walsh, University of Western Sydney, Australia

Sponsorship Chair
Kerry Holling, University of Western Sydney, Australia

Doctoral Consortia Co-chairs
Professor Anthony Maeder, University of Western Sydney, Australia
Dr Siamak Tafavogh, University of Technology, Sydney, Australia

Event Coordinator
Nicolle Fowler, University of Western Sydney, Australia
Welcome from the Organising Committee

On behalf of the Organising Committee, it is our pleasure to welcome you to Sydney and to the 2015 Australasian Computer Science Week (ACSW 2015). This year the conference is hosted by the University of Western Sydney and its School of Computing, Engineering and Mathematics.

A major highlight of the ACSW 2015 will be the Industry Research Nexus day on 27th January 2015. The aim is for industry leaders and academic researchers to come together and explore research areas of mutual interest. Many University research groups and 15 industries have confirmed their participation.

ACSW 2015 consists of 9 sub conferences covering a range of topics in Computer Science and related areas. These conferences are:

- Asia-Pacific Conference on Conceptual Modelling (APCCM) (Chaired by Motoshi Saeki and Henning Köhler)
- Australasian Computer Science Conference (ACSC) (Chaired by Dave Parry)
- Australasian Computing Education Conference (ACE) (Chaired by Daryl D’Souza and Katrina Falkner)
- Australasian Information Security Conference (AISC) (Chaired by Ian Welch and Xin Yi)
- Australasian Symposium on Parallel and Distributed Computing (AusPDC) (Chaired by Bahman Javadi and Saurabh Garg)
- Australasian User Interface Conference (AUIC) (Chaired by Stefan Marks and Rachel Blagojevic)
- Australasian Web Conference (AWC) (Chaired by Joseph Davis)
- Australasian Workshop on Health Informatics and Knowledge Management (HIKM) (Chaired by Anthony Maeder and Jim Warren)
- Interactive Entertainment (IE) (Chaired by Yusuf Pisan and Keith Nesbitt)

Social events are a very important part of a conference as these provide many networking opportunities. To foster networking we have included a reception with industry on 27th January 2015, a Welcome reception on 28th January 2015 and a conference dinner on 29th January 2015.

Organising a multi-conference event such as ACSW is a challenging process even with many hands helping to distribute the workload, and actively cooperating to bring the events to fruition. This year has been no exception. We would like to share with you our gratitude towards all members of the organising committee for their combined efforts and dedication to the success of ACSW2015. We also thank all conference co-chairs and reviewers, for putting together the conference programs which are the heart of ACSW, and to the organisers of the sub conferences, workshops, poster sessions and Doctoral Consortium. Special thanks to John Grundy as chair of CoRE for his support for the innovations we have introduced this year.

This year we have secured generous support from several sponsors to help defray the costs of the event and we thank them for their welcome contributions. Last, but not least, we would like to thank all speakers, participants and attendees, and we look forward to several days of stimulating presentations, debates, friendly interactions and thoughtful discussions.

Athula Ginige
University of Western Sydney

Paul Kennedy
University of Technology Sydney

ACSW2015 General Co-Chairs
January, 2015
CORE welcomes all delegates to ACSW2015 in Sydney. CORE, the peak body representing academic computer science in Australia and New Zealand, is responsible for the annual ACSW series of meetings, which are a unique opportunity for our community to network and to discuss research and topics of mutual interest. The component conferences of ACSW have changed over time with additions and subtractions: ACSC, ACE, AISC, AUIC, AusPDC, HIKM, ACDC, APCCM, CATS and AWC. Two doctoral consortia (ACDC and ACE-DC) and an Australasian Early Career Researchers Workshop (AECRW) reflect the evolving dimensions of ACSW and build on the diversity of the Australasian computing community. A specific industry day on the 27th January to facilitate academic / industry discussion and networking is a key feature of ACSW 2015.

In 2015, we are fortunate to have Professor Omer Rana, Associate Professor Pascal Hitzler and Professor Mark Sagar providing keynote talks to the conference. I thank them for their contributions to ACSW2015.

The efforts of the conference chairs and their program committees have led to strong programs in all the conferences, thanks very much for all your efforts. Thanks are particularly due to Professor Athula Ginige, Professor Paul Kennedy and their colleagues for organising what promises to be a vibrant event. Below I outline some of CORE’s activities in 2013/14.

I welcome feedback on these including other activities you think CORE should be active in.

The major sponsor of Australian Computer Science Week:
– The venue for the annual Heads and Professors meeting
– An opportunity for Australian & NZ computing staff and postgrads to network and help develop their research and teaching
– Substantial discounts for attendees from member departments
– A doctoral consortium at which postgrads can seek external expertise for their research
– An Early Career Research forum to provide ECRs input into their development

Sponsor of several research, teaching and service awards:
– Chris Wallace award for Distinguished Research Contribution
– CORE Teaching Award
– Australasian Distinguished Doctoral Dissertation
– John Hughes Distinguished Service Award
– Various “Best Student Paper” awards at ACSW

Development, maintenance, and publication of the CORE conference and journal rankings. In 2014 this includes a heavily-used web portal with a range of holistic venue information and a community update of the CORE 2013 conference rankings.

Input into a number of community resources and issues of interest:
– Development of an agreed national curriculum defining Computer Science, Software Engineering, and Information Technology
– A central point for discussion of community issues such as research standards
– Various submissions on behalf of Computer Science Departments and Academics to relevant government and industry bodies, including recently on Australian Workplace ICT Skills development, the Schools Technology Curriculum and the Defence Trade Controls Act.

Coordination with other sector groups:
– Work with the ACS on curriculum and accreditation
– Work with groups such as ACDICT, ACPHIS and government on issues such as CS staff performance metrics and appraisal, and recruitment of students into computing
– A member of CRA (Computing Research Association) and Informatics Europe. These organisations are the North American and European equivalents of CORE.
– A member of Science & Technology Australia, which provides eligibility for Science Meets Parliament and opportunity for input into government policy, and involvement with Science Meets Policymakers

The 2014 Executive Committee has been looking at a range of activities that CORE can lead or contribute to, including more developmental activities for CORE members. This has also included a revamp of the mailing lists, web site, creation of discussion forums, identification of key issues for commentary and lobbying, and working with other groups to attract high aptitude students into ICT courses and careers.
Again, I welcome your active input into the direction of CORE in order to give our community improved visibility and impact. CORE’s existence is due to the support of the member departments in Australia and New Zealand, and I thank them for their ongoing contributions, in commitment and in financial support. Finally, I am grateful to all those who gave their time to CORE in 2014, and look forward to the continuing shaping and development of the Australasian computing community in 2015.

John Grundy
President, CORE
January, 2015
The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as Volume 0). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

**2016.** Volume 38. Host and Venue - Australian National University, Canberra, ACT.

**2015.** Volume 37. Host and Venue - University of Western Sydney, NSW.

**2014.** Volume 36. Host and Venue - AUT University, Auckland, New Zealand.

**2013.** Volume 35. Host and Venue - University of South Australia, Adelaide, SA.

**2012.** Volume 34. Host and Venue - RMIT University, Melbourne, VIC.

**2011.** Volume 33. Host and Venue - Curtin University of Technology, Perth, WA.

**2010.** Volume 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.

**2009.** Volume 31. Host and Venue - Victoria University, Wellington, New Zealand.

**2008.** Volume 30. Host and Venue - University of Wollongong, NSW.

**2007.** Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.

**2006.** Volume 28. Host and Venue - University of Tasmania, TAS.

**2005.** Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.


**2002.** Volume 24. Host and Venue - Monash University, Melbourne, VIC.

**2001.** Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.

**2000.** Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUIC.


**1998.** Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.


**1996.** Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.

**1995.** Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.


**1993.** Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.

**1992.** Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).

**1991.** Volume 13. Host and Venue - University of New South Wales, NSW.

**1990.** Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).

**1989.** Volume 11. Host and Venue - University of Wollongong, NSW.

**1988.** Volume 10. Host and Venue - University of Queensland, QLD.

**1987.** Volume 9. Host and Venue - Deakin University, VIC.

**1986.** Volume 8. Host and Venue - Australian National University, Canberra, ACT.


**1984.** Volume 6. Host and Venue - University of Adelaide, SA.

**1983.** Volume 5. Host and Venue - University of Sydney, NSW.

**1982.** Volume 4. Host and Venue - University of Western Australia, WA.

**1981.** Volume 3. Host and Venue - University of Queensland, QLD.

**1980.** Volume 2. Host and Venue - Australian National University, Canberra, ACT.

**1979.** Volume 1. Host and Venue - University of Tasmania, TAS.

**1978.** Volume 0. Host and Venue - University of New South Wales, NSW.
## Conference Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDC</td>
<td>Australasian Computing Doctoral Consortium</td>
</tr>
<tr>
<td>ACE</td>
<td>Australasian Computing Education Conference</td>
</tr>
<tr>
<td>ACSC</td>
<td>Australasian Computer Science Conference</td>
</tr>
<tr>
<td>ACSW</td>
<td>Australasian Computer Science Week</td>
</tr>
<tr>
<td>ADC</td>
<td>Australasian Database Conference</td>
</tr>
<tr>
<td>AISC</td>
<td>Australasian Information Security Conference</td>
</tr>
<tr>
<td>APCCM</td>
<td>Asia-Pacific Conference on Conceptual Modelling</td>
</tr>
<tr>
<td>AUIC</td>
<td>Australasian User Interface Conference</td>
</tr>
<tr>
<td>AusPDC</td>
<td>Australasian Symposium on Parallel and Distributed Computing (replaces AusGrid)</td>
</tr>
<tr>
<td>AWC</td>
<td>Australasian Web Conference</td>
</tr>
<tr>
<td>CATS</td>
<td>Computing: Australasian Theory Symposium</td>
</tr>
<tr>
<td>HIKM</td>
<td>Australasian Workshop on Health Informatics and Knowledge Management</td>
</tr>
<tr>
<td>IE</td>
<td>Australasian Conference on Interactive Entertainment</td>
</tr>
</tbody>
</table>

Note that various name changes have occurred, which have been indicated in the Conference Acronyms sections in respective CRPIT volumes.
ACS W and AISC 2015 Sponsors

We wish to thank the following sponsors for their contribution towards this conference.

Host Sponsors
University of Western Sydney
www.uws.edu.au
Australian Computer Society
www.acs.org.au
Computing Research and Education
www.core.edu.au

Platinum Sponsor
Dimension Data
www.dimensiondata.com

Gold Sponsors
NTT Australia Pty Ltd
www.au.ntt.com
Hewlett-Packard Company
www.hp.com
Intersect
www.intersect.org.au
Cognizant Technology Solutions
www.cognizant.com

Silver Sponsors
SGI
www.sgi.com
SMS Management and Technology
www.smsnt.com
AARNet
www.aarnet.edu.au
Macquarie Telecom
www.macquarie telecom.com

Bronze Sponsors
Australian Access Federation
aaf.edu.au
NEC Australia Pty Ltd
au.nec.com
Squiz Australia
www.squiz.net/au

Talent RISE
www.talentrise.org
Espire Infolabs Pty Ltd
www.espire.com
CONTRIBUTED PAPERS
JMD: A Hybrid Approach for Detecting Java Malware

Adrian Herrera  Ben Cheney

Defence Science and Technology Organisation
Adelaide, Australia

Email: {adrian.herrera, ben.cheney}@dsto.defence.gov.au

Abstract
With the rapid rise in the number of exploits targeting the Java runtime environment, new tools are required to detect these malicious Java applications. This paper proposes one such tool, the Java Malware Detector (JMD). JMD takes a hybrid approach that combines symbolic execution, instrumentation and dynamic analysis to detect malware that subverts Java’s access control mechanisms. Using this approach, we aim to derive any trigger conditions that may exist before instrumenting and executing the malware in a controlled environment to observe whether it escapes the Java security sandbox. A key element of this approach is our use of existing open-source software platforms—specifically, Java Pathfinder and AspectJ. By using real-world Java malware samples we are able to evaluate the effectiveness of JMD. The results of this evaluation show that JMD’s instrumentation and dynamic analysis capabilities provide an effective tool for detecting a wide range of Java malware: we successfully detected malware variants that represent fourteen of the known access control-related CVEs disclosed over the past four years. However, our success in using symbolic execution to derive trigger conditions was limited, mainly due to the incomplete state of the String handling implementation in Java Pathfinder’s symbolic execution plugin.

1 Introduction
The number of exploits targeting the Java Runtime Environment (JRE) has been increasing at an alarming rate. During the 12 months from September 2012 to August 2013, a Kaspersky Lab report claimed to have detected over 14.1 million attacks that relied on a Java exploit—an increase of 33.3% on the previous twelve months [9]. Cisco found that Java exploits represented 91% of all Indicators of Compromise (IoC) in 2013 [4]. Figure 1 illustrates the recent escalation of the Java malware problem in terms of the number of JRE CVEs issued each year.1

The most common delivery method for these exploits is a Java applet [9]. A Java applet is a Java application that is accessed via a web browser and executed on the local host in a Java Virtual Machine (JVM) instance. To execute a Java applet, a web browser requires the installation of a Java plug-in. Java’s pervasive install base2 means that there is a high chance a user has a Java plug-in installed, making Java a popular target for drive-by download attacks (where the user either unwittingly executes a malicious applet or authorises its execution without fully understanding the risks). Furthermore, Java exploits can provide a relatively cheap path to reliable remote code execution compared with exploits that require address space layout randomisation (ASLR) and data execution prevention (DEP) to be bypassed [34]. These combined properties have made Java a popular target for malware authors.

The prevalence of Java malware means that accurate and timely detection methods have become crucial elements of effective computer network defence. Traditional malware detection (e.g. that provided by an anti-virus product) relies heavily on signature-based techniques. However, the weaknesses of signature-based detection techniques are well-known [17, 15]—they can only detect previously discovered threats and are unable to detect zero-day exploits. Additionally, malware authors can often evade anti-virus products through obfuscation.

To supplement signature-based detection methods, dynamic analysis techniques are often used to run malware in a controlled environment (a sandbox) where it can be monitored for malicious activity. While dynamic analysis may overcome the limitations of signature-based detection in some cases, it is still possible for malware to evade detection by remaining dormant until a particular trigger condition is met [1, 16]. In order to identify and solve trigger conditions within a given sample (so that code coverage is maximised during dynamic analysis), symbolic execution can be utilised to explore alternate code

---

1The statistics for this graph are sourced from the CVE Details website, http://www.cvedetails.com.

2The Oracle Java 7 Windows installer claims that “three billion devices run Java".
paths and attempt to derive trigger conditions that are required for a particular code path to execute [1].

We have combined the techniques of symbolic execution, instrumentation and dynamic analysis in developing a system aimed specifically at detecting Java malware. Our Java Malware Detector (JMD) takes Java bytecode as input, performs symbolic execution on the bytecode to derive the trigger conditions required to maximise code coverage, instruments the bytecode and then performs dynamic analysis on the instrumented sample\(^3\) to determine if the JRE’s access control mechanisms have been subverted. Note that JMD is not designed to detect Java malware that targets native code vulnerabilities (§3.1 discusses our rationale for taking this approach).

The main contributions made in this paper are:

- The design and implementation of a hybrid detection system. This detection system combines symbolic execution, instrumentation and dynamic analysis techniques to specifically target Java malware.
- The extension of Java Pathfinder’s symbolic execution engine (Symbolic Pathfinder) to find trigger conditions in Java malware.
- The design and implementation of a mechanism for detecting the subversion of Java’s access control mechanisms using Aspect-Oriented Programming (AOP) techniques.

The remainder of this paper is organised as follows: §2 provides background information relevant to the design of JMD; §3 discusses our design and implementation decisions for JMD; §4 evaluates JMD’s performance; §5 examines previous work in the area of Java malware detection; and §6 concludes the paper and suggests some ideas for future work.

2 Background

This section provides background material on the Java Runtime Environment (JRE), its security model (with a particular focus on access control mechanisms) and examples of how JRE vulnerabilities have been exploited in the past.

2.1 The Java Runtime Environment

‘The JRE’ (as defined by Oracle [24]) formally consists of the JVM and all associated libraries and components which enable the execution of applications written in the Java programming language. Notionally, the runtime environment for a Java program can also be abstracted into a number of layers, as illustrated in Figure 2 (adapted from [31]).

A Java application (the top layer) is compiled into machine-independent Java bytecode, which executes in a JVM (second layer) instance. As is suggested by Java’s ‘write once, run anywhere’ mantra, the JVM—which is available on several operating systems (OS)/architecture combinations—allows a Java application to be cross-platform and portable. Java applications can be deployed in a variety of ways, but Java malware typically takes the form of an untrusted Java applet\(^4\) (see §2.2.2 for more on the nature of untrusted applets).

\(^3\) Dynamic analysis may proceed multiple times, depending on the number of trigger conditions identified during symbolic execution.

\(^4\) Given that applets are the most common deployment vector for Java exploits [8], we use the terms Java application and Java applet interchangeably when discussing Java malware.

Figure 2: Runtime environment of a Java program, with some elements of the Java security architecture highlighted

Beneath the JVM lies the native layer, with which the JVM interacts when Java applications and APIs need to interface with native code. Java’s APIs call native layer code in order to make platform-specific system calls (e.g. for implementing operations on a local file system) and to interact with software written in another language (e.g. C, assembly, etc.).

2.2 Code Verification and Access Control

Java’s security architecture is realised by a set of language features and APIs which encompass areas such as cryptography, PKI, authentication and secure communication [23]; however, the features and APIs most relevant to a discussion of Java malware are those related to code verification and access control.

Java bytecode is verified at load-time to “ensure that only legitimate bytecodes are executed in the Java runtime” [23] (discussed further in §2.2.1). At run-time, the access control APIs mediate access to sensitive resources and operations (e.g. access to local files, sending/receiving arbitrary data over a network, etc.) in accordance with a defined security policy (discussed further in §2.2.2).

It is important to note that code execution in the native layer occurs outside the purview of Java’s access control APIs. As such, Java malware seeking to subvert Java’s security restrictions may choose to target either a vulnerability in the access control API (i.e. code being executed in the JVM layer) or a vulnerability in a native library called by Java (i.e. code being executed in the native layer).

2.2.1 Class Verifier

When Java classes are loaded by the JVM, the class verifier performs several passes over the bytecode in order to ensure the correctness of the class. This includes checking for forged pointers, stack overflows and underflows, access (public/protected/private) violations and ensuring type safety. If any of these checks fail, an error is thrown by the JVM.

CVE-2012-1723 [21] is an example of a vulnerability in the class verifier which allows malicious bytecode to perform a type-confusion attack. This attack is possible due to an invalid optimisation in the class verifier when a field access operation is performed [27]. Class verifier exploits such as CVE-2012-1723 are considered outside JMD’s scope.

2.2.2 Security Manager

The security manager (represented by an object of type SecurityManager) mediates all access control decisions for Java APIs. This ensures that an
application adheres to a particular security policy at run-time.

When a local Java application is loaded from disk and run, it is executed with full user privileges and without a security manager [23, 7] (unless the user explicitly installs one in their application code or supplies the -Djava.security.manager command-line flag). By contrast, untrusted applets (which are typically accessed over the Internet from an unknown source) execute in the presence of a security manager, which enforces a reduced set of privileges in order to prevent the execution of unsafe operations. The restricted environment in which untrusted applets execute is commonly known as the Java sandbox.

Whenever a potentially unsafe operation is attempted by a sandboxed Java application or API, the SecurityManager object checks whether the class has been assigned the relevant permission (represented by a Permission object). The AccessController class is the SecurityManager’s mechanism for checking these permissions. If the operation is not allowed in the current Java sandbox, a SecurityException is thrown. Examples of potentially unsafe operations that require a particular set of permissions include file (e.g., read, write, etc.), socket (e.g., connect, accept, etc.) and ClassLoader (e.g., create) operations.

The set of permissions that are applied to a sandbox are declared in a security policy file (represented at run-time by a Policy object). This policy file explicitly lists the permissions granted to a set (or multiple sets) of classes loaded from a particular location and/or cryptographically signed by a particular key. At run-time, the association between these sets of classes and their granted permissions is encapsulated within a ProtectionDomain object. The JRE provides a default security policy, which can be either supplemented or replaced by an administrator who wants to provide their own custom policy file when the JVM is started.

Some example vulnerabilities that affect the security manager include CVE-2008-5353 [19], CVE-2012-0507 [20] and CVE-2013-0422 [22]. Although these vulnerabilities are quite different (ranging from deserialization issues to insufficient package access checking), the malware that targets them shares the same goal—to manipulate or disable the security manager so that arbitrary code can be executed. Exploits which target these types of vulnerabilities are what JMD is designed to detect.

3 Design and Implementation

In this section we outline our design choices and JMD’s implementation details.

3.1 Assumptions

We have designed and implemented JMD to detect malware that successfully exploits vulnerabilities in Java’s access control mechanisms (which operate in the second layer of Figure 2). In particular, we target malware that escapes the Java security sandbox by disabling or subverting the run-time security manager (as discussed in §2.2.2). Malware that targets vulnerabilities in either the Java class verifier (as discussed in §2.2.1) or in the Java native layer (as discussed in §2.1) are considered outside JMD’s scope.

To put this in perspective, Gorenz et al.’s survey of Java vulnerabilities found that approximately half of the vulnerabilities patched between 2011 and 2013 had the ability to “bypass the sandbox and execute arbitrary code on the host machine” [8]. The top two vulnerability sub-categories in their sample set were “unsafe reflection” and “least privilege violation”, both of which relate solely to Java’s access control APIs and are thus potentially detectable by JMD. While the amount of malware targeting the Java native layer may be increasing [31], Java’s access control model remains a popular target for malware authors—such exploits provide attackers with a “write once, run anywhere” weapon that does not require further customisation for a specific platform and/or OS [8].

3.2 Overview

Figure 3 provides a high-level overview of JMD’s different stages.

JMD takes a compiled Java application as input (Java bytecode, either as a .class or .jar file). In most cases this will be a Java applet, although JMD is able to analyse any Java application provided that it either (a) extends java.applet.Applet or (b) contains a static main method.

By default, a sample passes consecutively through JMD’s three main stages of symbolic execution, instrumentation and dynamic analysis before reporting results to the user. However, JMD can optionally restrict its processing to certain stages, such as symbolic execution only, or instrumentation/dynamic analysis only.

The symbolic execution stage (see §3.3) attempts to determine what code paths exist within the sample, and to derive the trigger conditions that are required to execute them.

In the instrumentation stage, JMD injects custom logging code into the sample by using aspect-oriented programming (AOP) techniques (see §3.4). This logging code is designed to report access control violations, such as subversion of the security manager or unauthorised access to privileged APIs.

In the dynamic analysis stage (see §3.5), instrumented bytecode is executed within a virtual environment, in the context of a specific security policy. The instrumented bytecode detects and records any successful actions which require permissions that were not granted in the security policy. For example, if a sample is able to set the SecurityManager object to null—an operation that requires explicit permissions which are not given in JMD’s default security policy—then JMD can conclusively infer that an access control vulnerability in the JRE has been successfully exploited. Note that dynamic analysis may be run multiple times, depending on how many trigger conditions are found during the symbolic execution stage.

At the conclusion of JMD’s processing, results are returned to the user in an XML report.

The three stages are discussed in greater detail in the following sections.
3.3 Symbolic Execution

Symbolic execution is a mechanism for deriving the conditions required for a particular code path to execute [11]. For performing symbolic execution of Java bytecode, we use Symbolic PathFinder (SPF), which is an extension of the open-source Java PathFinder platform. In this section we focus on the differences between applying SPF to malware analysis and SPF’s standard usage. A more detailed description of SPF’s internals is available in [25].

SPF was primarily designed as a bug checking and test generation tool for Java applications. At a high level, SPF proceeds by replacing method arguments with symbolic values. Symbolic expressions are dynamically constructed and recorded as these symbolic values propagate through a given execution path.

These symbolic values and expressions are then used to derive the constraints required for a particular code path to be executed. Through a combination of searching and backtracking, SPF attempts to build constraints for all code paths to ensure maximum code coverage. Once SPF has finished processing the sample, a constraint solver attempts to determine the exact conditions required for each particular code path to be executed.

There are a number of technical issues that arise when applying SPF to the identification of trigger conditions in Java malware. For example, SPF cannot process bytecode unless it contains local variable debug information. A malware author would not normally include this information in the bytecode, as removing it increases the difficulty of reverse engineering and determining the malware’s intention. Therefore, this debug information must be synthesised and injected into the bytecode before symbolic execution in SPF can occur.

SPF also requires the Java application to have a static `main` method as an entry point. However, Java applets do not have a `main` method—they rely on alternate entry points being invoked by the JVM at runtime.

Additionally, SPF only allows method arguments to be treated symbolically. However, it is common for malware to query the external environment and to take a specific code path that depends on the result of this query. Examples of external environment queries are given in Table 1. Given that SPF does not provide a mechanism to treat the result of these method calls symbolically, modifications were required to enable the exploration of code paths (and hence the identification of trigger conditions) that depend on their results.

The following sections describe how we extended SPF’s behaviour in order to remediate these issues for JMD.

3.3.1 Symbolic Execution of Applets

As mentioned in §3.3, SPF only allows the execution of applications with a static `main` method as its entry point. This prevents the symbolic execution of Java applets, which require a browser or applet viewer to use the `init` or `start` methods as an entry point.

To overcome this restriction, a class containing a `main` method (based on Kurniawan’s `AppletRunner` class [12]) was implemented for use as an entry point to run the applet within SPF.

We also implemented both a `model` and a `native peer` class to model the standard `Applet` class in SPF. This enabled us to abstract away the underlying behaviour of a Java applet (which depends on functionality in the Java native layer; an element beyond SPF’s symbolic execution capabilities). While our model and native peer classes were mostly sufficient for modelling malicious Java applets (which typically contain an exploit without any graphical component), this approach would be insufficient in modelling an applet containing an exploit that depended on the graphical capabilities available in the Java API.

3.3.2 Taint-based Symbolic Execution

We have used concepts from dynamic taint analysis [18] to allow SPF to construct constraints and explore code paths that depend on the external environment. Traditional dynamic taint analysis is the process of marking (or tainting) data that originates from an external (and possibly untrusted) source and tracking that data during application execution. However, dynamic taint analysis can also be combined with symbolic execution to construct constraints representing only the parts of execution that depend upon the tainted values [29]. Examples of external environment queries for which we want to construct constraints are listed in Table 1.

SPF provides a number of symbolic `listeners` that “gather and display information about the path conditions generated during the symbolic execution” [25]. We extended one of these listeners (the `Symbolic SequenceListener`) to allow local variables that store tainted data to be treated symbolically (in addition to method arguments, which are already treated symbolically by SPF).

Rather than blindly marking all local variables as symbolic in SPF (which would quickly lead to path explosion, discussed further in §4.4), JMD includes an SPF configuration option that allows the user to specify the signature of each method call that they wish to treat as producing tainted data. The tainted data (we limit ourselves to `String` objects) returned by these method calls is then marked as symbolic. SPF then treats this tainted data the same way it treats a symbolic method argument; i.e. constraints are constructed and solved in order to determine the trigger conditions required for a specific code path to be executed.

For example, the code in Figure 4 comes from a decompiled and deobfuscated malware sample. Using our taint-based symbolic execution approach, SPF can determine that a different code path will execute depending on the value returned by `System.getProperty("os.name")` (i.e. the OS running SPF). This information can then be provided to JMD’s dynamic analysis stage, in which the (instrumented) malware would be executed four times, once for each result returned by `System.getProperty`.

3.4 Instrumentation

After SPF has completed symbolic execution of the Java malware, it can be instrumented and prepared for dynamic analysis. The malware is instrumented using AspectJ, which is an open-source AOP extension to the Java language [13, 10]. AOP is a programming paradigm which seeks to achieve “separation of cross-cutting concerns” [10], where cross-cutting concerns are software components that impact (cut across) multiple
Table 1: Examples of external environment queries

<table>
<thead>
<tr>
<th>Method call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applet.getParameter</td>
<td>Customise an applet’s operation via a name/value pair</td>
</tr>
<tr>
<td>System.getProperty</td>
<td>Query a system property</td>
</tr>
<tr>
<td>System.getenv</td>
<td>Query an environment variable</td>
</tr>
</tbody>
</table>

String s1 = System.getProperty("os.name").
  toLowerCase();
  if (s1.indexOf("win") >= 0) {
    ...
  } else if (s1.indexOf("mac") >= 0) {
    ...
  }
  else if (s1.indexOf("nux") >= 0) {
    ...
  }
  else {
    return;
  }

Figure 4: Decompiled and deobfuscated malware that alters its behaviour based on a tainted local variable

logical modules. The classic example of a cross-cutting concern is a logging library, which is usually implemented by linking a logger multiple times into several separate program modules. This results in increased levels of dependency and complexity, as the logging implementation becomes entangled with the core functionality provided by the program. By contrast, an AOP approach would maintain separation between a program’s core logic and the logging components during the software development process. The components are subsequently combined together at compile or load-time to create the final system—a process known as *weaving*.

For JMD’s purposes, we have exploited AspectJ’s ability to weave existing bytecode (at both compile and load-time) in order to monitor Java’s access control mechanisms. Figure 5 shows the important parts of the code which JMD weaves into its samples, with some of the finer details abstracted away or omitted. The SecurityMonitor aspect (i.e. the module expressing the cross-cutting concern, which is the monitoring of access control mechanisms) includes a pointcut\(^7\) anyCallOrExec (line 11). This pointcut selects all method calls and their execution as join points (i.e. locations where the instrumentation code is weaved) to ensure malicious activities are recorded as early as possible.

The advice (lines 13–16) is executed whenever the anyCallOrExec pointcut picks out a join point (i.e. after every method call returns control to the caller, or a method body completes). This advice calls two methods: checkSecMan and checkForIllegalPerms. However, the advice is extensible and additional checks can easily be added.

The checkSecMan method (lines 18–26) compares the security manager’s current state with its initial state (recorded on line 3). If the check returns an inconsistent result, an alert can be logged that indicates a compromise of the security manager.

checkForIllegalPerms (lines 28–40) ensures adherence to the permissions granted in the security policy file. The permissions for the currently-executing object are retrieved from the join point’s context (lines 29–31). These permissions are then compared with the initial permissions specified in the policy file (line 5). If permissions that were not originally granted to the application are present, an alert can be logged indicating an access control compromise.

These methods (checkSecMan and checkForIllegalPerms) attempt to verify the integrity of the Java security manager and its related classes. As discussed in §2.2.2, the ultimate goal of Java malware that seeks to subvert the Java sandbox is to disable (or change the state of) the security manager so that arbitrary code can be executed. By using instrumentation as described here to monitor the security manager and the set of permissions to which an application should have access, malware which successfully exploits a vulnerability in a Java access control API can be detected during JMD’s dynamic analysis stage.

3.5 Dynamic Analysis

After instrumentation takes place, the sample is ready for dynamic analysis. To facilitate this, JMD includes support for two major virtualisation platforms: VirtualBox and VMware. A Cuckoo Sandbox [5] plugin was also developed.

Before dynamic analysis can occur, a sandbox virtual machine (VM) must be appropriately configured—i.e. an OS and a JRE must be installed. Given JMD’s detection strategy of observing a sample’s behaviour within a controlled environment, it is important that the JRE used during dynamic analysis be susceptible to the vulnerabilities targeted by the sample. As such, a sandbox VM can be configured with multiple JREs installed. The desired JRE for each JMD invocation can then be specified in a configuration file.

In order to execute Java applets in the sandbox VM, an appropriate HTML container file must first be generated. This HTML file includes any applet parameters that were found during the symbolic execution stage (see §3.3) or that have been explicitly entered by the user. During application execution (which involves executing the HTML file in a web browser or the JDK’s appletviewer utility), the SecurityMonitor aspect detects and logs malicious activity targeting access control mechanisms as described in §3.4. If multiple combinations of applet parameters are found in the symbolic execution stage, then the applet is executed once for each combination. Once completed, JMD’s logs are retrieved from the sandbox VM, the VM is (optionally) reverted to a clean snapshot and the logs are processed.

JMD produces an XML report describing its results. An example XML report is given in Figure 6. The report contains information on the runtime environment (gathered in the SecurityMonitor’s constructor) and the specific classes that performed any (detected) malicious actions. The example in Figure 6 shows that both an unauthorised permission and the disabling of the security manager were detected

---

\(^7\)In AOP parlance, a *pointcut* is a program element which selects a particular *join point*—for example, a particular method signature like `System.out.println(String)` is a potential join point that could be expressed in a pointcut. Data from the execution context of this join point can then be queried and manipulated by third-party code within an *advice* block. For further details relating to AspectJ’s implementation of pointcuts, advice and join points, see [10, 13].
public aspect SecurityMonitor {

private final SecurityManager initSecMan = System.getSecurityManager();

private final PermissionCollection initPerms = Policy.getPolicy().getPermissions(SecurityMonitor.class.getProtectionDomain());

public SecurityMonitor() {
   // Gather information on the runtime environment

   
   pointcut anyCallOrExec(): call(* *.*(..)) && execution(* *.*(..));

   after(): anyCallOrExec() && !within(SecurityMonitor) { 
      checkSecMan();
      checkForIllegalPerms(thisJoinPoint);
   }

   
   private void checkSecMan() {
      SecurityManager secMan = System.getSecurityManager();
      
      if (secMan == null) {
         // Security manager disabled
      } else if (!secMan.equals(this.initSecMan)) {
         // Security manager altered
      }
   }

   
   private void checkForIllegalPerms(JoinPoint jp) {
      Class<?> clazz = jp.getStaticPart().getSourceLocation().getWithinType();
      PermissionCollection pc = clazz.getProtectionDomain().getPermissions();
      Enumeration<Permission> perms = pc.elements();
      
      while (perms.hasMoreElements()) {
         Permission p = perms.nextElement();
         
         if (!this.initPerms.implies(p)) {
            // Incorrect permission
         }
      }
   }

   }

   
   

Figure 5: AspectJ instrumentation code

by JMD. The XML report also gives a malicious/benign classification based on the behaviour detected by the SecurityMonitor (in this case it is malicious, due to the Java sandbox escape).

4 Evaluation

This section outlines our evaluation of JMD. We specifically focus on determining JMD’s accuracy in detecting malware that targets Java’s access control mechanisms. A discussion on possible evasion strategies is also provided.

4.1 Test Methodology

To test JMD’s accuracy in detecting Java malware we collected a range of malware samples (targeting Java versions 6 and 7) from a variety of publicly-available sources. These sources included: Metasploit modules⁸; the Contagio blog⁹; and other computer security blogs. This resulted in an initial sample set of 228 samples.

Each sample was uploaded to VirusTotal¹⁰ to determine the CVE number of the vulnerability targeted by the exploit. Where multiple VirusTotal scanners returned different results, a manual investigation (e.g. an Internet search for the hash and/or manual inspection of the sample) was used to determine the most probable CVE the sample represented.

From these results, the sample set was curated to remove broken samples and those that targeted out-of-scope elements (as discussed in §3.1—e.g. the class verifier and the Java native layer). The remaining 91 samples (which encompassed fourteen of the JRE access control-related CVEs disclosed over the past four years¹¹) were labelled with the vulnerable JRE version(s)—i.e. JRE 1.6u0 and/or JRE 1.7u0. Java 8 was not evaluated due to its relative immaturity.¹²

By testing JMD against the earliest releases of Java 6 and 7, we ensured that (where our technique was successful) JMD would detect exploits for which patches were released partway through the Java version’s lifecycle. The final list of samples is shown in Table 2.

These 91 samples were used as input to JMD’s instrumentation and dynamic analysis stages (see §4.3 for a discussion of our attempts to use the symbolic execution stage in our evaluation). Dynamic analysis was performed in both a stand-alone Ubuntu 12.04 32-bit sandbox VM and in Cuckoo Sandbox using a Windows XP 32-bit sandbox VM. The results of these analyses were collated and are the topic of discussion in the following section.

¹¹We attempted to quantify the total number of JRE CVEs from this period related to access control vulnerabilities (as opposed to vulnerability classes that are out-of-scope for JMD). However, we could not find public information sources that exhaustively correlated JRE CVEs with their vulnerability classes.

¹²At the time of writing, Java 7 is still offered as the default download at https://www.java.com.

⁸http://www.metasploit.com
⁹http://contagiodump.blogspot.com
¹⁰http://www.virustotal.com
4.2 Experimental Results and Discussion

Tables 3 and 4 show the number and percentage of samples correctly identified as malicious by JMD. As JMD detects exploits within the JVM layer (which is independent of the underlying OS), we observed identical results in both sandbox environments. It is important to note that although Table 2 lists both JREs as being vulnerable to CVE-2012-4681, the samples we had for this CVE were compiled with Java 7 and hence were unable to run in Java 6. Similarly, both JREs are vulnerable to CVE-2013-2465, but the particular sample we had for this CVE (from Metasploit) was only confirmed to work in Java 7.

JMD was able to achieve a 100% success rate for ten of the fourteen CVEs in our sample set across both JREs tested. However, some samples remained incorrectly classified by JMD. We subsequently examined these samples in greater detail.

CVE-2010-0094. The incorrectly classified CVE-2010-0094 sample was unique in our sample set in that it provides the user with an interactive GUI (via a text field and submission button). All of our other samples launch their exploit directly from the applet's init method (without requiring user interaction). Because this sample requires user interaction, JMD was unable to reach the exploitation and payload stages. Symbolic execution was also unable to assist because SPF does not model graphical elements (as discussed in §3.3.1).

Additionally, JMD was unable to instrument a particular method in this undetected CVE-2010-0094 sample. This was because the method was already approaching the maximum allowable method size, and the extra AspectJ instrumentation code put it over the limit—this is discussed further in §4.4.

CVE-2011-3544. This CVE relates to vulnerabilities in the JRE’s JavaScript engine. Half of the CVE-2011-3544 samples were incorrectly classified because they execute both their exploit and payload in the JavaScript engine. This differs from the detected samples, which only execute their exploit (disabling of the security manager) in the JavaScript engine before returning execution to the applet to deliver its payload (which JMD’s instrumentation
Table 4: Detection results for Java 7. This includes the number and percentage of samples successfully detected as malicious by JMD

<table>
<thead>
<tr>
<th>CVE</th>
<th># detected</th>
<th>% detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2012-0507</td>
<td>37</td>
<td>93</td>
</tr>
<tr>
<td>CVE-2013-0422</td>
<td>15</td>
<td>88</td>
</tr>
<tr>
<td>CVE-2013-0431</td>
<td>6</td>
<td>100</td>
</tr>
<tr>
<td>CVE-2011-3544</td>
<td>3</td>
<td>60</td>
</tr>
<tr>
<td>CVE-2012-4681</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>CVE-2012-5076</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>CVE-2012-5088</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>CVE-2013-1488</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>CVE-2013-2423</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>CVE-2013-2460</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>CVE-2013-2465</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

Total: 70 samples (90.91%)

correctly detects). For the SecurityMonitor to detect the incorrectly classified samples, the JRE's JavaScript engine must be instrumented in the same way the sample is—this is beyond JMD’s scope.

CVE-2012-0507. Our initial testing found that JMD could only detect nineteen (48%) of the CVE-2012-0507 samples. After some investigation, we found that this was due to many of the CVE-2012-0507 samples embedding a payload class file inside another class file as a byte array. This byte array undergoes an XOR deobfuscation routine (thus hiding the byte array’s true intent from cursory static analysis) before being passed as an argument to ClassLoader.defineClass. This embedded payload class is typically defined under an unrestricted protection domain (i.e., a Protection Domain with an AllPermission permissions object), thus providing the payload with access to sensitive resources and operations (e.g., write to the file system, open a network connection, etc.).

This privilege escalation was not detected by the SecurityMonitor’s checkForIllegalPerms method (as discussed in §3.4) because the obfuscated inner payload classes were not being weaved with our instrumentation code. In our initial implementation, we were only weaving precompiled bytecode (i.e., .class files); classes that were obfuscated and encoded in byte arrays (or by other means) and loaded at run-time were not instrumented. As a result, JMD was not able to log access control violations that were implemented in these inner payloads.

To rectify this issue, we added AspectJ’s load-time weaving [13] functionality to JMD in order to instrument classes as they are loaded by the JVM’s class loader. This increased our detection rate for CVE-2012-0507 samples from 48% to 93%.

4.3 Symbolic Execution Discussion

Unfortunately, JMD’s symbolic execution stage did not produce the impact we had hoped it would. This was due to a number of factors.

Firstly, the taint-based approach proposed in §3.3.2 focuses on String objects as tainted data. Therefore, we rely heavily on SP’s ability to solve string-based constraints. While much work has been done to incorporate string-based constraints into SP [26], the functionality required for SP to handle many string methods remains unimplemented. For example, we encountered malware that called the toLowerCase, toCharArray and split string methods. We were able to implement symbolic handling for some of these methods (such as toLowerCase) ourselves, but others remain unimplemented due to time constraints. We also frequently encountered malware that converted String objects to arrays (and vice versa). Unfortunately, we found SP’s handling of this conversion and support for arrays relatively immature, resulting in unsolved constraints.

We also found that SP had difficulties with the more heavily-obfuscated samples, especially those that made use of Java’s reflection APIs (e.g., to instantiate classes, execute methods, etc.). SP was often unable to construct constraints in these cases, potentially leaving execution paths unexplored.

Additionally, we found very few samples within our evaluation sample set that attempted to hide their payload behind a trigger condition. While obfuscation to prevent static analysis was common (e.g., randomising method names, field names and strings), hiding malicious activity behind trigger conditions (such as those discussed in §3.3 and [2]) was not. Our sample set only included one sample that hid its behaviour behind a specific applet parameter.

More commonly, applet parameters were used to directly inject metadata (e.g., they contained a URL to connect to, a port to connect on, a specific file to download, etc.) and were not used to construct constraints (i.e., they were not used at branch points to drive execution down a particular code path). For the one sample that did hide its exploit behind an applet parameter, SP was unable to solve its constraints and derive a valid applet parameter value. (In this particular case, the conversion of the applet parameter String value into an array type prevented SP from solving the constraints.)

However, the symbolic execution stage did provide useful information on payload creation and customisation (e.g., how the malware determined the underlying OS, such as the example in Figure 4) and applet parameter names (although in most cases SP was unable to solve the required constraints and derive the parameter values that would enable increased code coverage during dynamic analysis).

4.4 Possible Evasion Strategies

Denial-of-service (DoS) attacks are possible on each of JMD’s stages. The first DoS attack considered here affects JMD’s symbolic execution stage. Path (or state-space) explosion is a common problem for symbolic execution engines [6, 16], and occurs when the application contains a large number of branch points. As the number of branch points grows, the number of possible code paths increases exponentially. Each new code path necessitates an increase in computational resources (both in time and memory) in order to explore all paths and solve the constraints required to execute those paths. It is therefore possible for malware to hide an exploit from JMD’s symbolic execution stage by embedding it within a complicated code path (e.g., consisting of a very high number of branch points) that only triggers under specific conditions. In such a case, symbolic execution becomes computationally infeasible, and the exploit’s trigger conditions will not be derived. Thus, the exploit will not trigger during dynamic analysis and will remain undetected.

A DoS on JMD’s instrumentation stage is also possible, by exploiting the specified size constraints imposed on a Java class file. For example, bytecode crafted with 64 KB methods (the maximum allowable method size [14]) cannot be weaved with additional code by JMD’s instrument at such a stage. As such, any
malicious activity performed by these methods may go undetected during dynamic analysis.

Finally, it is possible for malware to examine itself to determine if it has been instrumented. For example, the malware could calculate a hash of itself at run-time and compare it to a hash calculated at compile-time (and stored in the bytecode). This comparison would return a different result (because the code introduced by AspectJ would alter the malware’s hash), leading the malware to infer that tampering had occurred. Alternatively, the malware could use the Java reflection API to determine whether any AspectJ classes had been woven into the bytecode. After determining if it had been instrumented, the malware could use its reflection API to generate a hash of the code’s cyclomatic complexity (and hence remain undetected during dynamic analysis. (The application of symbolic execution to Java’s reflection APIs could theoretically be used to defeat these evasion strategies.)

5 Related Work

While much work has been undertaken into malware detection in general, relatively little research has examined Java malware specifically.

The Jarhead tool developed by Schlumberger et al. [28] uses static analysis and supervised machine learning techniques to detect malicious Java applets. Features are extracted from the Java bytecode and supplied to a classifier that is able to classify a malicious applet based on a training set of known malicious and benign applets. These features range from code metrics (e.g. the number of instructions and the code’s cyclomatic complexity) to behavioural features (e.g. extending the ClassLoader class and the use of methods that are able to write files). A disadvantage of using a supervised machine learning algorithm is that the training data may be subject to overfitting. This could prevent the detection of zero-day exploits where either the relevant features have not been collected or there are not enough training samples to represent the new exploit technique. Additionally, because feature extraction is performed statically, the malware author may use obfuscation to hide key features (although the presence of obfuscation itself was a key feature in their results).

In [33], Wang proposed a dynamic analysis tool that records calls to the core Java API during execution. To achieve this, the core Java API’s source code is patched to record key method calls (e.g. System.setSecurityManager) and then recompiled. Rules are defined for specific CVEs so an exploit’s API trace can be matched to a rule set and hence a CVE. However, while this detects known exploits it does little in detecting zero-day attacks. Some heuristics for zero-day detection are proposed, however they are not evaluated. Additionally, patching the core Java classes may be against the Java license agreement.

Soman et al. [30] proposed a similar technique, in which security-related operations are logged as events that are supplied to a signature-based intrusion detection system. However, the key differentiator between this technique and Wang’s is that in [30] the JVM is instrumented to log security-related events (as opposed to patching the Java API in [33]). This has the advantage that system calls and calls to native code can be logged and used to detect attacks outside of the JVM. However, [30] still relies on a signature-based mechanism for detecting attacks.

An alternate approach for containing (rather than detecting) malicious Java applets was proposed by Chiueh et al. in [3]. Chiueh et al. proposed Spout, a transparent proxy that attempts to confine the damage of a malicious Java applet to an untrusted, disposable host (usually outside a firewall) that does not store sensitive data. This latter effect may be achieved in two ways. Firstly, it can be argued that the primary assumption in this approach is that “only the application logic can damage the host machine’s system resources” [3]. However, the authors of [8] found that during the period from 2011 to 2013 the 2D and AWT subcomponents were the second and fifth most vulnerable subcomponents in the Java language respectively. This potentially leaves Spout open to exploits that target vulnerabilities in these subcomponents. Symbolic execution for malware analysis has been a popular research area [1, 2]. However, this has typically focused on native x86-based malware; Java malware has received comparatively little attention. While the Java-based SPF has been used for a number of purposes (e.g. test input generation [25, 6]) to our knowledge it has not been used in the context of Java malware analysis and detection. Similarly, AOP techniques have also previously been applied to the security domain (e.g. to replace insecure function calls and log security-relevant data [32]). Once again, to our knowledge AOP has not been used in the context of Java malware detection.

6 Conclusions and Future Work

In this paper we have presented the Java Malware Detector (JMD), a hybrid approach that combines symbolic execution, instrumentation and dynamic analysis to detect malicious Java applications. We demonstrated that it is possible to implement such a system using a number of popular open-source software platforms—specifically Java Pathfinder and AspectJ. Our evaluation on real-world malware samples shows that JMD was able to successfully detect malware variants representing fourteen of the known access control-related CVEs disclosed over the past four years [30] and also shared there are known limitations in JMD’s detection capability, the design and extensibility of JMD’s instrumentation stage provides a platform upon which our results can be built and improved.

The application of symbolic execution to derive trigger points did not yield much success, but this was largely due to the incomplete state of SPF’s symbolic String handling (see §4.3) and other issues which could potentially be remediated by extra engineering effort on SPF.

While JMD is successful in detecting the subversion of Java’s access control mechanisms, there are a number of potential avenues to explore that could further expand JMD’s capabilities. These avenues include: remediating the evasion strategies discussed in §4; adding symbolic handling implementations for the full String API as well as other unimplemented types in SPF; exploring additional trigger points via symbolic analysis, such as date and time triggers and network communication (e.g. receiving commands from a botnet Command and Control server); and extending the dynamic analysis engine to include at-

tacks on the class verifier and Java native layer. Further evaluation of JMD on additional samples could also be performed.

References


[29] Schwartz, E. J., Avgieros, T. and Brumley, D. [2010], ‘All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)’, in


Hypervisor-based Security Architecture to Protect Web Applications

Dilshan Jayarathna  Udaya Tupakula  Vijay Varadharajan

Advanced Cyber Security Research Centre
Macquarie University
Sydney, Australia

Email: {dilshan.jayarathna, udaya.tupakula, vijay.varadharajan}@mq.edu.au

Abstract

Web based applications are very common nowadays where almost every software can be accessible through a web browser in one form or the other. This paper proposes techniques to detect different threats related to web applications by using a hypervisor-based security architecture. The proposed architecture leverages the hypervisor’s visibility of the virtual machines’ runtime state and traffic flows for securing the web application. The unique feature of the proposed architecture is that it is capable of doing fine granular detection of web application attacks, i.e. to the specific web page level, and protecting the application against zero-day attacks.

Keywords: web server, security attacks, hypervisor, virtual machine introspection.

1 Introduction

Web applications are part of everyone’s daily life, where you may be using one to read today’s news with rich content or to do online banking with complex functionalities. Similarity, web applications have become a constant target of many malicious attackers trying to exploit their vulnerabilities to obtain sensitive data to benefit financially or to gain access to systems that can be used for other attacks. According to research from the High-Tech Bridge Security Research Lab (HT Bridge 2014), web application vendors are still taking an average of 11 days to release patches for critical security vulnerabilities while the low risk vulnerabilities take an average of 35 days. However, according to WhiteHat security research (Grossman 2013), it took an average of 193 days to apply the patches leased by the vendor to make sure the reported vulnerabilities are no longer exploitable. In addition to web application threats, a study by Symantec (ISTR 2014) reported that there were 23 zero-day vulnerabilities discovered in 2013, which is 61% increase from 2012. Therefore, it has become increasingly important not only to make the web applications more secure, but also to detect attacks even before the vulnerabilities are identified and remediated.

Many security mechanisms have been developed to protect web applications against potential attacks. Some techniques such as Intrusion detection (IDS) and prevention (IPS) systems and Web Application Firewalls (WAFs) are targeted at network level while other techniques applied on the application itself where inputs are validated before the execution and make use of good coding practices. However such techniques are not effective against zero day attacks and fast restoration of the services.

In this paper, we propose hypervisor based techniques to counteract web server attacks. A hypervisor also known as a virtual machine monitor (VMM) (Rosenblum and Garfinkel 2005, Barham et al. 2003) is a software abstraction layer that enables multiple virtualised operating systems to run concurrently on a single physical computer. An instance of a virtualised operating system along with its applications is referred to as a virtual machine (VM). Use of VMMs in enterprise, government, and consumer applications is becoming increasingly important due to the diverse security requirements and the different levels of trust associated with different applications, systems and devices where VMs can provide virtual isolation between different environments.

While most of the VMMs provide application security from the network and host perspective, they are lacking in control over the contents specific to the critical applications running on VMs. This paper proposes techniques to protect web applications leveraging the hypervisor visibility of the traffic and runtime state of the web application. Since web servers are used for different applications, we interchange between the terms web server and web application in this paper.

This paper is organised as follows. Section 2 describes the related work in web application and virtualisation security. In Section 3 we focus on some specific attacks related to web applications and develop an attacker model and how it fits into VMM. Section 4 describes our proposed security architecture to counteract the security attacks against the web applications. In Section 4, we will also present how our model deals with the attacks that are considered in the attacker model. Section 5 describes our implementation of the security architecture and presents an analysis of the results. Finally, Section 6 concludes the paper.

2 Related Work

In this section, we describe some of the security tools that are currently available to protect web applications. We will also present different research areas related to the proposed architecture, which include new techniques to counteract web application attacks, virtual machine introspection (VMI) and domain name system (DNS) validation.

IDS and IPS protect web applications from known
attacks. A traditional IDS such as Snort (Roesch 1999) examines packets as they enter a network and uses various pieces of information from these packets to determine if a potential threat exists. Generally, this process involves pattern-matching the packets against a known database of threat patterns. Although this functionality is great, it is usually not enough to properly protect a Web server running complex web applications.

WAFs are designed to prevent web applications from attacks that common network firewalls and intrusion detection systems cannot detect. WAF can be configured to drop requests that appear to be HTTP but do not conform to HTTP standards such as RFC 2616 and 1945 or to limit the size of the HTTP body and headers in request. Many WAF products exist in the market today (OWASP WAP 2014) due to increased demand in protecting critical web applications. However such tools are not efficient in zero day attack detection, fine granular isolation of the vulnerable service and fast restoration of the services.

A number of researches published on anomaly-based detection of web attacks. Robertson et al. (2006) proposed an anomaly generalisation technique that automatically translates suspicious web requests into anomaly signatures. These signatures are then used to group recurrent or similar anomalous requests so that an administrator can easily deal with a large number of similar alerts. Sarrouy et al. (2009) approached it differently by examining the data which are sensitive to intrusions and controlling the constraints apply to them by comparing the data flow. Corona and Giacinto (2010) described an approach to detect server-side web attacks by applying a pattern recognition system to detect intrusion attempts on web servers. In another research study, Vigna et al. (2003) proposed a stateful IDS for web servers to perform early detection of malicious activity and possibly prevent more serious damage to the protected site. Ma et al. (2011) proposed a technique to detect malicious websites by applying online learning algorithm to analyse lexical and host-based features of their URLs.

There are other approaches to protect web sites from attacks such as injection and XSS attacks. Robertson and Vigna (2009) proposed a strongly-typed web development framework to build robust web applications against XSS and SQL injection attacks where user inputs are passed through specific sanitisation routines to ensure the integrity of the web documents and SQL queries. Samuel et al. (2011) built a reliable context-sensitive auto-sanitisation engine into web template systems based on type qualifiers to provide accurate context-sensitive sanitisation. Another approach to the taint analysis (Nguyen-tuong et al. 2005, Haldar et al 2005) by using the variables that have been ‘tainted’ with user controllable input to identify insecure information flows where user inputs are propagated and flow into possible vulnerable functions within web applications. Skrupsy et al. (2013) proposed TamperProof, which is deployed in a trusted environment between client and server and intercepts all communication between them to determine the validation that should to be performed on the server for any given HTTP request. However, this method only protects web applications from parameter tampering attacks. All these approaches attempt to mitigate these attacks by sanitising inputs, but none provides a way of identifying an attack that has not been recognised by the sanitisation process. Our method aims to detect a successful injection attack that are not detected by the existing sanitisation techniques by analysing the attack payload placed in HTTP response. As described above, most of the solutions apply protection mechanisms such as WAFs, IDS/IPS and pattern recognition before requests reaching the web application while the others look for anomalies within the web server to detect attacks. All approaches are relevant in the context of incoming attacks and compromise attempts within the web server. In our approach we examine both incoming and outgoing content to detect possible attacks and to verify that the attack actually compromised the application or the server. In addition, we also verify the integrity of the web server by validating associated processes. Our aim is to detect possible threats, which are previously unknown such as zero-day attacks, by applying aggressive detection mechanisms and examining the outcome of the attack. This allows us to deny access to the vulnerable application or the webpage and to build protection mechanism against subsequent attacks. However note that our model can easily block all the known malicious requests before forwarding them to the web server. For example, we can use attack patterns from the tools as Snort (Roesch 1999) or Bro (Paxson 1999) to block the malicious requests before forwarding them to the web server. However none of these tools are efficient against the zero-day attacks as the attack detection depends on the known attacks patterns. Hence this is the main focus of our work.

Several techniques were proposed to address web server or application vulnerabilities. Most recently, Chen et al (2014) proposed SafeStack to automatically patch stack-based buffer overflow vulnerabilities by virtualising memory accesses and moving the vulnerable buffer into protected memory regions. While the proposed method is not limited to web servers or applications, it can only address the stack-based buffer overflow attacks. In contrast, our model is capable of identifying any buffer overflow attacks by combining multiple techniques such as application and server availability monitoring, and process validation.

Yang et al. (2014) proposed new technique that can provide collection of attack information using virtualisation technology and stable web service to cope with external attacks. It uses a honeynet system largely composed of a number of dynamically created virtualised honey VMs and virtual server system. However, this system only works for known attacks such as TCP-SYN flooding and ARP spoofing attacks, and does not actively prevent any attacks.

A remote backup and recovery method was proposed by He et al (2013), which focused on web site protection and disaster recovery. It uses a mechanism based on Rsync and FTP protocols to automatically backup web servers and restore them after an adversary attacks a server successfully. However, it is only effective when the attack actually tampered web server files and does not prevent recurrence of the same attacks. In comparison, our model detects a range of attacks and uses VM snapshot to restore web server and blocking the identified attacks in a matter of seconds.

Most of the web applications hosting sensitive data are secured by Secure Socket Layer (SSL) to ensure the confidentiality of the information. However, it is becoming very common most of the websites that run on virtual environments terminate their SSL before the VM hosting the web server, typically on the load balancer, to gain more performance by decreasing the need to encrypt and decrypt traffic at the web server (Windom Sr. 2013) assuming the trust of the virtual environment. Therefore, inspecting the web server
traffic just before the VM is not difficult regardless of the web application is using SSL or not.

The method of inspecting a virtual machine from the hypervisor level for the purpose of analysing the software running inside it is called virtual machine introspection (VMI). VMI based architecture for intrusion detection system (IDS) leverages three properties of VMMs, namely isolation, inspection and interposition. VMM enables a virtual machine to be segregated in a way that it cannot access or modify the software running in a separate VM or in the VMM itself. This ensures that the IDS cannot be tampered even if the monitored host is completely subverted. The VMM provides the ability to directly inspect the hardware state of the virtual machine that a monitored host is running on. The VMM also provides the ability to interpose at the architecture interface of the monitored host, which offers even better visibility than standard OS-level mechanisms by enabling monitoring of both hardware and software level events. A good example of VMI IDS is the Livewire prototype.

A Formal model for VMI was described by Ploh et al. (2009). Another VMI model was described by Payne et al. (2007) have implemented XenAccess, a monitoring library for operating systems running on Xen, which provides virtual memory introspection and virtual disk monitoring capabilities without any modifications to the VMM or to VM being monitored. XenAccess was further developed as LibVMI, which is also compatible with Kernel-based Virtual Machine (KVM). The most recent work by Fu and Lin (2012) described a technique that can automatically generate the VMI tools by monitoring system wide instructions monitoring, which identify the introspection related data and redirect these data accesses to the in-guest kernel memory. Our architecture makes use of the LibVMI for detecting compromise of the web server by analysing the processes running on it.

Domain Name System (DNS) is a system for naming computers and network services that is organized into a hierarchy of domains. DNS is an essential component of the functionality web application of the Internet, where the primary purpose of the DNS is to translate easily memorised/readable Internet or private network domain and host names to IP addresses. DNS services play a vital role when publishing web applications using friendly/readable names as users always remember the website name access the required application. There is previous work such as Seifert et al. (2008) looked at the relationships between DNS and web server to identify malicious web pages. In this paper, we make use of an existing techniques described in Jayarathna et al. (2014) to validate any DNS anomalies of domain names that are included in HTTP requests/responses. This is an additional protection mechanism we use to secure the web application from different kind of threats such as DNS spoofing or cache poisoning attacks.

3 Attacker Model

There are a number of well-known threats against web applications. In this section we consider those attacks that can be addressed by the proposed security architecture to formulate an attacker model. In the first case, we consider unvalidated redirects or forwards. Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorised pages.

In the second case we consider injection attacks against the web server. Attacker injects untrusted data to an interpreter in order to execute unintended syntax/commands or accessing unauthorised data. Commonly used injection attacks are based on SQL queries and OS commands. It is one of the most common web attack mechanisms used by hackers to steal data, which was categorised as the top most dangerous web vulnerability by OWASP in 2013 and 2010, and made the top ten in 2004 and 2007 (OWASP Top 10 2014). In this paper we consider inband injection attacks where data is extracted using the same channel that is used to inject the SQL code or the script. This is the most straightforward attack, in which the retrieved data is presented directly on the application web page. For example, the web application may be vulnerable to certain injection attacks. Known attacks can be detected using signature based tools such as snort. However the attacker can easily alter the inputs to evade by the signature based detection tools. Hence there is need to secure the web applications from zero day attacks.

In the third case we consider buffer overflow attacks against the web server. For example, an adversary could exploit a web application coding bug or operating systems bug to mount a buffer overflow attack and gain control over the server. A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. Sending carefully crafted input to a web application, an attacker can cause the web application to overflow the allocated memory space and execute arbitrary code, possibly taking over the machine. Usually, buffer overflows lead to crashes and consequently causing denial of service.
These types of attacks are becoming increasingly difficult since major web server developers have heightened their security over the years. However, the addition of richer web application functionalities introduces the increased risk of untested code to be vulnerable to such attacks. Primary cause of such vulnerabilities is due to delay in applying patches on not only web applications but also the underlying operating system. In the following section we will describe how our model can deal with such attacks.

## 4 Protecting Web Application Attacks

Now we propose techniques to identify attacks that are considered in the previous section. First we present a high level overview of our model and then describe the architecture components in detail.

### 4.1 Architecture

Generally, in a standalone computing environment, the operating system has the full control over the execution of all applications and works as an interface between the applications and the hardware. In a virtual environment, the VMM controls the access to physical resources while each guest operating system manages its own applications. Our architecture makes use of the VMM capabilities to ensure secure operation of the web server.

![Figure 2: Attacker model mapped into a virtual environment](image)

Figure 2 shows how we can apply the attack model into a virtualised environment in order to identify web application attacks. For simple presentation, we consider that the web server (VM1) and one of the web application clients (VM3) reside on the same VMM. However note that operation remains same when the web application client (e.g. VM3) in different VMM similar to client 2. The only difference is between two models is that the virtualised environment will have its own LAN for VMs and internal clients will be on a different LAN segment. Therefore anything outside the VM LAN can be considered as external.

The proposed architecture leverages the hypervisor level visibility to monitor traffic flows in and out of the virtual machines. As shown in Figure 3, our architecture consists of three logical components, which are strategically placed in Dom0 for detection of attacks at fine granular level and fast restoration of the attacked services.

![Figure 3: Architecture overview and operation](image)

4.2 Logical components

In this section we will describe the important logical components of our architecture.

#### 4.2.1 Request/Response Capture Mechanism (RRCM)

RRCM consists of IDS component, which inspects all traffic flows using to and from virtual machines by simply sniffing the spanned virtual interface, which is mirroring the traffic of all the virtual bridge interfaces of the hypervisor. This reduces any performance impact of applying IDS alert rules directly into the active bridge interface. The second component of the RRCM is the Logger, which records content of the client request along with the time stamp, source IP address, domain name and the accessed web page in relation to the hosted web application for all alerts picked up by the IDS. Logger also captures the content of the corresponding responses, which are sent to the validation component for verification. We discard these captures after 24 hours to reduce the disk utilisation. But captures can be archived for longer period time if further analysis is required. Collected information is used for exploit diagnosis and deny further access to similar attacks.

#### 4.2.2 Validation and Attack Detection (VAD)

VAD can detect and categorise the attacks by validating the runtime state and communication of the web application. Let us consider how VAD can detect the attacks on the web application.

VAD matches requests and corresponding responses sent by RRCM, and checks the content of captures to verify that the suspected attacks actually compromised the web application. If the content
of the responses contains the proof of a compromise, the VAD flags the corresponding page as vulnerable and sends the information to PVMDRM (Protection and VM Data Restore Mechanism) component of the architecture. If VAD detected that the exploitation made a permanent change to the content website serving by inspecting the subsequent responses, VAD will also sends a restore request so that the changes done by the malicious requests are rolled back.

VAD also monitors the runtime state of the virtual web server and detect the compromise of the web servers. If the attacker exploits vulnerabilities in the web application or the operating systems in the virtual machine and install rootkits, such attacks are also detected by the VAD component. At regular intervals VAD checks what processes are running on the monitored web server. VAD keeps a track of processes running on the web server and uses this as reference to monitor the web server during runtime. Recorded process details are User ID (UID), Process ID (PID) and process running command (CMD). VAD component make use of the VMI interface to directly extract the runtime information from the monitored web server. If there is any variation from the default process list then the web server is considered to be compromised. Note that it is not an easy task for the attackers to alter the process list obtained by the VAD since it is running in the VMM. Once a compromise is detected, VAD sends a restore request to PVMDRM with the time stamp of initial connection of the attack obtained from the access logs recorded by the RRCM to perform a complete restore of the server.

VAD keeps a track of the usage of critical processes to identify possible buffer overflow attacks by analysing the process/memory utilisation trends and then map the time to the access logs recorded by the packet captures. A dynamic instrumentation tool Luk et al. (2005) is used to monitor all the program instructions to check their memory accesses.

VAD also checks the heartbeat of the server and the availability of published applications to identify a possible compromise and crash of the web applications.

4.2.3 Protection and VM Data Restore Mechanism (PVMDRM)
The PVMDRM takes a snapshots of the VM (without memory) at five-minute intervals and delete them automatically when the age of the snap shot is greater than 60 minutes except the each snapshot taken at first on every hour. It also deletes hourly snapshots after 24 hours keeping the first snapshot of the day for 30 days. Therefore the maximum number of snapshots it may keep at a given time is 64 (12/hour + 23 hours + 29 days). In this paper we mainly focus on the access logs recorded by the packet captures. A dynamic instrumentation tool Luk et al. (2005) is used to monitor all the program instructions to check their memory accesses.

VAD also checks the heartbeat of the server and the availability of published applications to identify a possible compromise and crash of the web applications.

If the information received from VAD indicates that application is vulnerable to detected attacks, protection mechanism applies fine granular deny rules directly into the VMMs bridge interface to block the vulnerable pages. The vulnerable parts of the applications have to be patched before making them available to the users. As already stated in Section 1, it takes considerable time before the patches are released by the vendors. Hence our model is able to prevent access to the vulnerable part of the application at fine granular level and make the unaffected services available to its users.

PVMDRM also has a collection of whitelisted URLs which web server or application is configured to redirect or forward requests. An IPS rule checks for all redirects and drops connections to any URL not whitelisted.

Any of the above actions triggers an alert to an external monitoring agent, which can be a third party security operations centre or a web application developer, who can verify the alert and take remedial action to rectify the problem. Once the problems are rectified, the drop rules can be removed to have the page available again. In case of a false-positive identification of a legitimate redirect is being blocked but not in the predefined list, it can be whitelisted to remove the block automatically.

4.3 Operation
Let us discuss a simple walk through of the operation of our model by considering the architecture overview diagram in Figure 3. Steps on the following Figure 4 should be read in reference to the numbered arrows on the Figure 3.

1. Client connects to the web server by opening up a web page
2. RRKM Logger captures HTTP traffic and records the time stamp of the initial request along with source IP address, domain name and accessed URL
3. IDS running in RRKM monitors HTTP content on network bridge using a spanned virtual interface and detects an injection attempt, which sends alert details to VAD
4. Web server responds to the request with requested page
5. RRKM Logger continues to record all HTTP requests and responses
6. Once the related response captured, RRKM triggers VAD to validate the responses related to the possible attack
7. VAD verifies the web server integrity by process validation and server/application monitoring (this is a continuous task, which doesn't depend on the previous steps)
8. VAD passes the identified vulnerability and/or compromise information to PVMDRM
9. PVMDRM
   a. Blocks the access to vulnerable page identified by VAD
   b. Verifies all the redirects against the list of redirects web application configured to do and blocks any unauthorized redirect requests (continuous task)
   c. Restores entire VM
10. PVMDRM alerts the external monitoring agent/administrator about the vulnerability, compromise, VM restore or URL block information.

Figure 4: Architecture in operation

4.4 Attack Scenarios
Now we consider the attack scenarios described in the attacker model.
In the first scenario, the web application redirects the client to an external URL. PVMDRM checks the redirected URL against the whitelisted URLs it has on record and identifies that the redirected URL is not whitelisted which is resulted in blocking the request going to the redirected URL. For example, the main website of a University is set up in a virtual environment and have redirects configured to go to its enrolment application or online learning applications hosted in a different server possibly hosted by a third party using an external service provider. When we have the proposed architecture applied to this virtual environment, IPS component verifies that the redirect to other application is whitelisted and allows the client to receive the redirect request. However, it blocks the URL redirect request that it identifies as not whitelisted and notifies the website developer via email. This URL may be for a phishing site or a social networking site. In the case of phishing site is being blocked, the architecture protects the user from going to a phishing site even without the knowledge of it is malicious. For the latter, a developer has added the redirect link of the University’s social media site page upon a request from the marketing department, but forgot to update the whitelist. Simply updating the whitelist allows the redirect.

In the second scenario, a particular web page was identified as vulnerable to an injection attack and blocked. For example, an attacker sends an inband injection attack against the simple web-based DNS lookup application hosted on a server protected by the proposed technique. Analysis of response messages identified that the response has the entire listing of the root file systems of the server. By matching the logged access requests, it identified and blocked the URL of the vulnerable web application by adding new IPS rule. Once the problems are addressed and verified, the IDS rule is removed to allow the application accessible again. Hence our model can raise alerts for the attacks that are not identified earlier.

In the third scenario, a web application is vulnerable to a buffer overflow attack, which compromises the server by escalating privileges to gain root access to the server. This attack is identified, the VM is restored and the vulnerable application is blocked. For example, an account firm’s website protected by the proxy architecture has an online application that can be used for calculating tax. An adversary mounted a buffer overflow attack against this application. The capture mechanism logged the connection to application, but IDS did not detect any anomalies. However, the VAD component identified that the calculator application is not responsive for a short period of time and the process validation detected that the calculator process was running as root, which was not in the original web server configuration. This generated a positive compromise alert and triggered a VM restore. The logs are analysed to determine the time the attack traffic was received. The VM is restored with the snapshot of known good state that was taken before receiving the attack traffic. Simultaneously, protection mechanism in PVMDRM applied deny rules to block further accessing to the calculator application until the problem is rectified and notified. This attack may have not been previously identified by any protection system due to the stealth nature of its behaviour.

Techniques applied in scenarios two and three can be categorised as capable of being detecting zero-day attacks. In all scenarios, the block can be temporary or permanent depending on the security level set.

5 Implementation and Analysis

We have implemented our model using Xen VMM and web server running as virtual machines on the VMM. Xen VMM started off as a research project at the University of Cambridge and was first introduced by Barham et al. (2003) as a high-performance resource-managed virtual machine monitor, which enables applications such as distributed web services, server consolidation and secure computing platforms. Xen is a native (or hypervisor-based) VMM where it runs directly on the hardware as lowest and most privileged layer. In Xen terminology domain” refer to a running virtual machine within which a guest OS executes. domain 0” (Dom0) boots with the hypervisor and works as the control interface with special management privileges which has direct access to underlying physical hardware. All other virtual machines are called domain U” (domU) in Xen terminology. Initially, on x86 architecture, Xen kernel code runs in Ring 0, while the hosted domains run in Ring 1 or Ring 3. Running operating system in Ring 1 or 3 instead of usual Ring 0 required operating system to be modified in order to suit the new privilege levels. This means only paravirtualized (i.e. modified operating systems) guests were able run on Xen. But Xen version 3.0 and above can use unmodified guest operating systems (e.g. Microsoft Windows XP) for hardware-assisted virtual machines (HVM) with supporting underlying hardware (e.g. Intel VT and AMD Pacifica).

5.1 Design Choices and Implementation

Components RRCM and VMDRM are placed in Dom0 of the hypervisor as RRCM needs to be in the data path of the bridge interface using the VMMs backend driver domain and PVMDRM require restoration and prevention abilities which can trigger commands available within the hypervisor. VAD component which is placed inside the Dom0 is used for process validation using LibVMI, and attack detection by validating the request and response messages.

RRCM captures all packets related to web application communication between DomUs, the Dom0 and the physical network interface. We achieve this by spanning the Dom0’s bridge interface (xenbr0), which is essentially the virtual switch of the hypervisor, to a separate virtual interface of Dom0 running in promiscuous mode to get an exact copy of the traffic flows in and out of the virtual machines.

We have analysed the default processes for Apache web server and MySQL running on a Linux server with Ubuntu 3.5.0-23 kernel. There are total of 63 processes in the clean state installation of Apache, PHP and MySQL on Linux. Figure 5 shows partial list of the processes running in the web server virtual machine and the highlighted process apache2 runs under root UID, which is located in specified path (/usr/sbin/apache2) is responsible for handling the client requests. We excluded the number of Apache child process running under wwww-data UID into the total count as primary process is responsible for launching child processes which listen to new connections and serve them when they arrive. Therefore the number of these child processes can change depending on the number of client connections. However, we keep a track of their Parent Process ID (PPID) to ensure that they are spawned from the same primary process. Figure 6 shows the partial list of processes of the web server and their corresponding address location obtained by the VAD component.
We have used Snort as the IDS/IPS components and SNMP to monitor the server heartbeat and web application availability. Pin tool by Luk et al. (2005) was used as the dynamic instrumentation tool to monitor all the program instructions to check their memory accesses. We used Metasploit to generate a number of injections attacks to verify the detection of known attacks.

5.2 Algorithms

This section describes algorithms used for various procedures of the three key components of the architecture.

5.2.1 RRCM Procedures

Algorithm 1 depicts the requests and responses capture process. It takes the packets from the spanned virtual interface and log access request information while detecting various attacks.

Algorithm 1 capRR (request/response capture in RRCM)

Input: spanned virtual interface packets
Output: None
1. webserver = web_server_IP
2. src_IP_list = empty list
3. alert_list = empty list
4. for all packets do
5. if destination = webserver then
6. add src_IP to src_IP_list
7. log src_IP, rtime, page
8. for all IDS_rules do
9. if IDS_rule_match = true then
10. log src_IP, domain_name, rtime, page, alert_type
11. add src_IP to alert_list
12. req_cap = capture upto next 1514 bytes
13. end if
14. end for
15. for all src_IP in src_IP_list do
16. if src_IP in alert_list then
17. res_cap = capture upto next 1514 bytes
18. call valReq (src_IP, req_cap, rtim, res_cap, page, alert_type)
19. end if
20. end for
21. end if
22. end for

5.2.2 VAD Procedures

Algorithm 2 shows high-level steps of validating response in relation to the attack source, content of the request, timestamp of the initial request, accessed page and detected attack type provided as inputs by inspecting the content of the response.

Algorithm 2 valReq (response validation in VAD)

Input: spanned virtual interface packets
Output: src_IP, req_cap, rtime, res_cap, page, alert_type
1. if res_cap has content matching the alert_type then
2. if page content changed then
3. call restoreVM (rtime)
4. end if
5. call blockPage (src_IP, page)
6. end if

Algorithm 3 explains the process validation procedure where it obtains a list running processes using VMI and compares against the previously recorded list of processes.
Algorithm 3 valProc (process validation in VAD)
Input: None
Output: None
1: vmi.plist = process list obtained using VMI
2: vm.plist = process list obtained from clean VM
3: for every x minutes do
4: vmi.plist = current process list obtained using VMI
5: if vmi.plist doesn’t match with vm.plist then
6: page = getPage (dt ime) // page retrieval from logs
7: call blockPage (page)
8: call restoreVM (rtime)
9: end if
10: end for

Algorithm 4 describes the process of the performing web server and application health checks, which also responsible of sending alerts.

Algorithm 4 checkHealth (application and server health check function)
Input: None
Output: None
1: for every x minutes do
2: poll webserver
3: if webserver is unresponsive then
4: record unresponsive time
5: end if
6: poll application y
7: if y is unresponsive then
8: record unresponsive time against y
9: end if
10: if unresponsive time > set threshold then
11: alertMsg(null, null, 5)
12: end if
13: end for

5.2.3 PVMDRM Procedures
Algorithm 5 shows the steps of identifying and restoring the correct VM snapshot closest before the attack time taking the initial request time of the attack as the input. It also alerts the external recipient of the action it has taken.

Algorithm 5 restoreVM (VM data restore function in PVMDRM)
Input: rtime
Output: None
1: restoreSnapT = snapshot_t [0]; // first snapshot time
2: for all all snapshots snapshot_t[x] before rtime do
3: if restoreSnapT before rtime then
4: restoreSnapT = snapshot_t[x]
5: increment x by one
6: end if
7: end for
8: if restoreSnapT after rtime then
9: alertMsg(null, rtime, 4)
10: end if
11: call restore (restoreSnapT); // hypervisor function
12: alertMsg(null, restoreSnapT, 3)

Algorithm 6 blockPage (page block in PVMDRM)
Input: page
Output: None
1: apply IPS rule to block page
2: alertMsg(page, null, 1)

Algorithm 7 below used for checking redirects against a list of whitelisted URLs, which calls the Algorithm 7 to block any unidentified URL redirects and send relevant alert.

Algorithm 7 redirectCheck (URL redirect check in PVMDRM)
Input: None
Output: None
1: RDwhiteList = list of URLs whitelisted to redirect
2: dURL = detected redirect URL
3: block = true
4: for each URL in RDwhiteList do
5: if vURL = dURL then
6: block = false
7: end if
8: end for
9: if block = true then
10: apply IPS rule to block dURL
11: alertMsg(dURL, null, 2)
12: end if

Algorithm 8 takes the inputs blocked page, restore time and alert type to formulate a correct message relevant to the action taken at the function calling it and sends alerts.

Algorithm 8 alertMsg (alert function in PVMDRM)
Input: page, restoreSnapT, type
Output: None
1: alertRecList = all alert recipients
2: for each recipient in alertRecList do
3: switch (type)
4: case 1:
5: send "page has been blocked" recipient
6: case 2:
7: send Unauthorised redirect to page has been blocked recipient
8: case 3:
9: send VM was restored to restoreSnapT to recipient
10: case 4:
11: send No snapshot found before rtime to recipient
12: default:
13: send server/application is unresponsive
14: end switch
15: end for

5.3 Example Scenario
This section describes the second attack scenario described in the Section 4.4 with detection steps as a practical example. The hostname mywebserver on following four figures is a host entry on the client machine pointed to a web server with IP address 192.168.61.17 running on a virtual infrastructure described at the beginning of Section 4.

First, let us consider legitimate operation for this scenario. Figure 7 below shows a simple DNS lookup application running on the web server using PHP as the scripting language. Figure 8 shows the intended output for DNS resolution of example.com.
Now let us consider the same scenario during the attack. In this case the attacker is able to inject malicious input to the server to retrieve sensitive information stored in the server. Figure 9 shows the inband injection attempt by adding "&cat /etc/passwd" command after the DNS name to read the content of the /etc/passwd file.

Figure 9: Inband injection to view /etc/passwd file

Figure 10 below shows the alert related to the detected password file retrieval attempt with alert time stamp. As highlighted on the figure, alert contains the timestamp of the incoming HTTP request on destination port 80 from the client IP address (192.168.61.2) to the web server IP address (192.168.61.17).

Figure 10: Alert of the password file retrieval attempt

Packet capture related to the identified attack is shown in Figure 11, which includes the accessed page (highlighted), vulnerable field and malicious input (at the bottom of the figure). It also has the same timestamp (second line highlighted) as the alert timestamp in Figure 10 above.

Figure 11: Request capture associated with alert

The unvalidated input shown in Figure 9 caused the web server to extract entire password file and display it on the client browser. Figure 12 shows a trimmed version of the output.

Figure 12: Output of injected command

Validating the response to the detected attack has the content of a password file, the system identified the page is vulnerable to that attack and blocked any future access to the identified URL (i.e. http://mywebserver/nslookup.html) containing the vulnerable web page by adding a new drop rule.

6 Conclusion

In this paper we have analysed different types of attacks on the web server and proposed techniques to deal with these attacks. We have shown that the proposed architecture can identify zero-day attacks by aggressively detecting attacks and analysing responses from the web application. This mechanism can be used to effectively safeguard web applications from such attacks and deny accessing the vulnerable page until a resolution or patch has been implemented. In the future work we will develop techniques for securing the web applications with dynamic web pages.

References


Annex: A Middleware for Constructing High-Assurance Software Systems

Tristan Newby\textsuperscript{1} \quad Duncan A. Grove\textsuperscript{1} \quad Alex P. Murray\textsuperscript{1} \quad Chris A. Owen\textsuperscript{1} \\
Jim McCarthy\textsuperscript{1} \\

\textsuperscript{1} Defence Science and Technology Organisation \hfill Edinburgh, South Australia 5111, \hfill Email: tristan.newby@dsto.defence.gov.au

Abstract

Cross Domain Solutions and Multi-Level Secure systems are becoming more popular as the benefits of merging data from different security levels becomes more widely understood. Software forming the Trusted Computing Base of such systems must maintain isolation between data from differing security levels while providing some means of bridging that isolation under strictly supervised conditions. We cannot expect to be able to build such trustworthy software using contemporary software development tools and techniques.

We describe the Annex Object Capability System, a tiny, security-focused software development framework and middleware for implementing high assurance application software on top of existing highly certified COTS \textmu{}kernels. By leveraging existing operating system provided process space isolation, we are able to provide the programmer with a simple, familiar environment for building complex, yet truly secure software.

1 Introduction

Historically, the mainstream computer industry has dealt with security problems via the path of least resistance, wallpapering over cracks as they appear. While this has sufficed until now, there is increasing evidence that computer security is in need of structural reform.

One prevailing change is a shift from open world to closed world security policies. For example, most corporate firewalls now block all traffic by default, using overrides to allow white-listed communication. Unfortunately implementing these types of policies on top of standard computer systems is fragile since these systems typically permit any actions as long as they are not explicitly disallowed. Trying to achieve closed world semantics using exclusion based mechanisms is difficult to get right, especially in a highly dynamic environment. Striking a good balance between under-constraint and over-constraint remains a difficult task, but there are mechanisms that can be used to manage this problem.

One approach gaining favour is the use of sandboxing to isolate programs, or even parts of programs, from each other so that a malfunction or compromise in one does not affect any others.

A more powerful extension is to combine a component based methodology with strong isolation and then use capabilities for controlled communication between otherwise isolated components. With capabilities, programmers are freed from having to protect against anything that could possibly happen and can instead focus on what a program must do to fulfil its purpose. Rather than needing to prevent operations after the ability to perform them has already been conceded, capability checks guarantee that explicit authorisation has been granted before any action is performed. With capabilities, these checks happen automatically because the authority to perform an action is inextricably bound with the mechanism for performing it.

One area where such trustworthy software is required is in Cross Domain Solutions (CDSs). CDSs can replace replicated computers and networking at multiple security domains with a single set of equipment that is able to process data at multiple sensitivities. At the core of such systems is a Trusted Computing Base (TCB) which must maintain the boundaries between the different domains. It may also provide some approved, audited, path for selected data to cross those security boundaries. The software that is used to build such a TCB needs to be trustworthy and amenable to formal proofs of its security correctness.

In this paper, we describe the design of the Annex Object Capability System, a middleware layer designed to be deployed on COTS secure \textmu{}kernels based operating systems, utilising their strong guarantees of isolation between components and message passing primitives. We envisage the primary deployment scenario for Annex systems as forming a TCB for larger systems such as CDS, however it is powerful and flexible enough to construct a workstation replacement.

Section 2 of this paper describes the high level Annex Object Model. Section 3 dissects the system’s components including the Object Capability Reference Monitor and elements that make up a standard object. A prototype implementation is discussed in Section 4 and Section 5 shows how all this interacts to produce real-world systems and devices. Section 6 examines the system’s security properties and performance characteristics while Section 7 places Annex in the context of related work.

2 Annex Object-Capability Model

The Annex Object-Capability Model is an object-capability system (Miller & Shapiro 2003), which combines capabilities and object oriented programming. At the most abstract level this equates to conceptual “circle and stick” diagrams like those shown in Figure 1. Nodes represent \textit{objects} and the directed graph...
of connected edges represent the capabilities that objects hold to other objects.

An object-capability system manages the complexity of capabilities by naturally decomposing complex systems into their component objects. Using capabilities to address each of the objects allows the construction of fine-grained access policies expressed through the programming model rather than imposed as an afterthought to system design.

An object-capability model provides a sound and intuitive interface for programmers that retained the strong security underpinnings of the µkernel rather than attempting to build the isolation mechanisms ourselves from scratch. The focus was on developing a useful, intuitive interface for programmers that retained the strong security underpinnings of the µkernel. For example, the Annex programming model explicitly disallows dynamic memory allocation. This removes several classes of bugs from occurring, and ensures that Annex systems extend conformance to the Separation Kernel Protection Profile (SKPP) (National Information Assurance Partnership 2007) into the programmer space.

3 Annex Components

An Annex system is comprised of several components working together to implement the model described in Section 2.
3.1 The Object Capability Reference Monitor

Annex’s Object Capability Reference Monitor, OCRM or simply monitor, provides an abstraction of the underlying operating system and acts as a reference monitor (Anderson & Co. 1972) for the Annex system. It is designed to be verifiable, hence it is small, and performs minimal functions. Primarily, it mediates delivery of messages between objects.

The monitor provides restricted versions of the scheduling and interprocess communications mechanisms that are provided by the underlying operating system and is built around the following major data structures and components: the object table, capability, catalogue, clist, message, message queue, promise, requests table and scheduler.

The object table lists all objects that exist on a given device. It allows the monitor to keep track of information about the objects including their status and how many method calls are outstanding for each object. When an object is created, its process space is created and initialised, an entry is made in the object table and a master capability is created with permissions for all methods exported by the object. This capability is returned to the object’s creator.

Figure 2: Capability structure

Capabilities are represented within the Annex monitor as shown in Figure 2. The DeviceID and NetworkID taken together form the IPv6 address of the host system. The ObjectID uniquely identifies the specific object on that device. There may be more than one capability to a given object on a particular host, so the CapabilityID is used to uniquely identify each capability to a particular object. The Password field is a large random number that makes forging valid capabilities computationally infeasible.

All capabilities that reference objects on the local device are stored in the monitor’s catalogue. The catalogue provides services for capability creation, derivation, validation and destruction, and storage of the authority associated with each capability in the form of a permission bit-vector.

Each object accesses capabilities indirectly via an index (or handle) into its clist, and does not have access to raw capabilities themselves. This prevents the object from directly reading or writing a capability’s fields, preventing the object from bypassing the controlled mechanisms for capability presentation or transfer by forging or altering a capability. This is also necessary to prevent violation of MLS policies as detailed by Boebert (Boebert 1984). The indirection involved for capability access using clists is illustrated in Figure 3.

The Annex monitor creates a pseudo-object, which allows objects access to their clist. If an object has a capability to its clist, it may be able to manipulate aspects of the list of capabilities that it has. For example, it may be able to change the “scope” (see Section 3.8) of a capability that it holds. Clists are a facet of the catalogue in the context of each individual object.

All communication between objects within the Annex system occurs by message-passing. Messages are atomic, meaning that they contain all information required to initiate the method call on the destination object including all parameters required by the call.

All messages between objects are passed via the monitor. The monitor uses a message queue to store messages that it has not yet passed on to the destination objects.

As Annex is a distributed system, it provides a means of making calls on objects, then continuing to do useful work as that method call is processed. When the method call is made, the monitor returns a token, known as a promise to the calling object. When the caller wishes to claim the result of the method call, it presents the promise back to the monitor, which notifies that caller when the results are able to be claimed.

In order to match method replies with their initiating calls, the monitor keeps track of them in its requests table.

Delivery of messages is handled by the scheduler. The Annex monitor is event-driven. Events are either an “interrupt” from a hardware component or a message arriving from an object. The monitor manages message events using a single message queue and deals with each message in turn. The monitor is single threaded, making it easier to ensure the correctness of both design and implementation.

The Annex monitor delivers as many of the messages that it has outstanding in its message queue as it can before accepting any new messages from objects, or processing hardware events (“interrupts”). Although the word “interrupt” is used, these events do not cause the Annex monitor to be interrupted or preempted. They are treated as data sources that have data ready to be processed by some registered object, and as such are scheduled in exactly the same manner as other events.

The object table provides subjects, the clist and catalogue together denote the sum total of the authority in the system, while the message queue and requests table enable the monitor to track the (flow of) content or data in the system.

3.2 Annex Objects

Objects are isolated units comprising both code and state. Each object resides in its own address space and has no direct access to any other memory, including shared memory or shared libraries. Objects may only communicate with each other by passing...
messages and must have a capability allowing such communication. Part of every object’s address space is reserved for use as a message buffer which is used to process both incoming and outgoing messages.

Although an object may make several different methods available to other objects, each object has a single entry point. When a new message is delivered to an object, the entry point function is invoked and it determines how to process the received message based on its content. Having a single entry point simplifies calling conventions.

Figure 4: Memory layout of a standard Annex object

A common object wrapper is provided for every object, abstracting the implementation details from object programmers and providing message-passing, task management and other object administration. Wrapping object code in platform specific management code allows object programmers to specify an object once, and have it run on any platform for which wrapper code exists.

Objects operate on an event loop. They are idle until they receive a message from the monitor. Once a message has arrived for an object, no others can be received until the object has yielded the CPU. This allows objects to use a single message-passing buffer for all calls that they make and receive.

In order to facilitate the required object isolation, objects are statically compiled executables.

3.3 Messages and Message Passing

Figure 5 shows the process of Object A sending a message to Object B. Object A must first construct a correctly formatted message in its message passing buffer (a). It then signals the monitor, which collects the message, placing it in its message queue (b). Before enqueuing the message, the monitor checks that Object A has a valid capability to Object B. Eventually the message progresses to the head of the queue (c). When Object B is able to receive it, the monitor places the message in Object B’s message passing buffer (d). Finally, Object B’s entry point function is invoked, and it can process the new message.

Object programmers are shielded from the details of message passing. A pre-processor is used on the object definitions at the compilation stage to generate remote procedure call (RPC) code for each method. The RPC code arranges for the marshalling and demarshalling of parameters and aids in providing a simple method calling syntax.

All messages are self-contained. No references other than capabilities can be passed between objects. As there is no shared memory, method calls are performed with call-by-copy semantics. While this does impose an increased burden on system efficiency, the increased security and stability from having no shared pointers outweighs this disadvantage.

Each message that invokes a method call contains: a capability providing the destination address, which the monitor uses to route the message to the appropriate device and object; a method identifier; and any parameters for the method call. The monitor on the device containing the destination object is responsible for checking the validity of the capability, and that the permissions allow invocation of the identified method. This ensures that the caller’s authority is limited to that of the specific capability presented.

There are four main message types. Method invocation (CALL) and method response (RETURN) mark the beginning and end respectively of a method call. Processor yield (ENDTURN) is used to stop a method call from processing, and puts the object into an idle state, enabling it to receive the results of an earlier method call that it made. RESUME messages are used to synchronise multiple outstanding method calls on a single object.

A CALL may be one-way, in which case the caller does not receive a corresponding RETURN message, or asynchronous, in which case the RETURN message is received some time later.

3.4 Tasks and Turn-Based Multitasking

In the Annex system, objects must always be in one of two mutually exclusive states. They are active while doing some computation, and idle once the computation returns or they are waiting for the result of a method call.

When an object receives a CALL message, a new thread of control, or task, is created. This task exists until it sends a RETURN message, at which point the task is destroyed and its stack and other resources are reclaimed.

While a task encompasses one complete method call on an object, a turn refers to the time period during which an object is continuously executing. A turn starts with either the invocation of a method or a return from waiting on a promise, and runs until either the method call completes or the object waits on another promise.

At any point in time, more than one task may exist for any particular object, but only one of these may be scheduled in any single turn. Each task explicitly releases control of the CPU by sending the monitor an ENDTURN message (e.g. by waiting on a promise), which also saves the task’s context. The context is restored when it is re-invoked after having received a RETURN message corresponding to a CALL message it has issued previously. The only task interleaving points are those involving ENDTURN messages.

Annex makes use of the preemptive multitasking that is a feature of modern operating systems to allow objects to run "in parallel", in addition to the coarse-grained cooperative multitasking described earlier, to avoid many of the consistency problems associated with preemptive multitasking. The Annex scheduler enables this by enforcing the rule that objects may not execute more than one turn at any point in time. This rule means that we can treat an object’s turn as a “critical section”. Objects can then be assured that their state (e.g. the values of global variables) will not be altered by another task during their turn.
This is achieved without explicit locking or other synchronisation primitives. As object execution can be modelled as a series of critical sections, program correctness can be much more simply formalised. To avoid multiple outstanding tasks interfering with global variables, these variables can often be moved on to the stack, becoming the equivalent of thread local storage.

Annex’s tasks and turn-based multi-tasking removes whole classes of bugs related to standard multitasking and multi-threading. It also allows Annex systems to scale effortlessly along with the number of available processing cores.

3.5 Requests and Promises

Since Annex is a distributed system, it provides a means of making calls on objects on remote devices while continuing to do useful work as that method call is processed. Annex uses the idea of promises (Friedman & Wise 1976) to implement this feature. An object receives a promise from the monitor when a method call is made. The result of the method call is claimed by a call to promiseWait, providing the earlier received promise as a parameter. As well as promiseWait, there are functions that let objects wait on some, or all, of a set of promises.

After the method call is made and a promise received, the object can continue to perform useful work, however, once a call to promiseWait (or equivalent) has been made, no further work is done by that task until the promise has been fulfilled. When the call to promiseWait is issued, the object sends the monitor an ENDTURN message. The object is then able to be scheduled to process another message. This may be either a reply to a previous call that the object made, or a new call, resulting in a new task being created.

The monitor maintains a requests table to associate RETURN messages with their earlier CALL messages. When the monitor receives a CALL message from an object, it creates a new requests table entry with a unique request ID and stores information identifying the source object and task. The request ID is then inserted in the CALL message before it is passed on to the destination object. The object sends its RETURN message back to the caller object, it includes the request ID. The monitor reads the request ID from the message, and uses that to look up the object and task that this RETURN message is destined for. The message can then be scheduled for delivery.

If the object making the call, and the object being called are on separate devices, then entries are made in the requests tables on each device. The request table entry in the caller’s monitor records the calling object and task, and is used to route the response back to the correct object and task, as in the local case. It inserts the request ID into the message, which is then sent to the remote device for processing.

When the message is received by the remote device, it records the existing request ID (Request ID 1), and the IPv6 address that the message came from in its requests table. A new request ID (Request ID 2) is created and inserted into the message before being delivered to the destination object.

When the RETURN message, containing Request ID 2, is delivered to its monitor, Request ID 2 is replaced with the original request ID (Request ID 1) and the message is sent back to the original monitor using the recorded IPv6 address. When the message is received by the original monitor, it is delivered as per the local version described earlier.

In this way it is possible to generate request IDs that traverse a network while remaining unique on both the caller’s device and the called object’s device.

3.6 Permissions

When objects pass capabilities to other objects, they may not wish to convey the same access rights (permissions) that they have. For example one object may wish to pass a capability that refers to itself, to another object but restrict the second object from calling anything but one particular method on it. This is done by generating a permissions mask, then passing it to the deriveCap system method on the capability of which we wish to derive a new version.

The supplied permissions mask is logically ANDed with the permissions of the capability used to make the call so that a derived capability can never gain more authority than the one from which it was derived.

3.7 System Methods

System methods are methods that are executed on behalf of a particular capability but are actually run by the monitor rather than the target object. An example is destroyCap, which removes that capability from the monitor’s catalogue of capabilities. System methods execute on the device that hosts the object that the capability refers to. All capabilities have a set of standard system methods attached to them, however access to these methods may be removed by permission restrictions.

3.8 Capabilities - Local vs Global Scoping

Capabilities held by objects exist in one of two mutually exclusive states, locally scoped or globally scoped. They are locally scoped by default and go “out of scope” at the end of the task that they are currently being used in and are removed from the object’s clist at the completion of that task. Once a capability has been removed from an object’s clist it may not be used in any further method calls.

For an object to retain a capability beyond the scope of the task to which it was passed, it must explicitly set the capability to be globally scoped. Globally scoped capabilities remain in the object’s clist beyond the end of the task that they were passed to and can be used by other tasks on that object, including those that they were not explicitly passed to. Objects may release globally scoped capabilities by setting them to be locally scoped. When that task completes they are removed from the object’s clist by the monitor.

As makeGlobalCap and makeLocalCap are methods that run on an object’s clist, an object must have a capability to its clist (with appropriate permissions) in order to call these methods. This means that we can construct objects that are unable to retain any of the capabilities that are passed to them. The ability to ensure that an object only ever has locally scoped capabilities (by not giving it a capability to its clist) is a confinement mechanism that assists in constructing systems based on the principle of least authority.

3.9 Error Handling

When a method call fails for whatever reason, provision is made for communicating details of the failure back to the calling object. When an error condition occurs inside an object method, the object’s author should return from the call using the
RETURN(ErrorCode) statement. This returns the ErrorCode information and some details about the failure to the caller, including the name of the object, the method ID, and the source code line that the error was generated at. The caller can access this information by calling annex_error(pPromise). The pPromise parameter is the promise that was supplied to the original call that failed.

4 Prototype Implementation

In order to rapidly design and develop prototype Annex systems, we have initially used a hand-rolled implementation of stylinux as the underlying operating system. Although some elements of the system have had to be built differently to how they would if running on a secure µkernel, almost all of the programmer-specified object code (exceptions being for hardware I/O and some code to handle object creation and message passing) will be able to run unaltered on top of other operating systems. Using Linux has also enabled us to use the same code-base to compile a system for use on an embedded ARM chip, and on standard x86(64)-based PCs.

We modelled the monitor and each object as separate processes and used dietlibc (dietlibc 2014) to statically compile each executable. This ensures that there are no shared libraries and all code for an object exists within its own process space.

In addition to having no shared code, we have been mindful of the fact that particular objects will need to be small in terms of code size in order to make formal verification a tractable proposition.

Permissions bit-vectors are 128 bits in size, with the first 16 bits reserved for system methods.

To implement the message passing, sockets were used. This does impose some overhead on the system, however, as we envisage Annex being used in niche situations (such as implementing a TCB in a CDS), the isolation, and hence security benefits, far outweigh the modest performance impact. In addition, there are mechanisms for mapping regions of memory between processes on secure µkernels that may prove to be more efficient.

One of the benefits of having a robust simulator for development effort is that standard debugging tools such as gdb are able to be used. We also utilised replay (Murray & Grove 2013), a tool for visualising graph modification over time. An Annex plugin allowed us to view the timing and content of messages.

5 Example Object systems

Annex has been designed so that software systems built using it are highly amenable to security proofs, and the example systems that have been developed demonstrate that benefit. The first example discussed is a software-only data diode which shows the simple power of the system. Then a much more complex software system, a graphical user interface, is described. Finally, an example hardware and software design is discussed as well as several real world devices that have been built using this design, demonstrating some new opportunities that become available in the high-assurance solution space due to our system design.

5.1 Data Diode

The concept of a unidirectional network link or data diode has been around for some time, but it was not until the 1990’s that a hardware implementation of a data diode became a reality. The Starlight Interactive Link Data Diode (Anderson et al. 1996) was certified to EAL7 under the Common Criteria in 2005.

Although the concepts are well understood, no software-only data diodes have been able to achieve such a high level of certification. This is due in large part to the complexity of software-based systems.

We contend that Annex, running on a secure µkernel, provides a framework that is highly amenable to both formally evaluating the implementation of the diode itself, and the means of passing data through it.

![Figure 6: A collection of objects using the diode (DD) to securely communicate in one direction](image)

Listing 1 shows the source code for a simplified version of a data diode that is restricted to passing 32-bit values. The diode sits between two disjoint graphs of objects and only allows data to pass in one direction between them.

```c
uint32_t local_data;

EXPORT write_up (IN uint32_t data)
{
    local_data = data;
    RETURN(DK);
}

EXPORT read_down (OUT uint32_t data)
{
    data = local_data;
    RETURN(DK);
}
```

Listing 1: diode.def

The diode is specified in a few simple lines of code. As long as the assumed object isolation properties are satisfied, and the object wrapper and monitor code shown to be secure, then it is a fairly simple matter to prove that this object cannot move data from the reader side to the writer side.

An object that wanted to write data to the diode would need a capability to the diode that contained the permission to call the write_up method.

diode.write_up(SYNC, diode.cap, data);

A diode object (a structure that contains pointers to the objects MPI stub) is associated with a capability (diode.cap). The SYNC keyword specifies that the calling object will immediately wait for the results rather than use a promise to claim them later.

Equivalently, an object that wanted to read from the diode would need a capability to the diode that contained the permission to the read_down method.

5.2 GUI

The development of a Graphical User Interface (GUI) subsystem demonstrates the power and flexibility of our object capability system. Each element to be drawn to the screen is represented by an object called
a guiUnit. This object type has attributes such as a background colour, border width and height, an image and/or text component, and placement. GuiUnits are connected together to form a tree, and each may only draw within the confines of its parent guiUnit.

Rendering to the screen is performed by a GUI server. This server queries the guiUnits to determine how to draw them. The server also reads mouse, keyboard and touch input and directs it to the appropriate guiUnit for processing.

While the performance of our Annex GUI subsystem is not comparable to modern high-performance interfaces, it has proven to be more than capable for the applications that we have used it for, including multi-level messaging, network management, cross domain cut and paste, covert channel monitoring tool and blue force tracking.

The covert channel monitoring tool is shown in Figure 7. Icons that launch other applications can be seen on the taskbar at the bottom of the screen.

5.3 Cross Domain Solutions

Annex has been used as the software framework for several prototype real world devices. These devices share a common basic architecture (shown in Figure 8) but address different aspects of cross domain security.

5.3.1 Annex Router

The Annex Router is conceptually similar to a VPN concentrator. In addition to the basic architecture described above it includes a second network interface at each security level, allowing access to an entire network at each classification. In this case Annex software is used only to create, remove and audit network connections. It should be noted that Annex software has no access to the network data itself, only the metadata used to route the encrypted packets.

5.3.2 Minisec

Two versions of the Minisec device were produced according to this architecture (a first prototype used a different architecture to achieve a different aim).

The Minisec2 is a touch screen device, pitched somewhere between a tablet device and a smartphone. It uses Qtopia as an operating environment on each of the at-level domains, providing services such as VoIP and email at each classification over encrypted channels. The device incorporates trusted buttons that allow the user to select which domain has access to the screen, touch input and audio output at any given time. The Minisec2 was designed for tactical military use, with two large hot-swappable batteries, making the device available without downtime for charging.
The Minisec3 moved the target market to a desktop/server space with a desktop PC form factor. Each domain was upgraded from its original Atom-based board to a Core i7 mini-ATX form factor. This allowed us to run unmodified Windows 7 or Ubuntu operating systems on each domain. The Minisec3 has buttons to allow the user to specify which is the active domain at any given time, and includes a greatly enhanced Annex domain which includes the GUI subsystem.

This form factor, and the strong isolation between the objects running in the TCB, lends itself to use with COTS user applications across different security levels. For example, Microsoft Word has been shown to run in separate domains while securely sharing a multi-level document using such a TCB to filter data to and from each domain (Owen et al. 2011).

Another example of newly developed capability that this architecture allowed was the concept of cut and paste data downgrade. As the Annex-based TCB has complete control of how data is moved between domains, it is able to selectively, and with appropriate user review, downgrade textual information from high to low.

6 Evaluation

Rather than reimplement the strong isolation properties provided by secure μkernels, we have explicitly chosen to leverage them to provide a programming environment that lends itself to more complex high assurance systems. Creating a generic framework in the monitor and object wrapper allows system programmers to focus on how their objects interact without needing to be overly concerned with low level details.

One measure that lends itself to system security analysis is the size of the code base. SeL4 consists of 8700 lines of C code and 600 lines of assembler (Klein et al. 2009). Using sloccount, the Annex monitor consists of 8616 lines of C code, while the object wrapper has 1560 lines of C code and 2 lines of assembler, and if we were to optimise the Annex code for use in a particular scenario, we would be able to cut down these numbers. While the count of lines of code does not by itself infer any security properties, it is indicative of a system designed with security in mind.

6.1 Formal Analysis

An initial formal analysis of Annex, based in the HOL theories of the Isabelle proof assistant (Nipkow et al. 2002), was commissioned. The motivation was to capture a formal model from an advanced prototype system, and to use it as a touchstone for discussions between modellers and developers in a future high assurance development effort. As such it helped initiate an artifact trail that could be used to produce the deliverables required in a high evaluation process.

The ensuing report (McCarthy 2013) quickly introduces a simple model for the distributed communication context in which Annex sits, and the capability, promise and message structures that control and utilise it. The report then concentrates on the two major elements of Annex, the monitor and the object wrapper code, whose properties must be fully analysed before any resultant systems could be formally verified.

For each element, the relevant data structures and methods thereon are modelled in detail, whilst the behaviour (in particular, message passing) is encoded in a state machine (SM) model using these structures in its state description.

Thus, for the monitor, all of the major components - the object table, capability, catalogue, clist, message queue and requests table - are SM state in this sense, and the scheduler is simply represented in the SM transition logic.

Similarly, for Annex objects, the task table, promise table, results table are SM state, and the turn-based multitasking is a consequence of the SM transition logic. To allow for a generic treatment (as opposed to the specific analysis of a particular application) the object methods are fed in as a collection of generic functions on the object data and an abstract computing stack.

Strong isolation allows graph theory to be used to provide inter-object authority bounds, while careful composition of permissions allows intra-object information flows to be verified. In this manner we hope to divide and conquer the evaluation problem.

6.2 Performance

We state quite clearly that Annex is not designed to develop a fully-fledged operating system that can compete with Windows, Ubuntu or OSX for interactive user experience. However, there are an increasing number of niche uses for which the performance tradeoff is a small price to pay for a much more robust security model.

The hub-and-spoke structure of the Annex architecture (see Figure 11) results in some limitations. The fact that all messages pass through the monitor, together with the call-by-copy semantics of method calls, means that there are bottlenecks in the system. We focus strongly on the side of security in the security-performance tradeoff.

Using processes to host objects and sockets to move messages between objects and the monitor means that each method call, from invocation to reply requires at least 6 context switches, however this impost will be lessened if we are able to deploy on a

Figure 10: Minisec3 demonstrating the concept of a single document being edited at multiple levels.

Figure 11: Conceptual, (left), vs actual, (right), message passing in the Annex system.
multi-core chip. Part of Annex’s design is that the number of objects that are able to run concurrently scales linearly with the number of available processing cores. Previous work (Newby 2008) looked at how to port Annex to multi-core chips such as the Cell processor or Intel’s Single-Chip Cloud Computer.

The data copying overhead required to move messages between monitor and objects may be able to be removed if we can use page-transfer mechanisms provided by the underlying µkern, while thoughtful architecting of systems can bypass much of this performance impost.

7 Related Work

The concept of isolation, bridged only through well defined interfaces, lies at the core of computer security (Aiken et al. 2006). As early as 1981, Rushby (Rushby 1981) identified the need for isolated processes in the development of multilevel secure systems on commodity computing platforms. Recently, there has been renewed interest in applying this principle to computing systems at various levels of abstraction.

In the software sandboxing space, Capsicum (Watson et al. 2010) is a capability system used by the Chromium browser on BSD-based systems to isolate each tab or web session. This stops problems with rendering one tab from affecting others. Similarly, uPro (Niu & Tan 2012) enables programmers to break their program into sections and define interfaces between them which are then checked by a run-time harness.

Annex owes its heritage to the E language (Miller 2006). In E, as in Annex, objects are referenced by capabilities, however, in E some objects may run together in a shared space called a vat. While method calls between vats use a distributed calling convention, calls internal to a vat do not. In contrast, in Annex all calls between objects use distributed calling conventions. In addition, E does not use permissions.

While these efforts are improvements over standard programs using shared memory, none are appropriate for high assurance systems.

The benefits of being able to access information of differing security classifications and the financial benefits of being able to remove multiple workstations from desktops is leading to renewed interest in Cross Domain Solutions. Policy makers are reducing the security requirements of such systems (at least for lower classifications of data) and commercial vendors are beginning to bring systems to market that are able to meet these reduced standards.

Galosi’s Trusted Services Engine (Galosi Inc. n.d.) connects to up to 4 networks at differing security levels and allows users to read from lower levels but prevents write-up. Raytheon’s High Speed Guard (Raytheon 2014) and Small Format Guard (Raytheon Trusted Computer Solutions 2014) are bidirectional filtering routers with rules that can be set prior to deployment. All of these systems use SELinux (National Security Agency n.d.) to control data flow, with Raytheon hosting theirs on Red Hat Enterprise Linux (Red Hat n.d.) (RHEL).

Thales’ Trusted Security Filter (Thales n.d.) is also a bi-directional filtering router with a predefined, non-configurable filter. The internal architecture of the device is unspecified, however an updated assurance evaluation (SERTIT n.d.) shows that the filter is specified in software.

Thales and Raytheon have also formed a partnership to provide the Australian Department of Defence’s Next Generation Desktop (NGD), a version of Raytheon’s Trusted Thin Client (Raytheon Trusted Computer Solutions n.d.). This product consists of a Distribution Console (DC) which acts as a VPN concentrator, sending out a single (low) level data stream and pre-configured virtualisation software that allows a client to connect to the servers. Keys appear to be pre-shared. The DC runs SELinux on RHÉL to maintain data separation.

SecureView (AFRL 2014), a collaboration between the US Air Force Research Labs, Intel and Citrix, is a CDS that uses XenClientXT as a bare metal hypervisor to isolate multiple virtual computers of differing classification.

The CDS’s described above are architecturally similar to the Annex hardware platform described in Section 5.3. However, where they use a single shared CPU and either complex virtualisation software or entire COTS operating systems to manage isolation, we use simple replicated hardware.

In 2007 the Separation Kernel Protection Profile (SKPP) was formalised by the Information Assurance Directorate within the NSA. In October 2008, Green Hills Integrity-178B (Green Hills Software Inc 2010) (a fixed variant of their real time operating system) was certified compliant against the SKPP. This compliance gives Green Hills the only operating system to be certified to Common Criteria EAL6 augmented.

NICTA’s seL4 (NICTA n.d.) is a formally verified µkernel. Its security properties, such as functional correctness, have been captured in an abstract specification and, through refinement proofs, a correspondence was shown with its underlying C implementation. It uses capabilities to partition and reference memory.

As both Integrity-178B and seL4 are µkernels they provide a bare minimum of primitives for a user-space program. By keeping this set small, they are able to guarantee that they behave correctly, but their programming interfaces are more difficult to work with than standard computer systems.

We see Annex as extending the trust that secure µkernels such as seL4 and Integrity provide, into a usable space for programmers. The guaranteed isolation for executing processes makes them perfect for hosting Annex object systems.

8 Conclusion

Designed to run on top of existing highly certified COTS µkernels, the Annex Object Capability System provides a strong platform for driving high level evaluation and certification beyond a separation kernel and into user applications.

While multi-level systems are gaining favour again, solutions to constructing the cross-domain component are often focused on supporting fine-grained security policy at the expense of a TCB that is too complex to effectively verify. Annex aims to solve this problem, combining true verifiable security with a familiar programming environment.

An approach to engineering high-assurance trustworthy systems has been shown for exemplar Cross Domain Solutions and Multi-Level Secure systems.

References


URL: http://doi.acm.org/10.1145/361011.361067


On the Effectiveness of Virtualisation Assisted View Comparison for Rootkit Detection

Toby J. Richer1 Grant Neale1 Grant Osborne1

1Defence Science and Technology Organisation
PO Box 1500, Edinburgh, South Australia 5111,
Email: toby.richer@dsto.defence.gov.au

Abstract

There is growing interest in tools for monitoring virtualisation infrastructure and detecting malware within Virtual Machines (VMs). View comparison, or cross-view validation, is a technique for detecting object hiding by malware. It involves comparing different views of system objects to find discrepancies that might indicate the use of object hiding techniques.

We present Linebacker, a system for performing view comparison on VMware vSphere VMs. Linebacker compares external (i.e. hypervisor level) and internal (i.e. guest operating system level) views of process, file and registry objects within VMs to detect rootkits that cloak such objects from the view of the guest operating system. We use Linebacker to compare the efficacy of the view comparison technique to sandboxing or API call monitoring approaches to rootkit detection. We also present a case study evaluating the performance impacts associated with using Linebacker to monitor VMs in a production environment. We present execution and analysis time metrics for this study and discuss feedback provided by users.

Finally, we analyse our results and make recommendations regarding the implementation of view comparison for real-world virtualisation infrastructure.

1 Introduction

There is growing interest in tools for monitoring virtualisation infrastructure and detecting malware within Virtual Machines (VMs).

Rootkits often use evasion techniques to avoid detection and thus increase their chance of persisting on a system. Evasion techniques may include manipulation of the way a system reports active processes, lists files and their contents, and displays registry keys.

View comparison (Wang, Vo, Roussev, Verbowski & Johnson 2004) is a technique for detecting rootkits by comparing internal and external views of operating system objects. The ability of the technique to detect rootkits has been demonstrated previously (Garfinkel & Rosenblum 2003, Wang, Beck, Vo, Roussev & Verbowski 2005, Quynh & Takefuji 2007, Jiang, Wang & Xu 2007, Jones, Arpaci-Dusseau & Arpaci-Dusseau 2008, Wang, Hu & Li 2011), but to our knowledge it has not yet been implemented within commercial monitoring tools (Adventium Labs 2014, Trend Micro 2014).

There are potentially several reasons why view comparison is not currently used in commercial monitoring tools. View comparison may not scale well to monitoring a large number of machines. The results of view comparison may be difficult to interpret, and may result in a high false positive rate when applied to production systems. Many current virtualisation assisted implementations of view comparison rely on modifications to existing commercial or open-source hypervisors (Garfinkel & Rosenblum 2003, Quynh & Takefuji 2007, Jiang et al. 2007, Wang et al. 2011, Jones et al. 2008), and it may be that adding this capability to commercial tools is too onerous. It may also be that the same or better performance (in terms of VM impact and detection of rootkits) can be achieved through the commercially available tools that use vShield Endpoint (VMware 2014) such as Bitdefender SVE (Bitdefender 2014), Sophos Antivirus for vShield (Sophos 2014) or Trend Micro Deep Security Antivirus (Trend Micro 2014).

We have developed a system for performing view comparison and live Virtual Machine Introspection (VMI)-based (Garfinkel & Rosenblum 2003) forensics on VMware vSphere VMs. Our system supports the “vanilla” VMware ESXi hypervisor and does not require additional drivers (other than VMware Tools) to be installed within monitored VMs. As a result, we have been able to evaluate the performance of our tools in a production environment and gather feedback on the user experience. We also used the Cuckoo malware analysis tool (Cuckoo Sandbox Developers 2014) to sandbox the execution of a sample set of common malware. This sandboxing process was used to generate profiles of the malware, which we used to compare the ability of API monitoring and view comparison to detect rootkits.

This work contributes an analysis of the effectiveness of view comparison detection techniques in detecting object hiding performed by rootkits, as compared to API call monitoring techniques. We assume that the data generated by Cuckoo is similar to the data used by API call-based detection techniques in commercial rootkit detection tools. This data includes windows kernel level activity associated with creation of processes and files, and any subsequent calls made to install filter drivers to hide them from the operating system. To our knowledge, our work is the first analysis of view comparison that compares it to another method of detecting object hiding, over a range of rootkits. Our work also contributes an analysis of view comparison performance across multiple VMs hosted on standard VM infrastructure. Our results provide new insights into the impact of implementing view comparison techniques in real-world scenarios.
Virtualisation infrastructure.

The next section describes previous work in view comparison and introduces Linebacker, our implementation of view comparison for the VMware vSphere virtualisation platform. Linebacker implements view comparison using standard APIs developed by VMware. Section 3 presents an evaluation of the efficacy of performing virtualisation-assisted view comparison, based on tests of Linebacker on a corpus of real-world malware samples. Section 4 presents a case study on the performance of Linebacker in a real-world scenario. Finally, Section 5 presents our conclusions and discusses potential further work in this area.

2 View Comparison

View comparison, also known as cross-view validation (Wang et al. 2004, Wang et al. 2005), is a technique for detecting rootkits. It involves comparing different views of system objects to find discrepancies that might indicate the use of evasion techniques by rootkits.

A rootkit is malicious software that allows an attacker to establish and maintain a persistent presence on a machine, while concealing their presence from legitimate users of the machine. Rootkits often hide objects such as files, registry entries, processes and kernel modules in order to avoid detection.

A simple technique for hiding files is to set the hidden bit for each of these files within the Master File Table (MFT). However, the details of such files are still available from the MFT, or by changing system settings to show hidden files.

A more sophisticated hiding technique employed by rootkits is to hook the system Application Programming Interface (API) functions used to enumerate a particular type of object and filter out any results referring to the object being hidden. The underlying data structures storing hidden objects are not modified by this filtering technique. These hidden objects can be detected by comparing the raw contents of these structures to the view presented by the system APIs. Objects that appear within the underlying data structures, but are not visible in API results might have been hidden by a rootkit.

Direct Kernel Object Manipulation (DKOM) rootkits hide objects by modifying data structures in the kernel directly. These rootkits can potentially be detected by comparing API results with raw data structure contents. In some cases they can be detected by comparing the contents of related kernel data structures storing similar information (Wang et al. 2005).

More recent rootkits have implemented new techniques to avoid being detected by view comparison (Kapoor & Mathur 2011). The Koutodoor rootkit blocks read/write access to files, rather than hiding them. The TDSS family of rootkits store files as orphan files in unused portions of the drive. They alter the Master Boot Record (MBR) to allow access to these files without them appearing within the filesystem. Another approach, proposed by Jiang et al. (Jiang, Wang & Xu 2010), is for rootkits to modify the fundamental structure of disk or memory to hide processes or files. This would also have the potential to hide processes from Linebacker, since it relies on the reconstruction of disk or memory by VMWare or forensics tools. However, it would be extremely difficult to modify these fundamental structures while enabling the normal operation of other software on the guest OS. Hiding information outside the file system, as done by the TDSS family of rootkits, achieves a similar goal more easily.

Some rootkits do not attempt to hide the presence of the files they have added or altered. One technique used by rootkits is to hide in Plain Sight (HIPS), by hiding objects within existing files or memory structures or in obscure sections of the filesystem. Rootkits that do not try to hide file objects will not be detected by view comparison, but will potentially be vulnerable to other established rootkit detection techniques.

Virtualised infrastructure provides a significant advantage over non-virtualised infrastructure for the application of view comparison. VMI allows VMs to be observed from the hypervisor, providing an external viewpoint that cannot be tampered with from within monitored VMs. Guest operating system data structures in memory or on disk can be parsed directly from this external view. View comparison can be performed by comparing this external view with the internal view provided by the guest operating system.

2.1 Virtual Machine Introspection

Virtualisation allows multiple VMs to run on a single physical server. Software called the hypervisor, or Virtual Machine Monitor (VMM), presents a virtual hardware platform to VMs and manages accesses to physical hardware. This allows isolation to be maintained between VMs sharing the same physical hardware.

Virtual Machine Introspection (VMI) (Garfinkel & Rosenblum 2003) is the observation of software running inside of VMs from the hypervisor level. The hypervisor controls all access to the virtual hardware platform and the underlying physical hardware, allowing disk, memory, CPU, network and other hardware contents or state to be observed directly, without relying on the operating system running within the VM. Such an operating system is commonly referred to as the guest operating system. VMI has previously been used in a number of applications including Intrusion Detection Systems (IDSs) (Garfinkel & Rosenblum 2003, Jiang et al. 2007, Jones et al. 2008), malware detection and analysis (Jiang et al. 2007, Litty, Lagger-Cavilla & Lie 2008), and forensics (Hay & Nance 2008, King & Chen 2005, Krishnan, Snow & Monroe 2010, Nance, Hay & Bishop 2009).

The key obstacle to VMI is the semantic gap between the state of a guest operating system and the raw data visible to the hypervisor. The challenge of reconstructing guest operating system semantics from this external view is well known and a number of methods have been proposed (Garfinkel & Rosenblum 2003, Jiang et al. 2007, Krishnan et al. 2010, Litty et al. 2008, Pföh, Schneider & Eckert 2009). Pföh et al. (Pföh et al. 2009) provide a useful categorisation of such methods into three “view generation patterns”: in-band delivery, out-of-band delivery and derivative. These three patterns or combinations thereof encapsulate all possible methods of bridging the semantic gap.

In-band delivery (Pföh et al. 2009) uses an agent within the VM to deliver information to the hypervisor. This method avoids the semantic-gap challenge by leveraging the guest operating system’s inherent knowledge of its own architecture. This method lacks portability and is easily subverted.

Out-of-band delivery (Pföh et al. 2009) generates an OS-level view from hypervisor-level state information using semantic knowledge obtained in advance. This knowledge might include kernel symbol tables, address space layouts or file system specifications.
The significant body of computer forensics research focused on reconstructing system state from forensic disk and memory images can be applied to overcome the semantic gap when using out-of-band delivery. Disk forensics techniques can be used to extract a variety of metadata (e.g., a file’s author or its modification, access and creation times) and digital evidence (e.g., emails, images, videos and documents) from a file system. There are several commercial tools that perform this type of analysis, such as EnCase Forensics (Guidance Software 2013). VMware uses delta files to track changes to the disks of VMs. In this case, changes in unallocated parts of the hard drive and modification of existing files will appear in these delta files, even if the file system is undisturbed. However, approaches to detect rootkits based on the delta files are beyond the scope of this work.

Memory forensics techniques can be used to gather system state information from volatile memory. Process and network connectivity information can be extracted from a memory image (Okolica & Peterson 2010). The Virtual Address Descriptor (VAD) data structure maintained by the kernel to keep track of allocated memory ranges can be used to enumerate the files and objects to which each process has references (Vömel & Freiling 2011, Russinovich, Solomon & Ionescu 2009). Aljaedl et al. (Aljaedl, Lindskog, Zavarsky, Ruhl & Almari 2011) discuss the large variety of information types able to be extracted from memory. A range of tools have now been developed to help with the analysis of volatile memory images, such as the Volatility Framework (Volatile Systems 2013).

In addition to forensics tools, memory and disk driver software on the VM monitor is often similar or identical to the driver software on the guest. In this case, the drivers on the monitor can be used as a template for reconstructing the memory and disk structures. This is the approach used in VMWatcher (Jiang et al. 2010).

Derivative delivery (Pfoh et al. 2009) generates an OS-level view using knowledge of the virtual hardware architecture of VMs. An example of this approach is monitoring the contents of CPU control registers to infer information about the state of the VM. This approach is portable between different guest operating systems and is much less likely to be subverted by malicious modification of guest OS. Unfortunately it is not portable across VM hardware architectures and can only extract limited information as prior semantic knowledge of the guest operating system cannot be used as a part of this approach.

2.2 Existing Implementations

View comparison based rootkit detection was first implemented by Strider GhostBuster (Wang et al. 2005), which compares the high-level view provided by the Win32 APIs to the lower-level view obtained by parsing data structures directly. These data structures are accessed by using a kernel driver, triggering a kernel memory dump or booting the monitored system from a WinPE boot CD. GhostBuster is able to detect hidden files, processes, kernel modules and registry entries. The file and registry view comparison aspects of this tool were subsequently released as RootkitRevealer, which used a kernel driver for its trusted view (Microsoft Research 2010).

Kernel Rootkit Trojan Detector (KeRTD) (Mahapatra & Selvakumar 2011) uses a number of techniques to detect and defend against rootkits. VMM coverage is used to compare and access control lists between kernel and user-mode to detect hidden objects.

A number of academic hypervisor-based IDS prototyping uses VM to provide the external view for their view comparison modules. Livewire (Garfinkel & Rosenblum 2003), XenKIMONO (Quynh & Takefuji 2007), VMwatcher (Jiang et al. 2007) and VMDetector (Wang et al. 2011) all read VM memory directly from the hypervisor to access kernel data structures. The contents of data structures listing kernel modules, processes and network connections are then compared to user-mode views of the corresponding objects to detect hiding.

Lycosid (Jones et al. 2008) uses a different approach to detect hidden processes. Process creation and deletion is inferred by observing related events, such as virtual address space creation and destruction, from the hypervisor. A hidden process causes the process count observed using this technique to differ from that reported by the guest operating system, allowing the hiding to be detected.

2.3 Our Implementation

We have developed a tool, named Linebacker, to perform view comparison and enable the use of standard disk and memory forensics on running VMs hosted on standard VMWare ESX servers. To our knowledge, none of the existing view comparison tools are designed to run on VMWare enterprise infrastructure; VMWatcher is able to perform view comparison on VMWare Workstation though development of this capability required access to VMWare source code. This tool is able to compare internal (i.e. in-band delivery) and external (i.e. out-of-band delivery) views of memory, filesystem and registry objects for Windows VMs. Linebacker relies on VMWare supported APIs to obtain the internal and external views required to perform view comparison. This approach eliminates the need to install an additional agent within each monitored VM, relying instead on installation of the standard set of VMWare Tools within the Guest.

The view comparison techniques we employ aim to detect hiding of malware artefacts from the guest operating system. Examples include filtering the results of API calls before they are returned to the operating system or modifying kernel data structure using DKOM. Hide In Plain Sight (HIPS) techniques such as creating a file or process with a legitimate looking name or setting system or hidden attributes on files to conceal them from the user are not the target of our view comparison system, but may be detected by it. Such techniques do not prevent the guest operating system from accessing the concealed objects, and are better addressed through analysis of the visible file objects in the operating system than by view comparison.

Linebacker obtains an external view of memory objects by parsing Virtual Machine Suspended State (VMSS) memory images. VMSS files are generated by VMWare hypervisors when VMs are suspended. These files store the metadata required to resume the execution of suspended VMs. This metadata necessarily includes an atomic snapshot of each VM’s memory at the time it was suspended. The VMWare vSphere Web Services API (VMWare 2014a) is used to momentarily suspend each monitored VM, triggering the creation of a VMSS file. The list of processes running in the VM is then parsed from this memory image using the open-source Volatility Framework (Volatile Systems 2013). This list is then compared to the corresponding internal list, which is obtained via the VMWare Tools guest addition using the VMWare VIX API (VMWare 2014b). Any processes that appear only in the external view are reported as hidden.


37
Similarly, Linebacker obtains an external view of disk objects by parsing disk snapshots. The VMware vSphere Web Services API is used to capture a snapshot of each monitored VM. These snapshots are then mounted using the VMware Virtual Disk Development Kit (VDDK) (VMware 2014a), allowing filesystem-level access to the contents of the VM’s disks. This allows the files and directories on each disk to be enumerated and compared to the corresponding internal view. The internal view is obtained using the VIX API to execute the `dir` command within the guest operation system. This command is run with flags to include any hidden files and to run recursively through all subdirectories. Any files that appear only in the external view are reported as hidden.

Linebacker also uses disk snapshots to reconstruct an external view of the non-volatile contents of the Windows registry (Russinovich, Solomon & Ionescu 2012). This is done by parsing the ‘hive’ files storing the registry using Hivex (Jones 2010). The corresponding internal view of the registry is obtained by using the VIX API to recursively enumerate all keys and values in the registry using the `RegEnumKeyEx` and `RegEnumValue` Windows API functions. These internal and external views can then be compared to detect hidden registry entries.

### 3 Detection

We conducted a set of tests in order to determine the efficacy of Linebacker’s view comparison techniques at detecting the evasion behaviour of rootkits. This involved using Cuckoo Malware Sandbox in conjunction with our own scripts to execute Linebacker and Cuckoo on a corpus of malware executables. Each sample was run on a Windows XP x86 VM and a Windows 7 x86 VM hosted on the VMware ESXi 5.5 hypervisor.

While Cuckoo is not specifically designed to detect rootkits, it is designed for wider analysis of rootkits. It does not use the same techniques to extract information as rootkit detectors, as it sandboxes the execution of malware rather than using API hooking (i.e. it does not install a system level driver that manipulates or reads API calls at a kernel level), but it provides the same type of information as is used by rootkit detectors such as GMER (GMER - Rootkit Detector and Remover 2014). As an analysis tool rather than a detector, it will provide all the metadata it can about the malware it is sandboxing without filtering this information down to a set of single alerts for each rootkit detected. Our position is that the most informative way of analyzing view comparison as a rootkit detection technique is to compare the total sets of raw data obtained by Linebacker and Cuckoo when tested against a broad range of object hiding techniques used by rootkits encountered on the internet, which are summarised in Section 2.3. We then ran them on our test Windows XP system, and removed those that did not run, or ran but had no effect on any aspect of the system (as measured by Cuckoo). In the case of the FUTo (Silberman 2006) rootkit, the version that we found did not run on our Windows XP test system, but we were able to find a cut-down version of the rootkit that employs the same techniques and ran successfully on Windows XP. This was tested later.

The information provided by Cuckoo includes:

- native function and Windows API call traces
- copies of files created and deleted from the file system
- child process and process activity logs

We use Linebacker’s view comparison techniques to provide metadata about:

- processes for which an `EPROCESS` structure can be parsed directly from an external VMSS memory image, but which are omitted from the results of Windows API calls used to enumerate processes internally;
- files that are visible when directly parsing an external snapshot of a VM disk, but are not listed internally;
- registry keys that are visible when directly parsing registry hive files from disk snapshots, but are not visible via the Windows API calls used to enumerate registry contents internally.

To compare the results of both techniques, we filtered out clear false positives from each set of results. The baseline registry, process and file system chatter were removed from Linebacker by removing any result that appeared in both the clean and dirty runs of Linebacker for a particular rootkit. In addition,
the dirty results for every test included malware files uploaded to the sandbox by Cuckoo and Windows prefetch files created as a result of executing the malware. These files were also filtered from the results. To determine if the remaining artefacts were part of the rootkit or unrelated, we consulted available references on the behaviour of each rootkit. Where necessary, we repeated the test procedure and manually inspected the “dirty” state of the VMs through a VMware console. Through this, we could determine as precisely as possible how much of the information provided by Cuckoo or Linebacker was directly related to the detection of a rootkit.

3.2 Results

The detection testing results obtained using Linebacker are summarised in Tables 1 and 2. For Windows 7, we have also indicated whether, based on any evidence of activity in Cuckoo, the rootkit has installed successfully. The results are described below, and analysed in more detail in Section 3.3.

Note: We refer to the memory view comparison results as “process” results, as we were only examining potentially hidden process information rather than all of the artefacts that could be extracted from memory.

Process Results: While Linebacker’s process view comparison tool detected three true positive hidden processes, a large number of false positives were removed through the filtering and verification process. Both Linebacker and Cuckoo detected a hidden process for the zeus.melt.exe rootkit on both editions of Windows tested. The name of the process is actually a random string, which changed each time we executed the rootkit. This behaviour is consistent with published descriptions of this rootkit’s behaviour. A hidden instance of smss.exe was also detected for two samples from the Zbot family, when run on Windows XP. Cuckoo did not detect this. On Windows 7, these two samples were not successfully installed.

As stated above, the FUTo rootkit was executed manually on both Windows XP and Windows 7. It was used to hide the notepad.exe process. On Windows XP the hidden process was detected by Linebacker, but was not detected by Cuckoo on Windows 7, though it was detected for two samples from the Zbot family, when run on Windows XP.

Disk Results: After filtering the results from Linebacker, and comparing the results with references on these rootkits, it was found that Linebacker had detected only two of the malware samples, Trojan-Spy.Win32.ZBot.dol.exe and QVOD, hiding files. This sample hid the directory C:Documents and Settings\ Test\ Application Data\ wsnpoem in Windows XP. This directory contains two files: audio.dll and video.dll. The wsnpoem directory is created by particular variants of the Zeus trojan. The audio.dll file within this directory is used to store stolen data to be sent back to the command and control server. The video.dll file stores the encrypted configuration for the trojan (ZeusTracker 2014). Linebacker also detected these files in Windows 7, in the directory C:Users[admin]Roaming\wsnpoem, though in this case Cuckoo also detected the files. Cuckoo observed the creation of three executable files by three variants of the Zeus/Zbot trojan: oembios.exe, ntos.exe and sdra64.exe, but did not detect the creation or presence of the files found by Linebacker on the Windows XP test machine. Linebacker did not detect the executable files found by Cuckoo as being hidden.

For the QVOD rootkit, a number of files were detected by Linebacker within the system restore directories. A manual verification revealed that these files were detected because the rootkit had set the “system” attribute for these files. This prevents them from being read by the dir /h command for anyone but the system administrator. While this is an example of Linebacker detecting rootkit activity, these files are only hidden from some users of the operating system, as opposed to the files hidden by Trojan-Spy.Win32.ZBot.dol.exe.

Linebacker detected files from other samples, on both Windows XP and Windows 7 test machines, but it was discovered during the verification process that these files were not associated with the malware under test. These files were either hidden from Linebacker’s internal view as a result of filesystem permissions, hidden with the “system” attribute as described above, or created in the time interval between the internal and external views of disk being generated.

Registry Results: The registry view comparison did not detect any hidden keys or values.

3.3 Discussion

Process Results: The large number of false positives removed through the filtering and verification process can be attributed to the small delay between capturing the memory image providing the external process listing and the generation of the internal view via VMware Tools. In practice it is not possible to synchronise these two views perfectly as a VM must be paused to allow an atomic memory image to be captured. We expect the number of false positives to increase with process creation and termination activity on the monitored system.

For example, a process in the system may terminate during the time it takes to download the VMSS memory image from the guest. This process would appear in the external memory image, but would not be reported in the internal process listing. Our tool would then detect this process as potentially malicious, even though it is simply the result of benign process creation and exit activity.

Another cause of process false positives is EPROCESS structures for terminated processes remaining in memory. These structures remain in memory until they are overwritten or wiped, which may not happen for some time after a process terminates. Such structures will be parsed as active processes in the external view, resulting in a false positive.

To our knowledge the false positives would be difficult to filter outside of a controlled test environment, as filtering could also filter out process hiding that a rootkit may use to preserve itself. Our results show the concept of memory-based view comparison has the potential to work. However it is our recommendation that a realistic security solution would require cross checking against additional detection capabilities (such as live in-memory monitoring of process behaviour — something that is not possible using the free VMware APIs).

Disk Results: Linebacker’s disk analysis detected object hiding from only one of the rootkits in the test set. The file created by this rootkit were not observed by Cuckoo. Though Cuckoo’s recording of API calls found more rootkits, and found more files associated with rootkits overall, this suggests that view comparison and tracking API calls may work as complementary approaches to rootkit detection.

As with the memory analysis, the disk view comparison produced a large number of false positives. An advantage of the disk view comparison technique is that false positives tend to appear in the same locations in the filesystem. False positives that occur
Table 1: Summary of hidden objects detected by Linebacker (Windows XP x86)

<table>
<thead>
<tr>
<th>Rootkit</th>
<th>Installed?</th>
<th>Process</th>
<th>File</th>
<th>Registry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backdoor.Win32.Ghost.binder.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backdoor.Win32.Ghost.20.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backdoor.Win32.Ghost.21.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backdoor.Win32.Ghost.23.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QVOD</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>TDL4</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Trojan-Spy.Win32.Zbot.dol.exe</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>Trojan-Spy.Win32.Zbot.aaak.exe</td>
<td>Y</td>
<td>smss.exe</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Wink.bg.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>zbot.exe</td>
<td>Y</td>
<td>smss.exe</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ZeroAccess-z (dumped.dll)</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>zeus.melt.exe</td>
<td>Y</td>
<td>meyw.exe</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FUTO (manual)</td>
<td>Y</td>
<td>notepad.exe</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Summary of hidden objects detected by Linebacker (Windows 7 x86)

<table>
<thead>
<tr>
<th>Rootkit</th>
<th>Installed?</th>
<th>Process</th>
<th>File</th>
<th>Registry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backdoor.Win32.Ghost.binder.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backdoor.Win32.Ghost.20.exe</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backdoor.Win32.Ghost.21.exe</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backdoor.Win32.Ghost.23.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QVOD</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TDL4</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Trojan-Spy.Win32.Zbot.dol.exe</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>Trojan-Spy.Win32.Zbot.aaak.exe</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Wink.bg.exe</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>zbot.exe</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ZeroAccess-z (dumped.dll)</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>zeus.melt.exe</td>
<td>Y</td>
<td>leby.exe</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FUTO (manual)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

as a result of file permissions can be filtered out using a fixed list of directories. File hiding that occurs as a result of rootkits, and appears in other directories, is unusual and flagging such activity should produce few false positives. A drawback of this approach to reducing the number of false positives is that an attacker could design malware that hides in parts of the system that have the highest level of security. Unless Linebacker is run with this level of access, files hidden in these areas will be assumed to be hidden via normal system permissions rather than hidden by an attacker. Running Linebacker with administrator credentials on all the systems it monitors could be seen as creating an additional method for these systems to be compromised.

A key issue with file-based view comparison is that it only detects rootkits that exhibit a certain class of file-hiding behaviour. Such behaviour is far from ubiquitous. Unsophisticated rootkits may not make any attempt to avoid detection, thus view comparison will not detect them. Rootkits that employ HIPS strategies will not be detected for similar reasons. Some modern rootkits, such as TDL-4, use alternative evasion techniques that avoid creating file system objects entirely. Later versions of the TDL rootkit store the exploit in slack space at the end of a drive, then overwrite the MBR to load the exploit before the rest of the operating system (Lau 2013). This technique circumvents the Master File Table (MFT), so the view comparison approach will not detect these files — they will not appear in internal or external analyses of the file table. However, forensic analysis of the external view of the disk could detect changes in slack space.

Despite these issues, view comparison does detect some rootkits and provides information about where they hide on the system, making it a valuable addition to an arsenal of complementary detection tools. However, if a new system for monitoring VMs were to be developed, an approach based on offline full analysis of disk changes would be able to detect a wider range of threats than view comparison alone.

Registry Results: Our registry view comparison detected only a small number of entries. These appeared in both the “clean” and “dirty” scans, so they are not the result of object hiding by rootkits. Cuckoo observed that registry entries were created by some of the rootkits in our sample. We believe that these entries were not hidden as there is little benefit to cloaking registry entries from the guest operating system. Cloaking these entries would prevent them from having their intended effect, such as automatically re-launching the rootkit following a reboot.

Overall: The Linebacker tools detected a small proportion of the rootkits we tested. These rootkits use a range of techniques to prevent their detection, from HIPS through DKOM to hiding in slack space. The view comparison technique is only designed to detect a subset of these techniques. To get protection against a broad band of rootkits, Linebacker would need to be combined with other detection techniques such as traditional antivirus techniques and MBR.
protection techniques. The Linebacker tools tend to generate a large number of false positives that must be filtered out in some way. The registry and hard drive in particular have large amounts of ongoing change and deletion even on VMs that are only running the malware under test. The inability to perfectly synchronise the external and internal views of Linebacker is difficult to determine (without manual verification) whether or not a process detected by the memory analysis tool is a false positive or not. File permissions and UAC in Windows 7 can also cause large numbers of false positives (i.e. preventing Linebacker’s internal view from accessing files which are not actually hidden in a malicious sense). This is highlighted by the fact that our testing against known malware samples generated thousands of lines of marked-as-hidden output that needed to be filtered out before meaningful analysis could take place. This could potentially be an even greater problem for scaling view comparison up from single machines to enterprise systems, though in enterprise systems there is the potential to compare output across a set of identical systems to identify the anomalous behaviour caused by rootkits (Bianchi, Shoshitaishvili, Kruegel & Vigna 2012).

4 Performance Case Study

We conducted a series of tests to determine the performance of the Linebacker view comparison tools on a small scale set of production VMware ESXi infrastructure. These tests involved recording execution time metrics for the memory and registry components of Linebacker. Due to limitations of the older version of VMware ESXi used on our infrastructure, we were not able to run the memory tool’s view comparison techniques, nor could we execute the disk analysis tool. This is because these tools rely on later versions of the VMware Web Services SDK.

Our tests monitored a set of 9 VMs running the Windows 7 operating system, which were hosted on a cluster of VMware ESXi 4.1 servers running on IBM HS22V blades. The VMs were used for general office work by volunteer members of our research group for one month.

Memory: The time taken to prepare the analysis of memory from a series of VMs varied erratically throughout the analysis based on current network and server loads. A single preparation time includes the time taken to suspend and resume a VM-under-analysis and then download its VMSS file. The total time for all VMs to be prepared in a memory analysis batch job and the time taken to analyse the memory snapshots are given in Table 3. The high standard deviation for the total batch time highlights how erratic the performance of VMware API calls can be. We suggest that this behaviour is caused by surges in network activity slowing the VMSS download or heavy server disk activity slowing the time taken to suspend the VM and then write its VMSS file to disk.

The relatively low times for processing snapshots show that the offline memory analysis is not the performance bottleneck. This is largely expected as Linebacker uses the well tested and robust Volatility framework to extract memory artefacts from the VMSS files by efficiently parsing in memory Windows kernel structures. This suggests that if it were possible to analyse the memory images in place, large performance gains could be made.

There are noticeable performance impacts that arise as a result of capturing an up-to-date memory snapshot for the purpose of forensically examining it via the Linebacker memory tool. These arise as a result of ESXi hosts only creating a memory dump when a VM is suspended. As the entire memory is imaged in this process, the time taken to suspend then resume a VM is related to the size of the memory provided to that VM. The time taken is also related to the current disk and CPU load on the ESXi host. We received several reports from users of noticeable hitching or pausing as a result of the memory analysis tool.

It is worth noting that performance impacts are worse in ESXi versions 5.5 and later, as they immediately delete VMSS files upon resuming from a suspended state. In versions 5.1 and earlier the VMSS file lingered after the resuming of the VM and could be downloaded in the background for later analysis. In more recent versions, the VMSS file can only be captured to disk if the VM is paused for the duration of the capture time. This means that the execution of monitored VMs must be halted while the memory snapshot is created and downloaded from the hypervisor.

Our suggestion would be that given the performance impacts of capturing the up-to-date memory snapshot and the time taken to capture it, this analysis would only be run at off-peak times.

Disk and Registry: The disk and registry components of Linebacker rely on a single shared disk snapshot for their external view. Capturing snapshots had a negligible impact on users of the VMs in our trial. In some cases a small pause in execution lasting less than a second was observed by users during snapshot creation, although most users noticed no impact on the execution of their VMs at all.

The analysis time for the registry component of Linebacker includes generation of an internal registry view, capturing a disk snapshot and comparing the external registry view stored in the snapshot with the internal view. We recorded the total batch time to perform this analysis for all VMs in our case study, as well as the time taken to perform each step of this analysis on individual VMs. These times are summarised in Table 3.

We consider the time taken to analyse all VMs to be reasonable given the number of VMs being monitored. We believe the time taken to perform individual analysis steps shows that this approach can feasibly be implemented in production environments. While monitoring larger systems of hundreds or thousands of VMs was outside the scope of our case study, we expect that a scalable implementation could be achieved by performing the monitoring in a distributed manner across the system.

It is important to remember that the execution of monitored VMs continues throughout the disk and registry monitoring process, with the possible exception of a small pause at the start of the snapshot creation process.

5 Conclusions

Our efficacy results show that virtualisation assisted view comparison detects a limited subset of modern rootkits. We attribute this to malware authors either hiding objects outside the file system, or taking a HIPS approach and relying on the presence of other filesystem activity to obfuscate their malware. Linebacker detected six of thirteen rootkits on Windows XP, and two of seven rootkits on Windows 7. However, five of the rootkits tested used no object hiding. Two rootkits hid rootkit objects by altering existing files, registry entries or processes rather than creating new ones. One rootkit hid objects outside...
Table 3: Performance case study results (in seconds)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mean Time</th>
<th>Median Time</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory batch preparation (all VMs)</td>
<td>1292</td>
<td>715</td>
<td>1018</td>
</tr>
<tr>
<td>Analyse memory snapshot (per VM)</td>
<td>21.48</td>
<td>20.57</td>
<td>3.14</td>
</tr>
<tr>
<td>Batch registry analysis (all VMs)</td>
<td>787</td>
<td>837</td>
<td>274</td>
</tr>
<tr>
<td>Internal registry view (per VM)</td>
<td>11.3</td>
<td>11.0</td>
<td>3.9</td>
</tr>
<tr>
<td>Capture disk snapshot (per VM)</td>
<td>14.9</td>
<td>16.0</td>
<td>4.3</td>
</tr>
<tr>
<td>Registry view comparison (per VM)</td>
<td>56.5</td>
<td>55.0</td>
<td>11.5</td>
</tr>
</tbody>
</table>

the filesystem. Each rootkit that was able to hide new files or processes without going outside the filesystem to do so was detected by Linebacker.

The registry analysis component detected no rootkit evasion behaviour. We suggest that this is likely caused by the fact that authors can effectively maintain a persistent key in the registry without resorting to installing filter drivers or applying other evasion tactics. There is a large amount of existing noise in the registry due to general Windows OS activity. Hiding in this background “white noise” is relatively easy. Furthermore, employing evasion techniques such as API manipulation within the registry is risky due to the high volume of activity in the registry. A filter driver applied here by a rootkit is likely to noticeably degrade system performance, which may arouse the suspicion of users or administrators.

The disk analysis detected little rootkit evasion behaviour. With several rootkits not using object hiding, and one rootkit hiding in slack space, there were few cases where rootkits used hiding techniques detectable by Linebacker.

The memory analysis component did successfully identify rootkit evasion behaviours, however we found it difficult to discern true positives from false positives without manual analysis of the target VM. This was primarily due to processes starting and stopping between internal and external scans, and EPROCESS structures remaining resident in memory for an indeterminate time. We attempted to rectify this by checking the process end time field in this structure, however we have found that the field is often left empty.

There are fundamental issues with the use of view comparison to detect rootkits. The difficulties we experienced in synchronising the internal and external views of disk and memory cannot be resolved without modifying the hypervisor to support more direct access to the state of executing VMs. A large number of false positives are caused by this sync issue, in addition to those caused by file access permissions. Finally, a large number of rootkits, as shown by our sample set, either employ a HIPS strategy or use alternative hiding techniques that our tool is not designed to detect.

Our performance results highlight that disk and registry view comparison and VMI techniques can be implemented on VMware vSphere infrastructure with minimal impact. We also identified that memory-based process view comparison and VMI causes noticeable hitching and performance issues on monitored VMs. These issues arise from the need to suspend VMs in order to create up-to-date VMSS files. Unfortunately, creating these VMSS files cannot be avoided as they contain the atomic raw memory snapshot required for perform view comparison analysis on a VM.

If it was to be used in rootkit detection, view comparison would be best used as part of a complementary set of approaches. In spite of the issues with view comparison, we feel that VMI approaches as a whole provide a valuable source of “trusted” external view computer security audit data. That is, VMI and the VMware APIs provide an accurate view of the processes, network connections, sockets, files and registry contents on VMs. VMI may be a good basis for detecting object hiding techniques that alter or work outside the filesystem itself, in addition to those detected by view comparison. Memory-based VMI causes performance impacts in vSphere deployments and as a result care must be taken in determining when the analysis will take place. The disk and registry VMI tools can be applied in vSphere infrastructure with minimal impact.

Our plan in future is to use Linebacker as a basis for analysis of multiple machines across a network. In this case, rather than performing analysis within Linebacker, it will act as a data source for existing analytic tools. By comparing multiple similar machines on the same network, we hope to identify anomalous behaviour in single VMs and track how this behaviour moves through a network of VMs.

References


URL: https://www.encase.com


URL: http://doi.acm.org/10.1145/1698750.1698752


URL: http://libguestfs.org/hivex.3.html


URL: http://doi.acm.org/10.1145/1866307.1866314


Hardware Trojans – A Systemic Threat

John Shield, Bradley Hopkins, Mark Beaumont, Chris North
Defence Science Technology Organisation
PO Box 1500, Edinburgh 5111, South Australia

Abstract
Hardware trojans are a systemic threat that can impact the operations and infrastructure of corporations and government organisations. In this paper, we evaluate a credible and organisation-wide hardware trojan threat from compromised network cards. Our research examines the systemic threat of hardware trojans with an actual hardware trojan implementation to evaluate the impact. Our hardware trojan can degrade network services inside a corporate network, controllable from outside the network. An external activation mechanism is used to activate the trojan; the implementation bypasses data encryption, firewall packet inspection, and is agnostic to software protection and the operating system.

Keywords: Hardware Trojan, Networking, Systemic Effects, Threat Estimation

1 Introduction
Hardware Trojans are intentional, malicious modifications to electronic circuitry designed to disrupt operation or compromise security – including circuitry added into Integrated Circuits (ICs). These ICs underpin the information infrastructure of many critical sectors including the financial, military, and industrial sectors. Consequently, hardware trojans pose a security risk to organisations due to the broad attack surface and specific organisations’ reliance on ICT infrastructure. Hardware trojans can be difficult to prevent and even more difficult to detect (Beaumont, Hopkins and Newby 2011). Most of the current security protection mechanisms implicitly trust the hardware, allowing hardware trojans to bypass software or firmware security measures (Goertzel and Hamilton 2013). Hardware trojans inserted during fabrication or design stages can become widely dispersed within an organisation and pose a systemic threat.

There has been significant research exploring the threat of hardware trojans, particularly targeting the exploit of a single computer or electronic device (Rajendran and etc al. 2010, Chakraborty and etc al. 2009, Tehranipoor and Koushanfar 2010, Beaumont, Hopkins and Newby 2011). However, the research does not explore the broader effects that can be achieved through the systemic design of a hardware trojan attack. Such threat estimation requires additional considerations, such as trojan coordination, supply chain logistics, organisational processes, core-business, and security policies.

This paper contributes the construction, threat estimation, and analysis of a hardware trojan as a systemic-wide effector. Understanding the threat these hardware trojans pose to organisations paves the way for future security systems that will defend organisations against this threat. We present some of the key differences when a hardware trojan threat is scaled to an organisation level and highlight these differences with a network chip hardware trojan as an example. Describing our example trojan with a hardware trojan taxonomy (Tehranipoor and Koushanfar 2010) it is small, adds a functional change, externally activated, and modifies the bandwidth of the network card.

The remainder of this paper is outlined as follows: Section 2 describes the threat model; Section 3 discusses scaling considerations for the hardware trojan; Section 4 discusses related work in hardware trojans; Section 5 outlines our implementation of a network hardware trojan; Section 6 evaluates our hardware trojan and how it fits the threat model; Section 7 provides our conclusions and proposes future work.

2 Threat Model
Supply chain vulnerabilities are the key vector for hardware trojans to be placed into an organisation’s information infrastructure. A major vulnerability point in the supply chain is global manufacturing, which provides a pathway for hardware trojans to be placed in COTS (commercial off the shelf) information infrastructure. Whilst the designs for some ICs may be produced by trusted local engineers, the majority of IC and electronic component development and consequent manufacturing occurs in facilities outside of the control of the design vendor. These facilities are therefore considered untrusted and provide the opportunity for an adversary to add hardware trojans during manufacture, or further through the supply chain (Samuel 2008). The impact of this can be widespread due to the limited number of manufacturers. During 2013, the top foundry company supplied around 46% of the global market and the top 13 companies supplied 91% of the global market (IC Insights 2014). The problem of global manufacturing is exacerbated for countries such as Australia that lack the local industries and infrastructure needed for producing ICT hardware (Beaumont, Hopkins and Newby 2011). Furthermore, organisations usually have preferred suppliers and procurement procedures, which can assist an adversary in inserting hardware trojans into an organisation. In this paper, we are more interested in chip level hardware trojans; where the capability is inserted at, or prior to, chip masking. However, we don’t preclude...
other hardware trojans that require the addition, or modification, of physical circuits performed later in the supply chain. Once insertion of hardware trojans is achieved through the supply chain, a viable pathway for compromising an organisation’s information infrastructure is created.

Through physical or logical disruption, a hardware trojan can affect the confidentiality of information, integrity of information and availability of services throughout a business or organisation – undermining the operations and even reputation of the business. The impact of a hardware trojan attack can be long term and far reaching, in-part because there are few current security measures that can detect or counter the effects of malicious hardware (Beaumont, Hopkins and Newby 2011).

In this paper, we specifically explore the threat of a network hardware trojan to an organisation. Network chips are an ideal insertion point for a systemic hardware trojan, due to the supply and distribution characteristics. The number of suppliers for communication chips is limited. In 2013, Broadcom had a 40% market share in these chips (Wheeler and Bolaria 2013). Furthermore, network chips are ubiquitous in all critical infrastructure including, PCs, servers, switches, communications infrastructure and embedded devices. These factors significantly increase the likelihood that a network hardware trojan can gain widespread penetration within an organisation.

Network infrastructure forms a critical component of an organisation’s operations, even when the core business is not technology orientated. Examples include: email communication between internal staff and external stakeholders; accessing corporate information, such as client information, inventory and schedules; and software usage, which can either be in the cloud or require network access for licensing reasons. Network services are so pervasive in organisations that minor delays or outages can have cumulative impacts on all staff and external stakeholders, which can be crippling to an organisation’s core business.

3 Hardware Trojans at Scale
Previous research (Rajendran and etc al. 2010, Chakraborty and etc al. 2009, Tehranipoor and Koushanfar 2010, Beaumont, Hopkins and Newby 2011) looked at threats to individual electronic devices, but did not estimate the hardware trojan threat to an organisation’s processes and systems. When evaluating a hardware trojan at a systemic level instead of a device level there are three key differences that come from scaling:

- **Insertion** - The method of insertion should gain widespread penetration into an organisation. This widespread penetration and delivery of a hardware trojan ameliorates uncertainty of where and whether the trojan will be placed.

- **Activation** - Hardware trojan activation to achieve a systemic effect needs to consider infrastructure and security policies as hurdles to activation. The timing and reach of the activation mechanism also needs to be considered to achieve desired coordination and affect multiple disparate hardware trojans.

- **Effect** - Once activated, the hardware trojan needs to compromise organisational-wide processes rather than specific functionality on a single machine or device. The trojans need orchestration and coordination to have wide-reaching effects that cannot be achieved alone by a single hardware trojan.

**Insertion** vectors for achieving systemic effects need to achieve broad penetration to deal with the unpredictability of placement and provide scalability of the threat. Targeting specific machines or classes of components may not be possible, due to unpredictability of where the hardware trojan is eventually placed. To be effective, the compromised component needs to be generic and widespread within an organisation, although the trojan may only need a few instances to be effective. A widespread trojan improves the chances that it will be placed in a critical location.

Hardware trojan insertion vectors requiring physical interaction with individual machines will not scale to the desired penetration levels to achieve the kinds of systemic effects that are the focus of this paper. Methods for insertion of a widespread trojan need to occur before or during mass production, or during supply chain logistics. Insertion would ideally occur through compromised IP cores, chip designs, or added as part of the manufacturing process.

**Activation** of a systemic hardware trojan to achieve coordinated effects needs to account for the infrastructure and security policies of the organisation. Organisational security infrastructure can hamper specific types of activations signals, such as network data being blocked by firewalls and gateways. Organisational security policies can also block many side channel activation signals that require physical access or software access to the machine.

Consequently, the activation signal needs to be resilient, widespread and easily propagated to overcome the uncertainty of the hardware trojan’s placement and for coordinating the activation for multiple instances.

**Effect** of a systemic hardware trojan is most severe when it impacts the organisation as a whole, namely its core-business and processes. Factors that contribute to severity include sublety and enduring nature of effect, time cost incurred to discover, criticality of affected equipment and ability to remediate affected equipment with compatible hardware trojan free replacements. A traditional hardware trojan defines success as compromising very specific functionality of a machine or process. However, the compromised machine and the functionality may be unused within an organisation, or its effects may not reach beyond a single individual machine or person.

4 Related Work
There has been significant research work into describing and classifying hardware trojans (Rajendran and etc al. 2010, Chakraborty and etc al. 2009, Tehranipoor and Koushanfar 2010, Beaumont, Hopkins and Newby 2011). However, the literature is heavily weighted towards
hardware trojans that are designed and described as a threat to a single computer or device. In the literature, there has been minimal implementation or technical analysis for hardware trojans operating at a larger scale.

Some research implementations of hardware trojans are: CPU based hardware trojans (King and etc al. 2008, Wang and etc al. 2012) that can steal passwords, or break privilege protections; encryption hardware trojans (Lin, Burleson, and Paar 2009, Agrawal and etc al. 2007, Jin and Makris 2009) to extract secret keys; methods of DoS (Denial of Service) on general circuitry (Shiyanoavskii and etc al. 2009, Wei and Potkonjak 2013); adding communication channels using USB peripheral trojans (Clark, Leblanc, and Knight 2009); and adding communication channels using a network card trojan (Farag, Lerner, and Patterson 2012). These previous research implementations focus on how the hardware trojan can impact individual computers or devices, while our paper explores the wider effects of a hardware trojan implementation on an organisation.

Previous research into defence mechanisms against hardware trojans focuses on security of the hardware design and in-built detection methods (Tehranipoor and etc al. 2011). Some of these systems use in-built delay monitoring and power monitoring of the design against pre-calculated values (Wei, Kai, and Potkonjak 2012, Narasimhan and etc al. 2012). Data guards can also be used to prevent trojan activation by scrambling input data (Waksman and Sethumadhavan 2011). Additional software tools can be used to verify that the EDA tools create correct designs (Potkonjak 2010). These defence mechanisms against hardware trojans will eventually need to be evaluated in the context of systemic defence. Differences in implementation and operation of systemic hardware trojans could significantly impact effectiveness of the defence. Our work outlines some of the possible differences.

Previous research into hardware trojan implementations and defence mechanisms, only considers the security measures and impact for the immediate computer or device. It fails to consider defence mechanisms that can be implemented through procedures or guard electronics on an organisational scale. The lack of scale in the defence mechanisms is a side effect of hardware trojan implementations being narrowly focused on individual devices and machines. We hope to address this deficiency by exploring the design characteristics of large scale trojan threats.

5 Implementation

Our network hardware trojan performs remotely activated degradation of service, targeted at a RTL8111E Realtek Ethernet Controller chip.

5.1 Design Goals

Our design goals for the network hardware trojan were: easy insertion into the supply chain, simplicity in the design, small footprint to increase difficulty of detection, and broad and decentralised activation mechanisms.

Supply Chain: The trojan was designed to be easily inserted post-design during the chip manufacturing, or through another supply chain vulnerability. This is achieved by requiring no modifications to the original logic design and only accessing external signals on the IC for the design and implementation of our trojan. This fits our described threat model whereby manufacturing provides the opportunity for the broadest dissemination of a hardware trojan threat.

Simplicity: Although the chip handles gigabit ethernet, the trojan mechanisms used are simple and low frequency. This simplicity reduces the size and improves the robustness of the trojan system, and assists in making it easier to add to a manufactured chip.

Small: A smaller design creates a smaller footprint in the silicon, which is more difficult to detect. This in turn increases the chance that the trojan hardware activates and is used, and also increases the length of time it is present before potential detection and removal.

Systemic Activation: The activation signal needs to enable decentralised and widespread activation. This signal also needs to overcome common security measures such as firewalls and gateways.

5.2 Design

Hardware trojans are usually composed of a trigger and a payload (Chakraborty and etc al. 2009). The trigger is the activation mechanism and the payload generates the effect. Prior to triggering, a hardware trojan lies dormant without interfering with the operation of any electronics. The trigger mechanism for our network hardware trojan is based on a communication channel in network packet timing, while the payload is an adjustable degradation level of the ethernet channel through noise injection into the ethernet controller’s clock.

5.2.1 Trigger

The trigger mechanism, for the network hardware trojan, uses the Ethernet controller chip’s activity LED light as a method to access the packet timing. The activity LED is used to give a very broad indication to a user of the current network traffic. For the RTL8111E chip, network activity causes the LED light to cycle on then off over a 160 millisecond period. There is a delay between these 160 millisecond cycles when there is no network activity. The behaviour of the Ethernet Controller is shown in Figure 1.

![Figure 1: Ethernet controller chip LED behaviour](image)

The timing behaviour of the activity LED is used as a communication channel to trigger the hardware trojan. Sending network packets at different intervals allows a user to modulate the period of the LED activity, which can then be used to encode the data to trigger the
hardware trojan. Figure 2 shows how different network activity is able to modulate the period of the activity LED.

This simple timing channel contains noise from normal network traffic. To overcome the impact of noise, a sufficiently long activation code can prevent false positives. Repeated signalling can also overcome noise in receiving the signal. In a few cases, the LED timing channel is absent, due to continuous network activity. However, most systems do not continuously communicate and a signal can be received during any breaks in normal communication, provided a sufficiently robust protocol is utilised.

![Figure 2: Communication timing channel through trigger packets](image)

For the RTL8111E chip, the period of the activity LED can be obtained with a coarse sampling of the signal, as shown in Figure 3.

![Figure 3: Sampling of LED Timing](image)

The samples are matched against two pre-defined trigger sequences implemented in our hardware trojan. There is an activation sequence that increments a counter, which controls the network degradation level, and a deactivation sequence that resets the trojan.

The left half of Figure 4 shows logic for sampling. This generates a pulse every 21 milliseconds, based on the counter size and the 25MHz input clock from the RTL8111E chip (CLK). The sampling is also synchronised to the rising edge of the LED output of the RTL8111E chip.

The right half of Figure 4 shows logic for sample matching. The sample rate pulses are used to clock in the LED state into a shift register. To determine whether the sequence matches the trigger or reset signals, comparators are used to check for the rising edges for pre-defined sequences in timing.

5.2.2 Payload

The network hardware trojan payload performs a degradation of network services. It uses noise injection into the ethernet controller chip’s clock circuitry in the form of a bias voltage. This voltage slightly changes the resonant frequency on the external crystal. The change in frequency desynchronises the clock of the ethernet controller chip with the ethernet channel. This causes bit errors in the ethernet channel. Figure 5 shows simple bias voltage circuitry that can be directly fed into the crystal. Our demonstrator (described later) uses a pulse-width modulation (PWM) source where the pulse width sets the bias voltage. Figure 6 shows how an adjustable PWM can be generated using a small number of gates. Figure 7 shows where the hardware trojan injects the bias voltage into the standard crystal clock circuit. The adjustable bias voltage allows for variable degradation of the ethernet channel.

![Figure 4: Trojan Circuitry – Trigger](image)

![Figure 5: Trojan Circuitry – Payload Potentiometer](image)

![Figure 6: Trojan Circuitry – Payload PWM Generator](image)
The hardware trojan is designed to minimise the size needed for the implementation, facilitating easy hardware modification and making it more difficult to detect. For our demonstrator, we implemented the hardware trojan in functionally similar firmware instead of circuitry.

5.3 Demonstrator

For our demonstrator of the hardware trojan, we used the ENW02A-1-BC01 Gigabit Ethernet PCI-Express Card. The RTL8111E controller chip is part of this card. We implemented our hardware trojan externally on a PIC16F690 Development Board and attached it to the pins of the controller chip. The experimental setup can be seen in Figure 8. An actual hardware trojan would be added inside the ethernet controller chip, most likely during manufacture.

The trigger detection sequence utilises the PIC timers to measure the period between rising edges. These periods are compared using coarse values against an expected sequence of delays.

The payload for the implemented version of the hardware trojan is achieved via an adjustable PWM output signal generated by the PIC. This is injected through a resistor onto one of the clock crystal inputs (Figure 7).

6 Evaluation

6.1 Network Performance Adjustability

Figure 9 shows the range of degradation effects the hardware trojan can implement. The range of settings allows for a spectrum of disruption, from minor network slowdown to complete disabling of access.

The effects of our network hardware trojan are extremely difficult for users and IT support staff to debug even while activated. During our testing there was no error reporting to the user of a problem (under the Ubuntu OS), until the hardware trojan was set to completely disable the network card. Furthermore, the amount of reported TCP/IP packet loss was minimal even under high network degradation. This is attributed to the operation of the Transport Control Protocol (TCP), whereby packet losses are treated as being caused by congestion and this reduces bandwidth. Academic studies (Kumar 1998) have shown that dropping 1 in every 70 packets causes degradation of bandwidth performance in the order of 50%. These characteristics make our trojan difficult to isolate, even whilst operating.

6.2 Threat Effectiveness

The network trojan is well suited to implement a systemic effect. It has the desired insertion, activation, and effect characteristics to provide a coordinated and disruptive organisation-wide attack.

Insertion: The network hardware trojan can be easily inserted into operation within an organisation. Firstly, there are a limited number of suppliers of ethernet controller chips making it easier to compromise the supply chain. One company supplies 40% of the communication chip market (Wheeler and Bolaaria 2013). Consequently, a potential compromise within one company could result in significant hardware trojan penetration within an organisation’s infrastructure. Such a compromise may be from a dissatisfied employee, organised group, or state sponsored actor. Secondly, network enabled devices are ubiquitous in today’s
organisational environments, providing numerous and widely spread device locations that can be compromised.

**Activation:** The common security policies within organisations will not limit our network hardware trojan and the activation method scales up well. Firstly, the activation method of the hardware trojan occurs before any software protections and is not reliant on software or other hardware. This allows the trojan to work irrespective of where the ethernet controller chip is located. Secondly, the signal is also based purely on packet timing and ignores data content, allowing it to bypass the data encryption, packet inspection and port blocking commonly found on firewalls and gateways. It will also work in the presence of packet encryption. Finally, the activation signal scales easily using the existing network hierarchy. The signal is simple enough to easily replicate and can activate all intermediate network card trojans as it propagates through the network. The signal can be blocked by noise in the form of other network traffic. However, the activation can still be received, provided a sufficiently robust protocol is utilised.

**Effect:** Our network hardware trojan is targeted at disrupting organisational operations. Firstly, a degradation of networked services can significantly reduce organisational efficiency by slowing down communication, information access and limiting software usage. Networked devices can include: servers, desktops, gateways, routers, faxes, phones and printers. Secondly, multiple network cards are chained together in connecting any service and it only takes a single network card within the chain to adversely affect the service. Thirdly, the variable effect of the trojan with network degradation and possibly intermittent behaviour encourages temporary workarounds rather than directly addressing the problem. These workarounds may prove more costly for an organisation in the long run, taking up more time and encouraging departure from standard procedures. Departure from standard procedures may create new security vulnerabilities, such as using a personal email for business activities due to the corporate email being too slow. Finally, the trojan can remain unidentified for a long period. Degradation of networked services can come from any number of factors and in most cases, denial of service attacks aside, these are typically the result of a single hardware, or software failure. Having a wide attack surface, coupled with an intermittent effect will severely retard technical support within the organisation identifying the nature of attack and consequently isolating it.

7 Conclusions & Future Work

In this paper, we have outlined a natural development for hardware trojan research into systemic effects. Currently, there is a narrow focus on the threat estimation to individual machines, with little analysis on systemic effects of hardware trojans targeted at an organisational level. For our contribution, we have designed and implemented a network hardware trojan to better characterise the threat posed by organisation-wide hardware trojans. The trojan we designed was small, easy to implement and can be leveraged to provide coordinated and variable disruption to most organisations.

Our research has demonstrated that there are key differences, in insertion, activation, and effect, when scaling hardware trojan effects from an individual computer to an organisation. We have outlined the key differences as a basis upon which later work can build upon in researching and developing appropriate defence mechanisms.

Future work will look at additional system-wide threats and recommended policies and protection mechanisms that could be implemented by organisations.

8 References


Contributed Posters
Information Privacy Concerns of Real Estate Customers and Information Security in the Real Estate Industry: an Empirical Analysis

Deepa Mani, Alireza Heravi, Kim-Kwang Raymond Choo, Sameera Mubarak

School of Information Technology & Mathematical Sciences,
University of South Australia
Adelaide, South Australia
deepta.manil@mymail.unisa.edu.au, alireza.heravi@mymail.unisa.edu.au,
Raymond.choo@unisa.edu.au, Sameera.mubarak@unisa.edu.au

Abstract
An organisation in the real estate industry that uses information systems for its daily business operations for storing and transferring customers’ data is a potential target of cyber-attacks. However, real estate is an understudied industry in terms of information security and customers’ information security privacy concerns. In this paper, we surveyed 82 Australian real estate customers to explore their privacy concerns when providing personal information to real estate agencies and the conditions that they are willing to provide such information. We also interviewed 20 real estate businesses to understand their current information security practices. Our findings suggested that customers are naturally concerned when providing personal information to real estate agencies and that trust plays a key role. Our findings also highlight the need for real estate organisations to enhance their information security practices.

Keywords: Information privacy, Information security, Real estate organisations, Social penetration theory.

1 Introduction
Information privacy (e.g. ensuring the security and privacy of user data) is a topic of ongoing research and policy interest, particularly when our data are increasingly collected by a wide range of public and private sector organisations. In Australia, for example, customers provide or disclose personal and financial information (e.g. copies of their bank statement and passport) when submitting rental applications or when they are selling their properties to real estate firms. Customers are generally not aware how their information (electronic, scanned or physical documents) will be stored / secured, when and how their information will be disposed of and the security of the devices used to access, store and disseminate personal information. Similar to other service industries, real estate organisations use information systems for their daily business operations but information security is generally not their business priority. This is particularly true for small and medium-sized real estate businesses and many of the organisations outsource their information technology/security functions. To reduce the risk to customers’ data, this paper aims to contribute to an in-depth understanding of information security practices in real estate organisations. As shown in Figure 1 this paper consists of two studies.

2 Study 1
To understand customers’ concerns regarding information privacy when providing personal information and the conditions in which they are willing to do so, we use the five constructs outlined in Figure 2. In that the perceived risks/benefits and self-disclosure construct are based on social penetration theory. Note that the aim of this study is not to test the theory; rather it is used to guide the study.

2.1 Theoretical Background
The Social Penetration Theory suggests that relationships develop from lower levels of intimacy (superficial self-disclosure) to higher levels of intimacy (greater level of self-disclosure), and finally to disengagement (withdrawal of disclosure) (Altman & Taylor 1973). This theory links self-disclosure to interpersonal relationship development using a cost-reward approach. Individuals evaluate their relationship/interactions with others and if the associated costs are perceived to be less than the rewards, then the relationship/interaction is considered satisfactory and vice versa (Giri 2009).

2.2 Hypotheses
This study focuses on the information privacy, which is defined as: “the interest an individual has in controlling, or at least significantly influencing, the handling of data about themselves” (Clarke 2006). Information privacy concerns have attracted the attention of academics, and government agencies as rapid advances in information and communications technologies (ICT) have facilitated wide scale collection, aggregation and analysis of data (Malhotra, Kim & Agarwal 2004). Cost-rewarding and self-disclosure are two factors discussed in the theory described in Section 2.1. Cost-rewarding refers to the calculation of costs (risks) and rewards (benefits).
In this study, we adopted the term “risk-benefit” as this is more reflective of the customer-real estate organisation relationship. We will investigate the risk’s effect on customers’ information privacy concerns when providing personal information to the real estate agency in this study. Therefore, we posit the following hypothesis:

H1: Risks perceived by customer will have a positive effect on his/her information privacy concerns on providing personal information to the real estate organisation.

H2: Benefits perceived by customer will have a positive effect on his/her information privacy concerns on providing personal information to the real estate organisation.

Self-disclosure is defined as “any message about the self that a person communicates to another” (Wheelees & Grotz 1976). In this study, we investigate whether being open or closed (high/low level of self-disclosure) affects one’s information privacy concern when providing personal information to the real estate organisation. Therefore, we posit the following hypothesis:

H3: The degree of self-disclosure by the customer will have a positive effect on his/her information privacy concerns on providing personal information to the real estate organisation.

In the cost-rewarding calculation, it is believed that trust is used, and high trust correlates with low cost and vice versa (Dwyer, Hiltz & Passerini 2007). According to Metzger (2004), high trust reduces the perceived risks of self-disclosure. In this study, we investigate whether customers’ trust in the real estate organisation affects their information privacy concerns when they provide personal information. For this reason we posit the following hypothesis:

H4: Customer’s trust in the real estate organisation will have a negative effect on his/her information privacy concerns on providing personal information.

In the current ICT-enabled society, most of our communications take place over the Internet, and real estate companies are no exception. Customers are likely to email these organisations regarding property inquiries, provide electronic or scanned copies of documents, etc. (Mani, Choo & Mubarak 2014), and such information will be stored on the organisation’s systems. Therefore, customer’s general online privacy concerns may have an impact on his/her information privacy concerns when the customer provides personal information. Therefore, we posit the following hypothesis:

H5: The degree of online privacy concerns of customer will have a positive effect on his/her provision of personal information to the real estate organisation.

2.3 Data Collection

An online questionnaire was sent to students and staff of the University of South Australia. A total of 82 respondents who had rented/bought/sold a property through a real estate company in South Australia in the last 12 months participated in the survey. The scales were refined based on the pilot study results. The final questionnaire consists of eight sections. Section one collects the demographic information about the participants – see Table 1. The second section is designed to determine the level of trust in the real estate organisation(s) that the respondents had dealt with. The third and fourth sections are designed to understand the perceived risks, and the respondents’ information privacy concerns. The last four sections focused on information sensitivity, perceived benefits, disclosing personal information, and online privacy concerns of the respondents.

<table>
<thead>
<tr>
<th>Category</th>
<th>Subcategory</th>
<th>Frequency</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
<td>Male</td>
<td>50</td>
<td>61%</td>
</tr>
<tr>
<td></td>
<td>Female</td>
<td>32</td>
<td>39%</td>
</tr>
<tr>
<td>Age</td>
<td>18-24</td>
<td>15</td>
<td>18%</td>
</tr>
<tr>
<td></td>
<td>25-34</td>
<td>40</td>
<td>49%</td>
</tr>
<tr>
<td></td>
<td>35-44</td>
<td>18</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td>45 or older</td>
<td>9</td>
<td>11%</td>
</tr>
<tr>
<td>Country of origin</td>
<td>Australia</td>
<td>25</td>
<td>31%</td>
</tr>
<tr>
<td></td>
<td>Malaysia</td>
<td>9</td>
<td>11%</td>
</tr>
<tr>
<td></td>
<td>India</td>
<td>10</td>
<td>12%</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>38</td>
<td>46%</td>
</tr>
<tr>
<td>Education</td>
<td>High School</td>
<td>12</td>
<td>14%</td>
</tr>
<tr>
<td></td>
<td>Bachelor Degree</td>
<td>16</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Honours Degree</td>
<td>6</td>
<td>07%</td>
</tr>
<tr>
<td></td>
<td>Masters Degree</td>
<td>31</td>
<td>38%</td>
</tr>
<tr>
<td></td>
<td>Doctorate Degree</td>
<td>16</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>1</td>
<td>01%</td>
</tr>
</tbody>
</table>

Table 1: Respondents’ demographic details

2.4 Findings

The individual constructs of the data were tested for reliability, convergent and discriminant validity. The
calculations were performed using the SmartPLS software package. For reliability analysis, Cronbach’s alpha was tested to analyse the constructs’ internal consistency measure for each construct. The constructs ranged from 0.75 to 0.95. Since all the constructs exceeded the 0.70 cut-off values, the recommended threshold for construct reliability is exceeded.

<table>
<thead>
<tr>
<th>Construct</th>
<th>CR</th>
<th>AVE</th>
<th>PR</th>
<th>PB</th>
<th>SD</th>
<th>TR</th>
<th>OPC</th>
<th>IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>0.97</td>
<td>0.95</td>
<td>0.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB</td>
<td>0.87</td>
<td>0.77</td>
<td>0.25</td>
<td>0.88</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>0.86</td>
<td>0.62</td>
<td>0.24</td>
<td>0.39</td>
<td>0.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR</td>
<td>0.88</td>
<td>0.66</td>
<td>-0.13</td>
<td>0.22</td>
<td>0.15</td>
<td>0.81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPC</td>
<td>0.87</td>
<td>0.70</td>
<td>0.34</td>
<td>0.15</td>
<td>0.20</td>
<td>-0.5</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>IPC</td>
<td>0.91</td>
<td>0.66</td>
<td>0.56</td>
<td>0.09</td>
<td>0.14</td>
<td>-0.38</td>
<td>0.47</td>
<td>0.81</td>
</tr>
</tbody>
</table>

*Table 2: CR, AVE, and Inter item correlations*

The convergent validity was also assessed using the Average Variance Extracted (AVE). The AVE and the Composite Reliability (CR) of all the constructs were above 0.5 and 0.6 respectively, indicating sufficient convergent validity as shown in Table 2. Likewise, the square root of the AVE reveals that the value of each construct is larger than its correlation with other constructs and, thus, satisfies the discriminant validity of the constructs. Since all the items had adequate reliability and validity, all the measurement items were used to test the structural model. The Partial Least Squares (PLS) and bootstrapping test were respectively used to determine the hypothesised path (and the path coefficient, $\beta$) and to estimate the path significance using $t$-values. The computed results are shown in Figure 3.

![Figure 3: Results of the Structural Model Testing](image_url)

Note: The dotted line indicates that the item is not significant. * indicates that the item is significant at $p<0.01$

### 2.5 Discussion

In study 1, we examined perceive risk/benefit, self-disclosure, trust and online privacy concerns regarding customers’ information privacy concerns. The analysis results suggest that three out of five hypotheses are supported. H1 and H2 propose there is a positive relationship between perceived risk/perceive benefit and real estate customers’ information privacy concerns. H1 ($\beta=0.356, P<0.01, t=6.967$) was supported but H2 ($\beta=0.008, P=NS, t=0.183$) was not. This indicates that while customers’ perceived risk influences their concerns about information privacy, the perceived benefits do not. Therefore, it can be concluded that customers are more concerned about the potential risks of providing personal information rather than the benefits that they might gain when dealing with a real estate organisation. To our surprise, H3 ($\beta=0.047, P=NS, t=1.259$) was not supported. This suggests that self-disclosure does not have a significant impact on information privacy concerns. In other words, being open or closed (high/low level of self-disclosure) does not affect one’s information privacy concerns when providing personal information to a real estate firm. H4 ($\beta=-0.291, P<0.01, t=6.700$) was supported; as expected when individuals trust the company they are dealing with, they will be less concerned about their information privacy. Finally, H5 ($\beta=0.325, P<0.01, t=8.162$) was supported; indicating that online privacy concerns have a significant effect on information privacy concerns. Therefore, people who do have concerns about privacy when they are online are more likely to be concerned about information privacy when dealing with a real estate agency.

### 3 Study 2

In study 2, we analysed real estate organisations’ information security practices in terms of people, process, and technology. For data collection, 50 South Australian real estate organisations, members of Real Estate Institute of South Australia (REISA), were contacted by email. Of these 50 organisations, 20 participated in the semi-structured face-to-face interviews. Each interview took approximately 30 minutes to do and they were audio-recorded and transcribed verbatim for analysis using comparison and contrast methods.

#### 3.1 Current Information Security Practices in Real Estate Organisations

To protect a real estate company’s business it is necessary to address the three important aspects of business operations, namely people, process and technology. These aspects play a major role in protecting the confidentiality and integrity of customer information.

##### 3.1.1 People

People are the real estate organisations’ employees who use the technology and access customer information. To protect customer information, it is important that employees have adequate training/qualification in information security. All 20 participants reported that their organisations provide induction training for all new employees. However, only six participants indicated that their organisations include a basic information security module in the induction training. It is important for real estate organisations to conduct regular information security training for all employees and create a culture of security to ensure that employees are kept abreast of recent cybercrime threats and is equipped to respond to such threats (Imgraben, Engelbrecht & Choo 2014).

##### 3.1.2 Process

An effective process needs to have policies that are specified as well-defined documents. Since the core business of real estate organisation is not in information technology, it is unsurprising that these organisations do not have any information security policies. Half of the participants were either not aware whether there is an acceptable IT technology use policy or reported that their
organisation does not have such a policy or bring their own devices (BYOD) policy. This is despite BYOD being a norm in all the 20 organisations. The real estate organisations collect significant amount of personal information about their customers; therefore, it is essential to have policies for document retention and disposal. Of the 20 interviewed participants, six mentioned that their organisation never deletes customer information. Twelve participants reported that their organisations kept customer information for five to seven years, and the remaining two organisations reported that customer information is kept for only a year. However, when asked about deleting electronic or scanned files, only two organisations reportedly used wiping software to delete such files. It is necessary for real estate organisations to have an appropriate media disposal best practice, particularly for electronic media, as data can be forensically recovered from unwiped media (Quick, Martini & Choo 2013).

3.1.3 Technology
Smart mobile and portable devices are commonly used in the real estate industry. Several organisations allowed BYOD. However, only half of the interview participants indicated that their organisations have an acceptable BYOD policy in place. When asked about security protections on the mobile devices used for work purpose, 85% indicated that their devices are protected with a password, 55% reported having antivirus software installed on their mobile devices, and only 20% mentioned they have remote wiping apps installed. Over 85% of participants reported incidents involving malware attacks on their work computers and other hardware and/or had their hardware such as laptops, CCTV cameras, and mobile and other portable devices lost or stolen.

3.2 Discussion
Our findings suggest that the level of employee security awareness is generally inadequate. For example, using mobile and portable devices for both work and personal purposes is a norm in these organisations, but they did not have any security measures in place to ensure the security and privacy of data that such devices could be used to access once lost, stolen or compromised. Only half of the interview participants reported that their organisations have acceptable mobile device use policies. Anti-virus software is common in personal computers but not on their mobile devices. In our study, 13 (65%) participants reported incidents of malware infection on their hardware which did have antivirus installed. This highlighted the fact that installing antivirus software is not sufficient on its own. It is necessary to ensure that the software is updated regularly with the latest signature, as well as the need to introduce security hygiene. It is also necessary for organisations to educate the importance of incident reporting as well as how to report such incidents, so that future events may be prevented.

4 Conclusion
In this paper we found that: (1) individuals are more concerned about the potential risks than the benefits that they may gain when providing personal information to a real estate organisation; (2) there is no difference between an open and a closed (high/low level of self-disclosure) individual regarding information privacy concerns when dealing with a real estate organisation; (3) the more a customer trusts a real estate organisation, the less concerned he/she will be about information privacy; and finally real estate customers who were more concerned about their privacy when they were online would be more likely to worry about their information privacy. In study 2, our findings highlighted the need for such organisations to invest more in information security, and understand that information security is not a cost because it can deliver business benefits. It is clear from study 1 that the customers are generally concerned about how their information is protected. However, findings of study 2 indicate that real estate organisations are not taking the necessary precautionary steps to protect customer private information.

5 References
Correcting flaws in Mitchell’s analysis of EPBC

Binbin Di, Leonie Simpson, Harry Bartlett, Ed Dawson, Kenneth Wong
Science and Engineering Faculty
Queensland University of Technology
2 George St, Brisbane 4000, QLD

binbin.di@hdr.qut.edu.au, lr.simpson@qut.edu.au, h.bartlett@qut.edu.au,
e.dawson@qut.edu.au, kk.wong@qut.edu.au

Abstract

Efficient error-Propagating Block Chaining (EPBC) is a block cipher mode intended to simultaneously provide both confidentiality and integrity protection for messages. Mitchell pointed out a weakness in EPBC and claimed that this permits a forgery attack. This paper corrects a flaw in Mitchell’s analysis and shows that the attack is no better than brute force of the integrity check vector.

Keywords: block cipher, authenticated encryption, EPBC, forgery attack.

1 Introduction

Efficient error-Propagating Block Chaining (EPBC) (Zuquete and Guedes 1997) is a mode of operation for block ciphers that is intended to provide authenticated encryption (AE). EPBC can be used with any block cipher. The plaintext is divided into blocks as defined by the selected block cipher. A predefined Integrity Check Vector (ICV) is appended to the plaintext message and the message is then encrypted in EPBC mode. When the ciphertext is decrypted, the receiver checks the correctness of ICV. Any change to the ciphertext should propagate erroneous decryptions to all subsequent ciphertext blocks, resulting in the decryption to an incorrect ICV (Zuquete and Guedes 1997), as shown in Figure 1. Messages with an incorrect ICV are rejected by the receiver.

![Figure 1: Integrity mechanism (Recacha 1996)](image)

Recently, Mitchell analysed EPBC, pointing out a weakness in the integrity mechanism and proposed a forgery attack based on this weakness (Mitchell 2007). He claimed that knowing sufficient plaintext/ciphertext pairs permitted the inner vectors, used to conceal plaintext patterns, to be disclosed with very high probability. Once these inner vectors were known, a forgery could be constructed. However, we show that his calculation is inaccurate, and the probability of a successful forgery is no better than that of guessing the ICV.

2 Description of EPBC

EPBC is a mode of operation for an n-bit block cipher, for even n, say n = 2m. Two secret keys denoted K and K′ are used. One key, K, is used for encryption and decryption. Let ek (P) denote the encryption of the plaintext block P and dk (C) denote the decryption of the ciphertext block C under the key K. The second secret key, K′, and a sequence number S are used to generate a pair of secret n-bit initial vectors denoted by F0 and G0, where F0 = ek (S) and G0 = ek (F0), which are used for encryption and decryption of the first block.

The EPBC encryption operation is defined as follows:

\[ G_i = P_i \oplus F_{i-1}, \quad 1 \leq i \leq u, \]
\[ F_i = e_k (G_i), \quad 1 \leq i \leq u, \]
\[ C_i = F_i \oplus g(G_{i-1}), \quad 2 \leq i \leq u, \]

where \( C_i = F_i \oplus G_i \) and g is a function applied to the two m-bit halves of the n-bit block. More precisely, suppose \( X \) is an n-bit block, where \( X = L || R \). \( L \) is the high order m-bit block and \( R \) is the low order m-bit block (|| denotes concatenation). Then g is defined as follows:

\[ g(X) = (L \lor \overline{R}) \lor (L \land \overline{R}) \]

where \( \lor \) and \( \land \) denote the bitwise inclusive or and logical and operations respectively, and \( \overline{X} \) denotes the bitwise inverse version of \( X \). This process is shown in Figure 2.

![Figure 2: EPBC encryption (Mitchell 2007)](image)

The decryption operation is simply a reverse process of the encryption as follows:

\[ F_i = C_i \oplus g(G_{i-1}), \quad 2 \leq i \leq u. \]
\[ G_i = d_k (F_i), \quad 1 \leq i \leq u, \]
\[ P_i = G_i \oplus P_{i-1}, \quad 1 \leq i \leq u. \]
where \( F_i = C_i \oplus G_i \).

Verifying the integrity is done simply by checking the last \( l \) bits of recovered plaintext (where \( l \) is the length of the ICV). If this matches the expected value of the ICV, the message is regarded as authentic.

3 Review of Mitchell’s analysis

Mitchell’s forgery attack (Mitchell 2007) on EPBC aims to forge a ciphertext in such a way that the forgery is not detected by the integrity mechanism. This is an existential forgery (Preneel 1998). In order to achieve this, the attacker has to construct a message such that the last ciphertext block will decrypt to the correct ICV value. The inner vectors, \( F \) and \( G \), in EPBC ensure the integrity protection by propagating inaccurate decryptions from any tampered ciphertext blocks through to the end (Zuquete and Guedes 1997). Zuquete and Guedes (1997) note that the forged ciphertext blocks must be constructed to adjust values of the inner vectors during the decryption process, to permit correct decryption of the ICV.

The function \( g \) in EPBC is critical in protecting the contents of the inner vectors from discovery. Mitchell’s analysis is composed of two stages: investigating a vulnerability of the function \( g \) which can be used to reveal the inner vectors and then using this knowledge to construct a message which will not be detected as a forgery by the integrity mechanism of EPBC.

3.1 Mitchell’s analysis of function \( g \)

This stage aims to use knowledge of a series of plaintext and ciphertext pairs to disclose the inner vectors, \( G \). Knowledge of \( G \) permits a forgery attack on EPBC mode. The process of constructing a forged ciphertext is outlined in Sect. 4.

Properties of the function \( g \) are used to reveal the contents of the inner vectors \( G \). Suppose \( X \) is an \( n \)-bit block, where \( X = (x_1, x_2, \ldots, x_n) \) and \( R = (x_{m+1}, x_{m+2}, \ldots, x_{2m}) \). Also, suppose \( g(X) = L \parallel R \) where \( L = (x_1, x_2, \ldots, x_m) \) and \( R = (x_{m+1}, x_{m+2}, \ldots, x_{2m}) \). Because \( g \) applies bitwise operations to the two \( m \)-bit halves \( (L \text{ and } R) \) of each block, it can be treated as \( m \) parallel operations on pairs of bits \( (x_j, x_{j+m}) \), where \( x_j \) is the \( j \)-th bit of the block and \( x_{j+m} \) is the \( (j+m) \)-th bit of the block, for \( j = 1, \ldots, m \). Table 1 (modified from Mitchell, 2007) shows the set B of possible output pairs \( (x_j, x_{j+m}) \) that can be obtained after applying \( g \) to each possible set A of input pairs \( (x_j, x_{j+m}) \). Sets in column A are grouped by the number of alternatives in each set. We will explain later why we have separated group 2 into subsets 2a and 2b.

Assume also \( (x_j, x_{j+m}) \) is a bit pair in inner vector \( G_{i-1} \), where \( j \) \((1 \leq j \leq m)\) is a randomly chosen bit position in an \( n \)-bit block. There are four possible values for this bit pair (listed as group 4 in Table 1). Mitchell (2007) notes that the set of output bit pairs from the function \( g \) can never include the specific bit pair \( (0, 1) \). Thus, the pair in position \( (x_j, x_{j+m}) \) of \( g(G_{i-1}) \) can only take one of the values listed (for group 4) in column B of Table 1. Because of this, we can also narrow possible bit pairs in position \( (x_j, x_{j+m}) \) in \( G_{i-1} = P_{i-1} \oplus C_i \oplus g(G_{i-1}) \) from four to three, where the bits in position \( (x_j, x_{j+m}) \) of \( C_i \oplus P_{i-1} \) determine which set of three pairs is relevant in each individual case. Similarly, when \( G_{i-1} \) runs through the function \( g \), either three (50% chance) or two (50% chance) alternatives result for the bit pairs in position \( (x_j, x_{j+m}) \) in \( g(G_{i-1}) \).

<table>
<thead>
<tr>
<th>Group</th>
<th>Input pairs A</th>
<th>Output pairs B</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>(0, 0) (0, 1) (0, 0) (1, 1)</td>
<td>(0, 0) (0, 1) (0, 1)</td>
</tr>
<tr>
<td>3</td>
<td>(0, 0) (0, 0) (0, 1) (1, 1)</td>
<td>(0, 0) (0, 1) (1, 1)</td>
</tr>
<tr>
<td>2b</td>
<td>(0, 0) (0, 1) (1, 1)</td>
<td>(0, 0) (1, 1)</td>
</tr>
<tr>
<td>2a</td>
<td>(0, 0) (0, 1)</td>
<td>(0, 0) (1, 1)</td>
</tr>
<tr>
<td>1</td>
<td>(0, 1) (0, 1)</td>
<td>(0, 1)</td>
</tr>
<tr>
<td></td>
<td>(0, 0) (0, 0)</td>
<td>(1, 0)</td>
</tr>
</tbody>
</table>

Table 1: Input/output possibilities for the function \( g \)

(Mitchell, 2007, modified)

If the bit pair in position \( (x_j, x_{j+m}) \) of \( g(G_{i-1}) \) has two alternatives, so will the bit pair in this position in \( G_{i+1} = P_{i+1} \oplus C_i \oplus g(G_{i-1}) \). Mitchell (2007) argues that the output bit pairs in \( g(G_{i-1}) \) will then either have two alternatives (5/6 chance) or one alternative (1/6 chance). Finally, if there is one alternative in the input, the output pairs of the function \( g \) have only one alternative. According to this argument, the possible alternatives for each bit pair in \( G_{i+1} \) will eventually be reduced to a single (known) alternative if sufficiently many staggered plaintext/ciphertext pairs \( (C_1, P_1), \ldots, (C_{e+2r-1}, P_{e+2r-1}) \) are known.

Based on Table 1, Mitchell (2007) proposed a matrix (shown in Figure 3) for the probability of transitions between the different groups in Table 1. The entries in row \( i \) and column \( j \) in the matrix denote the probability that there are \( j \) possible output bit pairs from the function
g, given that there were \( i \) possible input bit pairs. For example, for a set of three input bit pairs (3rd row) the output will be either three bit pairs (3rd column) with 50% chance or two bit pairs (2nd column) with 50% chance.

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 0 & 0 \\
2 & 1/6 & 5/6 & 0 & 0 \\
3 & 0 & 1/2 & 1/2 & 0 \\
4 & 0 & 0 & 1 & 0 \\
\end{array}
\]

**Figure 3: Mitchell’s transition probability matrix**

Mitchell used this theory to launch an attack. We discuss the methodology in section 4.

### Table 2: Probability of a unique possibility for a bit pair and a 128-bit block (Mitchell 2007)

<table>
<thead>
<tr>
<th>( \nu )</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>0.71027</td>
<td>0.95305</td>
<td>0.99341</td>
<td>0.99978</td>
<td>0.99980</td>
<td>0.99987</td>
</tr>
<tr>
<td>( q' )</td>
<td>Very small</td>
<td>0.00467</td>
<td>0.01400</td>
<td>0.02485</td>
<td>0.07878</td>
<td>0.09888</td>
</tr>
</tbody>
</table>

### 3.2 The flaw in Mitchell’s analysis

Determining the inner vectors (\( F_i \) and \( G_i \)) is critical to the success of Mitchell’s forgery attack. We reviewed his process for obtaining the inner vectors and found that this process cannot uniquely determine the inner vectors of EPBC. Two alternatives remain for every inner vector of plaintext/ciphertext blocks, there is a 99% chance that a bit pair will be known.

Mitchell used this theory to launch an attack. We discuss the methodology in section 4.

### Table 3: Probability of two alternatives for a bit pair and for every pair in a 128-bit block (Mitchell 2007)

<table>
<thead>
<tr>
<th>( \nu )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>10</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p' )</td>
<td>0.5</td>
<td>0.75</td>
<td>0.875</td>
<td>0.9375</td>
<td>0.99980</td>
<td>0.99984</td>
</tr>
<tr>
<td>( q' )</td>
<td>Very small</td>
<td>0.00000001</td>
<td>0.0000194</td>
<td>0.0016075</td>
<td>0.882389</td>
<td>0.986103</td>
</tr>
</tbody>
</table>

### 4 Mitchell’s Forgery Attack

Based on his analysis, Mitchell (2007) explains how a forged ciphertext message can be derived by controlled deletion of blocks in a legitimate ciphertext message. Assume the attacker has obtained the values for two of the inner vectors \( G_i \). Blocks can be deleted anywhere between the first ciphertext block and the second last
4.1 Attack application
We demonstrate this attack for a seven block ciphertext $C_1, C_2, \ldots, C_7$. Assume that the inner vectors $G_i$ and $G_0$ are known and that the final plaintext block, $P_7$, is the ICV.

Following Mitchell’s process, we constructed a forged ciphertext by deleting ciphertext blocks $C_4$ and $C_5$, modifying $C_6$, and leaving $C_7$ unchanged. After decryption, the forged ciphertext $C_1, C_2, C_3, C_4', C_5'$ generates the correct value for the ICV. This forgery attack has been demonstrated for a specific example by coding it in C programming language, using AES with a block length of 128 bits.

4.2 Success rate of revised attack
Mitchell claimed that with the knowledge of over 100 consecutive plaintext/ciphertext blocks ($\nu > 50$), the inner vectors $G_i$ would be revealed with very high probability (Mitchell 2007). However, as we showed in Sect. 3.2, the number of possible values for each inner vector $G_i$ can only be reduced to $2^9$ alternatives. The deletion attack described above requires two $G_i$ values to be known.

Therefore the probability of a successful forgery following this method is $2^{-128}$. This contradicts Mitchell’s claim that the forgery is guaranteed to succeed. Therefore Mitchell’s attack is no better than making random changes to the ciphertext (insertion, deletion or substitution) and hoping that the final block decrypts to give the correct ICV.

If the length of the ICV is $l$ bits, then the probability of successful brute force attack on the ICV is $2^{-l}$. If $l < 128$ bits then this approach has higher success probability than Mitchell’s forgery attack.

4.3 Comparison with key recovery attacks
Recall that EPBC uses two keys, $K$ and $K'$. Suppose we use a cipher with a block length and a key size both of 128 bits. Key $K$ is used to firstly encrypt a sequence number $S$ to obtain $F_0$, and then encrypt $F_0$ to obtain $G_0$.

Key $K'$ is used to encrypt $G_i$ to obtain $F_i$ for each message block. We compare Mitchell’s attack against exhaustive search on either or both keys.

Suppose $S$ is known to the public and that a number of pairs of plaintext/ciphertext blocks are known to the attacker. Then it can be shown that exhaustive search on both keys requires $2^{256}$ guesses. Checking each of these guesses will require at least one decryption, so the complexity will be around $2^{256}$. Knowing both $K$ and $K'$ allows the attacker to decrypt all ciphertext messages and impersonate either sender or receiver to communicate with the other one.

Now consider the key $K$. If this key and at least the first two plaintext/ciphertext pairs are known to the attacker, the relevant inner vectors can be revealed and a forgery attack conducted following Mitchell’s process. The probability of guessing this key correctly is $2^{-128}$. The correctness of the guess is verified by the receiver accepting the forged message.

Finally, consider the key $K$. If the attacker knows three consecutive plaintext/ciphertext pairs, it can be shown that this key and the inner vectors for these blocks can be revealed with a complexity of roughly $2^{231}$ encryption/decryption operations. The knowledge of key $K$ and these inner vectors guarantees the success of a forgery attack.

5 Conclusion
We reviewed Mitchell’s forgery attack on EPBC and found a flaw in his estimation of the probabilities of correctly obtaining the inner vectors. Knowledge of these inner vectors allows a forgery to be constructed. We show that, regardless of the number of known plaintext/ciphertext blocks, the possible values for each inner vector can only be reduced to two alternatives per bit pair, rather than being uniquely determined as claimed by Mitchell. When the block length of the underlying cipher is 128 bits, the number of alternatives is reduced from $2^{128}$ to $2^{64}$. The success rate of Mitchell’s forgery attack is therefore $2^{-128}$. This is no better than a brute force attack on the ICV, and worse if the length of the ICV is less than 128 bits. If the block cipher has a 128-bit key this is also comparable to exhaustive search on the key $K'$. For all of these attacks, the attacker does not know whether the modified ciphertext will be accepted before sending it.

Alternatively, the attacker can construct a forged ciphertext that is guaranteed to be accepted if either the second key $K$ or both keys are known. However the calculation complexity of finding these keys is prohibitive ($2^{231}$ for finding $K$ and $2^{256}$ for finding both keys).

Our results indicate that EPBC is in fact secure against Mitchell’s forgery attack. Additionally, we recommend that the ICV should be no shorter than the block length, to reduce the success rate of brute force attacks on the ICV.

6 References


Internet-wide Scanning Taxonomy and Framework

David Myers¹ Ernest Foo² Kenneth Radke³

¹ Email: d1.myers@connect.qut.edu.au
² Email: e.foo@qut.edu.au ³ Email: k.radke@qut.edu.au

Abstract

Industrial control systems (ICS) have been moving from dedicated communications to switched and routed corporate networks, making it probable that these devices are being exposed to the Internet. Many ICS have been designed with poor or little security features, making them vulnerable to potential attack. Recently, several tools have been developed that can scan the internet, including ZMap, Masscan and Shodan. However, little in-depth analysis has been done to compare these Internet-wide scanning techniques, and few Internet-wide scans have been conducted targeting ICS and protocols.

In this paper we present a Taxonomy of Internet-wide scanning with a comparison of three popular network scanning tools, and a framework for conducting Internet-wide scans.

Keywords: Internet-wide scanning, Taxonomy, Framework, Industrial Control Systems, Critical Infrastructure, SCADA, ZMap, Masscan, Shodan.

1 Introduction

With the exhaustion of the IPv4 address pool, and the slow adoption of IPv6, researchers have the opportunity to conduct Internet-wide surveys for research. In the past few years, there have been several Internet-wide scans conducted by different organisations worldwide. With recent advances in Internet-wide scanning tools, computational power, and network bandwidth, the required time to scan the IPv4 address space has been dramatically reduced. It is now possible to scan the entire public IPv4 Internet in as little as three minutes (Graham, 2013b). Several tools currently exist which allow scans of the IPv4 internet, including ZMap, Masscan, Unicornscan and Shodan.

ICS have been moving from traditional serial communications to switched and routed corporate networks, either directly connected through ethernet or through devices to enable serial to ethernet conversion. These ethernet networks allow for easy access, management and operation of the devices, however connection to corporate networks can allow the devices to be directly accessible from the Internet (Hoover, 2013).

However, there is currently no framework for conducting Internet-wide scans, no comparison between commonly used Internet-wide scanning techniques has been made, and few Internet-wide scans have been conducted for Internet accessible ICS devices.

2 Background

The Electronic Frontier Foundation (EFF) EFF SSL Observatory project conducted an Internet-wide scan for study (Electronic Frontier Foundation, 2014). From this dataset, the EFF were able to ask key questions about the existing state of SSL Certificates on the internet, including the number of trusted Certificate Authorities (CAs), number of signers, and frequency of use. The EFF found there were a large amount of weak and vulnerable certificates.

The Internet Census (2012) was a distributed scan of the IPv4 Internet using the Carna Botnet, which infected over 400,000 embedded devices (Anonymous, 2012). Using the NMap Scripting Engine, the botnet was designed to initially scan random addresses, attempt a telnet login, and upload a small binary to infected devices which then was used to scan the Internet. This distributed method of scanning the Internet dramatically reduced the time of the scan, from potentially months to hours.

Mining your Ps and Qs was a distributed scan of the IPv4 Internet using the NMap network mapping tool; the largest network survey of TLS and SSH servers at the time. The goal of the project was to search for TLS certificates with problems related to inadequate randomness upon generation (Heninger et al., 2012).

ZMap, created by a team from the University of Michigan (Durumeric et al., 2013), provides several improvements over traditional port-scanning programs such as NMap, used during the EFF SSL Observatory project, Internet Census, and the Mining your Ps and Qs scans (Electronic Frontier Foundation, 2014; Anonymous, 2012; Heninger et al., 2012). The ZMap tool dramatically reduces the time of scanning from days to as little as 45 minutes (Durumeric et al., 2013). In response to the release of ZMap, a new Internet-wide scanning tool Masscan was developed, which further reduces the time of scanning the Internet to a theoretical 3 minutes (Graham, 2013b). The development of these two tools has made conducting an Internet-wide scan easier, cheaper and more effective. Both ZMap and Masscan are tools designed specifically for conducting scans of the Internet, and provide significant performance improvements compared to NMap, a tool built for intensive local network scanning.

In comparison to conducting Internet-wide scans, the Shodan project allows users to bypass conduct-
ing scans themselves, and use information gathered from conducting ports scans and banner grabbing to find information about specific target devices. Users search the Shodan database using an interactive web interface, and can search using queries designed to restrict searches to a type of device, port, or geographical location. Shodan captures information about many devices, including SCADA, ICS, IP cameras and routers (Shodan, 2014).

“KATSE” was a scanning system designed to scan the nation of Finland to constantly search for exposed, Internet-connectedICS and analyse the systems for possible vulnerabilities. KATSE, a several component scanning system, scanned devices which were found using Shodan (Tilliakinen, 2014).

The ZMap team at the University of Michigan released in August, 2014, an analysis of traffic dataset received by a darknet over a 16 month period. Through the use of libpcap, the traffic was analysed and the team found that scans conducted targeting 10% or more of the IPv4 address space did not use ZMap or Masscan (Durumeric et al., 2014).

Existing research projects which conduct complete 3-way handshakes and surveying methods have been conducted in the past few years. Several new methods of scanning the internet have been developed, dramatically reducing the time required to conduct a full Internet-wide scan.

However, limited global Internet-wide scans have been conducted specifically targeting ICS. Furthermore, limited comparison between the new Internet-wide scanning methods have been conducted. The majority of Internet-wide scans described have used different methodologies and tools while conducting scans of the Internet, several developing their own tools to fulfill their research needs. Thus there is a need to analyse and compare techniques through the development of a Taxonomy and develop a Framework for conducting Internet-wide scans. Additionally, there exists the potential to scan the Internet to view the current landscape of publicly available ICS, through scanning the public IPv4 Internet for industrial control system protocols.

3 Taxonomy of Internet Scanning Methods

Through our investigation of Internet-scanning tools and previously conducted Internet-wide scans, we have seen Zmap, Masscan and Unicornscan share a large number of similarities. As such, we have distilled the following categorical breakdown of the Taxonomy of Scanning Methods. We have then compared the properties of ZMap, Masscan and Unicornscan using our taxonomy, as shown in Table 1.

3.1 Scanning Method

We define Scanning Method in Internet-wide scanning as the method used by the scanners to connect, check port availability and disconnect from a target host.

We categorize Scanning method into two categories: scanners which conduct SYN-scanning, and scanners which conduct complete 3-way handshakes.

Zmap utilises separate sending and receiving threads for packet transmission, and uses SYN-scanning for sending packets (Durumeric et al., 2013).

Masscan makes use of SYN-scanning, and like ZMap and Unicornscan, uses separate sending and receiving threads to transmit packets and receive responses.

Unicornscan conducts a full three-way handshake while conducting a scan, and breaks down the process of conducting scans into three processes. The main process “Unicornscan” is used to control the scan and keep track of packets, “unisend” which sends a SYN packet to the scan target, and “unilisten”, which listens for the SYN-ACK response, and sends information back to the master process “Unicornscan”.

3.2 Packet Transmission

We define Packet Transmission in Internet-wide scanning as the method used by the scanner to send and receive packets. We categorise Packet Transmission in to three categories: scanners which use the kernel TCP/IP stack, scanners which implement their own self-contained “user-mode” TCP/IP stack, and scanners which bypass the TCP/IP stack completely.

ZMap generates and sends packets using a raw socket at the Ethernet layer, which reduces kernel overhead and bypasses the TCP/IP stack. By generating and caching the Ethernet layer packet, ZMap prevents the Linux kernel from performing a routing lookup, arp cache lookup, and netfilter checks for each sent packet (Durumeric et al., 2013). Masscan uses a user self contained TCP stack, separate from the Linux kernel. In addition to this function, Masscan makes use of a kernel module “PF_RING” to improve packet transfer and capture speed. Unicornscan’s method has similarities to Masscan, using a user TCP stack outside of the kernel.

3.3 Randomisation

We define Randomisation as the ability of the scanning tool to generate a random permutation of the IPv4 address space, preventing iterative scanning of the IPv4 address space.

Traditional network scanning tools, such as NMap, iteratively scan through a list of IP addresses. Due to the methods new scanners use to generate packets, more traffic is generated and transmitted faster, reducing the time required to conduct an Internet-wide scan. However, this results in the possible overload of a destination network, potentially causing issues to the normal operation of that network (Durumeric et al., 2013).

ZMap uses a mathematical method for generating a random permutation of the IPv4 address pool, using modular mathematics. ZMap iterates over a multiplicative group of integers, ensuring the scanner will reach all IPv4 addresses, with exception to the address 0.0.0.0, an IANA reserved address (Durumeric et al., 2013). ZMap has recently been improved by including parallelisation generation of IP addresses over multiple cores, allowing faster address generation (Adrian et al., 2014).

Masscan creates random permutations of the IPv4 address pool using a custom cryptographic algorithm “Blackrock”, based on a Feistel network to encrypt an index. The Blackrock encryption function is based on Data Encryption Standard (DES) (Graham, 2013a).

ZMap and Masscan both have the ability to “seed” the randomisation element of the scans, allowing the random permutation of IP addresses to be repeatable.

From using Unicornscan to perform restricted logical network scans, we found Unicornscan does not have a randomisation function, and scans iteratively through addresses in the specified network range.

3.4 Scan Distribution

We define Scan Distribution in Internet-wide scanning as the ability for the scanner to conduct dis-
tributed scans from multiple source hosts. We cate-
gorise Scan Distribution into two categories: scanners
which can conduct distributed scans, and scanners
which cannot conduct distributed scans.
Both ZMap and Masscan have the ability to con-
duct distributed scans of the Internet, and use the
term “Shard” to describe the distributed hosts. ZMap
and Masscan have similar methods of conducting dis-
tributed scans; first a “seed” is set to specify the same
randomised address permutation over all hosts, then
assign multiple IP addresses to scan from. Unicorn-
scan does not have the ability to conduct a distributed
scan from multiple hosts.

3.5 Blacklisting and Whitelisting
We define Blacklisting in Internet-wide scanning as
a user created or edited list used to exclude IP ad-
dresses from scans, resulting with any address listed
in a blacklist not being scanned at any point. We de-
fine Whitelisting in Internet-wide scanning as a user
created or edited list used to specify a network range
to scan, resulting in only that address or range of
addresses being scanned.

We categorise Blacklisting and Whitelisting in to
categories: scanners which can use blacklisting,
scanners which can use whitelisting, scanners which
can use both blacklisting and whitelisting, and fi-
nally scanners which can use neither blacklisting or
whitelisting.

ZMap can use both blacklisting and whitelist-
ing for Internet-wide scans (Durumeric et al., 2013).
Masscan can use blacklisting in the form of an “ex-
clude file”, but not whitelisting. Like ZMap, black-
listing is configured through a configuration file, ac-
cepting the same format as ZMap (Graham, 2013b).
Unlike ZMap and Masscan, Unicornscan does not im-
plement either blacklisting or whitelisting.

3.6 Modularity
We define modularity in Internet-wide scanning as
the scanner being extensible with internal or exter-
nal modules, to increase the functionality of the scan-
er or provide some additional benefit. We categorise
modularity in to two categories: scanners which are
modular, and scanners which are not modular.
ZMap is a modular scanner, internally having a
series of extensible probe modules which can be cus-
tomised for different types of probes and payloads,
such as the UDP probe module (Durumeric et al.,
2013). In addition to the internal probe modules,
ZMap has output handlers which allow the scan re-

tults to be pushed into external modules to provide
additional processing. Neither Masscan or Unicorn-
scan have the ability to be extended with modules.

3.7 Scanning Speed
We define scanning speed in Internet-wide scanning
as the speed it is theoretically possible to conduct an
Internet-wide scan using a scanning tool. We cate-
gorise scanning speed in to three categories: 1gigE,
scanners which can theoretically scan up the limit of
1gigE Ethernet Cards, and 10gigE scanners which
can theoretically scan up to the limit of 10gigE Eth-
ernet Cards. These differences are determined on net-

The receiving component of ZMap utilises libpcap,
a library for capturing network traffic and filtering
results. ZMap can send packets close to the theoreti-
cal limit of a 1gigE Ethernet Card, approximately 1.5

millions packets-per-second (Mpps) (Durumeric et al.,
2013). Recently, ZMap has been further developed
and optimised, improving the performance of address
generation and “PF_RING” resulting in the ability
to scan using a 10gigE network card, reaching similar
speeds to Masscan at 15Mpps to 25Mpps (Adrian et al.,
2014).
Through the PF_RING module, Masscan can use
speeds to scan at a maximum of 15Mpps, or up to 25Mpps using a dual-port
10gigE Ethernet card (Graham, 2013b).
Unicornscan uses a similar method of sending
packets compared with Masscan, and additionally
uses the libpcap library for receiving network traffic. (Lee and Louis, 2005).
Using the same library, Uni-
cornscan would be able to send packets at the same
rate as ZMap, at approximately 1.5Mpps over a 1gigE
Ethernet card.

3.8 Speed Limiting
We define Speed Limiting in Internet-wide scanning
as the ability to slow or limit a scan’s speed, in order
to conduct the scan slower if necessary. We categorise
Speed Limiting in to four categories: limiting speed
by rate of scan in packets per second (pps) or band-
width (G,M,Kbps), limiting speed by duration of scan
(seconds), limiting speed by number of results, and
limiting speed through a combination of methods.
ZMap has the ability to limit the speed of a scan
by rate in packets per second (pps) or bandwidth (G,M,Kbps), limit number of hosts
and results, and by total time. Both Masscan and
Unicornscan can limit the rate of the scan in packets
per second, however neither Masscan or Unicornscan can limit by amount of hosts, results, or time.

4 Internet-wide Scanning Framework
In this section, we present our framework for conduct-
ing Internet-wide scans. We present our framework in
four sections: a scan policy, a primary scan, secondary
scan, and scan analysis.

4.1 Scanning Policy
While developing the Internet-wide scanning tool
ZMap and conducting internet-wide scans as part of
research, the team at the University of Michigan de-
veloped a list of seven recommended practices for fu-
ture researchers to use as guidelines for “Good
Internet Citizenship” (Durumeric et al., 2013). We
followed this list of recommended practices where
it was feasible while developing our policies. The
ZMap team’s guidelines for “Good Internet Citizen-
ship” were used as a base for implementing our two
policies for ensuring all required parties are aware of
any Internet-wide scans.

We have defined a clear policy to be used for com-

1. Request Ethics approval where necessary.
2. Inform and Discuss the nature and extent of the
scans with the ISP.
3. Coordinate network usage with the ISP to pre-
vent any disruption to normal network operation.
4. Coordinate with the ISP to ensure any emails
will be received and processed by the scanning
team.
Table 1: Comparison of ZMap, Masscan and Unicornscan using our Taxonomy of Internet Scanning Methods.

<table>
<thead>
<tr>
<th>Properties</th>
<th>ZMap</th>
<th>Masscan</th>
<th>Unicornscan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Method</td>
<td>SYN-scanning</td>
<td>SYN-scanning</td>
<td>3-Way Handshake</td>
</tr>
<tr>
<td>Packet Transmission</td>
<td>Bypass Kernel</td>
<td>User-mode TCP/IP</td>
<td>User-mode TCP/IP</td>
</tr>
<tr>
<td>Randomisation</td>
<td>Uses Randomisation</td>
<td>Uses Randomisation</td>
<td>No Randomisation</td>
</tr>
<tr>
<td>Distributed Scanning</td>
<td>Can conduct</td>
<td>Can conduct</td>
<td>Can not conduct</td>
</tr>
<tr>
<td>Black/Whitelisting</td>
<td>Both Black &amp; Whitelisting</td>
<td>Blacklisting</td>
<td>Neither</td>
</tr>
<tr>
<td>Scanning Speed</td>
<td>1gigE (10gigE as of August, 2014)</td>
<td>10gigE</td>
<td>1gigE</td>
</tr>
<tr>
<td>Speed Limiting</td>
<td>Combination (Rate, Duration, Results)</td>
<td>Rate of Scan (pps)</td>
<td>Rate of Scan (pps)</td>
</tr>
<tr>
<td>Modularity</td>
<td>Is modular</td>
<td>Is not modular</td>
<td>Is not modular</td>
</tr>
</tbody>
</table>

In addition to this policy for working with internal groups, we have defined a clear policy for working with scan-traffic recipients to receive, and process any requests for information or requests to opt-out of any future scanning activities.

1. Maintain a constant, clear contact point for receiving any information or opt-out requests, through use of web pages, reverse-DNS and contact email.
2. Respond to information or opt-out requests promptly after receiving the request, ensuring responses are taken seriously.
3. Immediately add opt-out requests to an IP address blacklist for future scans, and update blacklist as soon as possible.
4. Refine the address blacklist as needed if a repeat request is received.

### 4.2 Primary Scan

The Internet-wide scanning framework has two scanning stages, a primary scan and a secondary scan. The primary scan is conducted to find a broader range of hosts to be narrowed down by the secondary scan. In the primary scan, an Internet-wide scan is conducted using ZMap, Masscan, or Unicornscan, against a port or ports necessary to obtain a range of IP addresses for research. The main outcome of the primary scan is a list of IP addresses to be used for the secondary scans.

### 4.3 Secondary Scan

The Internet-wide scanning framework uses a secondary scanning stage to further identify the initial hosts, in order to identify or gather more required information from the hosts. The secondary scan is conducted using the outcomes from the primary scan on a second port or ports which are used by common services, that have the potential to identify the devices. These services, when queried, can provide information or banners containing software versions and device information such as device name and type. These ports include the web server port TCP/80, Simple Network Management Protocol (SNMP) port TCP/161, Telnet port TCP/23 and File Transfer Protocol (FTP) port TCP/21. These protocols are commonly used to interact with ICS devices, for managing, accessing and uploading and downloading of files to the devices. Using this list of ports, scanning a list of IP addresses can return banners and status of the device.

### 4.4 Scan Analysis

The Scan Analysis section of the framework is for gathering insight from the results gained from the primary and secondary scans, such as statistical information, and geolocation information. Statistical information can be gathered from using standard UNIX tools. ZMap, Masscan and Unicornscan output files to multiple formats, and by default use a human-readable format for viewing files. Extensions to these tools, such as banner grabbing modules, have the ability to output to the human readable to ascii format. Using unix tools such as grep, wc, diff, and comm, information can be observed from the results; such as how many IP addresses are in a range, and the number of times a server appears in a banner grab.

Our scan analysis stage uses geographical IP address information, retrieved from regional internet registries (RIR). Geolocation software uses databases of IP address data gathered from the RIR's to allow users of the software to search for geographic information related to an IP address, such as the approximate geographical location (Fiori, 2014). The results obtained through conducting primary and secondary scans can be processed through a GeoIP Server, and used as input to generate geographic maps of results to visually display scan results.

### 5 Discussion

While designing the Framework for Primary and Secondary scans, we initially considered sending messages to industrial control system protocols. Based on the responses we receive from the messages, we could quickly eliminate what devices were not ICS. However, this method of scanning would require us to craft packets to send commands as a payload specifically designed for the destination protocol. A scan using this method of crafting packets could be construed as an attempted attack on the destination system. In addition to the construed nature of the scan, it is possible that commands sent to a destination industrial control system could potentially interrupt the function of the device. Due to possible misinterpretation of the messages, and the potential of interrupting the function of the devices, we eliminated this method of scanning as a possible way of identifying ICS. Instead, we designed the Secondary Scan as a method for identifying ICS without the use of specialised or crafted payloads.

### Acknowledgements

This work was supported in part by Australian Research Council Linkage Grant LP120200246, Practical Cyber Security for Next Generation Power Transmission Networks.
References


DOCTORAL SYMPOSIUM
Real-Time and Interactive Attacks on DNP3 Critical Infrastructure Using Scapy

Nicholas R. Rodofile\textsuperscript{1} Kenneth Radke\textsuperscript{2} Ernest Foo\textsuperscript{3}

Information Security Discipline, Queensland University of Technology, Email: \{n.rodofile\textsuperscript{1}, k.radke\textsuperscript{2}, e.foo\textsuperscript{3}\}@qut.edu.au

Abstract

The Distributed Network Protocol v3.0 (DNP3) is one of the most widely used protocols, to control national infrastructure. Widely used interactive packet manipulation tools, such as Scapy, have not yet been augmented to parse and create DNP3 frames (Biondi 2014). In this paper we extend Scapy to include DNP3, thus allowing us to perform attacks on DNP3 in real-time. Our contribution builds on East et al. (2009), who proposed a range of possible attacks on DNP3. We implement several of these attacks to validate our DNP3 extension to Scapy, then executed the attacks on real world equipment. We present our results, showing that many of these theoretical attacks would be unsuccessful in an Ethernet-based network.

Keywords: substations, Distributed Network Protocol 3.0, DNP3, critical Infrastructure security, Scapy

1 Introduction

Despite the prevalence of Industrial Control System (ICS) networks connected to corporate IT networks, there are limited techniques that can be used to correctly detect cyber-attacks or forms of intrusion on industrial networks. To develop signature-based Intrusion Detection Systems (IDSs), attack signatures are required to help identify malicious network traffic. By using real critical infrastructure equipment for the attacks, it shows the realism of the attacks. Prior to this paper, not only did few sets of DNP3 attack signatures exist, but there was no easy way to create malicious network traffic. Hence, there is a need to enhance tools, such as the interactive packet manipulation Python library Scapy, that exist for traditional IT communication networks, such as Scapy, to include control system protocols. Our DNP3 extension of Scapy allows sets of DNP3 attack signatures to be easily created.

The outline of our paper proceeds as follows: Section 2 describes the relevant background information of previous research in Critical Infrastructure security for DNP3. In Section 3 we provide an overview of DNP3. We describe our attack tool that executes our DNP3 extension to Scapy in Section 4 and our attacks and their results in Section 5, and we then conclude in Section 6.

2 Related Work

There is currently ongoing research to help secure control systems using DNP3 and various other control system protocols. Lee et al. (2014) simulated a DNP3 attack using OpenDNP3. The research involved a small scale testbed. The testbed simulated actuators and sensors of a hydro-power system. The attacks using DNP3 were conducted on computers simulating DNP3 systems and were not tested on real equipment used in industry.

Researchers at the Sandia National Laboratory Urias et al. (2012) created testbeds to demonstrate attacks on SCADA Systems by exploiting network vulnerabilities in a corporate network. The teams managed to perform several attacks on a small scale testbed. The testbed simulated actuators and sensors of a hydro-power system. The attacks were general to an industrial network.

East et al. (2009) identified 28 attacks and 91 threats through their taxonomy and analysis of the DNP3 specification. These attacks were all theoretical attacks, based on the specification of DNP3. Although there is clearly research interest in ICS, in particular DNP3, there has been little research into attacks that work on real-world critical infrastructure equipment, to aid in the development of IDS attack signatures. We selected three of East et al. (2009) theoretical attacks, the Length Overflow attack, Address Alteration attack, and Configuration Capture attack, as we found these attacks interesting. We tested each of the attacks on real-world equipment using our DNP3 extension to Scapy.

3 DNP3 Overview

A typical electricity distribution company's substation will have an operations centre, making use of master devices, to manage multiple slave devices running in outstations that are usually at a remote location. These devices can be configured to communicate using DNP3 over an Ethernet connection using TCP/IP. The DNP3 frames, shown in Figure 1, are layered within a TCP frame.

A DNP3 frame begins with a start (START) field to identify the beginning of the frame, which is then followed by the length field (LEN), used to provide the length in octets of the entire DNP3 frame. The control (CTRL) field defines the frames direction, transaction initiator, and function. The destination field (DST) identifies the destination for the frame, whereas the source field (SRC) identifies the source of the frame. A Cyclic Redundancy Check (CRC) field in the data link layer provides integrity for the other eight octets of the data link segment. A CRC field is inserted after every sixteen octets (data...

The Transport segment, as shown in Figure 1, is used to data reassembly in the receiving device for fragmented DNP3 Frames. We implemented the Transport segment in our DNP3 Scapy extension, allowing us to manipulate and corrupt the reassembly process in the receiving device.

The application fragment begins with an application control (App CTRL) field, which is made of several sub-fields. The unsolicited (UNS) field flags that the fragment is an unsolicited response. If the UNS flag is unset, then the fragment is associated with a sequence number. The sequence (SEQ) field is used to assure the segments are not duplicated, missing and that they are in order, as the sequence number increments on each fragment. The Function Code (FN Code) field is used to identify the purpose of the fragment. There exists 34 defined function codes for application requests. A response DNP3 frame would contain additional the Internal Indications (IIN) to indicate the state and conditions of a slave device.

4 Attack Tool
To ensure that an attack tool is able to create appropriate attack signatures for the development and testing of IDSs, we identify the necessary requirements for our DNP3 extension by analysing the 34 theoretical attacks described in East et al.’s 2009 taxonomy, by which we identified the following three critical requirements for our DNP3 attack tool. Spoofing frames, Capture frames, and Modify Frames. These requirements allow the user to send or modify messages to target devices, in order to manipulate the target’s functionality or cause it to malfunction.

The DNP3 extension allows a user to craft a DNP3 frame, and layer the frame with each DNP3 segment to manipulate the fields inside the crafted frame. To create each of the DNP3 frame segments, we extended the Packet class of the Scapy library, and closely followed the DNP3 specification (IEEE Standard for Electric Power Systems Communications-DNP3 2012). Our Scapy extension is able to process the data chunks and generate the CRC values for each of the data chunks when a segment within the DNP3 frame is updated. To determine if the IIN field is required, we checked the DIR field in the data link control function. As part of our extension we did not implement the DNP3 data objects, instead we must construct the object payload from raw hex values.

5 Attacks
To evaluate our Scapy implementation of DNP3, we implemented a series of theoretical attacks described by East et al. (2009). The successful execution of each attack required the interception of all TCP frames, followed by the modification of the DNP3 payload, before forwarding the reconstructed frame on to its intended destination.

5.1 Setup
To perform our attacks, we used a testbed setup consisting of critical infrastructure equipment used in the real world. The testbed setup made use of the one-to-one network architecture (see Figure 2). The testbed used a Supervisory Control and Data Acquisition (SCADA) gateway as the master device and an Intelligent Electronic Device (IED) as the slave device. To connect the master and slave devices, we used three interconnecting layer 2 industrial network switches. The master was attached to switch 1, and the slave to switch 3. The attacker laptop was connected to switch 3. To allow us to monitor the experiments, we configured a mirror port to listen to all network traffic on the master device’s port. All of the mirrored traffic was captured on a separate laptop running Wireshark. We had the ability to see, via Wireshark and switch mirror ports, what was being sent to and from the slave IED and the master SCADA Gateway, and being received and sent by the attacker.

As part of the setup, we configured the master to request class object variables from the slave device using a “READ” DNP3 request message. The master would request data using DNP3 class variables, identified as Class 1, Class 2, Class 3, then finally Class 123. The slave device would send a DNP3 response message containing the relevant values of the requested classes after each DNP3 request. To physically indicate DNP3 communication, we also configured a mechanism to illuminate an LED on the slave, in response to a button push on the LED’s corresponding button. The mechanism involved (see Figure 2) the slave IED sending an unsolicited DNP3 response to the master device. The Confirm is followed...
by a “SELECT OPERATE” DNP3 message from the master, resulting with the LED on the IED to turn on. Therefore, if the LED on the IED illuminates then a message has flowed from the slave device to the master device, advising a change of state. Our physical indicator can represent something as significant as a circuit breaker opening or an indication of a running motor.

5.2 Eavesdropping

Before any other attack could successfully be executed, we needed to take control of the communications between the slave IED and the master gateway. Therefore, for our first attack, our objective was to intercept, and eavesdrop all network traffic between the master and slave devices, through the use of Address Resolution Protocol (ARP) cache poisoning.

Results As a result of the ARP cache poisoning, the attacker intercepted all frames, then redirected the frame on to it’s intended destination. During the eavesdropping attack, there was a manipulation of the MAC addresses by the attacker using our tool, this can be seen in Figure 3. As can be seen, the Toshiba (MAC xx:80:e2) on the first line of Figure 3 is attempting to send a TCP message to a Destination (MAC xx:80:e2). However, the message first goes to our attacker machine, Toshiba 58:77:b3, on row one of Figure 3. The attacker’s machine then updated the addresses and then the same message is sent from Toshiba 58:77:b3 on to the intended recipient, being xx:80:e2, on row two of Figure 3. Similar manipulations can also be seen in the pair of messages on rows 3 and 4 of Figure 3.

Figure 3: Eavesdropping Attack - Wireshark capture screen-shot of communication on the Attacker’s machine

During the faithful forwarding, the physical indication mechanism was successful as the LED did illuminate on the button press from the slave device, but we note there was an increased delay between the button press and the actual lighting of the LED. This meant that the delay added by the frame manipulation of our attacker computer was inside the tolerances of the critical infrastructure devices we tested.

5.3 Address Alteration

The second attack performed was East et al’s destination address alteration attack (East et al. 2009). The objective of the attack is to intercept and modify the DNP3 destination address of each DNP3 frame (shown in the DST field of Figure 1). Note: the DNP3 destination address, for example “10”, is different from the ethernet destination address (which is a MAC address). Our master device was assigned the address of ‘0’, whereas the slave device was assigned the address of ‘10’. For the address alteration attack, we passed the intercepted frame into a function where the DNP3 destination address is modified to be ‘2’. According to East et al, changing the destination of a the DNP3 frame may cause other devices to reply or the intended device will fail to receive the message. We performed this attack to see if our extension is capable of altering the DNP3 destination address. After updating the DST field, we then had to recalculate and update the DNP3 data link CRC field (see Figure 1).

Results After analysing the network capture from the attacker, we were able to see that all DNP3 addresses passing through the attacker were updated regardless of its DNP3 destination address. Our mirror capture of the master device shows all messages sent to the master had the destination address of ‘2’. During the address alteration attack our physical indication mechanism failed meaning the LED did not illuminate on the button press from the slave device.

This attack has some interesting practical limitations. If the intention is to forward a frame meant for one physical device to another physical device, this will not work as the surrounding TCP connection would need to be established with the second device. Therefore, the only way this attack could have a result of an action being taken by a different master, would be if there were multiple masters configured on the one device.

As a further development for this test, we configured a second address on our device, but the slave device had a limitation that, although it could have multiple masters, each of the masters needed to be at a separate IP address. Our intention was to have two physical indication mechanisms, for the two different addresses on the one device. The attack would mean that pressing the button for one physical indication mechanism would activate the other physical indication mechanism. The limitation that different IP addresses are required for different DNP3 addresses, means that the TCP layer would send the response to the updated DNP3 address to a different physical device. This attack may work in a serial environment, but there seems no conceptual way for the attack to work in an ethernet-based environment.

5.4 Length Overflow

For our next experiment, the objective of the attack is to intercept and modify the length field of a DNP3 payload to a new length of 44 octets. The CRC field also needed to be recalculated and updated to reflect the new length value. We conducted this attack on every frame containing a DNP3 payload. East et al expects that the length overflow attack will result in “data corruption, unexpected actions and device crashes” (East et al. 2009).

Results We analysed the traffic capture from the attacker’s interface. We can see that all DNP3 frames from the slave device had the length field updated to 44. Analysis of the new frames using Wireshark reveal that manipulated frames are flagged as malformed. Further, since the DNP3 message is part of the TCP payload, adjusting the length mean that some of the encapsulating TCP frame is now shown as part of the DNP message. Wireshark flagging an error is only an indication that the target device would also reject the frame. However, during this length overflow attack, our physical indication mechanism failed as the LED did not illuminate on the button press from the slave device meaning that the DNP3 destination address testing also rejected these frames. As such, the attack becomes a Denial of Service (DoS) attack. We also inspected each device’s commissioning tool, and found no indication of errors or data corruption. It
may seem that there is potential, if there were multiple DNP3 payloads inside the one TCP frame, that the first DNP3 message could consume part of the second message and still be a valid message. However, since the CRC check at the end of a DNP3 message is every 16 octets and at the start of the (next) DNP3 message frame after 8 octets there is almost no combination of two messages that would result in a working combined message due to length alteration.

5.5 Configuration Capture

The objective of this attack is to assert that the configuration file of the target outstation is corrupted. East et al. (2009) believed that this would cause the master to transmit a new configuration file to the slave. allowing the attacker to intercept the configuration file to use in a replay attack.

To implement this attack we set the 5th bit in the second Internal Indications (IIN) field (see Figure 1) to true. We performed this attack to see if firstly our extension is capable to updating the IIN field of a DNP3 fragment without interrupting the unsolicited response, and secondly to capture the upload of a new configuration.

We conducted two attacks. For the first attack we were able to intercept all messages between the master and slave devices, but we only manipulated frames that did not contain an unsolicited message. This means that our physical indication mechanism was excluded from the attack. For the second attack, the attacker intercepted all DNP3 frames, manipulating the 5th bit of the second octet in the IIN of all frames.

Results During the first configuration capture attack, the physical indication mechanism was successful as the LED did illuminate the button press from the slave device. After analysing the traffic capture from the attackers interface in Wireshark, we noticed there was no traffic indicating the transfer of configuration files in response to the solicited messages which did have their IIN field updated. During the second configuration capture attack, our physical indication mechanism failed, meaning the LED did illuminate on the button press from the slave device. After analysing the captured traffic from the attacker, again there was no traffic indicating the transfer of configuration files.

The equipment we were using had a full deployment configuration used in critical infrastructure protection, before the specifics of our tests. Even so, the master did not have configurations for the slaves stored, and thus there was no potential for this attack to result in the master sending new configuration files. The configuration of the slave takes several minutes from an engineering workstation, all of which time the device is not functioning. Since such a downtime is unacceptable in a critical infrastructure environment, many critical infrastructure environments may not store configurations on their master devices for automatic re-deployment at unspecified times due to this corruption assertion. Our result and the standard configuration of these critical infrastructure devices suggests that East et al.’s theoretical attack would rarely work.

6 Conclusion

In conclusion, we presented our DNP3 Scapy extension and implement a series of East et al.’s 2009 theoretical DNP3 attacks on real critical infrastructure equipment. In Table 1 we outline the number of frames intercepted by the attack tool, the number of intercepted frames that contained a DNP payload that were modified beyond changing the IP addresses the intended recipient’s IP address, and the result of the physical response mechanism as outlined in Section 5.1. For each of the attacks conducted we observed that our attack tool was able to intercept frames from both the master and slave devices. As can be seen by the results of our tests, many of East et al.’s theoretical attacks have limited chances of success on real world equipment. However, other attacks are more plausible and we will present these attacks in future work.

The results from our attacks on the critical infrastructure equipment validate that Scapy is a suitable tool for attack signature generation for DNP3, as our tool was able to create and parse DNP3 frames in real-time. In closing, our tool will assist with the development attack signatures for IDSs.

Acknowledgment

This work was supported in part by Australian Research Council Linkage Grant LP120200246, Practical Cyber Security for Next Generation Power Transmission Networks.

References


<table>
<thead>
<tr>
<th>Attack</th>
<th>Intercepted</th>
<th>Modified</th>
<th>LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forwarding</td>
<td>400</td>
<td>0</td>
<td>True</td>
</tr>
<tr>
<td>Address Alteration</td>
<td>50</td>
<td>30</td>
<td>False</td>
</tr>
<tr>
<td>Length Overflow</td>
<td>400</td>
<td>15</td>
<td>False</td>
</tr>
<tr>
<td>Config Capture</td>
<td>100</td>
<td>8</td>
<td>True</td>
</tr>
<tr>
<td>Config Capture 2</td>
<td>100</td>
<td>18</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 1: Results of Attacks. Only DNP3, not ethernet, modifications shown.
Author Index

Bartlett, Harry, 57
Beaumont, Mark, 45
Cheney, Ben, 3
Choo, Kim-Kwang Raymond, 53
Dawson, Ed, 57
Di, Binbin, 57
Foo, Ernest, 51, 57
Grove, Duncan, 25
Heravi, Alireza, 53
Herrera, Adrian, 3
Hopkins, Bradley, 45
Jayarathna Dilshan, 15
Mani, Deepa, 53
McCarthy, Jim, 25
Mubarak, Sameera, 53
Murray, Alex, 25
Myers, David, 61
Neale, Grant, 35
Newby, Tristan, 25
North, Chris, 25, 45
Osborne, Grant, 35
Owen, Chris, 25
Radke, Kenneth, 61, 67
Richer, Toby, 25
Rodofile, Nicholas, 67
Shield, John, 25
Simpson, Leonie, 57
Tupakula Udaya, 15
Varadharajan Vijay, 15
Welch, Ian, iii, vii
Wong, Kenneth, 57
Yi, Xun, iii, vii
Recent Volumes in the CRPIT Series

ISSN 1445-1336

Listed below are some of the latest volumes published in the ACS Series Conferences in Research and Practice in Information Technology. The full text of most papers (in either PDF or Postscript format) is available at the series website http://crpit.com.

Volume 137 - Database Technologies 2013
Edited by Hua Wang, University of Southern Queensland, Australia and Rui Zhang, University of Melbourne, Australia. January 2013. 978-1-921770-22-7.
Contains the proceedings of the Twenty-Fourth Australasian Database Conference (ADC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 138 - Information Security 2013

Volume 139 - User Interfaces 2013
Edited by Ross T. Smith, University of South Australia, Australia and Burkhard C. Wünsche, University of Auckland, New Zealand. January 2013. 978-1-921770-24-1.
Contains the proceedings of the Fourteenth Australasian User Interface Conference (AUIC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 140 - Parallel and Distributed Computing 2013
Contains the proceedings of the Eleventh Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 141 - Theory of Computing 2013

Volume 142 - Health Informatics and Knowledge Management 2013
Contains the proceedings of the Sixth Australasian Workshop on Health Informatics and Knowledge Management (HHKM 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 143 - Conceptual Modelling 2013
Contains the proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling (APCCM 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 144 - The Web 2013
Edited by Helen Ashman, University of South Australia, Australia, Quan Z. Sheng, University of Adelaide, Australia and Andrew Trotman, University of Otago, New Zealand. January 2013. 978-1-921770-25-8.
Contains the proceedings of the First Australasian Web Conference (AWC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Contains the proceedings of the Australian System Safety Conference (ASSC 2012), Brisbane, Australia, 23rd – 25th May 2012.

Volume 147 - Computer Science 2014
Edited by Bruce Thomson, University of South Australia and Dave Parry, AUT University, New Zealand. January 2014. 978-1-921770-35-7.
Contains the proceedings of the Australasian Computer Science Conference (ACSC 2014), Auckland, New Zealand, 20 – 23 January 2014.

Volume 148 - Computing Education 2014

Volume 149 - Information Security 2014

Volume 150 - User Interfaces 2014

Volume 151 - Australian System Safety Conference 2013

Volume 152 - Parallel and Distributed Computing 2014
Edited by Bahman Javadi, University of Western Sydney, Australia and Saurabh Kumar Garg, BH Research, Australia. January 2014. 978-1-921770-34-0.

Volume 153 - Health Informatics and Knowledge Management 2014
Contains the proceedings of the Seventh Australasian Workshop on Health Informatics and Knowledge Management (HHKM 2014), Auckland, New Zealand, 20 – 23 January 2014.

Volume 154 - Conceptual Modelling 2014

Volume 155 - The Web 2014

Volume 156 - Australian System Safety Conference 2014
Contains the proceedings of the Australian System Safety Conference (ASSC 2014), Melbourne, Australia, 28 – 30 May 2014.