LINUX PATCH MANAGEMENT: WITH SECURITY ASSESSMENT FEATURES

Soranut Midtrapanon, Gary Wills
School of Electronics and Computer Science, University of Southampton
{sm6n17, gbw}@soton.ac.uk

Keywords: Patch Management, Linux, Software Inventory, CVE Scanning, Security, Puppet, Mcollective

Abstract: The lack of patch management has been identified as the main reason for many ransomware attacks. The cost of patch management is still an obstacle for many small and medium-size businesses. There are many open source, free of charge, patch management systems but these require many pre-configuration steps making them complicated to use. Hence, this paper presents a patch management system that is cost-effective but also efficient in terms of set-up time. We have written the system in Python with Puppet and Mcollective to aid the configuration steps. An additional feature of this system is the ability to assess the security of the system being patched, using CVE scanning.

1 INTRODUCTION

Patch management is the process of timely updating of existing systems with software patches to plug known security vulnerabilities or to improve service performance (Souppaya, and Scarfone, 2013). It is recommended that security patches be applied within 14 days of being issued. However, this is not enforced and this has led to an increase in ransoms (Adamov, and Carlsson, 2017) (Rajput, 2017). For example by leveraging the EternalBlue exploit, well-known ransomware WannaCry was able to infect many systems worldwide (Mansfield-Devine, 2017).

The National Health Service (NHS) in the UK was a victim of this ransomware, even though the EternalBlue patch was released months beforehand, many of the unpatched NHS systems were subject to ransomware encryption which resulted in many of the networks being shut down, this would not have happened had an effective patch management system been in place (Ehrenfeld, 2017) (Hoeksma, 2017). Ransomware affects many systems not only Windows based systems, in 2017 the EREBUS ransomware infected the outdated kernels and web application stack of Linux based systems (O'Brien, 2017) (McAfee, 2017).

Even though having good patch management is a known method for basic security hygiene in order to reduce cyber-attacks; it is still a neglected area for many non-IT specialist companies. This is especially the case in small and medium-sized enterprises (SMEs), where the cost of setting up and maintaining a patch management system can be prohibitive (Mansfield-Devine, 2016) (Renaud, 2016) (Goucher. 2016).

This paper presents an approach that enable SMEs to address the issues of cost and maintenance of a patch management system, for multiple Linux-based servers.

2 BACK GROUND LITERATURE

A patch management system allows system administrators to install updates on their managed systems (Gerace, and Cavusoglu, 2009), (Rankin, 2017). The patch management lifecycle involves acquiring information about the patch from software vendors; identify critical patches, performing patch installations and verifying results (Dadzie, 2005). In addition, aspects such as assessing potential security risks on managed systems, collecting an inventory of the software on the platform and hardware used, pre-assessment of patches prior to being installed, and prioritizing the order in which patches are installed (Mell *et al*, 2005). They also give two mechanisms for patch management.

- **The manual patching method:** In this method system administrators are required to perform all patch management related tasks manually. This includes monitoring newly released patches, running scripts, logging changes and performing spate analysis of their system.

- **Automated patching:** System administrators utilize software to perform most if not all patch management tasks automatically.

The automated approaches are more efficient than manual methods of patch management. In addition, the automated patching method is recommended by the National Institute of Standards and Technology (NIST) (Mell *et al*, 2005), as it allows patching a large number of systems within the recommended time and importantly reduce the risk of human error (Dey *et al*, 2016]. Several other standards also point out the need for good patch management. For example, it is a compulsory component for compliance for PCI Security Standards and NIST SP 800-53 (Souppaya, and Scarfone, 2013) and is part of the UK's Cyber Essentials scheme to mitigate cyber-attack incidents in SMEs (Mansfield-Devine, 2016).

Moreover, it has been established by US-CERT that by maintaining managed systems so that they remain up-to-date, that 85% of all cyber threats can be avoided (US-CERT, 2015). Therefore, when patch management procedures are implemented correctly, numerous system patches can be managed in a timely manner (Arora *et al*, 2006).

2.1 Automated patch management software architecture

Patch management software provides an effective method for patching systems automatically. The software has the ability to keep all managed systems up to date by applying patches promptly. Client-Server architecture for patch management enables users to manage all systems and view reports through a central management console. The patches are held on patch management servers, and the patches are installed on client machines (which can be other servers).

Generally, there are three methods for identifying and installing the patch required on client machines. A tool may use one or more of these techniques together to patch a client machine (Souppaya, and Scarfone, 2013), each of these approaches can be summarized as follows:

1. Agent-based patch management: In this approach, an agent is installed on each of the managed clients. A different agent is required for each type of platform. The agent undertakes the tasks required for patching: getting the patch information, installing the patch, and verifying successful completion of the task. The agent required administrative privileges on all machines.
2. Agentless patch management: A remote server regularly scans the servers under its control and carries out the necessary patch management if it finds out of date applications/software. Hence, there is no need to install an agent on the client. However, this does increase the network traffic (increase bandwidth), and only works on a local network, as remote scanning can be blocked.
3. Passive Network Monitoring: Similar to agentless scanning technique, where the patch management server(s) scan the internal networks, but they can also identify unmanaged and unpatched systems but do not patch them automatically. The limitations are that they can only work on unencrypted networks and version detectable applications.

Agent-based approaches is the preferred method over agentless approaches as it has fewer limitations, but importantly can be made to work on most installations. While the passive approach is used to extend features of existing systems.

2.2 Technical challenges and issues

There are several challenges and issues arising from applying patch management approaches.

1. **System downtime**: In some circumstances, it is necessary to stop or restart software services when patches are being applied or after applying patches. This could have consequences for organizations fulfilling Service Level Agreements (SLAs) (Le, *et al.*, 2014). However, there are patching approaches that can be applied to live systems to achieve zero downtime; for example, Oracle's Kspice[i], RedHat's Kpatch[ii], and SUSE's Kgraft[iii] (Kashyap, *et al*, 2016). Moreover, these techniques also support verification, which is undertaken immediately after the patches have been installed without stopping or restarting the service.
2. **Failures and side effects due to installed patches**: Installing a patch can also introduce new errors such as inconsistency in system configurations, permission issues, software bugs and new vulnerabilities (Okhravi, and Nicol, 2008). Organizations normally have to restore to the safe fall-back state and then wait for a manual update to be issued (Le, *et al*, 2014) (Kashyap, *et al*, 2016). Hence, before installing a patch, system administrators may perform manual testing first (Gerace, and Cavusoglu, 2009).
3. **Multiple architectures and platforms:** There are many operating systems and network architectures used by industry and commerce that require specific procedures or commands when installing patches. Many of the commercial packages do cater for this and work well in this multiplatform/multi-architecture environment. There are still occasions when the system administrators need to perform manual patching tasks (Souppaya, and Scarfone, 2013).

3 EXISTING APPROACHES

Section 2 outlined the approaches that could be undertaken for patch management systems. This section focuses on identifying existing systems and the gaps that led to our design.

3.1 Why Linux?

Linux is still the most popular system on which to host web-based applications. Unfortunately, it is also the one on which it is most likely to also find the most portion of out of date software. In addition, there are also a number of Linux hacking tools available to remotely exploit unpatched systems, (Delasko, and Chen, 2018). Microsoft has developed Windows Server Update Services (WSUS) to support the updating of patches on Windows-based systems. This is deployed in such a way that the system is updated regardless of the user choices (Palumbo, 2015).

3.2 Commercial Tools

There exist many commercial automated patch management tools. The most commonly used are those that use a centralized system that is controlled via a web-based or desktop application. Some will also assess third-party applications. Our analysis focuses on security and customizable features as shown in Table 1. It is important to pretest all third-party patches prior to deployment on a system, even if the patch has been developed especially for the system (customized patch), it should go through a system of verification first, as any patch could introduce new vulnerabilities (Okhravi, and Nicol, 2008). System administrators like to control and verify the process of patch management, hence it is important that tools provide Application Programming Interfaces (APIs) supporting originations to integrate the patch management tools into their systems and processes. Additionally, organizations like to buy one tool for multiple platforms (Mansfield-Devine, 2016). The prices shown are only for guidance.

Table 1 Shows that patch management tools with multi-platform support are now important, this differs from previous surveys (Seo, *et al.,* 2005) (Seo, *et al.*, 2006). Notably that only about half of the vendors provide pre-built and pre-testing third-party application patches, they mainly mitigate this by offering software catalogues of verified updates that can be installed. Very few vendors provide a mechanism for a security verification process of custom patches before they are applied, they also do not provide patching-related APIs and the cost is based on a subscription model per managed device which can become a prohibitive cost for SMEs (Mansfield-Devine, 2016).

3.3 Non-commercial software

The three common open-source tools for patch management on Linux installations are vFense[iv], FAI Linux Project[v], and Spacewalk[vi]. vFense is a standalone patch management tool, while the others are automated tools design for system configuration that has the ability to update software. All three tools use an agent-based approach. This survey (Table 2) aims to show the tools' main feature, the number of steps required to configure the setup of the system, and whether or not the tool is currently support. Table 2 shows that

1. All the tools require numerous steps to complete the setup and initial configuration. Also, vFense involves the administrator to build the client agent from scratch.
2. The tools use the Common Vulnerabilities and Exposures (CVE) descriptions to analyze security vulnerabilities.
3. Only provided by Spacewalk provides any form of software inventory and an essential feature for patch management.
4. Typically system administrative staff access is via a web-based user.
5. With any open-source project, there is a risk that the project is deprecated and receives no further developer community support.

Spacewalk provides suitable patching related features, has good support from the community, and takes a moderate number of steps to complete the configuration, but does not support Ubuntu.

A common problem for the systems that have deployed the agent-based patching technique, is that they are complex to use, and require many steps to configure the systems. Therefore, our system plugs these gaps creating a Linux based patch management tool that is cost-effective, supports all of the current Linux distributions, and is customizable and easy to configure.

4 ARCHITECTURE

An orchestration tool allows us to control and concurrently manage multiple servers. Yet many of the existing open-source tools that have good community support and are intended as configuration management tools that allows system administrators to create configuration files that can then be distributed the managed servers. Many of them also come with plug-ins that provide orchestration functionality.

Table 1 Commercial patch management software

Patch management software	**Multi-platform**	**Third party pre-tested**	**Custom patch add-on**	**API**	**Pricing per year**
IBM Bigfix[vii]	Yes	No	Yes	Yes	£2.05 per client
Ivanti Patch[viii]	Yes	No	No	Yes	N/A
RedHat Satellite[ix]	No	No	Yes	Yes	N/A
SolarWinds Patch Manager [x]	No	Yes	No	No	£2,745 per 250 nodes
Flexera Corporate Software Inspector[xi]	Yes	Yes	No	No	From £2,194 per 100 nodes
Kaseya VSA[xii]	Yes	No	No	No	N/A
GFI LanGuard[xiii]	Yes	No	No	Yes	£6.67 - £17 er node
ManageEngine Patch Manager Plus [xiv]	Yes	Yes	No	No	£493/£650 per 100 nodes
ZENworks Patch Management [xv]	Yes	Yes	No	No	N/A
CMS Patch Manager [xvi]	Yes	Yes	No	No	From £50 per month
BMC BladeLogic Server Automation[xvii]	Yes	No	No	Yes	N/A
KACE [xviii]	Yes	Yes	No	No	£8,000 per 100 nodes

Table 2 Open Source patch management software

Tools		vFense	FAI Linux Project	Spacewalk
Features	Functionality	An open-source application. Main purpose is to install patches and perform related tasks	An automated configuration management that can install patches and perform related tasks	An open-source Linux. Management system that can install patches and perform related tasks
	CVE Vulnerability scans	Yes	No	Yes
	Inventory of applications	No	No	Yes
	Web –based User interface	Yes (But deprecated)	No (terminal user interface)	Yes
	Distribution to all Linux distribution?	Yes	Yes	No (Red Hat only)
Ease of Use	Steps required to Configure (difficulty)	Large (Require numerous steps)	Moderate to low	Moderate
	Currently supported?	No Last update was in 2016	Yes	Yes

These files contain the specified configurations and settings for the managed systems. The tool will also monitor any changes to the configuration of the managed servers and apply updates automatically. Hence, we can take such tools and adapt them to meet our requirements for patch management. We choose to use the open source version of Puppet for our configuration management tools (Walberg, 2008). It addition, Puppet has good community support. However, the open-source version does not come with a web user interface so have we implemented these to allow users to administer and manage the patching process. An additional benefit of Puppet is the native implementation of a server orchestration tool called Mcollective, which can be used to undertake tasks on all Linux distributions (Rankin, 2017). In addition, Mcollective is customizable allowing for custom plugins to perform automated tasks. Mcollective uses the Puppet Certificate Authority (Puppet CA) to support secure communication features over TLS/SSL. Mcollective, obtains a certificate from Puppet CA for each specified administrator on a Puppet master server. Mcollective uses the certificated to construct a secure channel through which only the authorized administrator can control the Puppet agents installed on managed servers, see Figure 2.

Table 3: Development tools and libraries

Tool/library	Description
Celery[xix]	A Python message queue library to real-time tasks in the background.
Eventlet[xx]	A Python thread pool concurrent management library, to enable us to run Celery on Windows
Django-celery-results[xxi]	A Django Python library for storing all Celery task results which can be queried to display task statuses.
RabbitMQ[xxii]	A message queue for handling Python background, controlled by Celery.
Fabric[xxiii]	A SSH connection Python library. It is the necessary for use to establish an SSH connection to a Puppet master server, and to run commands and authentication method for the Public key.
Bootstrap[xxiv]	It is a responsive web user interface framework. We use this in the implementation of the user interface to set its looks and feels.
Toastr[xxv]	A JavaScript library for displaying notifications to the users. We use this to patching task notifications.

The system architecture design is shown in Figure 1. The main system is implemented using the Django (Python) web-framework and a web user interface front-end. The core process communicates with the embedded database system (SQLite3) as this natively supports Django. Authorized users can access the system via a web browser.

In order to undertake the task of patch management, the core system connects and communicates with the managed systems using a combination of Puppet and Mcollective functions through an SSH connection. Each communication task is executed as a background task (Celery) combined with a central message queue (RabbitMQ), this enables the administrator to continue with other activities without needing to wait for the initial task to finish.

The process of establishing an SSH connection is that an authenticity verification takes place on the server, it confirms that the server's public key matches the expected user's public key. In order to ensure a successful connection to the managed server, a corresponding SSH passphrase is required. We use the Mcollective and its native plugins to execute Linux-based commands, which retrieves all the information from the connected systems. SSH connections to the Puppet master server uses the public key authentication found in the Fabric Python library. We have manually tested all these commands with various Linux systems to verify these commands work in our system and give the expected results.

As we use Puppet to undertake the host discovery task by connecting to our managed servers, and then simultaneously uses Mcollective to run the Linux commands on the managed server, it is therefore, necessary to ensure that a Puppet environment is set up first. At the core of the system, we process Django views, which handles the requests. It also renders user interfaces from template/static files and undertakes the database operations.

Table 3 shows the tools and libraries used in our system for the Django web-based development and the Python background task implementation. There were a number of methods for implementing the CVE scanning of the managed servers. The first was to use a locally hosted CVE database, but this would have resulted in an extremely large size database, which would be hard to maintain, so we decided to use a Public CVE API. We did try using the unstructured word search facilities with the API but obtained too many false positive results. We settled on the CVE API provided by RedHat which allowed us to search by package names and versions. However, it is limited and does is not supported Ubuntu, for which we used the word search.

5 TESTING AND EVALUATIONS

This section presents the testing and evaluation of the patch management system. The testing showed all implemented functionalities work as expected. While the evaluations compared our system with current patch management systems.

5.1 Testing processes

We conducted unit tests and full system integration tests. We used a white box approach to verify the internal processes and logic of the system (Pressman. 2010).

Unit testing allowed us to ensure that the functions we developed were as designed; we used the automatic test facilities that comes with Django, this also come with automatic testing of views and user interface functionality. We developed 96 test cases for testing both views and internal functions; this gave us the ability to conduct regression testing and to use a test-driven approach to development.

We had designed the system so that the patch management core functions could be automatically tested separately from the system integration. The integration tests were undertaken using manual operations, using a Vagrant-based Puppet testing environment. A Test Pyramid method (Cohn, 2010) was integrated into the testing in order to reduce the total number of test cases required. Hence, the unit-test case cannot be reused. Therefore, the integrations testing was undertaken without having to know the internal logic (Pressman, 2010).

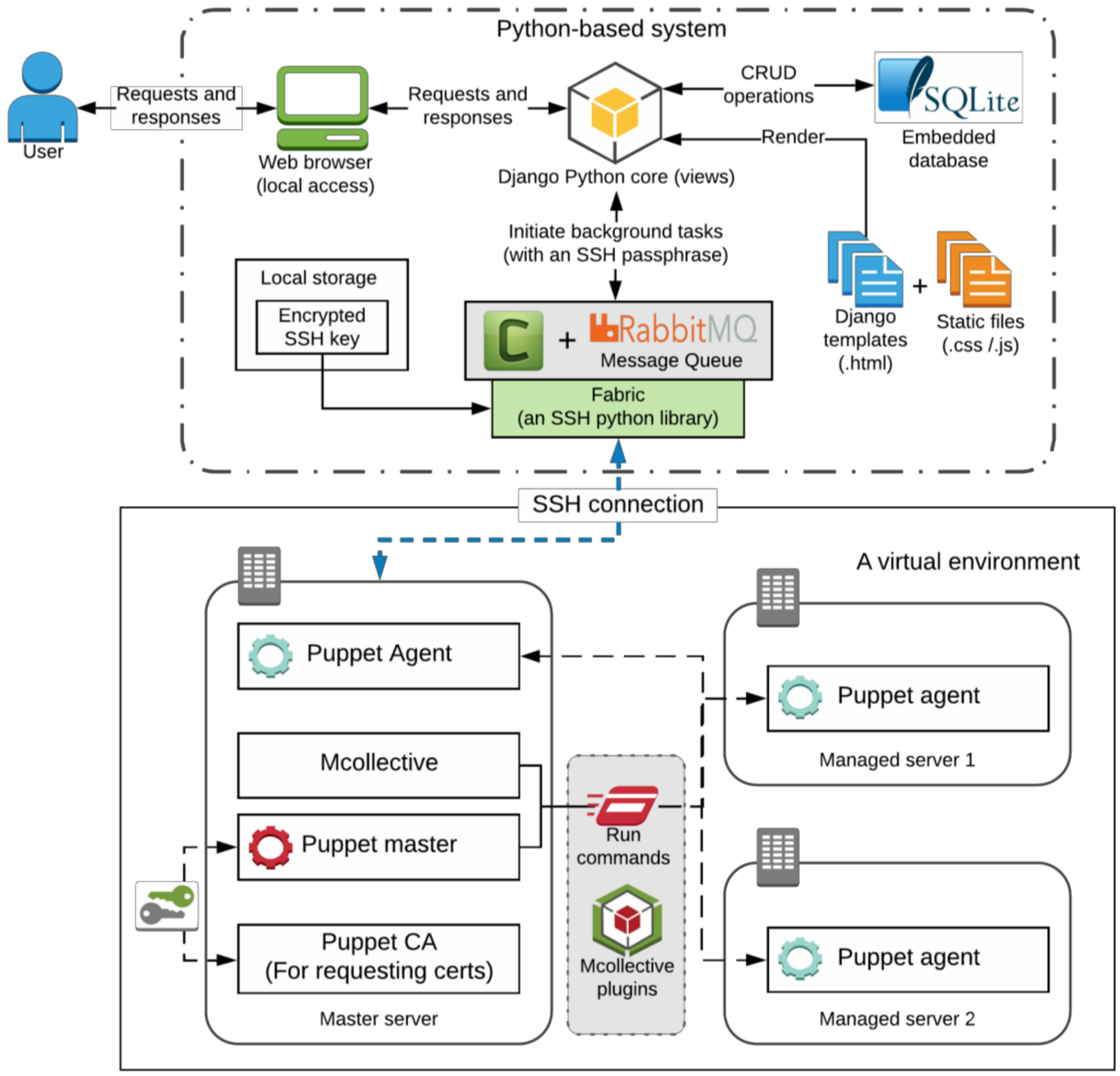

Figure 1: System architecture

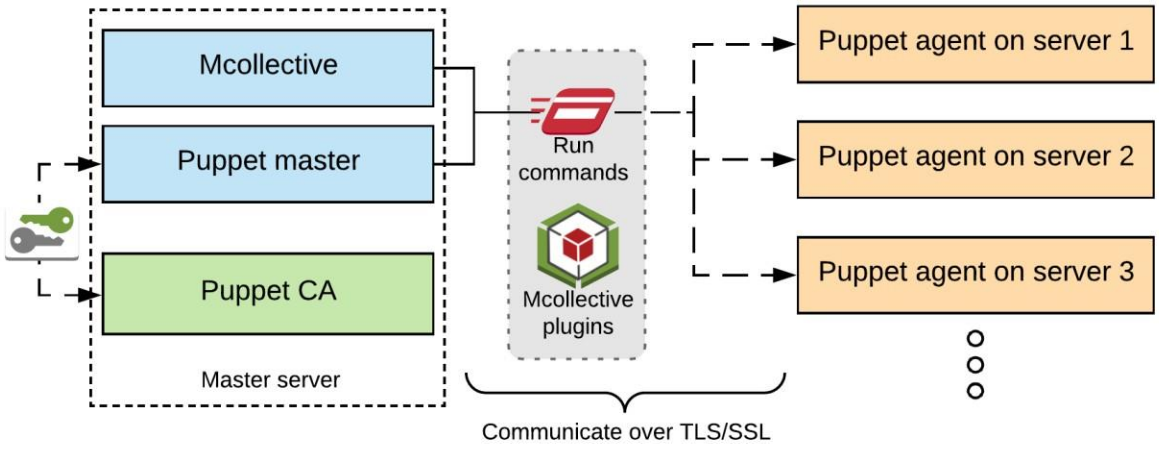

Figure 2: Overview of how Puppet and Mcollective works together

5.1 Evaluation

In this section, we present the evaluation of the project in terms of the features, customizability and cost. The main aim of this project was to develop an efficient Linux patch management system that would fit the budget of an SME

Section 3 identified a number of commercial systems with indicative costs. Our system can provide the necessary patching functionality and security assessment. This will substantially reduce the cost to an SME as they are saving in terms of investment costs (as our system is open source).

In section 3 we identified Spacewalk as the best open-source patching tool available to date. Therefore, we compared the patching features of our system and Spacewalk, see Table 4.

As can be seen from Table 4, our system has two main advantages over Spacewalk. The first is that we support all Linux distributions. We have achieved this by utilizing the Puppet/Mcollective. Secondly, by using Puppet we can connect to and discover all managed systems, also by using Mcollective to run commands simultaneously, we have reduced the number of steps required in the configuration process.

In addition, our system is designed for future development, by enabling more features to add by customizing and extending new plugins.

Table 4 Comparison of open-source system features

Feature	Our system	Spacewalk
Web –based User interface	Yes	Yes
CVE Vulnerability scans	Yes	Yes
All Linux distributions	Yes	No
Steps required to Configure (difficulty)	Low	Moderate

6 CONCLUSIONS

Our patch management system has been implemented and the evaluation shows that it is a cost-effective and usable open-source tool.

Our patch management tools fill the gaps found in the current patch management software, it provides vital patching capabilities on Linux systems with security assessment features which support the retrieval of essential details and CVE information from managed systems for further analysis.

The advantage of using Puppet and Mcollective to integrate into the core is that we have significantly reduced the number of steps required in the initial configuration process as we are able to support to all Linux distributions with customizability and extensibility. Furthermore, we provide a user-friendly interface that allows anyone to interact with the patch management system without the need to be an expert at understanding Linux commands.

Moreover, by being open-source it is an affordable patch management tools, it can be used to increase security awareness by the use of the CVE scanning and the necessity of keeping systems up-to-date, especially to SMEs on a limited budget.

Future work is in developing further the use of CVE scanning with public APIs. We are looking into better ways to scan and analyses CVEs. For example, machine learning coupled with Neuro-linguistic programming can improve the interpretation of the CVE descriptions and improve the effectiveness of the CVE scan. This could lead to efficiently discovering known vulnerabilities in a system.

7 LIST OF REFERENCES

Adamov, A. and Carlsson, A. (2017) 'The state of ransomware. Trends and mitigation techniques', *2017 IEEE East-West Design and Test Symposium (EWDTS)*. Sept. 29 2017-Oct. 2 2017. pp. 1-8.

Arora, A., Caulkins, J.P. and Telang, R. (2006) 'Research Note—Sell First, Fix Later: Impact of Patching on Software Quality', *Management Science*, 52(3), pp. 465-471.

Cohn, M. (2010) Succeeding with agile: software development using Scrum. Pearson Education.

Dadzie, J. (2005) 'Understanding Software Patching', *ACM Queue - Patching and Deployment*, 3(2), pp. 24-30.

Delasko, S. and Chen, W. (2018) 'Operating Systems of Choice for Professional Hackers', *ICCWS 2018 13th International Conference on Cyber Warfare and Security*. Academic Conferences and publishing limited, p. 159.

Dey, D., Lahiri, A. and Zhang, G. (2015) 'Optimal Policies for Security Patch Management', *INFORMS J. on Computing*, 27(3), pp. 462-477.

Ehrenfeld, J.M. (2017) 'WannaCry, Cybersecurity and Health Information Technology: A Time to Act', *Journal of Medical Systems*, 41(7), p. 104.

Gerace, T. and Cavusoglu, H. (2009) 'The critical elements of the patch management process', *Commun. ACM*, 52(8), pp. 117-121.

Goucher, W. (2011) 'Do SMEs have the right attitude to security?', *Computer Fraud and Security*, 2011(7), pp. 18-20.

Hoeksma, J. (2017) 'NHS cyberattack may prove to be a valuable wake up call', *BMJ*, 357.

Kashyap, S. *et al.* (2016) 'Instant OS Updates via Userspace Checkpoint-and-Restart',*USENIX Annual*

Technical Conference. pp. 605-619.

D Le, J Xiao, H Huang, H Wang (2014) 'Shadow patching: Minimizing maintenance windows in a virtualized enterprise environment', *10th International Conference on Network and Service Management (CNSM) and Workshop*. 17-21 Nov. 2014. pp. 169-174..

Mansfield-Devine, S. (2016) 'Securing small and medium-size businesses', *Network Security*, 2016(7), pp. 14-20.

Mansfield-Devine, S. (2017) 'Ransomware: the most popular form of attack', *Computer Fraud and Security*, 2017(10), pp. 15-20.

McAfee (2017) *Malware and Threat Reports: Threat Advisory - Ransomware-Erebus* (PD27141). McAfee, CA, USA: McAfee Labs Knowledge Center.

Mell, P., Bergeron, T. and Henning, D. (2005) 'Creating a patch and vulnerability management program', *NIST Special Publication*, 800, p. 40.

O'Brien, D. (2017) *Internet Security Threat Report (ISTR) Ransomware 2017: An ISTR SpecialReport*. Symantec Corporation, Mountain View, CA, USA.

Okhravi, H. and Nicol, D. (2008) 'Evaluation of patch management strategies', *International Journal of Computational Intelligence: Theory and Practice*, 3(2), pp. 109-117.

Palumbo, T. (2015) 'Patch Management: The Importance of Implementing Central Patch Management and Our Experiences Doing So', Proceedings of the 2015 ACM Annual Conference on SIGUCCS. St. Petersburg, Florida, USA. 2815561: ACM, pp. 105-108.

Pressman, R.S. (2010) *Software engineering: a practitioner's approach*. 7th edn.: Palgrave Macmillan.

Rajput, T.S. (2017) 'Evolving Threat Agents: Ransomware and their Variants', *International Journal of Computer Applications*, 164(7), pp. 28-34.

Rankin, K. (2017) 'Hack and /: sysadmin 101: patch management', *Linux Journal*, 2017(279),p. 5.

Rankin, K. (2017) 'Hack and /: Orchestration with MCollective, Part II', *Linux Journal*2017(273), p. 5

Renaud, K. (2016) 'How smaller businesses struggle with security advice', *Computer Fraud and Security*, 2016(8), pp. 10-18.

Seo, J.-T. Choi D-S, Park E-K, Shon T-S, Moon J. (2005) 'Patch Management System for Multi-platform Environment', *Parallel and Distributed Computing: Applications and Technologies. PDCAT 2004.* Berlin, Heidelberg. Springer Berlin Heidelberg, pp. 654-661.

Seo, J.-T. Kim Y-J, Park E-K, Lee S-W, Shon K, Moon J.. (2006) 'Design and Implementation of a Patch Management System to Remove Security Vulnerability in Multi-platforms', *Fuzzy Systems and Knowledge Discovery.FSKD 2006.* Berlin, Heidelberg. Springer Berlin Heidelberg, pp. 716-724..

Souppaya, M. and Scarfone, K. (2013) 'Guide to enterprise patch management technologies', *NIST Special Publication*, 800, p. 40.

US-CERT (United States Computer Emergency Readiness Team) (2015) *Top 30 Targeted High Risk Vulnerabilities* (TA15–119A). Washington, DC. Available at: https://www.us-cert.gov/ncas/alerts/TA15-119A (Accessed: 29 June 2018).

Walberg, S. (2008) 'Automate system administration tasks with puppet', *Linux J.*, 2008(176),p. 5.

8 END NOTES

[i] https://ksplice.oracle.com
[ii] https://access.redhat.com/articles/2475321
[iii] https://www.suse.com/documentation/sles-12/book_sle_admin/data/cha_kgraft.html
[iv] https://github.com/vFense/vFense
[v] https://fai-project.org
[vi] https://spacewalkproject.github.io
[vii] https://www.ibm.com/security/endpoint-security/bigfix
[viii] https://www.ivanti.com
[ix] https://access.redhat.com/products/red-hat-satellite
[x] https://www.solarwinds.com/patch-manager
[xi] https://www.flexera.com/enterprise/products/software-vulnerability-management/corporate-software- inspector
[xii] https://www.kaseya.com/resource/kaseya-patch-management
[xiii] https://www.gfi.com/products-and-solutions/network-security-solutions/gfianguard/specifications/patch-management-for-operating-systems
[xiv] https://www.manageengine.co.uk/patch-management/knowledge-base/overview.html
[xv] https://www.microfocus.com/products/zenworks/patch-management
[xvi] https://www.cloudmanagementsuite.com
[xvii] http://www.bmc.com/it-solutions/bladelogic-server-automation.html
[xviii] https://www.quest.com/products/kace-systems-management-appliance/patch-management- security.aspx
[xix] http://www.celeryproject.org
[xx] http://eventlet.net
[xxi] https://pypi.org/project/django_celery_results
[xxii] https://www.rabbitmq.com
[xxiii] https://www.fabfile.org
[xxiv] https://getbootstrap.com
[xxv] https://github.com/CodeSeven/toastr