Investigation into remote monitoring of complex network systems

MSc Professional Computing
Dissertation Stage

Jonathan Andrew Haddock
Staffordshire University
h000112a@student.staffs.ac.uk
jonathan.haddock@bcs.org

A project submitted in partial fulfilment of the requirements of Staffordshire University for the degree of Master of Science

September 2014
Supervised by James McCarren

1 Abstract

This project looked to address the problem of IT system outages by reducing the amount of time the system would remain down. RESO (**RE**duce **S**ystem **O**utages), an Android smartphone app, provides system administrators a secure way to view system health and remotely instigate corrective action.

Given the wide variety of network monitoring systems in use, RESO does not aim to only detect the state of the IT environment's health, but instead will take data from other applications which could then be extended to output data based on the required XML specification. Projects such as Nagios/Icinga can be easily extended to output XML in the required format.

A proof-of-concept system has been designed and built showing how such a configuration would work - further work would be required to make the solution fully operational.

Contents

2 Acknowledgements

The author would like to acknowledge the feedback provided by those anonymously responding to survey questions. Thanks are also given to Jason Bramley and Andrew Cassidy who were interviewed as part of this project to better understand requirements of IT administrators. No fees were paid to any respondents who gave their time voluntarily.

The Internet community at sites such as stackoverflow.com and various blogs has been invaluable at finding solutions to problems as this author has learnt Java, providing examples and often saving many hours.

Thanks also go to James McCarren, this author's supervisor at the University of Staffordshire, for his guidance and support during the course of this project.

Part I

Introduction

3 Introduction

The challenge of this project will be to develop new monitoring approaches suitable for a rapidly changing industry. The medium used for remote monitoring is also changing along with developments in voice and hand controlled interfaces. The security implications will also be considered, specifically "if an IT Manager can monitor a system remotely, can it also be influenced remotely?".

A number of existing monitoring solutions are available, both commercial and open-source (products such as Nagios, GFI Max Remote Management); this paper will critique their approaches also investigate the wider issues relating to system health monitoring.

Part II will review existing literature on the topics of project management, monitoring systems, cloud computing and encryption while also investigating existing network monitoring solutions. Results of primary research can be found in part III (sections 15 and 16) with an analysis of this author's survey and interviews can be found. This research was critical to understanding the requirements of the IT administrator community.

Based on information gathered in earlier sections the system design and implementation is discussed in part IV with testing and final conclusions found in parts V and VI respectively. This document ends with references and an appendix detailing questions from the survey/interview and a glossary of terms.

4 Background

Over the last 50 years there has been a fundamental change in computer networks. Whereas historically IT within a business was a self-contained system, with servers and clients being located together or geographically close, the Internet changed how the system operates. Now it is not uncommon for clients to be a significant distance from their server, or for servers in different offices to need to communicate. It is not unreasonable for clients to communicate with a server on a different continent. For the IT Manager, a problem exists in how to manage a system where the client may be in Australia while the server is in London. Additionally, IT professionals are often expected to travel between sites so are not always at their computer in order to monitor and remotely connect to the network.

System downtime may not be an entire network but simply one area where there is a form of denial of service. For example, a busy shipping office may lose income if a printer failure caused by lack of ink prevents them from producing shipping notes. As another example, the failure of a company's main PBX can cause loss of customer confidence if the business is particularly phone orientated. The estimated financial loss caused by system outages varies, however, research by Emerson Network Power in 2011 suggests data centre down time costs $5,600 per minute (Emerson Network Power, 2011). In order to remain of value to the company, an IT manager needs to minimise downtime and receiving near-instant notifications of problems can assist in this.

It is this author's opinion that mobile technology is an under-utilised area of technology that, despite being key to receiving email on the move has not benefited the administrator fully.

5 Hypothesis

It is this author's view that mobile devices are underutilised by network administrators when monitoring networks and responding to detected problems. Although responding to an incident from a remote, mobile, device has security implications it is also this authors view that risks can be mitigated or brought to a more acceptable level.

6 Business case

Businesses often require a formal business case for any expenditure, especially given the economic climate of 2014. If an organisation were to invest in a project such as this the following business case may be appropriate. Where costs are used they are taken from responses to interview questions (see section 16).

6.1 Summary

Implementing a network monitoring system with support for remote fixes

As the organisation's reliance on IT grows it becomes increasingly necessary to ensure the uptime of systems providing services to customers. It is estimated the cost of downtime of all systems for one business day would be £17,000, not to mention the cost to the company's reputation.

This business case outlines the proposal to implement a network monitoring solution, detailing the benefits in a range of business areas. Investing in the preferred route, providing the necessary hardware and software solution, is suggested to cost approximately £800 plus in-house development time (existing department smart phones would be used, saving costs). Conversely, the cost of not implementing the system is significantly more. Research has shown that an IT health monitoring system is essential to the success of the business as this allows problems to be dealt with when detected, rather than once end-users begin reporting the fault. Interviews with other IT professionals showed that in their business the costs of implementing a monitoring system were far outweighed by the benefits and savings provided by the system.

The Company's IT department would be responsible for implementing this project.

6.2 Who will this benefit?

The organisation
More and more of a company's activities are computer based. Computers are involved in the majority of processes from communicating with customers by email, receiving and logging telephone queries to finance and service delivery. Put simply, if there is an outage of the

system at least one company employee is not able to work and the company is losing money. System outages are also frustrating for users, potentially causing them to look for employment elsewhere.

A Network Monitoring System (NMS) will enable IT to better respond to problems, potentially before they become apparent to end users and customers, providing a better service and overall experience.

Customers
Consistent system uptime allows the company to serve its customers, be that when they phone customer services or use the company's website. System outages, such as to a core website feature as shown below, can cause customers to turn away which in turn leads to lost revenue.

Where a customer is unable to use the company's services there will be a negative impact from a publicity perspective with customers taking to services such as Twitter and Facebook to vent their frustration. Negative publicity in turn causes lost revenue as the company's reputation drops.

IT Department
Currently the IT department is responsible for monitoring all services required and offered by the company. Performing these checks by hand significantly adversely affects the department's ability to provide user support and innovative solutions. By reducing the administrative burden the IT department will be able to better serve the company, presenting an increased return on investment within that team.

6.3 Summary of benefits

- IT department is notified of systems experiencing an outage
- IT department can monitor the overall health of the system, detecting factors that will become a problem if left unchecked
- Proactive, preventative maintenance
- Fixes can be applied remotely, without a traditional network connection

The final point is key. For a large number of problems the resolution can be triggered through a single command (or collection of commands in a script). By being able to remotely trigger these commands from a dashboard, systems can be repaired regardless of the location of the administrator providing they have a data connection. With the proliferation of mobile networks offering 3G and 4G services this is more possible than ever before.

6.4 Impact on the IT team

The IT team spend approximately half a day each day making checks on servers to ensure they are operating correctly and performing tasks key to the business' IT resilience. Implementing a monitoring system will free up members of the team to better service the Company's users and allow them to design and implement new projects and resources.

As an example, the table below shows estimates of the IT manager's day-to-day work both with and without the system.

Task	Time estimate	Time with monitoring system
Check server disk space	3 minutes *per server*, 30 minutes total	2 minutes
Check Anti-virus environments	15 minutes	2 minutes
Audit servers for "hack attempts"	5 minutes *per server*, 50 minutes total	2 minutes (read report)
TOTAL:	**1 hour 55 per day**	**6 minutes per day**
Saving of 1 hour 49 minutes *per day*. This illustration is based around 10 servers		

6.5 Cost estimate

Software and licensing costs
Assumed as zero cost
It is intended to use Free Open Source Software (FOSS) for the majority of the system. While this is not commercially supported there is expertise within the existing team to implement and maintain the proposed system.

Commercial systems, such as GFI Max Remote Management cost upwards of £5 per monitored device, by using FOSS there's a clear cost saving. Despite the costs of the system being low, training on the system would incur a cost in either time or course fees.

Hardware cost estimate
£800 (ex-VAT, as at September 2014)
It is proposed the monitoring system runs on a dedicated physical server in order to reduce complexity in the deployment. Although a Virtual Machine (VM) could be used this would potentially negatively skew the system's output dependant on the hypervisor's load. A hypervisor is software that allows a computer to run multiple operating systems, simultaneously, on the same hardware. Examples of hypervisors are VMware ESXi and Microsoft Hyper-V.

- Dell R220 server
- Intel 3.4GHz processor
- 8GB RAM
- 2x 500GB SATA HDD, RAID 1
- Dual NIC
- 1 PSU

Existing department smart phones would be used as these are capable of running Android an any such app which is developed.

Total cost estimate
£800 plus in-house development time

6.6 Estimated savings

Presently when the IT manager is away on business it is necessary to hire in third-party contractors to manage the system unless there are other employees with relevant expertise. By moving to this system, which allows the network manager to remotely apply fixes from his mobile device, a saving of £800 per day in hire charges can be immediately realised. Similarly travel costs may be saved where problems occur out of hours with the network manager being able to respond within minutes from the mobile device versus the time and cost of having to get to and use a computer to connect in. The cost of this is harder to quantify, however, with every minute of downtime costing the company both financially and reputationally the cost savings should be significant.

6.7 Resource estimate

Research and development time is required to design, install and implement the system in addition to producing the necessary mobile device components. It is estimated this would take 80 man-hours.

6.8 Risks

This project has few dependencies, notably dedicated hardware and project time, resulting in low risks. In the event that project team members were unavailable the project would be delayed, however, this would not have a noticeable impact on the Company's users or customers.

Overall, this project can be classed as *low risk*.

6.9 Consequences of project rejection

Rejecting this project offers no benefit to the organisation, and in fact harms it, as system outages will need to be reported by users at which time there is already a problem. Unplanned system downtime equals lost revenue for the Company and although this can't be predicted it is known outages can cost the Company upwards of £17,000. Additionally, the Company will *not* save money by rejecting this project as members of the IT department will need to be

dedicated to monitoring system health which will result in a net loss in available man-hours for other tasks. In turn this may require the hiring of additional staff with a consequent increase in company costs.

As mentioned, system outages also have negative implications from a public relations and reputation perspective. While the costs of this cannot easily be quantified there is a potential drop in consumer-base which should be considered.

Part II

Literature review & existing solutions

7 Project Management

For irregular tasks, such as developing an application or building a house, various project management techniques are used to ensure a project completes successfully. Similarly, routine tasks are subject to written procedures which are followed. In project management, a project is split into stages with guidance given for each stage. Generally, a project ends once the task is appraised and signed off as complete, a process which may include a "snagging list" where the customer identifies areas for modification. Not every project reaches the completion stage with projects being deemed too costly or suspended.

Although there are numerous models for project management there are common elements across the schemes. *Analysis* of the tasks and *requirement gathering* often form the first tasks of a project allowing *planning* to follow. The initial requirements and analysis will be used in the design phase and later to evaluate the success of the project. The authors of *The Pragmatic Programmer* tell us that it is unreasonable to expect to understand all the requirements of a project by merely listening to the end user. Instead, the programmer needs to observe the users to fully comprehend system requirements and completely meet the needs of the business (Hunt and Thomas, D., 1999, p.204).

Project management is essential from a financial perspective as costs and the duration of the project are determined and maintained by following set controls. Perhaps most importantly, project management provides a direction for the project and by monitoring progress the direction can be maintained, avoiding the risk of a runaway project that doesn't fulfil the required outcomes.

7.1 Traditional / Waterfall

The traditional model, also known as waterfall, takes feedback from one project stage and feeds it in to the next. Despite waterfall being a widely recognised model, and one still taught in schools, Royce explains in his paper that an iterative approach is best. Contrary to popular belief, Royce does not advocate the use of the waterfall model in his paper, merely

outlining it in his opening pages (Royce, 1970).

In environments where goals are static and not subject to change the waterfall method does have its uses. Requirements are outlined at the outset and work is conducted based on the specification - there is little scope for change within the waterfall model. Software development is a dynamic process where requirements change regularly making the traditional method less applicable to modern software development.

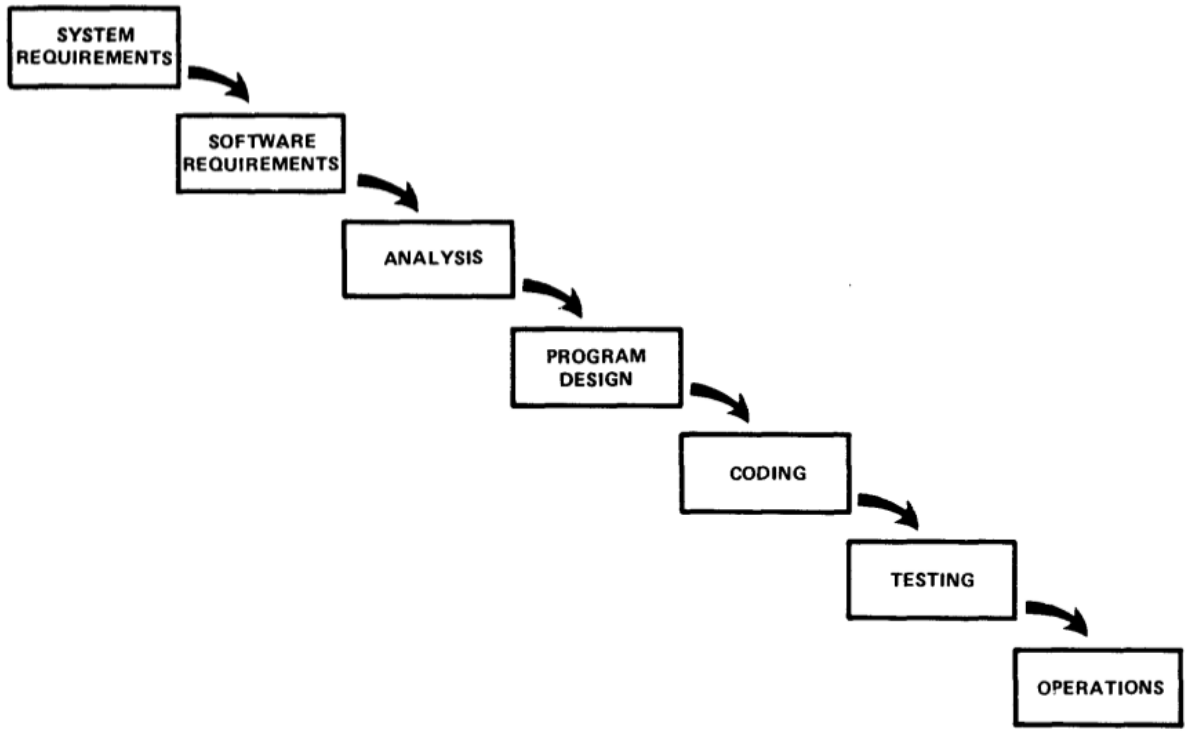


Figure 1: The waterfall project management methodology (Royce, 1970)

Waterfall is criticised for placing software testing at the *end* of the process, often when the project budget is near to used if not overspent (figure 1).

7.2 Agile Software Development

Agile software development expects specification changes and appreciates that not all requirements can be discovered prior to project commencement. A focus of Agile is to provide software in a short time frame on the understanding the customer would rather have working software than reams of documentation. Collaboration is also a value of the model and is encouraged both within the development team and with the customer. By consulting the

customer regularly, developers are expected to react to changes in requirements in preference to following a rigid plan as would be expected within the waterfall model.

Agile is relatively young, formalised in 2001 with the publication of the Agile Manifesto (Beck, K. et al., 2001). The manifesto does not discard practices that are prioritised in other management styles but instead values them differently. In the case of documentation, the manifesto states that "working software" is more valued than "comprehensive documentation", a stance that can make software developed under Agile methods more difficult to support. The methodology is based on 12 principles which include:

- *Frequent software updates* - software should be released regularly. Progress is measured by the working state of the software.
- *Changing requirements shouldn't be discouraged* - requirements are expected to change and this should be welcomed.
- *Attention to detail* - programmers should strive to write code in line with best practices and by following a good design.

Agile also details how project stakeholders should communicate, explaining that face to face communication is best and teams should be able to organise themselves. Customers should be consulted regularly, both when determining the original requirements and as the project progresses; Agile highlights a satisfied customer is key to the project.

7.3 Test driven development

Test Driven Development (TDD) could be considered as working backward when compared with other approaches. Traditionally code has been written to a specification and then tested by the developer or QA teams once the code is available. In TDD the test is written first, based on the specification provided. Once a test has been written the full test suite is run and the new test should fail as there is no code to support it yet; if the test passes then it should be re-written. Following a failure run of the new test the code is written and the test suite run again. So long as the test passes the code can then be refactored, tidied and moved to the correct location in the program logic and the next test is written. The first revision of the code is designed to pass the test and is not expected to be perfect.

Another goal of TDD is to remove duplication wherever possible. Firstly, duplication is inefficient: it's difficult to manage multiple instances of the same logic and makes source code larger. Admittedly, storage is cheap by modern standards, however, compilation time

may not be. Secondly, if a change is required it must be made in multiple places. For example, in *Test driven development: by example*(Beck, 2002) the author gives an example of a system with a dependency on a SQL server. If the SQL server type were to change (for example from Microsoft SQL to Oracle) it would be necessary to make multiple changes if code was needlessly duplicated.

Figure 2 explains this process more clearly.

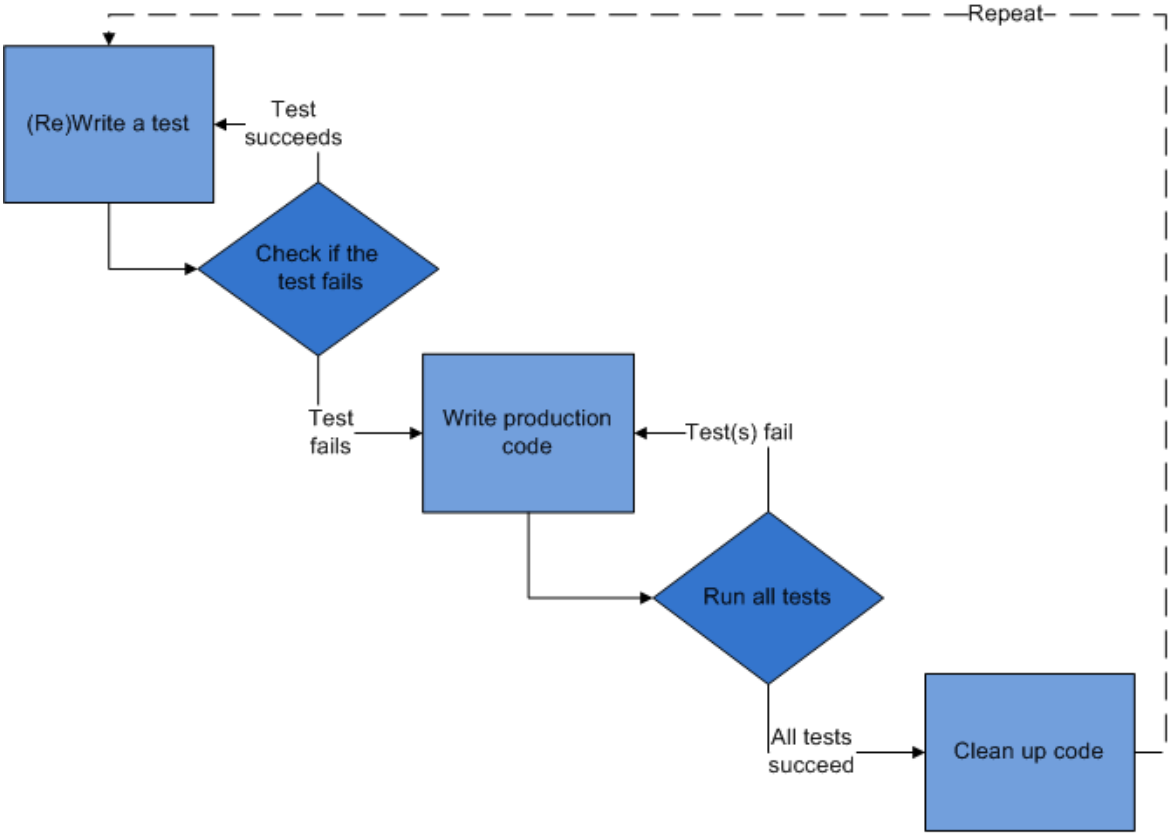


Figure 2: Cycle of test driven development(Excirial and Renier, M, 2009)

7.4 Iterative and incremental development

Despite the waterfall model shown by Royce in *Managing the development of large software systems*(Royce, 1970) it should be noted the author himself states that although the "waterfall" model is valid it "is risky and invites failure" (Royce, 1970, p.329) and he spends the remainder of the document advocating iterative practices. In Iterative and Incremental Development (IID) a project is split into smaller portions which are worked on, tested and adapted before moving on to the next segment. Not all developers consider the segmenting

of a project to be beneficial as this can lead to software with a poor architecture as the system has not been considered as a whole (Atkinson and Hummel, O., 2012). This author considers the problem is largely dependant on the practice of an individual developer - if the developer considers the system as a whole when dividing the system then this issue is reduced.

IID has become associated with the Agile development movement, however, it seems to pre-date this by some time. Possibly the most famous project to use IID was the NASA project "Mercury" and we see in *Iterative and Incremental Development:A Brief History* the approach has been around since at least 1968 (Larman and Basili, V.R., 2003). Cockburn gives clear examples of how iterative and incremental development can be performed individually highlighting how the methods work better together(Cockburn, 2008). *Incremental* approaches involve segmenting the system and building each component, later integrating it into the whole system as each component is completed. *Iteration* is the process of improving what already exists following feedback or the results of testing.

Figure 3 shows this author's understanding of this process based on an amalgamation of the diagrams in *Using Both Incremental and Iterative Development* (Cockburn, 2008).

8 Monitoring systems

All monitoring systems require configuration. A common problem when configuring a monitoring system is that in order to be effective the administrator needs to know the acceptable values for each check. For example, an administrator could configure a hard disk free-space check to only alert when 1GB is free. This may not be a problem for a print server configured with few resources, however, could cause a Microsoft Exchange server to stop routing mail because the minimum required disk space is 10GB. System misconfiguration can lead to system downtime and when the misconfigured system is the one designed to assist in preventing downtime the system has failed in its goals.

Nagios is a powerful monitoring system which this author has used previously to monitor an estate of around 20 servers plus switches and printers. Although the solution performed the required task, alerting administrators to problems, the configuration was threshold based. Although an alert was flagged the system was not able to provide additional guidance, therefore administrators were left to troubleshoot. While this was not a problem in a medium sized environment it wouldn't scale to the data centre.

This author is also familiar with GFI Max Remote Management which, again, is threshold

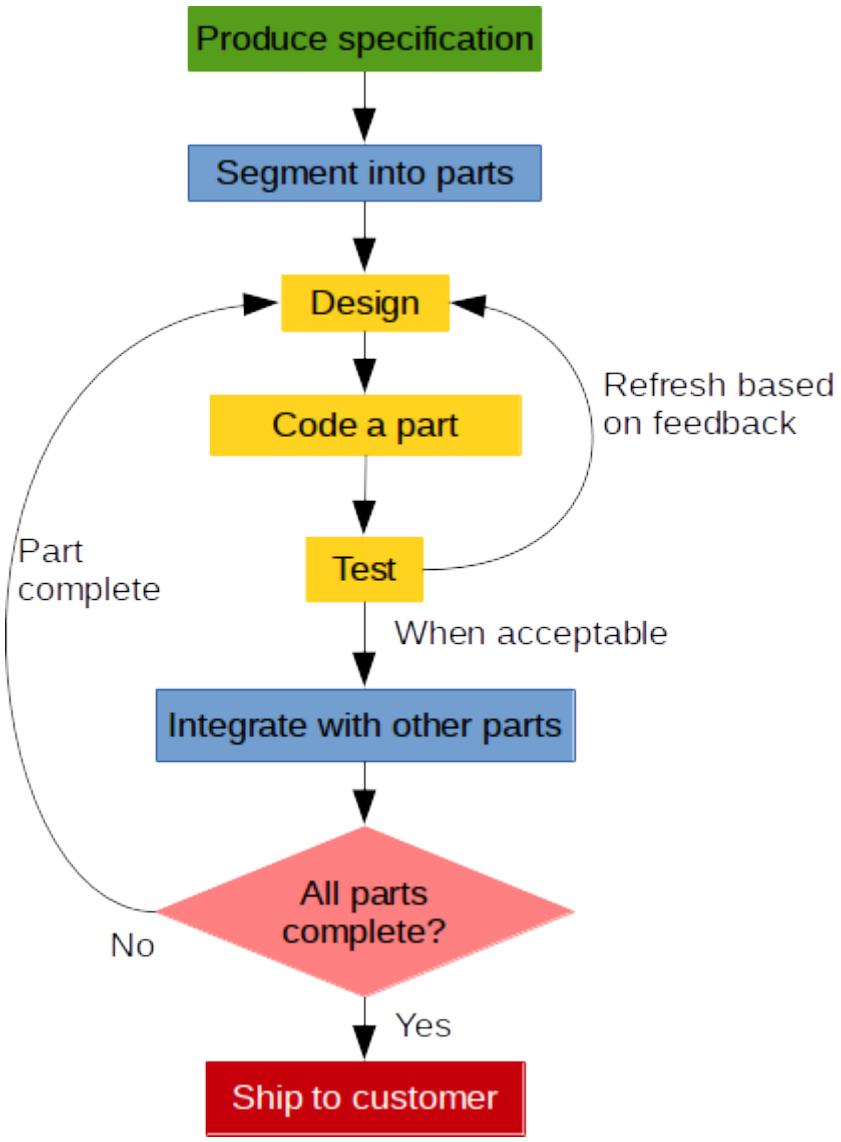


Figure 3: Cycle of IID based on Cockburn (2008)

based. Both GFI and Nagios allow the administrator to provide custom scripts, facilitating a wide range of checks, but as with the overall configuration these scripts are not immune to error. It has been observed that some checks are left disabled, particularly performance checks, as these appeared to be generating false positives. On reflection, it is more likely the check was misconfigured, the defaults not being appropriate to the server being monitored. A lack of historical baseline will have contributed to the configuration error and Sun et al. discuss this in their paper (Sun et al., 2006).

Sun et al. outline 3 weaknesses of existing monitoring software as *configuration*, *reasoning* and a *lack of historical data* (Sun et al., 2006). To be correctly configured an administrator

must have expertise in managing the system so the systems optimal baseline must be considered. The administrator must set thresholds appropriately for each check (see the disk space example above) enabling the system to alert on discovering a problem. The authors explain *reasoning* as the problem an administrator encounters when receiving a single alert - knowing the alert has been triggered still leaves the administrator the task of determining the root cause of the problem. Finally, a *lack of historical data* makes it near impossible to determine the optimal baseline for a system, making configuration difficult but also making any alert solely down to the threshold set potentially months before.

Sun et al. produced InteMon which was designed to monitor large data centres and clusters by analysing data and highlighting deviations from the measured norm. The authors describe an example victory for InteMon where although the temperature of the data centre was within tolerance the air intake was found to have excessive humidity causing the air to be over-cooled (Sun et al., 2006, p.51). Traditional systems may not have noted this problem, given temperature wasn't effected, however, system administrators were prompted to check the air intake before the air conditioning's reheat cycle elevated the room temperature. InteMon flags anomalies by comparing present data with historical data, allowing it to generate an alert should a problem be noted.

It is not unusual for monitoring systems to generate support tickets when detecting a problem, however, as Tang et al. note in their paper, transient events, for example a spike in CPU or memory usage, can cause a ticket to be raised. Once a ticket is created human interaction is required, inflicting costs as their IT staff are required to investigate and resolve. Where an administrator does not need to take any action the ticket can be classed as a *false positive*. Research showed that for the monitored environment 75% of transient alerts could be cleared automatically after 20 minutes so these were classed as false positives(Tang et al., 2013).

> The term *false positive* is widely used and understood to be something which has flagged incorrectly. For example, a false positive alert in an IT monitoring system might be a notification that a server is off-line when it is not. Some transient alerts may also be classed as a false positive as there is no work for an administrator to do.
>
> Less common is the term *false negative*, the occasion that a problem is not detected. Misconfiguration can lead to many false negatives, perhaps a service not being highlighted as unavailable. Checks need to be relevant and checking a web service is available to "localhost" is not the same as verifying the site is available to external users (an example of a misconfiguration which may produce a false negative).

9 Cloud computing

Cloud computing is being increasingly adopted by companies as a means to provide IT facilities to their employees. Generally there are two types of cloud - *public* and *private*. Computing has changed dramatically in recent years following the removal of the requirement for clients and servers to be near to each other. As evidence of this growth, it is expected the UK cloud computing market alone could be worth approximately £10.5 billion in 2014 - a growth of over £4 billion in 4 years (Smith, 2012, p.2).

Whereas monitoring a system used to be as simple as checking on a handful of servers "in the next room", an organisation's server estate may now be significantly bigger and geographically spread thanks to good communications links. To further complicate environments, servers may be held in a data centre to which the administrator has no physical access.

Cloud computing allows access to resources (files, databases, software) from a multitude of devices at numerous locations rather than previous reliance on locally installed applications and stored files (Smith, 2012, p.2). The concept of supplying services via the Internet is not new, previously being referred to as Application Service Provision (ASP) in the 1990s. Dramatic changes in the provisioning of services, particularly what is required of the customer, has transformed ASP into cloud computing, now an IT buzzword and growing industry.

Public clouds are managed by a third party and an organisation is billed for their usage. Examples include Microsoft Azure and Amazon EC2. A benefit of the public cloud is reduced cost as costs are shared by many organisations through "rental" fees. Microsoft Azure, an example of a public cloud, offers numerous services at low cost including cloud hosted databases.

Private clouds are accessed only by the organisation which "owns" it, meaning that organisation is responsible the initial setup and ongoing maintenance. de Chaves, S.A. et al. comment in their paper (de Chaves, S.A. et al., 2013) that security of a private cloud is "not a major concern when compared to a public cloud" due to access being provided through internal interfaces. They don't appear to consider the wider implications relating to system misuse once the cloud resources are compromised. In the same paper, the authors comment that monitoring has not kept pace with the expansion of cloud services and that there isn't a single solution to this problem.

The term *hybrid cloud* has also come into use and describes a scenario where some resources are accessed from dedicated servers while others exist in a cloud hosting environment. Hybrids may be used where there is a need to keep highly sensitive data fully controlled or as part of a migration from a private to public cloud setup. Additional processing power can also

be obtained through *cloud bursting*, a situation where some processing is passed off to often more powerful cloud based servers.

10 Existing solutions

Where possible, academic case studies and research have been referenced as part of this evaluation, however, these were not always available. Some vendor provided case studies have been referenced but should be considered excessively biased in favour of the vendor. Personal experience of this author is also included.

Existing solutions largely fall into two groups, paid-for and free, and examples of both categories have been examined. Using Google Trends, which shows the total number of times a precise search term has been used in a given time period using the Google search engine, the following graph has been produced. This graph suggests Nagios is the most common system monitoring solution investigated (and, therefore, potentially used) although it should be noted that this graph is not a guaranteed indicator as to which system is most in use. GFI Max Remote Management doesn't register on the graph, potentially due to this product being offered via a Managed Solution Provider which then re-brands the product.

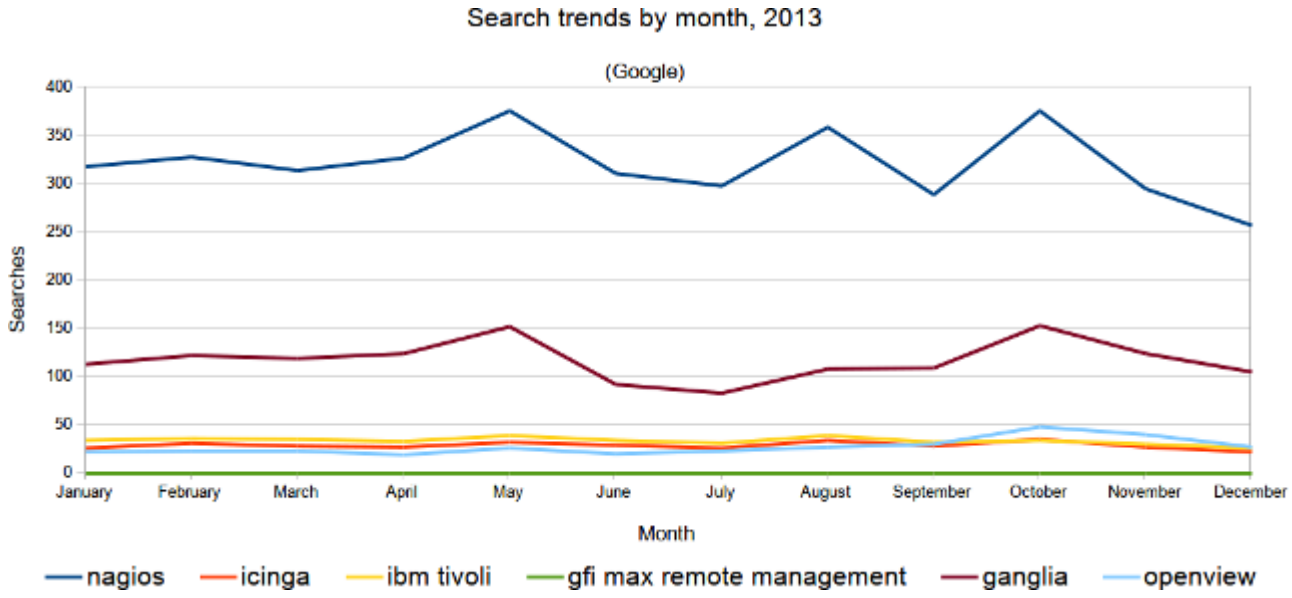


Figure 4: Most common search terms - monitoring systems

Monitoring can either be *active* or *passive*. For active monitoring, an agent or utility is installed on the server or device and this agent submits its data to a central monitoring server. GFI Max Remote Management is one example of active monitoring. Where there

is no additional software installed on the device, monitoring is considered to be passive and results are obtained from "the outside". Smokeping and Nagios are examples of passive monitoring.

Active monitoring may provide more detail as the full Operating System (OS) is exposed to it. Conversely, passive monitoring has to rely on firewall exceptions and services on the monitored device exposing themselves and their information to the network. Some tools, such as NSClient++, used with Nagios, expose additional information to passive solutions. In this example, Nagios itself is still considered passive as it is not responsible for the agent installed on the device.

10.1 Nagios

Of the papers reviewed, a number chose to expand on Nagios, an open source monitoring solution (it should be noted there is now a commercial version of Nagios, Nagios XI). The main criticism appears to focus on the difficulty of configuration (Issariyapat et al., 2012)(Wu et al., 2013). To configure checks there are numerous text files to which details must be added. In this author's experience the configuration is also case-sensitive so specifying a server's name as `FileServer` in one place but `Fileserver` in another (an easy mistake) will potentially result in an error.

When troubleshooting problems it has already been indicated that historical information is of benefit (Sun et al., 2006). Nagios provides this feature although in a primitive fashion (it was criticised by Issariyapat et al. in their paper). Issariyapat et al. cite another problem with Nagios is a lack of consistent primary keys (as a monitored node can change name, adding historical monitoring via a plug-in is problematic as historical data would be lost on name change) (Issariyapat et al., 2012, p.2772).

Nagios' greatest strength appears to be its flexibility and architecture. Plug-ins and modules can be added to provide additional functionality. The Nagios Core documentation states there are a "several methods of integrating Nagios with the management software you're already using" (Nagios Core Development Team, 2010, p.325) and shows the architecture of Nagios as in figure 5. Administrators need not change the main source code of Nagios to add a feature as integration points allow for this instead. This design is advantageous as third-party additions are not lost each time an update to Nagios is released.

Given this extensibility it is possible to monitor virtually anything in Nagios so long as a script can be written - the tool can be used to monitor more than just traditional computers. Monitoring server room temperature, for example, in addition to the availability of servers

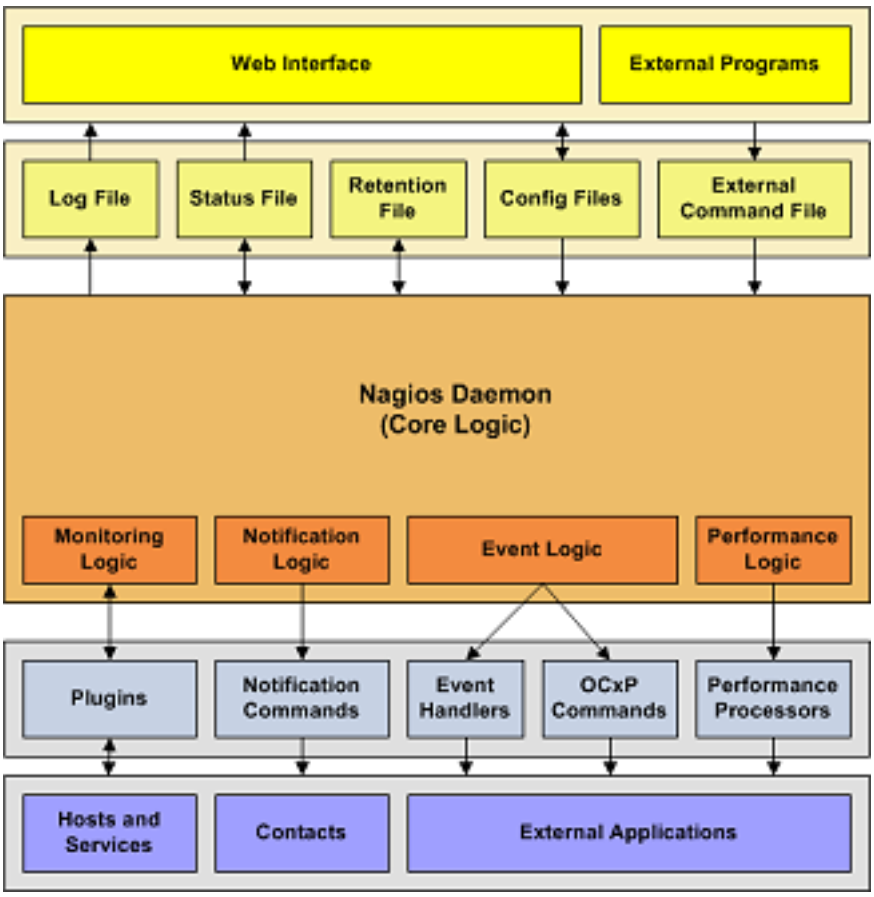


Figure 5: Nagios architecture (Nagios Core Development Team, 2010)

can quickly alert administrators to excessive temperatures in a server room which are the likely cause of some service problems.

Issariyapat et al. highlight that a simple ICMP ping check as provided by `check_ping` is not sufficient to confirm if a host is up or down as many devices now do not respond to ping. Thanks to Nagios' flexibility they produced an additional check, `check_accessibility` which monitors availability of a host based on ping and TCP / UDP port checks. The web-based interface to Nagios is also limited so Issariyapat et al. produced NetHAM which generates output based on results provided by Nagios. NetHAM is also capable of showing what links are active between devices, using information obtained via Simple Network Management Protocol (SNMP), and considers redundant links. This allows the authors to show *Switch A* is connected to *Server 1* by 2 connections and that one connection is done. This is not possible in Nagios which only supports a basic parent/child relationship.

Icinga is a fork of the Nagios project and aims to address some of the criticisms mentioned above. It aims to be backwards compatible with Nagios, maintaining compatibility with

plug-ins and extensions.

10.2 Smokeping

Smokeping focuses on providing graphical representations of latency to an administrator. In figure 6 we see a clear indication that latency is greater during a backup (the graph has been annotated to highlight this) and due to the clear nature of these graphs administrators are able to quickly troubleshoot problems. That said, troubleshooting was only possible because the system administrator knew the effect the backup had on network connectivity to the server in question - it was still necessary for a human to interpret the graph.

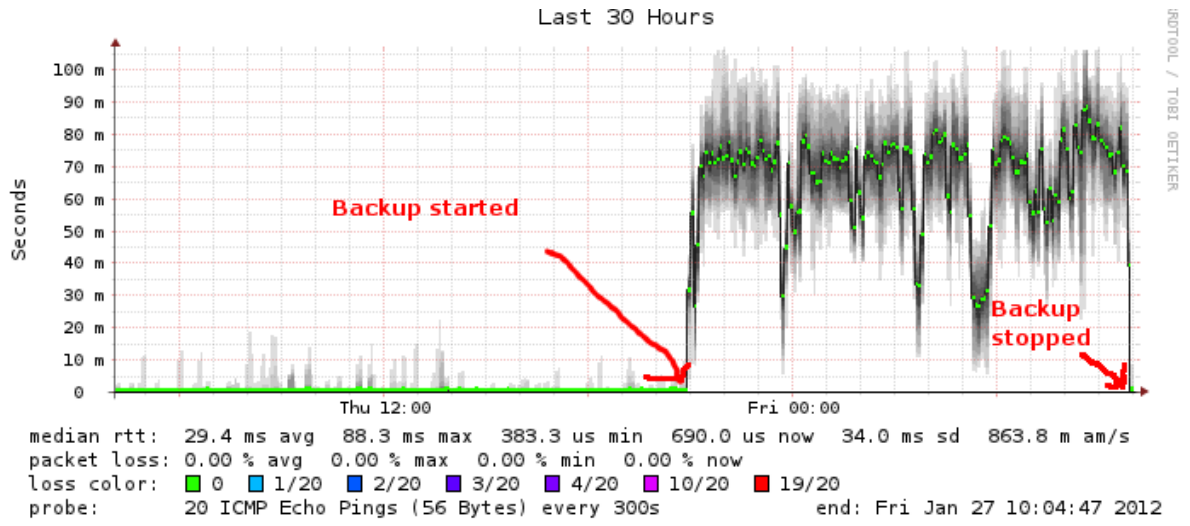


Figure 6: A Smokeping graph showing ping latency to a server before, during and after its backup

Smokeping is not as comprehensive in its checks as Nagios although there are many plug-ins available providing numerous checks. Similar to Nagios, Smokeping's configuration is based on text files and the syntax is equally awkward (albeit, in this author's opinion, easier to become proficient with). Slave nodes can be configured which read their configuration from the master host. This approach has the advantage that not only saves the administrator time but also that multiple Smokeping hosts can be placed on a network so a variety of measurements can be taken. This is especially useful if the physical LAN is large and slowness is only detected in certain geographical areas.

Through changes to the configuration it's possible to adjust the size of packet sent for each test (by default, tests repeat every 5 minutes). The type of packet and test can also be adjusted, Smokeping provides support for standard ICMP ping, HTTP and generic connection

tests and ICMPv6 ping. Cody-Kenny et al. used Smokeping to monitor round trip time for ICMPv6 packets over a wireless sensor network and they showed packet loss was virtually 0, despite increasing the packet size. Smokeping also showed round trip time increased with packet size.(Cody-Kenny et al., 2009)

Alerts can be sent by email or Simple Network Pager Protocol when Smokeping detects increased latency.

10.3 Ganglia

Unlike the solutions reviewed above, Ganglia is an active solution with a daemon[1], `gmond`, running on each monitored node. Ganglia then uses multicast traffic to ensure each node is aware of the overall health of the cluster (Massie et al., 2004, p.821). As `gmond` listens on a well-known multicast address on each node there is no need to manually configure membership of the Ganglia monitoring system - it is essentially configured by auto-discovery.

A second daemon, `gmetad`, is used to gather information from multiple nodes and can be used to federate information from multiple clusters. A web front end is used to view performance and health data which is stored in RRD format. `RRDTool`, the same tool as used by Smokeping, is then used to produce graphs of this data.

Massie et al. studied the impact on cluster resources and found this varied based on cluster type. Across 3 cluster types CPU usage for `gmond` was less than 1% with both physical and virtual memory usage below 17MB (Massie et al., 2004, p.830, table 4). Due to its low overheads, Ganglia is well suited to the multi-node cluster with a very small, likely insignificant, percentage of resources being lost to the monitoring system.

Wu et al. used a combination of Nagios and Ganglia in their project: Ganglia to collect data and Nagios to query that data and raise alarms as required (Wu et al., 2013). This, again, demonstrates how one solution is not sufficient.

[1] Linux and similar systems have "daemons", Windows systems have "services"

10.4 GFI Max Remote Management

GFI Max Remote Management is an *active* solution. In order to monitor systems, an agent is installed on the device and checks are configured. Windows and Linux agents exist.

Each monitored system reports back to a central server which provides an overview of system health via a web-based dashboard. Alerts can be sent via email or SMS text message and an API exists enabling 3rd party services to access data. In addition to providing an overview of system health, the dashboard provides remote control features in the form of a remote shell or a full interactive view of the computer. The latter is provided by another integrated product called Team Viewer. Although remote control is provided, it is not possible to remotely instruct a service to stop/restart in the event of a problem. For service management, the agent can attempt to restart the service before alerting the administrator. Compared to passive solutions like Nagios this is a distinct advantage of an active design.

Due to the agent based nature of this solution it's possible to obtain more detailed information on each system as all data is exposed to the client (subject to the agent's access rights). Additional checks can be configured through user provided scripts, authored in any language supported on the monitored OS.

An Android app exists for the system but has been heavily criticised on the Android Play Store as being an incomplete solution with a poor interface. The mobile app doesn't offer remote control and must be connected to the central dashboard by use of an API key, making it difficult to terminate a user's access. An iPhone app is also available.

No academic case studies were available so the above is based on this author's own experience of the product. Numerous vendor provided case studies are available and these, predictably, are biased to show how this product saves their clients money. It should be noted that one case study, with CCR Technology group, stated they had been able to "reduce their workforce". This shows a wider implication of IT health monitoring systems in that companies can reduce staff numbers due to less work being required. (GFI Max, 2012).

11 Encryption

Encryption can be divided into 2 categories, *symmetric* and *asymmetric*(Simmons, 1979), and is key to sending information through an insecure environment such as the Internet (Bryan and McDermott, J., 2011). For symmetric encryption there is only one key which is shared between parties (hence this method also being known as *shared key*). Symmetric key encryption is explained in figure 7.

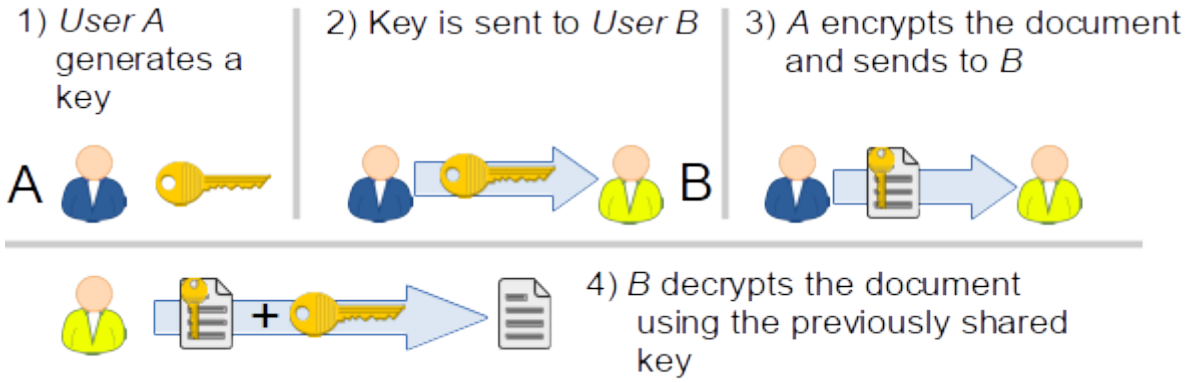


Figure 7: Symmetric key encryption

A weakness of symmetric encryption is found not in its mathematical algorithms but in its basic premise - anyone with the shared key can decrypt the data (see figure 8) so how do you protect and distribute the key? One solution to this problem is to avoid it altogether by using asymmetric encryption (Burke et al., 2000, p.1), illustrated in figure 9, but the simplest solution is to strictly control access to the key. Initial key exchanges must be secure (Parnerkar et al., 2003) and can be performed using physical media, such as a CD ROM, in person. The media can then be physically secured to restrict access by unauthorised parties. Although this approach reduces risk it does not solve the problem of a key compromise exposing all encrypted transmissions.

Asymmetric encryption involves the use of 2 keys *per entity* in order to help protect against this issue. The concept of a *public*[2] and a *private* key are fundamental to this mechanism. Each party generates a public/private key pair and public keys are exchanged. The method of exchange does not matter so can be via an insecure mechanism including widespread publishing. Private keys should be protected and not be shared under any circumstances. It is computationally infeasible to derive the user's private key from the public key (Parnerkar et al., 2003).

When a file is encrypted with a user's public key it can only be decrypted with the corre-

[2]Public keys should not be confused with shared keys. Although a public key is shared the terms refer to components in different encryption schemes

sponding private key and the reverse is true. For this reason it doesn't matter if a user's public key is intercepted as the encrypted data cannot be decrypted with it. Figure 9 explains this more clearly.

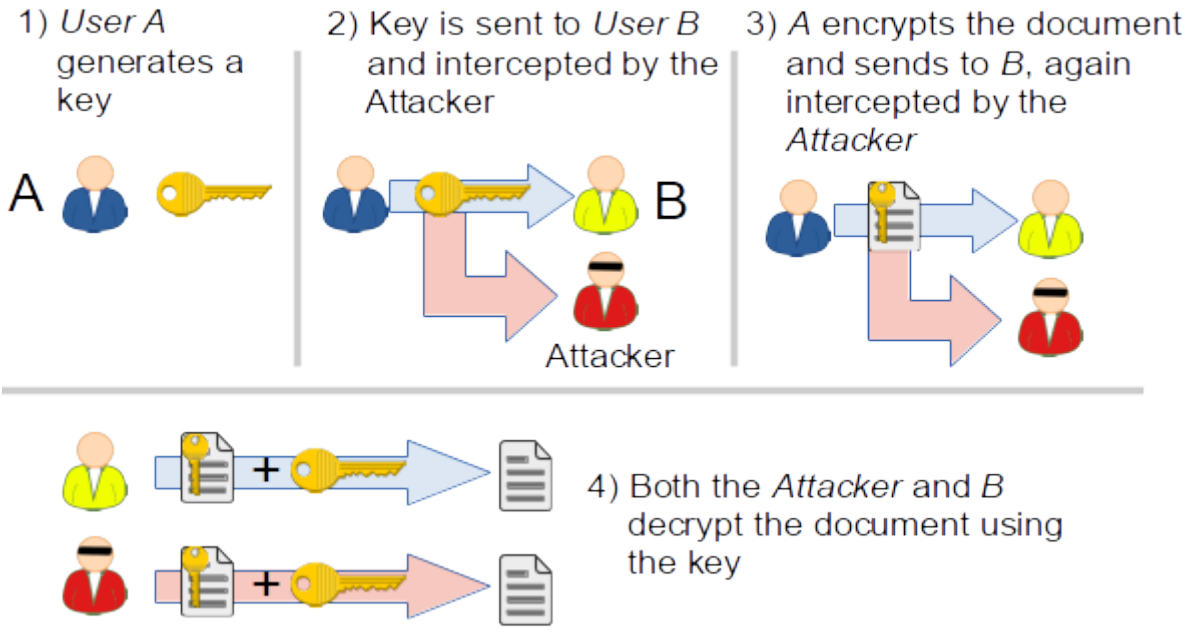


Figure 8: Symmetric key encryption suffers from problems if the key is intercepted

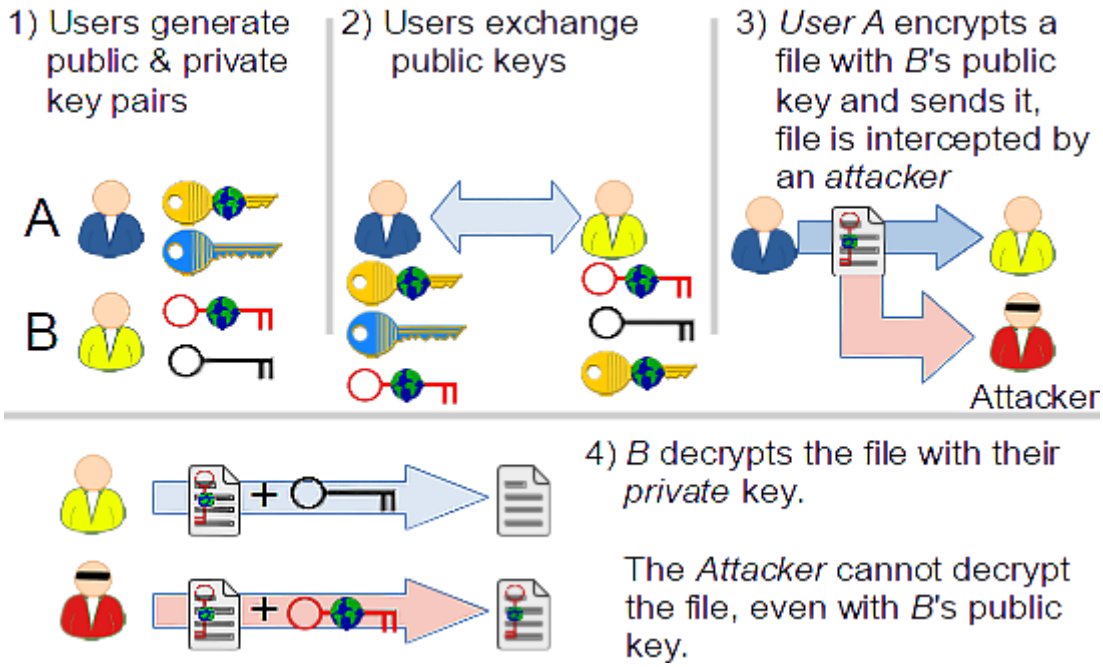


Figure 9: Asymmetric encryption doesn't suffer the same interception problems

11.1 Encryption on mobile devices

Mobile devices are valuable targets for information thieves due to their portability and small size. The news regularly features stories about data loss following devices being left on pubic transport and while losing a device and its contents is always a financial problem the impact on companies can be reduced by encrypting storage. Options for encryption vary dependant on platform; some OSes have encryption built in (for example, BitLocker in some editions of Windows) whereas others require features to be added from other vendors (such as Truecrypt although it should be noted the Truecrypt project was closed May 2014).

Although Android devices have featured encryption as part of the OS for some time, Wang et al. ported EncFS to Android as part of their project to offer protection of data on mobile devices (Wang et al., 2012). Their research showed there was a negligible overhead involved in reading from the encrypted file system yet a 20x overhead when writing.

Symmetric encryption is less processor intensive so is preferable from a mobile device's perspective, where battery life should be considered. To prevent the key being brute forced (i.e. "guessed" by systematically attempting values) a strong key should be used. If using a key based on a text string the text should not be a standard dictionary word.

12 Literature review conclusion

Ruling out the waterfall methodology was accomplished early on, in part due to Royce explaining how the method was flawed in his paper but also due to concerns that a non-iterative system can result in problems at the testing stage. It is this author's belief that testing should take place throughout a project. Agile was also excluded as it doesn't appear to fit with a single developer with no defined customer - there's no scope for regular meetings and nothing to release until the project meets the requirements outlined in section 19. Test driven development, although powerful, was not appropriate here as the author was learning to use the relevant language during this project so would not have sufficient knowledge to create relevant tests prior to writing code.

This author generally employs an iterative and incremental approach when working on projects as this allows results to be seen during development with feedback acted on immediately. As such this approach was chosen for this project.

A single tool is not sufficient for monitoring today's systems which are often specific to a

particular need. For example, Ganglia is better suited to clusters where each node must know about every other node at all times. Nagios has been shown to be regularly criticised when it comes to configuration but is equally praised for its extendibility. Both active and passive approaches to monitoring are important, again implemented based on need.

The challenge of any monitoring system is to keep false-positives and false-negatives to a minimum to ensure an accurate overview of the environment is presented at all times. It is not uncommon to use multiple tools to achieve this goal. Tools should have a minimal performance impact on the systems being monitored and provide historical data for analysis wherever possible. Clearly storing historical data requires additional storage capacity, however, there is no need for this to be stored on expensive, fast, disk arrays. Historical data is not critical to business practice so storing it online on slow media should not cause a problem.

Where support tickets are generated automatically by monitoring systems there is a financial implication as a human administrator must investigate the "fault". As has been mentioned, false-positives will have a negative impact on an IT Department's "bottom line" in financial terms.

With IT systems becoming increasingly more dynamic thanks to cloud computing and practices such as "cloud-bursting" it is necessary for monitoring systems to also become dynamic. Whereas cloud computing providers may pass this responsibility off to consumers it is reasonable for the provider to require an overview of each OS instance running on its infrastructure (if for no other reason than billing purposes). Monitoring is not simply the domain of the IT administrator but increasingly that of the accounts department too.

Encryption is only as strong as its encryption keys so measures must be taken to protect these. If the key is compromised the encryption schema is moot and the message revealed so care must be taken to protect key exchange (Parnerkar et al., 2003)(Burke et al., 2000).

Part III

Research

13 Research methods

There are numerous research methods, the most commonly referenced being *qualitative* and *quantitative* although *basic* and *scientific* are also mentioned (Glenn, 2010). Glenn tells us basic research (also known as fundamental or pure) is often driven by the researcher's curiosity and may have no end point or goal in mind. As a result it can be difficult to receive funding or support from sponsors. Basic research can provide a foundation for further research.

Scientific research follows a process, often aiming to support or disprove a hypothesis. It should be noted that a hypothesis is not "proven" as it is not possible to absolutely state something is true (Glenn, 2010). Scientific research may be shared verbatim to allow others to repeat and validate the results. A definition of scientific research is:

> To be termed scientific, a method of inquiry must be based on gathering observable, empirical and measurable evidence subject to specific principles of reasoning. A scientific method consists of the collection of data through observation and experimentation, and the formulation and testing of hypotheses.
> *(Glenn, 2010, p.12)*

Although it is possible to statistically measure whether IT administrators feel there is a benefit to using mobile devices to monitor system health through use of surveys and interviews it is not possible to divine the features of such a system using scientific methods. Feature requirements will be subjective and it's unlikely that all individuals would agree on the chosen features.

Qualitative research was typically used for social sciences until the 1970s where after it became a multi-discipline research method. Its purpose is to gain an in-depth understanding of why and how the study matter is. There was a view in the 1930s that social research had all the theories it required and that qualitative research could only build on these, however, as observed by Glasser and Strauss, A.L. not all the theories of the "masters" fit the current social climate. Instead they suggest there is guidance available for generating theories (Glasser and Strauss, A.L., 1967, pp.10-11).

A qualitative researcher will aim to immerse themselves in the subject matter, contrasting with quantitative methods where remaining independent is the focus (Lapan et al., 2012). It is acknowledged the perspective of the research subject is important, i.e. how the individual views the topic, and the researcher may also have an effect on what is observed. Observations are targeted at specific groups that are relevant to the topic area (Glenn, 2010). By focussing research in this way more meaningful results can be obtained.

Quantitative research aims to examine the cause and effect of phenomena and aim to remain independent from the research so it can be quantified (Lapan et al., 2012, p.7). Research is often aggregated to a set of numbers which can be analysed. This method of research may be used to justify an existing concept making it similar to scientific research.

13.1 Research methods conclusion

Lapan et al. state "qualitative research, as contrasted with quantitative studies, places more emphasis on the study of phenomena from the perspective of insiders." (Lapan et al., 2012). This project is particularly interested in the needs of a specific group, IT system administrators, who would be the "insiders" in this context. The bulk of this project will be researched using qualitative methods in order to develop an in-depth response to the research questions.

Initially the research question to be answered is "do IT administrators feel there is a benefit to using mobile device apps to monitor the health of their network?". This will be investigated using a quantitative approach through the use of a survey (see section 15) in order to determine the opinion of IT administrators and feed into the next research question.

"What features would be required for a mobile app, should it be deemed one was desirable?" forms this project's second research question. Consideration will be given to multiple areas including security and implementation. To obtain more detailed views from those working in the industry some professionals would be interviewed. Qualitative research is in use for this section of the project.

14 Workplace observations

It has been observed by this author that some checks are left disabled when correct expected values are not available to the configuring administrator. Of note are server performance

checks which this author has seen left disabled. Performance checks are important for flagging a range of issues and when left configured on one server were the only indication the server had been compromised (the attacker was using the server to mine Bitcoins causing processor utilisation to be excessive).

Similarly it has been noted checks are not always adjusted following configuration changes. GFI Max Remote Management includes a "hacker check" designed to check the number of audit failures in a server's security log. If the alert value is 1,000 prior to installing a tool to limit brute force attacks the alert value should be decreased to avoid the protection mechanism masking a successful attacker (since fewer attacks are possible it is necessary to investigate attempts earlier than before).

15 Survey research

The survey was circulated to the author's work colleagues, other professional associates and via the BCS (the chartered institute for IT) private LinkedIn group. Questions can be found in Appendix A. Survey responses were collected anonymously using the SurveyMonkey platform with the collection of IP addresses disabled (these could be used to trace responders as a number of public IP addresses are known). There were no paper copies of the IT system circulated.

It should be noted that a high representation of GFI Max Remote Management is likely due to survey responses from this author's employer where this tool is the chosen solution. This presents a form of bias in the results.

15.1 Reasoning behind questions

A reasonably broad spread of questions was included to provide an overview of IT monitoring systems used by IT Professionals. Respondents were asked to indicate their job role as this may influence their exposure to tools. A junior technician may not be provided access to the monitoring system or its configuration.

It was important to determine if support tickets were automatically generated as this could be a desired feature in a solution. False positives are also queried, in part as a follow on to research by Tang et al. (2013), but also because it's possible current systems suffer an issue of raising invalid support tickets.

The survey ends with a question about notification methods with a view to determine what suits professionals best. The result to this question should help direct this paper's research.

15.2 Survey response analysis

Although 43 individuals responded to the survey not all of them responded to every question (no questions were mandatory). Importantly the majority of questions were answered by the majority of respondents. Most individuals identified themselves as being "in-house IT support engineers/technicians" or as an "IT support engineer providing services to 3rd parties" (71% combined) showing the majority of those answering the survey didn't have overall management responsibility for the systems they worked with. This makes sense given most organisations only have 1 IT manager but numerous supporting technicians. In addition to IT support personnel the survey saw responses from system architects, database architects and IT consultants.

Please indicate network monitoring systems with which you have experience

More people had used Nagios (54.55%) than other systems although it did not have majority. Solarwinds Orion came second with 45.45% of respondents having used this. Interestingly, GFI Max Remote Management, which prides itself on being quick to setup, had only been encountered by a third of those responding (there were 33 responses to this question in total). Figure 10 shows full details of responses to this question.

GFI Max Remote Management is primarily sold to organisations which provide IT services to 3rd parties. 33.33% of respondents had used this tool while 34.21% indicated they worked in this area of the industry. Results for GFI Max Remote Management may have been enlarged due to colleagues of the author responding to the survey; there was no limitation placed on the number of responses per organisation. GFI's product is used extensively at the author's workplace.

Nagios' result is interesting as this product is often criticised for its complicated configuration (Issariyapat et al. (2012)Wu et al. (2013)), however, as a free solution it is possible more organisations are prepared to implement it (trading product cost for the cost of man-hours). There is a commercial branch of Nagios, Nagios XI, and this survey didn't delineate between the two. Icinga appears to be less used than expected despite being a fork of Nagios suggesting the "parent" product still had a strong following.

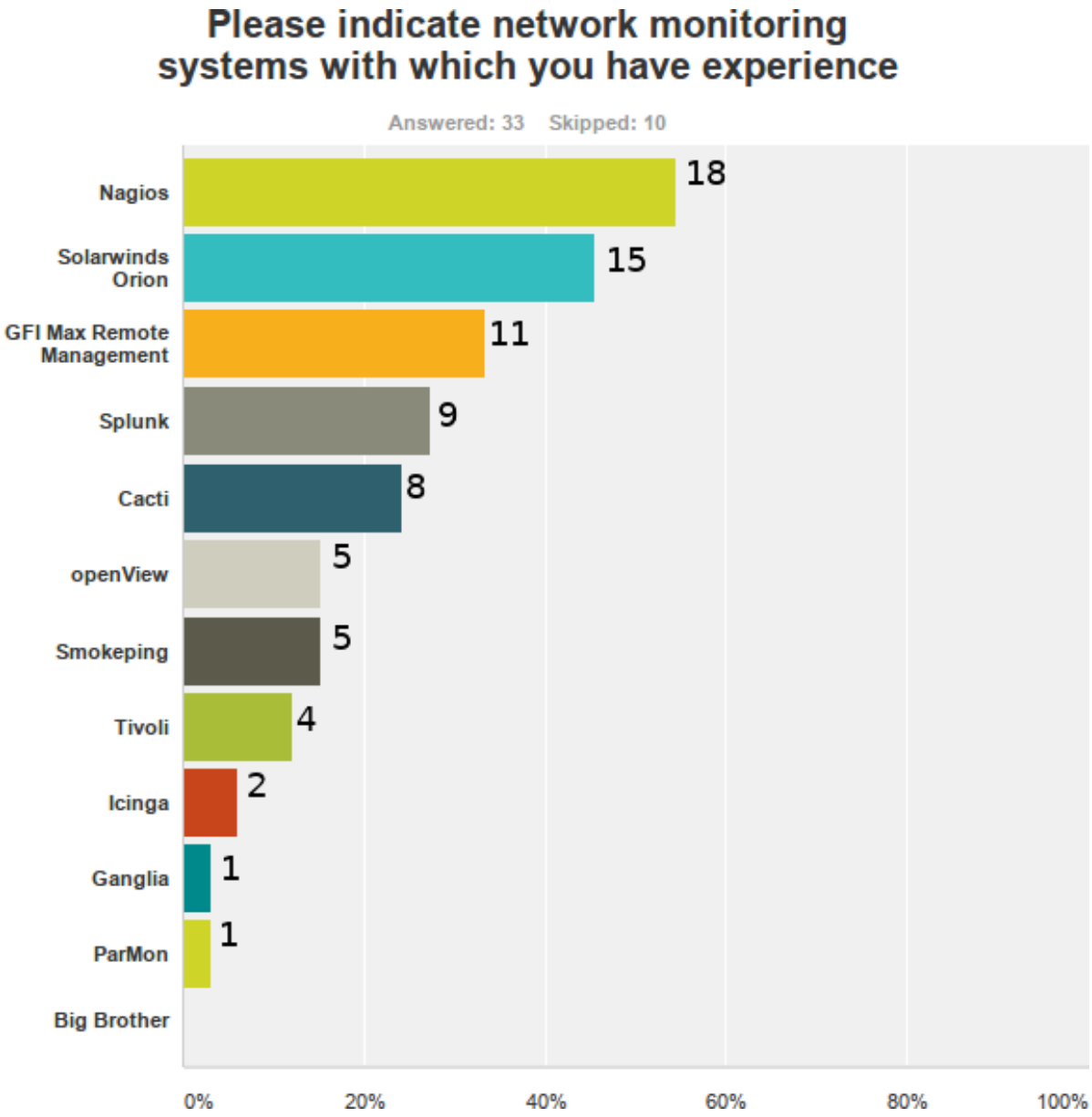


Figure 10: Monitoring systems by users familiar with them

Both Solarwinds Orion and GFI Max Remote Management are licensed per monitored node and offer fairly easy configuration (this author has experience of both) so it is not surprising these tools are found in the top 3.

In addition to the monitoring systems suggested by the survey it should be noted some users had also used Spiceworks, ServerDensity, ArcSight, RM Event Master, Zabbix and internally developed solutions.

Costs to the business

Costs to the business appeared to vary, largely based on the business type. From the responses it is clear a number of industries were represented including local government, educational establishments and businesses. Organisations where projects are worked on locally using locally installed tools indicated the cost of 1 hour's downtime would be minimal as employees would continue working on their local copies. The same respondent indicated that if they suffered an entire IT outage for a day, citing the building burning down as an example, their company would lose at £30,000 per day (£1,000 per employee per day).

Other answers given by someone providing support services to 3rd parties indicated an hour's downtime might cost their company tens of thousands of pounds whereas a day's outage would have both financial and reputation costs. Those working in education suggested although the financial costs would be low there would be a negative impact on the organisation as a whole and management may perceive this as a greater problem.

At the upper end of the scale, IT outages were predicted to cost in the order of millions of pounds.

When the monitoring system detects a problem, is a support ticket automatically raised in your helpdesk/management system?

72% of those responding to this question indicated that a ticket was automatically logged in their systems for further investigation. Of this 72% the majority of systems logged the fault immediately. Figure 11 shows the number of automatically raised tickets estimated to be false positives. While the percentages of false positives generally seemed to be below 50% it should be noted this particular statistic doesn't explicitly consider scale. Whereas 10% of 100 tickets being a false positive may not be considered a problem, 10% of 1,000,000 (100,000) would involve much more work. Ultimately this question confirmed that false positives are still an issue.

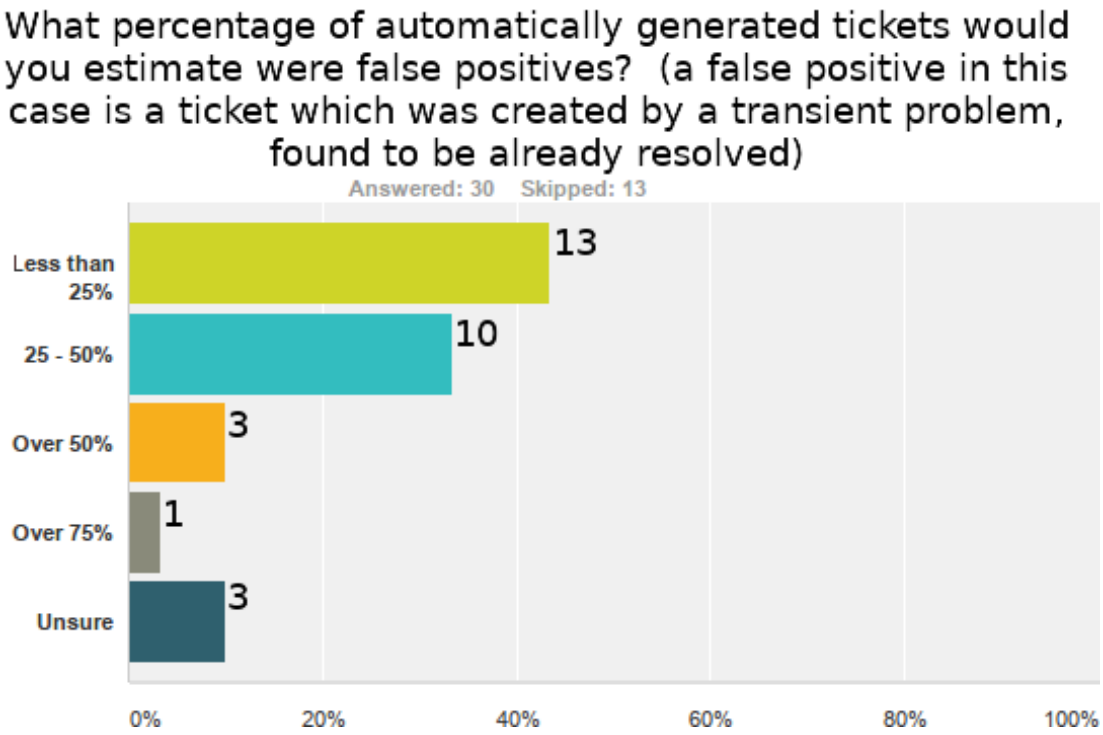


Figure 11: Percentage of automatically generated tickets estimated to be false positives

Preferred notification methods

Perhaps of most relevance to this project was the survey's final question about response methods. Overall email was found to be the preferred method of notification followed by SMS text message (see notification preferences, right). This was also found to be the case in interviews with IT professionals (see section 16). One individual commented the notification method would ultimately depend on the type of problem being reported - a paper jam would not require as urgent a response as a disk failure.

1. Email
2. SMS text message
3. Notification via smart-phone app
4. Automated phone call
5. Notification via desktop software
6. Pager message
7. Wait for users to report the issue

Notification methods in ascending order of preference

Arguably the first 4 options could all be routed to a mobile device / smart-phone and it would be interesting to see responses in the event that a smart-phone app was able to interact with the monitored environment.

15.3 Conclusion

Responses to the survey demonstrated there is a reliance on IT systems by businesses and there are costs, both financial and to reputation, associated with downtime of systems. Although costs vary dependent on industry there was always a cost. A number of monitoring systems were used although no monitoring system held a majority among those queried. In hindsight, the question relating to monitoring systems should have been reworded to determine which systems were actually used versus simply having been trialled.

The survey did not ask what features IT professionals would like available in future products. It is possible to determine what features might be used based on the products IT professionals stated they had experience of although there is no guarantee each feature of each product is used, most likely organisations use a subset of features.

For some products, for example office productivity suites, it is clear commercial offerings have the greater market share. This survey has shown this isn't necessarily the case in the network monitoring domain. Further study would be required to determine if this is due to companies placing low budget priority on tools of this nature, however, this is outside the scope of this project.

"Notification via smart-phone app" was shown to be the third preference for administrators receiving information on system health showing that a mobile app is desired but would not be the primary resource for administrators as solutions stood today.

16 Interview responses

As part this project's primary research two IT professionals were interviewed to obtain more detail on their thoughts about monitoring systems. It was noted their answers were similar indicating there is a need for these systems.

Monitoring systems clearly avoid excess costs to companies by helping to prevent downtime. One interviewee, an administrator for a telecomms provider, explained that for each minute of downtime costs vary based on the type of phone call that wasn't possible. For example, downtime varies from a "few pence to in excess of £7 lost revenue every minute the system in unavailable" (Cassidy, 2014). Cost of downtime can be expressed thus:

$$Cost = callincome_{perminute} \times calls \times downime_{minutes}$$

There have been a maximum of 15 concurrent phone calls through that system. If each phone call was to a premium rate number at £7 per minute, an outage of 5 minutes would cost the company £525 in lost revenue, before loss of reputation and any fines are taken into consideration. Another IT professional highlighted the cost caused by a loss of reputation for a 3 day outage at their business and estimated this may cost the business £50,000 (Bramley, 2014).

In both cases it was determined the cost of the monitoring system was offset significantly by the savings it provided. Of the respondents, one used Icinga, an open source solution, for which the cost was only the hardware and electricity required to run the system. Another used GFI Max Remote Management which has a license cost per device. The decision to use GFI was partly a result of needing to know the software as it was sold to the company's customers as a hosted service. There were also marketing benefits to saying "this is what we use" (Bramley, 2014). Usability was mentioned by respondents along with the problem of Icinga's text configuration files. GFI has a much more user-friendly interface which contributed to its purchasing decision.

It was noted interviewees relied primarily on email alerts when a problem was detected although SNMP alerts were also used. In the event an outage prevents email from being sent, one organisation has configured their system to send SMS text messages. Although overview dashboards exist these weren't the primary method for monitoring problems but are used to obtain further information. It was felt a smart-phone app (or similar) would be beneficial for showing urgent or critical alerts (accompanied by the device vibrating or making a sound)(Bramley, 2014)(Knibbs, 2014). Another respondent suggested he was happy with email and SMS unless an app were produced which provides additional controls, for example triggering a reboot of a server, as this "would definitely help" (Cassidy, 2014).

16.1 Conclusion

Although smart-phones and tablets are increasingly becoming part of our lives, merely receiving alerts to them is not sufficient to cause their adoption for IT monitoring notifications over more traditional methods. Adding additional capabilities to an app so it doesn't only report events will assist in adoption of this medium.

It was also apparent that more than just Windows Server systems are in use so monitoring systems need to provide solutions for multiple platforms.

17 Research conclusions

"Do IT administrators feel there is a benefit to using mobile device apps to monitor the health of their network?"

Survey and interview based research has shown there is a definite cost to the business in terms of system downtime, verifying this widely held belief. Being able to restore a system quickly, with the assistance of a monitoring system, clearly is in the business' interest and seemingly that of IT professionals.

Mobile device app notifications were rated the third notification preference according to the survey (see section 15.2) although comments in interviews and conversation indicated an app alone was not necessarily enough as alerts could already be received by email. The type of problem should be considered by the app with an audible alert perhaps being generated for high priority problems but not those of a non-critical nature (Knibbs, 2014). Importantly the app needs to offer more than just another "dashboard view" with options to rectify the problem being accessible from the app itself (Cassidy, 2014).

Overall it would appear administrators did see a benefit in using mobile device apps. Given this notification method wasn't the top preference it was clear current offerings didn't provide administrators everything they required to prompt a shift in preference.

Part IV

Solution

As highlighted by de Chaves, S.A. et al.(2013), and this author's research, there is no single solution to the problem of monitoring cloud computing. Differences in how services are provisioned vary between suppliers, and services can be provided on demand, so tools must support dynamic configuration. There are also security implications in monitoring cloud services as firewalls must have additional rules created to permit monitoring probes. Fortunately, firewalls can be configured to only permit probes from specific sources.

An additional challenge posed by the cloud is particularly relevant when a company's cloud is provided by a vendor, rather than an in-house, private, cloud. Where a company doesn't directly control the hypervisor which is running its VMs it is necessary to be able to remotely issue scripted commands from the RESO Control Server. The reason for this is simple: if an administrator has to logon to the cloud provider's web management console it would be very difficult, if not impossible, to use RESO to take corrective action. Fortunately some providers have addressed with issue. Microsoft Azure offers the "Microsoft Azure Powershell Cmdlets" which include commands like `Restart-AzureVM` and `Start-AzureVM` (Washam, 2012). Support from other vendors varies.

It is also necessary to secure communications between the monitoring system and the administrator's device. If communications can be intercepted it would be possible for an attacker to determine if their attack was working and adjust their actions accordingly. Similarly, if remote control was provided by the remote application it would be necessary to prevent unauthorised access to this system.

Surveys conducted as part of this project showed administrators generally relied on email for alerts and a mobile app was not highly sought after just for the purposes of showing alerts. As noted during interviews, if the app permitted the administrator to take some form of corrective action it would be considerably more useful (Cassidy, 2014) and as such this project will look to produce a proof of concept solution to this end.

18 Security considerations

As RESO gives the user a portal into a remote network there are a number of issues to consider. This section looks to address those security concerns, explaining how the system protects itself.

18.1 The device

Security of the device should be considered as through the device a user has the ability to remotely control a company's infrastructure. Although RESO doesn't enforce it, devices should be encrypted, a facility found in Android for some time. Devices should also be protected with a password or PIN to prevent misuse although it is not reliably possible for a third party app to determine if this is present. As a result RESO does not check because the result cannot be trusted.

Mobile Device Management solutions such as Sophos MDM and Mobile Iron have the ability to control device features from a central console, being able to prevent the use of bluetooth, storage cards etc. RESO doesn't implement these control features, however, it would work on a device which had been secured in this fashion.

Overall, device security is the responsibility of the IT Administrator for the company.

18.2 The user

While the device may be protected by a password, PIN or facial recognition there is still a possibility the user might unlock the device and hand it to another person. This would leave the person holding the device in control of the network. For this reason it is necessary to also authenticate the user before any instructions are sent from the device. A simple prompt for the password when confirming the user's desired action has been included to protect against an unauthorised user using an authorised device (figure 12).

It should be noted the password should be approved/rejected at the control server's end, rather than on the handset, in order to ensure the approval cannot be changed by the end user.

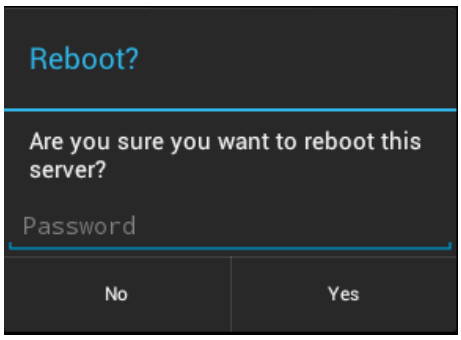


Figure 12: RESO: Screenshot of a confirmation prompt including the password field

18.3 Cloning the device and disgruntled employees

Without using particularly obscure tools it would be possible to move RESO from a trusted device, such as the device assigned an administrator by his employer, to an untrusted one (their personal device perhaps). In the event the app is copied to another device the company immediately loses control of their deployment, their attack surface is widened and it's not possible to enforce device policies through the third party solutions discussed above.

When the device is assumed to be trusted there is nothing preventing an unauthorised device being used to control the system. In the event an employee "goes rogue" with their assigned, or unauthorised, device there would be no mechanism to prevent their actions. To combat this RESO uses a device identifier which is unique to each device. A technical explanation of this identifier can be found in section 20.2.

Using an identifier allows devices to be assumed *untrusted* placing control of approved devices back with the system administrator. To trust a device the system administrator must add the device's identifier to the RESO Control Server's white-list. This method is not entirely fool proof as an attacker with the correct encryption key could take all the relevant elements of the device ID, encrypt it and transmit it to the control server. Further hardening of the system would be possible through digitally signing the message (see section 18.5).

In the event an employee "goes rogue" the device identifier can be removed from the white-list rendering the device useless from a remote control perspective. The employee's password can also be removed from the system, preventing them from using another device.

18.4 Remote control - arbitrary commands

A key factor is the security of the remote control mechanism as it is clear an organisation would not want unauthorised users accessing the remote control mechanism (doing so would present the potential to remotely attack the company, perhaps performing a Denial of Service (DoS) attack. When dealing with user provided input it's important to consider the possibility the user is not friendly and will try to subvert the system for malicious means. For example, consider the insecure code in listing 1:

```
1 public static void processCommandInsecure(String command) {
2         ProcessBuilder systemCommand = new ProcessBuilder("bash", "-c",
              command);
3         try {
4                 Process runSystemCommand = systemCommand.start();
5         } catch (IOException e) {
6                 //Do nothing
7         }
8 }
```

Listing 1: Java: example insecure code to execute commands

This code essentially says "run BASH (a standard Linux shell) and instruct it to run `command`" and is flawed because it doesn't allow for the dangerous nature of user-supplied input. A "good" user would provide safe commands to be executed, for example `ping 8.8.8.8`, however, a malicious user can use this access vector to run arbitrary commands (`halt` to shut down the Linux system, `rm -rf /` in an attempt to erase files on the system etc.). There is no sanitisation of input making the code inherently dangerous - user-input *must not* be trusted.

In the control server application this threat is significantly reduced by only permitting certain commands to be executed. Rather than producing a white-list of shell commands, this is further obfuscated (not itself a security mechanism) by only accepting short codes. This is further explained in section 19.3 but the code is included here:

```
1 public static void processCommand(String command) {
2         ProcessBuilder systemCommand = new ProcessBuilder("");
3         if (command.equals("exit")) {
4                 System.exit(0);
5         } else if (command.equals("rebootHH")) {
6                 systemCommand = new ProcessBuilder("bash", "-c", "
                      VBoxManage -q controlvm DNS01 reset");
7         } else if (command.equals("startHH")) {
8                 systemCommand = new ProcessBuilder("bash", "-c", "
                      VBoxManage -q startvm DNS01");
9         }
10
11        try {
```

```
12                  Process runSystemCommand = systemCommand.start();
13          } catch (IOException e) {
14                  //Do nothing
15          }
16  }
```

Listing 2: Java: "safely" processing user input to remotely execute commands

Should console input be required, allowing an administrator to specify specific commands, then an alternative product should be used. SSH is a good example of a tested product that can be used for this purpose.

18.5 Distributing command details

At present the RESO app and Control Server have the commands that can be executed hard coded, i.e. there is no way to expand what can be done remotely without recompiling the source. Long term this is not a viable solution and certainly does not scale so it is necessary to develop a mechanism for distributing a set of command short codes per server. For the purposes of this project such a mechanism has not been implemented, however, it is discussed here.

How the remote command is securely passed to the control server is discussed in section 19.2.2 but should an attacker be able to intercept and modify the distribution of commands sent from the control server to the app there would be security concerns. For example, if an attacker changed every command to have the same or an invalid short code, or perhaps removed the short codes altogether, the app would immediately be compromised. System administrators would unknowingly be executing the wrong command against a server without any reason to believe they were potentially harming the system. Consider this scenario:

1. Mobile device connects to a compromised wireless access point and contacts the RESO Control Server
2. RESO Control Server sends mobile device the latest list of servers, their status and commands
3. Attacker intercepts the transmission and adjusts all short codes, not necessarily knowing what they mean, to a new value
4. Mobile device receives and processes the compromised list

5. System administrator attempts to respond to an alert and remotely executes a different command to the one he's expecting

Protecting against this attack, known as a modification attack, would require RESO and the control server to digitally sign their transmissions making it immediately obvious if the transmission has been tampered with. Signing requires support for asymmetric encryption as a public/private key pair is required. Asymmetric encryption is explained in figure 9 and signing in figure 13.

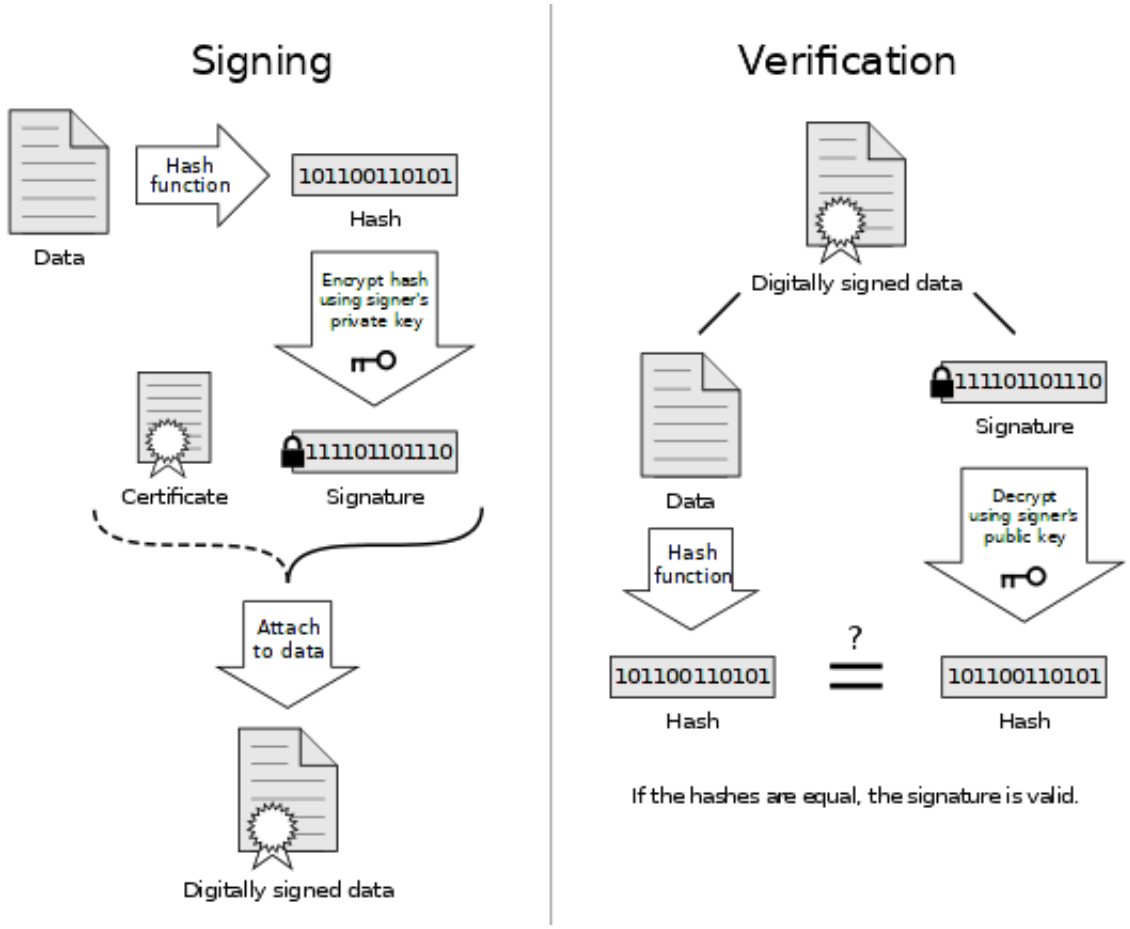


Figure 13: Signing messages with digital signatures (ACDX, 2008)

The altered communication would not have the correct hash and the app would discard the information. Alternatively the aggressor could elect to replace the message entirely and sign the message with his own *private key*. In this event the app would not have the aggressor's *public key* resulting in the information being discarded as there would be no method to verify the transmission.

19 System design

Given administrators are not always physically connected to a network it is necessary for them to be capable of performing their tasks when not constrained by network cables. The prolific coverage of 3G capable mobile networks, and potentially WiFi hotspots, mean an administrator is never truly cut off from their systems unless they lack a device with which to connect. Increasingly there's a move to working from home so it is necessary to communicate with the corporate LAN from an external location.

As a result, it was determined that an application for a mobile device, commonly referred to as an *app*, would be produced as part of this project. The app will look to harness the flexibility of the mobile device and provide administrators with a useful tool for not only viewing their network's health but also to initiate remedial action (for example rebooting a server) where possible. It is expected remedial action would be easier on a virtual system as the host would still be available to execute commands. As cloud computing heavily uses virtual machines such a design is useful in the cloud computing space.

The overall operation of the app can be seen in figure 14 while the control server's operation is shown in figure 16. Network deployment scenarios can be found in section 21.

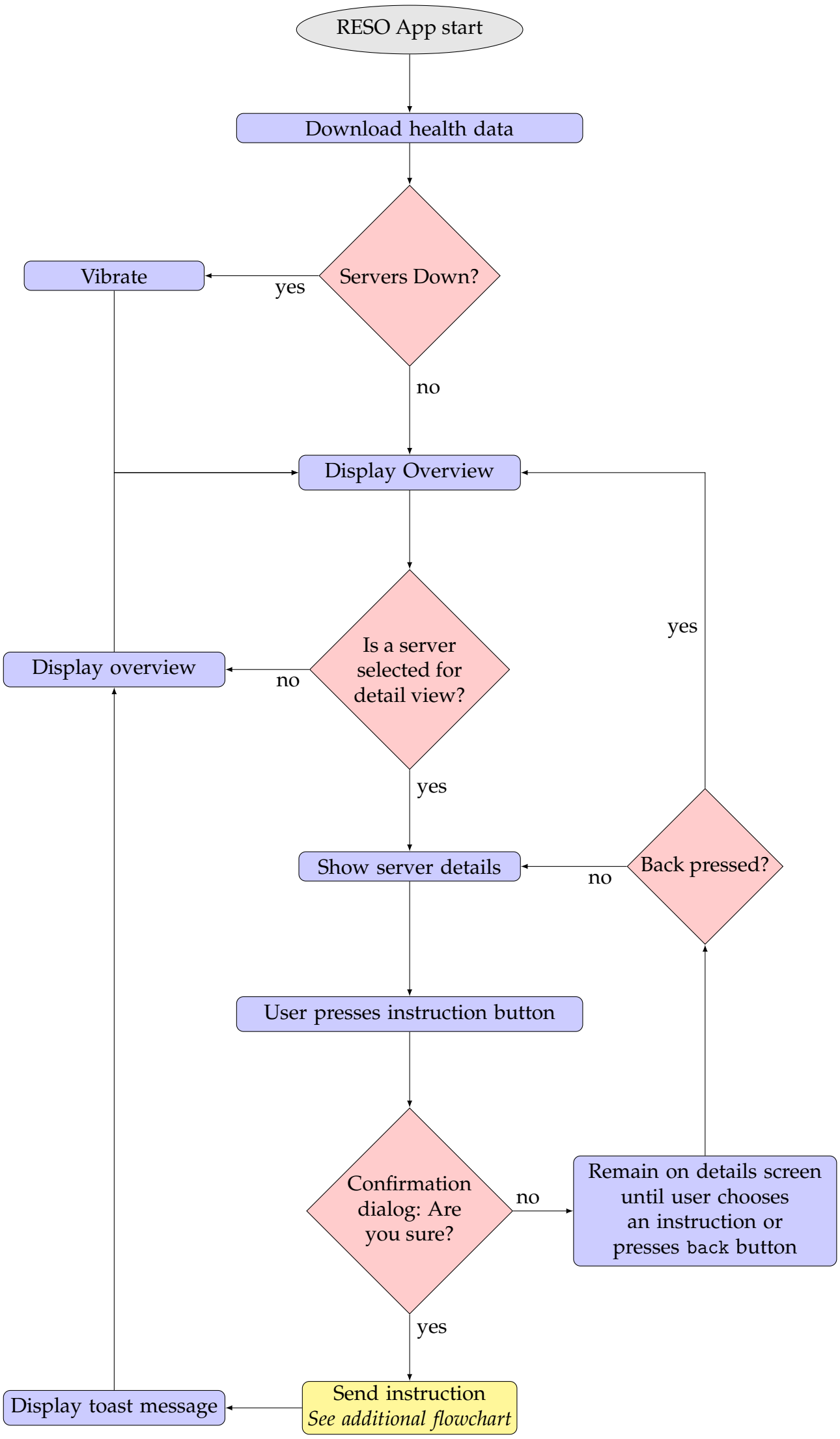


Figure 14: RESO App process diagram

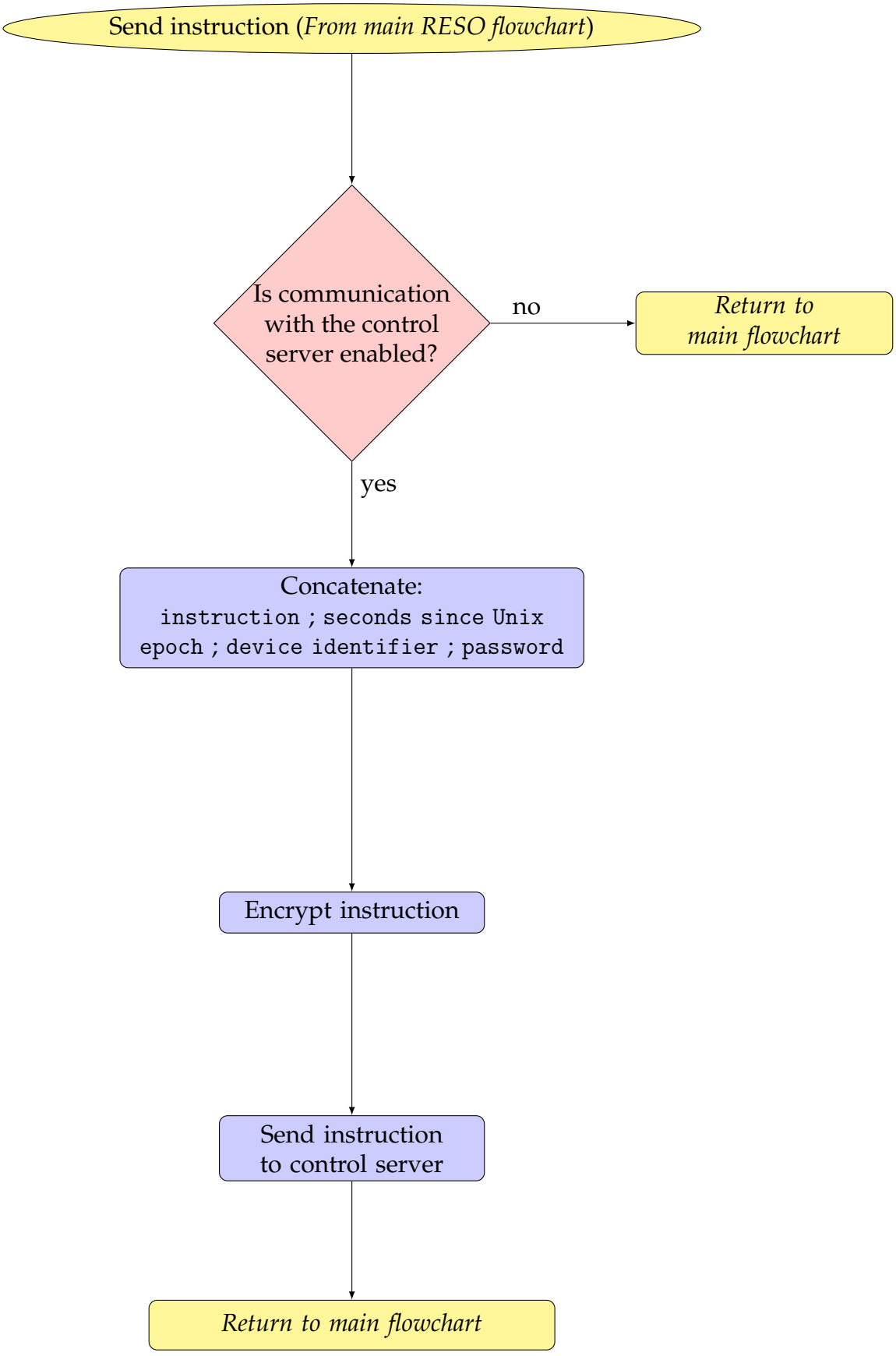


Figure 15: RESO App: *send reboot instruction* process diagram

19.1 Mobile platforms

The three most used mobile device OS (in no particular order) are Google's Android, Microsoft's Windows (Phone and RT) and Apple's iOS.

While Apple's offering arguably gained popularity in the consumer space, and initially in businesses, it is viewed by some in the IT industry as a primarily home-user grade system. When business finances are considered the premium paid for an Apple device is also noticeable making Apple devices controversial choices for businesses in the current financial environment. To Apple's credit as iOS runs on restricted hardware it should be easier to produce a successful app for these devices.

Microsoft's mobile platform has the weakest adoption and has been criticised by some for having few apps in the Windows Marketplace. In this author's experience there are few Windows Phone handsets in use by IT professionals making it a poor choice for this project.

Android is available on multiple devices with varying specifications. There are 2 different ways to produce an app for Android. One option is to write in native C, using the Android Native Development Kit (NDK), although this requires the app be compiled for the relevant CPU. More commonly Android apps are written in a combination of Java and XML using the Android Software Development Kit (SDK).

BlackBerry is also worth mentioning because although it once was the de-facto standard for business mobile communication due to the encryption of its devices it has now fallen behind. The company has posted numerous consecutive quarterly losses and is moving to focus on its infrastructure services. This project has chosen to ignore BlackBerry due to its uncertain future and decreasing adoption.

For this project the target platform will be Google's Android with the app written in Java and XML. Not only does this choice offer a relatively fast turn around and steady learning curve it's also the route recommended by Google who state, relating to native code:

> Before downloading the NDK, you should understand that the **NDK will not benefit most apps**. As a developer, you need to balance its benefits against its drawbacks. Notably, using native code on Android generally does not result in a noticeable performance improvement, but it always increases your app complexity. In general, you should only use the NDK if it is essential to your app-never because you simply prefer to program in C/C++.
> *(Google, 2014)*

Android is also open source for commercial use making it a low cost platform to develop for as there are no license fees to pay.

19.2 Requirements

Overall, the system must:

- Work on multiple hardware platforms
- Allow secure communications between the mobile device and monitoring server
- Permit an administrator to securely instigate corrective action
- Use *push notifications* to reduce the impact on battery life
- Be extensible to support multiple systems

Production of a monitoring system is outside the scope of this project, however, it would be possible to extend Nagios / Icinga to process notifications using the new system. This project focuses on producing a mobile app to securely communicate with a monitoring system, interacting with it to remotely trigger fixes.

19.2.1 Hardware support

Unlike Apple's iOS which only operates on a limited set of hardware, Android operates on a significantly wider set of devices. Hardware specifications vary widely between manufacturers and devices so there is a challenge in ensuring the mobile app operates the same across a multitude of devices. Fortunately for developers much of this is already done through use of the Android SDK. Key visual differences will occur where screen resolutions and themes vary.

It is a requirement of this project that the app is usable on a variety of Android devices, regardless of resolution.

19.2.2 Secure communications

In order for two people to communicate it is often necessary to talk in rooms where others are present. As such it would be unwise to discuss confidential matters plainly, perhaps speaking in a made up language. When two computers need to communicate the same is true - doing so will often be overheard. For example, sending data across the Internet does not guarantee the traffic won't be intercepted and analysed. To be secure it's important the medium itself (e.g. the Internet) doesn't need to be secure; this reduces costs and increases the ability to communicate. Moreover, just because an environment is secure today it is not valid to assume it will always be so; a wire tap could be installed at any point (or in the case of switched networks a mirror port could be created to receive a second copy of the traffic).

Combating this problem requires the information to be made unreadable by third parties. This project will encrypt the data using *symmetric encryption* to ensure the data can only be read by the correct third party (i.e. those with the encryption key). Symmetric encryption was chosen as it is less processor intensive, thus less draining on the mobile device's battery. As mentioned, it is critically important to protect the encryption key, ensuring it isn't made available to non-legitimate third parties.

Despite the data exchange being encrypted it would be possible to capture the data and replay it. While the interceptor could not understand the data they were retransmitting the server would repeat any processes on the data, potentially leading to a denial of service. If the replayed instruction was to reboot a server the server would restart again. To combat this more than just the instruction will be transmitted:

- **Timestamp**
  The control server will compare the timestamp of received instructions with that in the encrypted data. If there is more than 5 seconds difference the instruction will be ignored. This reduces the duration of a replay attack to 5 seconds.
- **Instruction**
  Rather than transmitting the full instruction (e.g. `service apache2 restart`) an instruction ID will be sent, reducing bandwidth requirements while also preventing data leakage in the event the packet is decrypted.
- **Device identifier**
  This enables the control server to determine if a trusted device is being used. This is discussed more in section 20.2
- **User password**
  The user is prompted to provide their password prior to any actions being taken, this is added to the message sent to the control server.

Use of a timestamp to help secure communications is not a new concept. Timestamps are used with Kerberos, an authentication mechanism used in a number of operating systems including Microsoft Windows (Russinovich and Solomon, D.A., 2009, p.517). Kerberos is defined in RFC 4120 (Neuman et al., 2005) and MIT advise that clocks should be synchronised within 5 minutes of each other to "prevent intruders from resetting their system clocks in order to continue to use expired tickets" (MIT, 2007). RESO takes a similar approach in order to prevent a replay attack (its equivalent of using an expired ticket).

In the future it would be possible to convert the system to use asymmetric encryption.

19.2.3 Instigate corrective action

For the purposes of this proof of concept the only corrective actions offered to the administrator will be to start or reboot the server. If the solution were to be extended it would be necessary to offer additional options as it is not always desirable to reboot an entire server. As examples, additional corrective actions which could be added:

- Stop or start a service
- Open SSH session (for greater control)
- Open RDP session (for greater control)
- Allocate additional resources to a VM

19.2.4 Push notifications

With traditional computing, where a device is connected to a continuous power supply, it is not unusual for programs to poll on schedule. An example of this is an email client set to check for email every 5 minutes. The drawback to this is that a check is made whether any new data is available or not although this isn't usually a concern for a permanently powered device.

Mobile devices have a battery life which requires consideration. Regularly polling a remote host for information reduces battery life, especially as most of the time there will be no change. To combat this, push notifications are available which send instructions to the device on an as-needed basis. Push notifications are sent to the device when a change has occurred and can prompt the device or user to perform a sync to obtain the latest information.

For Android the most common method for a push notification is via GCM which will be used in this project.

19.2.5 Interoperability

Research (section 15) by this author has shown there are numerous monitoring systems in use so this system would need to provide a method for third party systems to exchange data. For this purpose XML is used and the third party system would need to output XML based on the specification shown in appendix C.

As RESO is not monitoring the system, only using the output of another, there is no need to send data back to the network monitoring system.

19.3 Control Server

The RESO control server will take instructions sent by the app and process them accordingly. This reduces the need to forward management ports for monitored hosts (e.g. virtualisation hypervisors) to the Internet (see deployment scenarios in section 21).

As mentioned, the control server is never sent the full command to be executed so the "short code" it is sent must match a command which is already known. This increases the security of the system by making it more difficult to run an arbitrary command (for example to begin an attack on a remote system or to create a local user). A process diagram for the control server is shown in figure 16.

For the purposes of this project a proof-of-concept control server was developed in Java which has one "short code" built in (to reboot a certain VM in the test lab). The XML health data file was distributed via the Apache HTTP web server, rather than by the control server. In the lab, this control server would need to have been published to the Internet as shown in figure 21.

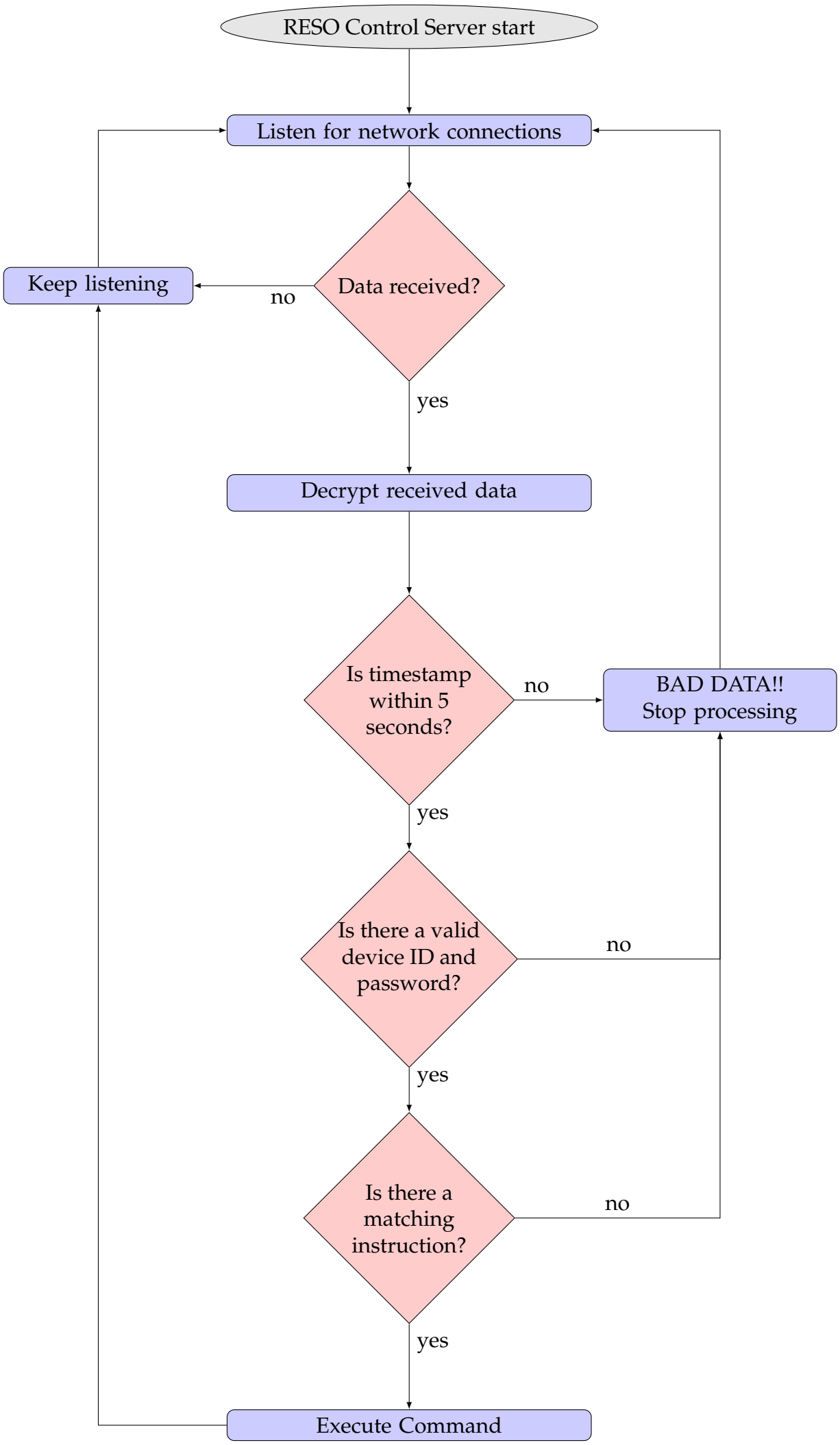


Figure 16: RESO Control Server process diagram

20 Solution development

As discussed in section 12, the IID methodology was employed for this project. Individual parts were developed and tested before integrating with the remainder of the project. In some cases this involved creating an entirely separate code base to ensure code worked as expected. Source code was managed using the Subversion Version Control System (VCS) allowing code revisions to be tracked and reverted if necessary.

In addition to developing parts separately there were a number of iterations of each layout. The initial design for the server list screen can be seen in figure 17 but it was deemed this didn't provide an intuitive interface: it was not apparent that touching the status "square" would open further information on the server. The final result can be seen in figure 24.

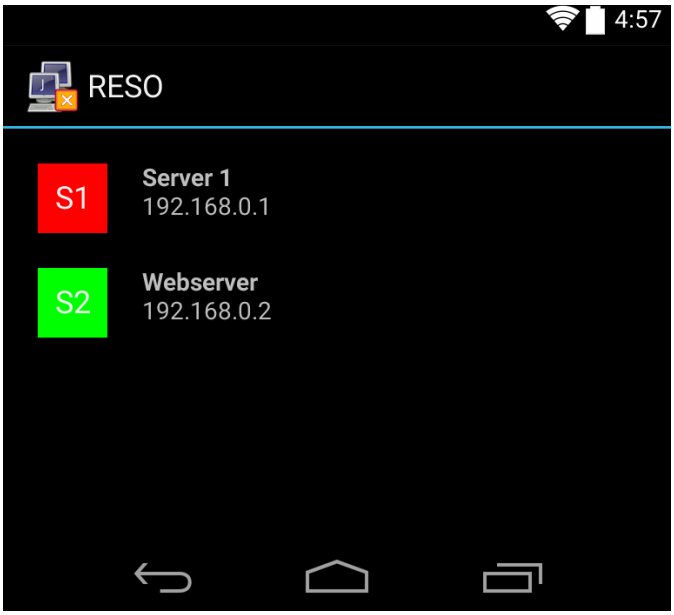


Figure 17: RESO: Initial design for the server list

Development was conducted using the JetBrains IntelliJ integrated development environment which provided syntax highlighting, integration with the Android SDK and, most importantly, integration with Android Virtual Devices. This tool allowed code to be tested in both an emulator, which was used for most of the development, and to be transferred to mobile devices for "real" testing. More details on the lab used for development can be found in section 22.

20.1 Generating the UI

Each "screen" in Android is known as an *activity* and must be defined in `AndroidManifest.xml`. Activities comprise of a Java class and an XML layout file which provides what the user sees and more complex or dynamic layouts can be generated using Java rather than a specific XML file. Moving between activities is accomplished by tapping UI elements (for example a button) or using one of the Android built in "buttons" (sometimes physical, sometimes on-screen) such as "back". An activity's layout file can be defined within its Java class by overriding the `onCreate` method as shown in listing 3.

```
1 @Override
2 public void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.about);
5 }
```

Listing 3: Defining an activity's layout

Layout files must be named `filename.xml` and the name must be lower case.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                  android:layout_width="match_parent"
4                  android:layout_height="match_parent">
5      <TextView
6              android:id="@+id/labelIcon"
7              android:layout_width="wrap_content"
8              android:layout_height="wrap_content"
9              android:text="Application icon from \nhttp://openclipart.org/
                   detail/36691/tango-network-offline-by-warszawianka"
10             />
11     <TextView
12             android:id="@+id/labelAcknowledgments"
13             android:layout_width="wrap_content"
14             android:layout_height="wrap_content"
15             android:layout_centerHorizontal="true"
16             android:layout_below="@id/labelIcon"
17             android:layout_marginTop="10dp"
18             android:text="@string/labelAcknowledgments"
19             android:textSize="8pt"
20             />
21 </RelativeLayout>
```

Listing 4: XML to generate the About activity layout

Activity layouts can be constructed separately from program code making it convenient to debug without the complication of program logic. Alternatively, dynamic layouts can be

constructed using Java - this technique is used to display the server health overview activity by parsing the XML.

20.2 Device identifier

Device identifiers are used as a rudimentary way to authenticate the device - the device identifier is sent along with the instruction to the control server and unless the identifier is on the white list no further processing is done on the instruction. If the device identifier was solely specified by a human, all an attacker, perhaps a disgruntled employee, would need to do is obtain a colleague's identifier and set their device's identifier accordingly. To combat this the identifier consists of three parts:

1. **Device IMEI**
   If the mobile device is a phone on the GSM network it should have an International Mobile Equipment Identity (IMEI) which is unique to it. This cannot be changed by the user.
2. **Device hardware serial**
   The device's serial number is extracted if possible and used in the identifier
3. **User specified identifier**
   This is an optional component, particularly used for devices which don't have an IMEI or hardware serial (for example Android emulators used in testing and development or tablet computers with no mobile network connectivity)

Each of these strings is concatenated, if they exist, in the order above, to generate the device's identifier which should be unique (certainly if an IMEI is used it will be). This is a achieved through the code in listing 5.

As can be seen in line 12 of listing 5 a *shared preference* is used to store the user specified device identifier. This is set on the *advanced preferences* configuration page as shown in figure 18.

```
1  public String deviceIdentifier() {
2          String deviceId = "FAKE_ID";
3          TelephonyManager telephonyManager = (TelephonyManager)
               getSystemService(Context.TELEPHONY_SERVICE);
4          if (!telephonyManager.getDeviceId().isEmpty()) {
5              deviceId = telephonyManager.getDeviceId();
6          }
7
8          if (!Build.SERIAL.isEmpty()) {
9              deviceId = deviceId + Build.SERIAL;
10         }
11
12         SharedPreferences sharedPrefUserDeviceId = getSharedPreferences("
               deviceIdFromUser", 0);
13         if (!sharedPrefUserDeviceId.getString("deviceIdFromUser", "").
               isEmpty()) {
14             deviceId = deviceId + sharedPrefUserDeviceId.getString("
                   deviceIdFromUser", "");
15         }
16         return deviceId;
17     }
```

Listing 5: Java used to determine the device's identifier

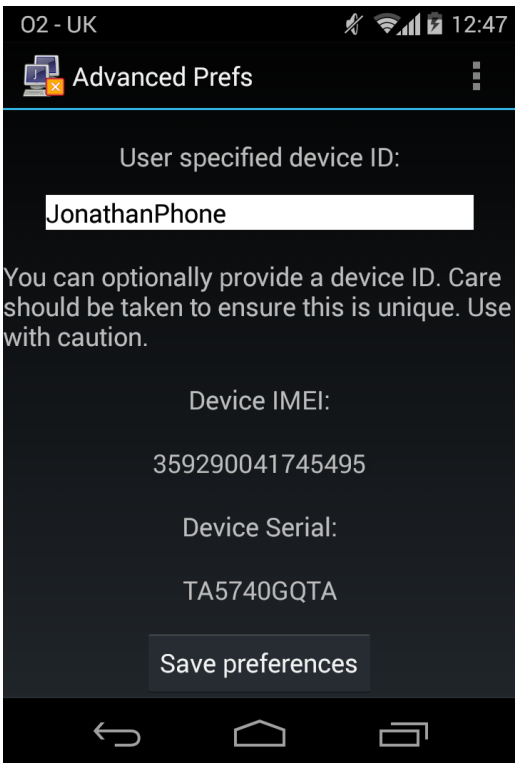


Figure 18: RESO: Advanced preferences

20.3 Adding encryption

Once communication between the app and control server had been established in plain text it was necessary to introduce encryption to meet security requirements. Initial research suggested it was necessary to use the SpongyCastle library, however, after further investigation it became apparent Java's built-in cryptography routines could be used instead. AES encryption was chosen as the encryption standard and was tested in a separate Java (not Android) project. It should be noted this code was taken from contributions by Fernandes (username *bricef*) on GitHub (Fernandes, 2014).

Once the encrypt/decrypt code was confirmed as working`AES.java` was then created within the Android project and an immediate problem was noted:

```
1 public static byte[] encrypt(String plainText, String encryptionKey)
    throws Exception {
2         Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding", "SunJCE");
3         SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF
              -8"), "AES");
4         cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(IV.
              getBytes("UTF-8")));
5         return cipher.doFinal(plainText.getBytes("UTF-8"));
6 }
```

Listing 6: Android: Calling a missing encryption provider

Whereas the code in listing 6 worked correctly in a Java project it was not compatible with the encryption setup within the Android implementation, instead throwing the following error:

```
java.security.NoSuchProviderException: Provider not available: SunJCE
```

It is not necessary to specify an encryption provider for the method `getInstance` so this was removed resulting in listing 7. This code is the same in both the Android app and the RESO Control Server running in the lab and works correctly.

```
1 public static byte[] encrypt(String plainText, String encryptionKey)
    throws Exception {
2         Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
3         SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF
              -8"), "AES");
4         cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(IV.
              getBytes("UTF-8")));
5         return cipher.doFinal(plainText.getBytes("UTF-8"));
6 }
```

Listing 7: Android: Fixed encryption code

20.4 GCM

The GCM component of RESO was initially built in a test app in order to test it without the interference of additional code. To use GCM it's necessary to stipulate additional permissions are required within the `AndroidManifest.xml` file as shown in listing 8. This listing shows a working `AndroidManifest.xml` excerpt, however, when first implemented, following an online tutorial, an error was introduced while re-factoring the code which broke the required Google permission. Permissions such as `com.google.android.c2dm.permission.RECEIVE` and `com.jonathanhaddock.gcmtest.permission.C2D_MESSAGE` must be represented exactly as written else the GCM system will not work. By developing this component outside of the main RESO project it was possible to debug the issue much easier as there was less code to review.

```
1 <uses-permission android:name="android.permission.INTERNET" />
2 <uses-permission android:name="android.permission.GET_ACCOUNTS" />
3 <uses-permission android:name="android.permission.WAKE_LOCK" />
4 <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"
      />
5
6 <permission android:name="com.jonathanhaddock.gcmtest.permission.
    C2D_MESSAGE" android:protectionLevel="signature" />
7 <uses-permission android:name="com.jonathanhaddock.gcmtest.permission.
    C2D_MESSAGE" />
```

Listing 8: Android: Permissions in AndroidManifest.xml

Android uses a permissions system that dictates what an app can do and these are defined in the `AndroidManifest.xml`. In listing 8 there are a numbers of permissions defined (note these are only an excerpt of what RESO requires):

- `INTERNET` - Allows access to the Internet, required so the device can receive the push notification
- `GET_ACCOUNTS` - to use GCM it's necessary to have a Google account, this permission allows the list of accounts to be queried
- `WAKE_LOCK` - Allows the app to prevent device CPU from sleeping, this is optional
- `com.google.android.c2dm.permission.RECEIVE` - permits the app to act as a broadcast receiver
- `com.jonathanhaddock.gcmtest.permission.C2D_MESSAGE` - prevents other apps registering to receive RESO's push notifications

Once the code was confirmed as working, and the handset could receive a push notification from the GCM servers, the code was moved into the main RESO project. Initial testing was conducted within an Android emulator but this was unsuccessful as the emulator didn't have a Google account added (and couldn't be configured to use one). Further testing was conducted on the developer's mobile phone instead.

How GCM works with RESO is shown in figure 19.

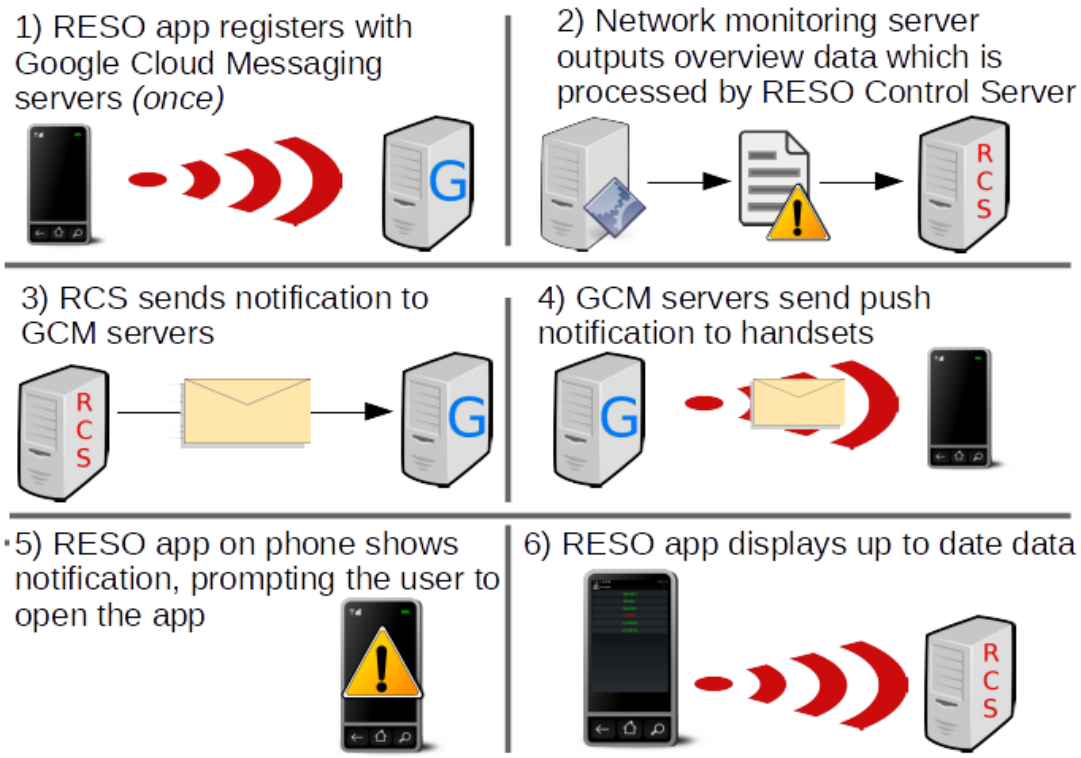


Figure 19: Google Cloud Messaging and RESO

21 Deployment and usage scenarios

RESO is designed to be usable by any system administrator but particularly those that are on the move or have to support multiple, geographically disparate, offices. If an administrator receives a phone call while en route explaining that a primary server had gone down, prior to RESO it would be necessary to find sufficient connectivity in order to link to the corporate network using a laptop. Instead it's now possible to simply tap a button from the mobile handset to boot a backup server reducing the company's down time significantly. Similarly an administrator may be pulled to 2 different incidents and clearly only 1 can be attended in person. RESO allows the administrator to begin corrective action ahead of their usual arrival time and potentially while still on the move (and hours away) or at a remote location.

Although the solution presented here is a powerful one there are occasions where it shouldn't be used. These cases are not necessarily a reflection on the solution itself but can come down to the basic principle of security - if you want something secure *don't* connect it to a network. For example, RESO should not be used to help monitor and track highly sensitive systems (computers controlling nuclear reactors) as this would immediately provide a route to the system from the Internet. Even if RESO was 100% secure, which no software can be, common sense would dictate that putting the nuclear reactor controls on the Internet would be a bad idea.

RESO is dependent on connectivity from either mobile or wireless networks so clearly cannot be used in areas where neither is available.

There are 2 options for deploying the RESO Control Server - either in a De-Militarised Zone (DMZ) or by forwarding the control server's ports directly to the Internet. As figure 20 shows, by placing the control server in a DMZ the firewall is able to tightly control the traffic from the control server. While this may appear an additional complication at first this design prevents the control server being successfully used as a pivot point to attack other areas of the network. Should the control server be compromised, the attacker gaining full control, it would be used to launch further attacks against the company's infrastructure. A large number of these attacks will fail as the firewall will not be configured to permit DMZ servers open access to the trusted LAN.

In smaller networks there is often not a DMZ so the server would be published directly to the Internet per figure 21. As the server is behind the firewall it can be used to attack other assets on the trusted LAN . With more networks moving to use virtualisation this potentially means an entire network can be brought down by targeting a handful of hypervisors.

Due to the high cost of server OS licenses companies may be hesitant to dedicate a server

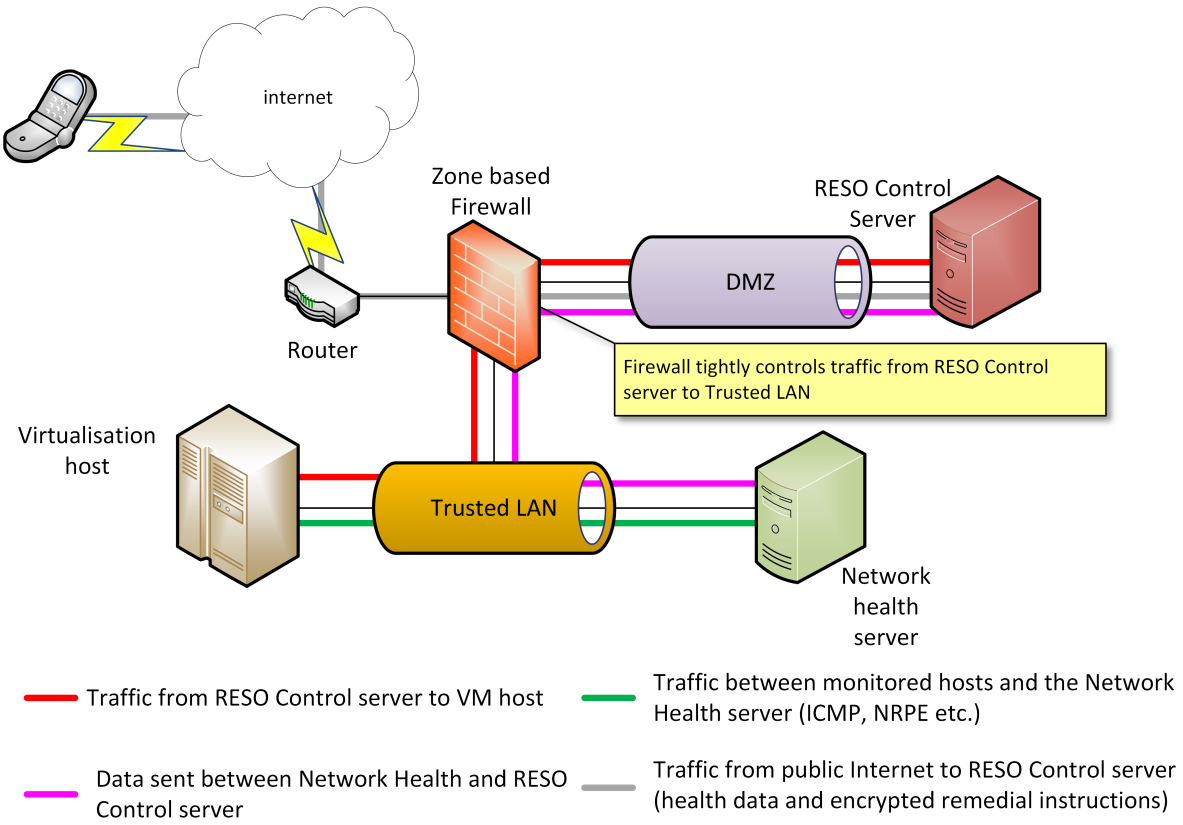


Figure 20: RESO Control Server deployed in a DMZ

to use as a RESO Control Server. In the "forwarded port" configuration the software on the server, not just the RESO Control application, is just as vulnerable as in the DMZ, but the server's placement makes the scenario significantly more dangerous.

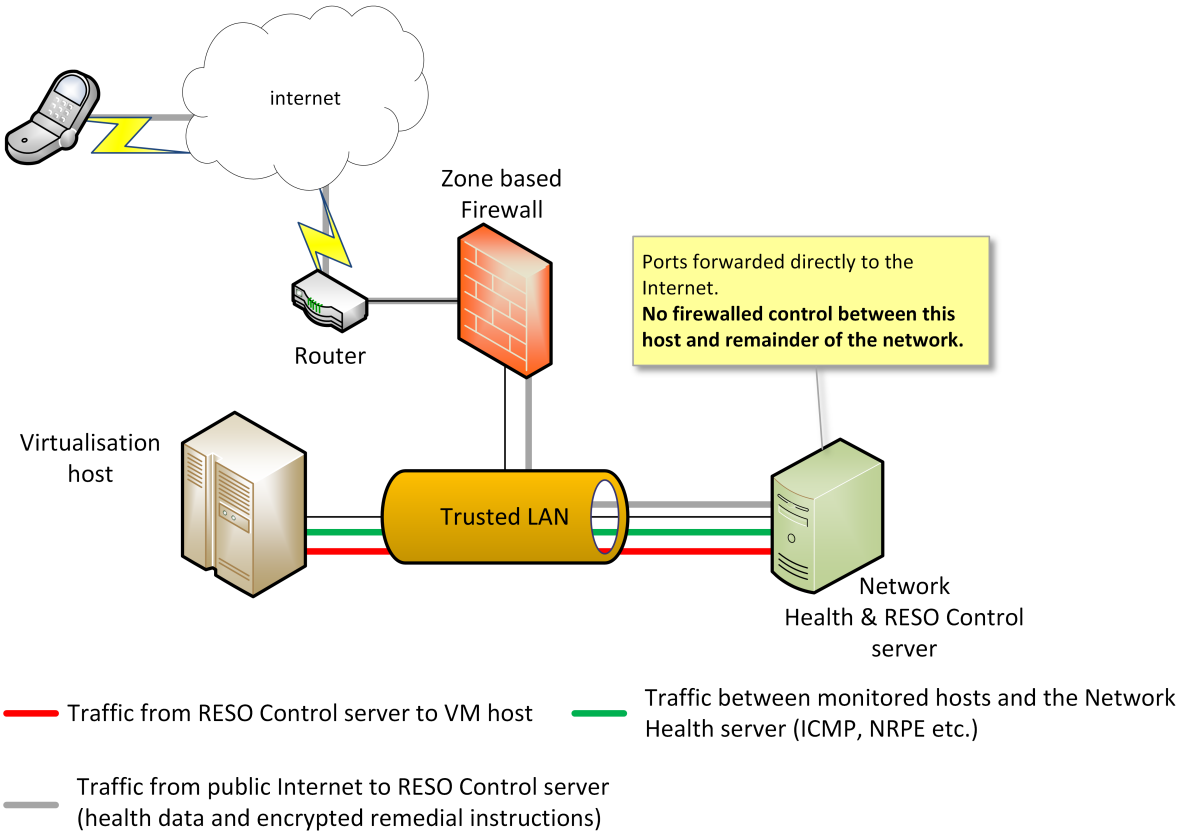


Figure 21: RESO Control Server deployed with ports forwarded directly to the Internet

Part V

Testing and validation

22 Testing

For testing, a lab was constructed comprising of the following parts:

- Network infrastructure (switch and wireless access point)
- Server running a hypervisor, the RESO control server and an emulated Android handset
- Handset running an "approved" RESO instance
- Handset running an unauthorised RESO instance

The XML file, parsed by the RESO app to show server states, was hosted on an external web server although this is not shown in figure 22 which shows the lab network.

For these tests the RESO control server was never exposed to the Internet, however, as a simple port forwarding would permit access from 3G or 4G networks there should be no difference accessing the control server over a wireless LAN to accessing it via a mobile provider's network.

Testing was largely conducted by a process of trial and error. Each time a change (or related collection of changes) was made, for example adding an additional layout element, the app was recompiled and tested on emulators and physical handsets. Sometimes subsequent changes required modification of functionality that was considered working. When linking with the RESO Control Server, transmissions between it and the app were initially in plain text so the system could be shown as working. For more details about the development of this interaction see section 20.3.

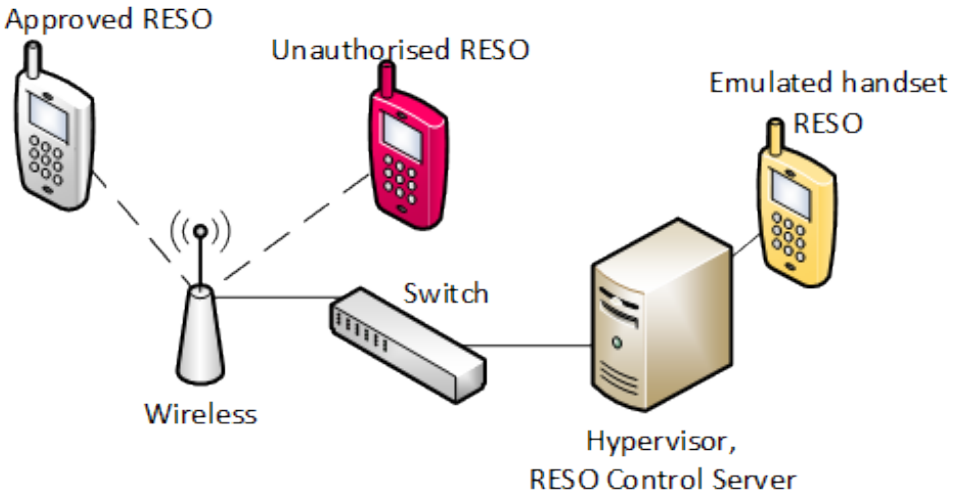


Figure 22: Testing lab network diagram

22.1 Hardware platforms

While testing in development 2 physical devices were used: the Asus Nexus 7 (2012) tablet and the HTC One V phone. Hardware specifications for these can be found in table 1.

	Nexus 7 (tablet)	**HTC One V (phone)**
Screen resolution	1280x800	800x480
CPU	NVIDIA® Tegra3™ Quad-Core, 1.2 GHz	Qualcomm Snapdragon MSM8255, Single Core 1 GHz processor
Audio?	Yes	Yes
Vibrate circuit?	No	Yes
Connectivity	WiFi	WiFi, 3G, GPRS, GSM

Table 1: Android test hardware specifications

The app's design took these differences into consideration. For example, if a server is down a sound is played and, where a vibrate circuit is present the device will vibrate. The vibrate circuit is not solely relied upon as it is not a required standard for Android devices.

Different screen resolutions have been catered for by use of scrolling layouts where required, an example can be found in figure 23 which shows the app's preferences screen. On the tablet the whole form is displayed whereas on the phone it is necessary to scroll to even see the full "save preferences" button. Despite this difference the app is still usable on each device.

It should be noticed that differences in visual style are due to the themes installed on each device.

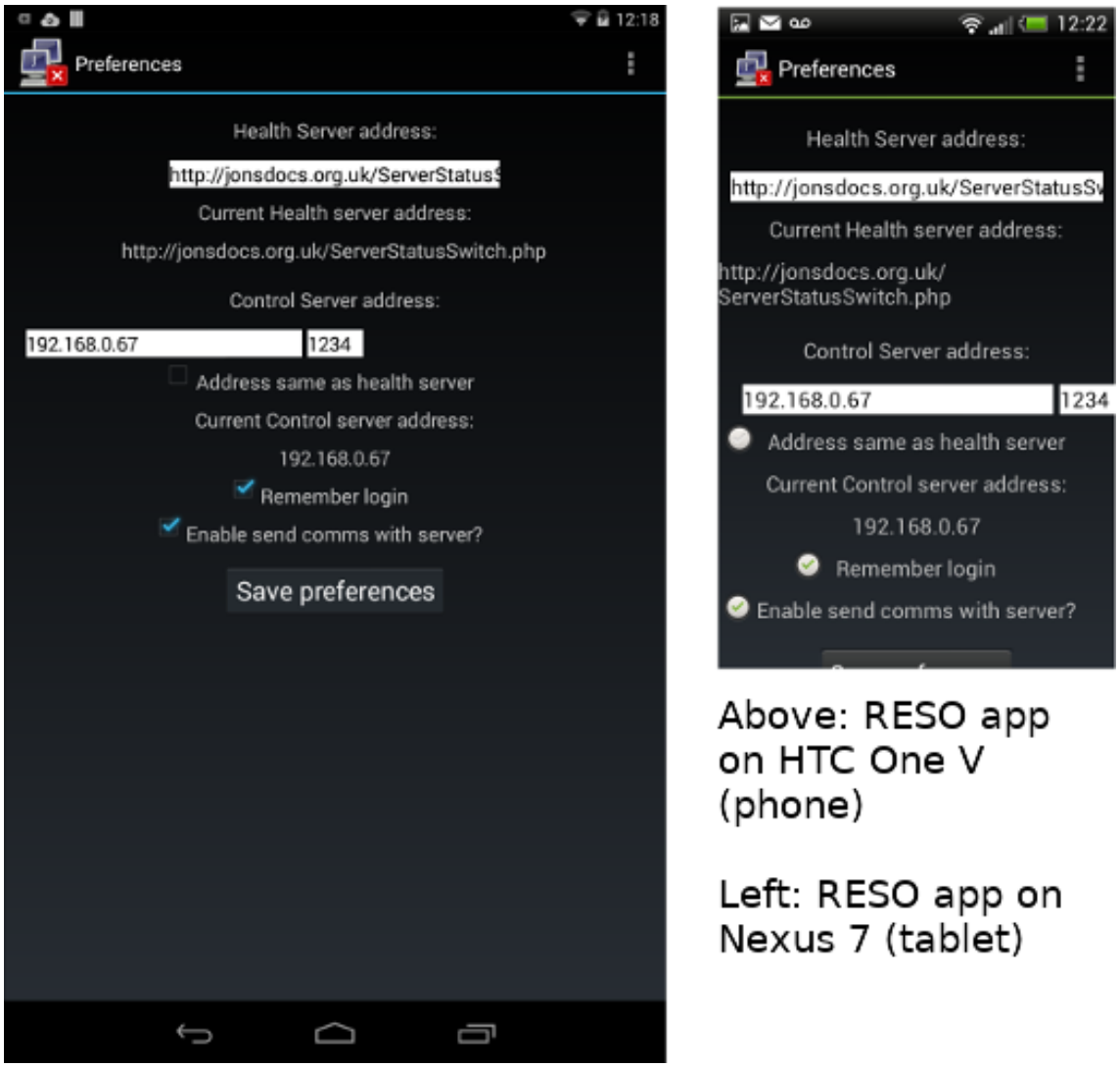


Figure 23: Screenshots showing RESO's preferences screen on different devices, part way through development

Figure 24 shows another difference caused by screen resolution - the app's menu covers key areas of the screen. If the menu were to be permanently open this would be considered a major problem in terms of usability, however, the menu is triggered by the user in order to access further settings within the app.

As the app is usable on a number of devices with different hardware specifications this requirement has been met.

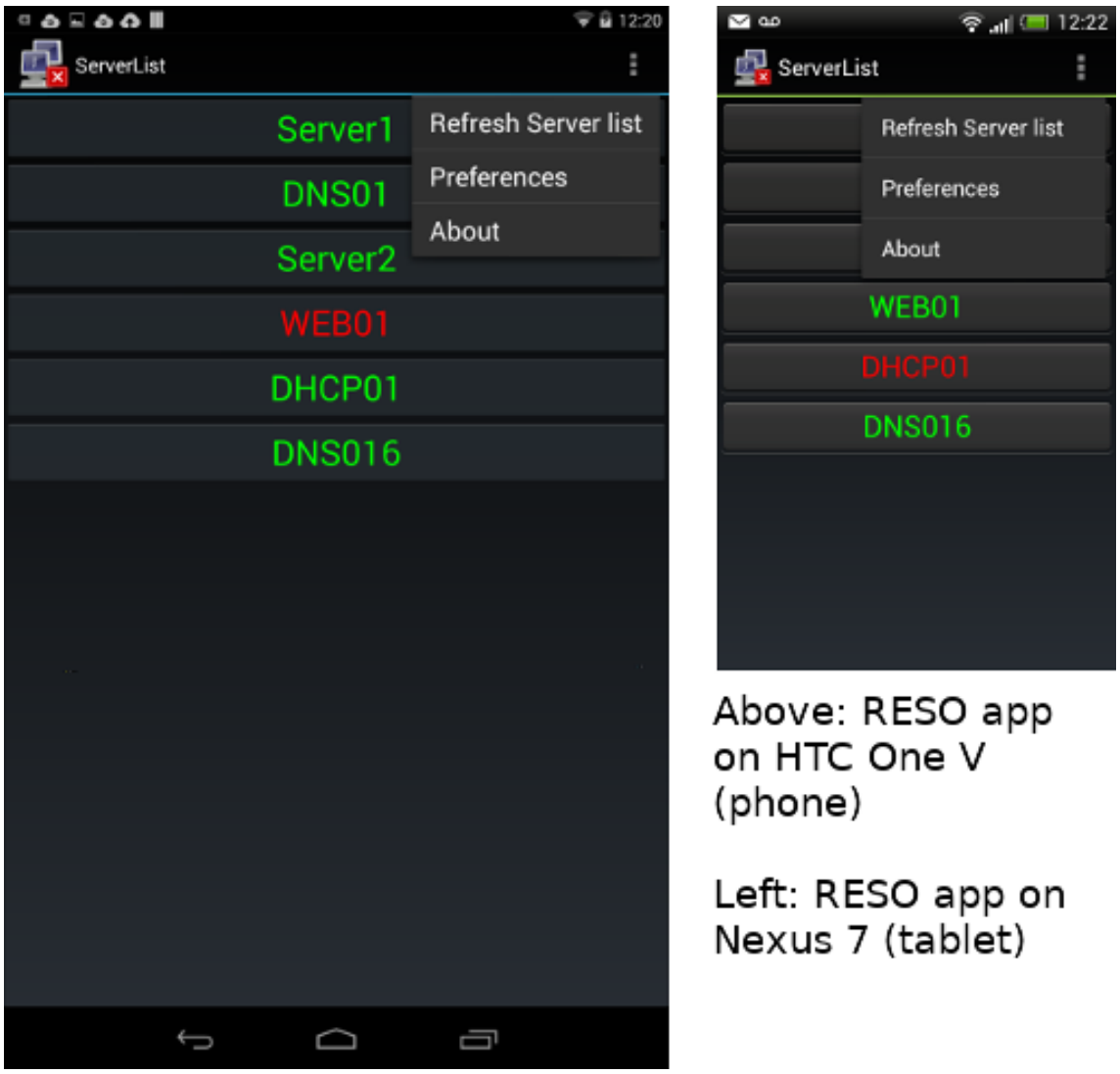


Figure 24: Screenshots showing RESO's server list with an open menu on different devices

22.2 Encryption

It is not possible to disable encryption within the app as this would be inherently insecure, however, for the purposes of testing a build was produced without encryption enabled. Using Wireshark, a network packet protocol analyser, traffic was captured between the mobile device and the RESO Control Server. Figure 25 shows this captured traffic - text in red (on the right) is the message sent from the mobile app to the control server which is clearly readable as `rebootHH;1412280045;359290051743494TA4750GQTEJonathanPhone;This is a password` : the instruction followed by a Unix timestamp, the device identifier and a password (in a final product a password should not be sent in plain text, instead a salted hash

should be used. For this proof of concept the password is being passed as plain text). As this information is in plain text it would be possible to tamper with the timestamp, setting it to a later value, and replay the packet.

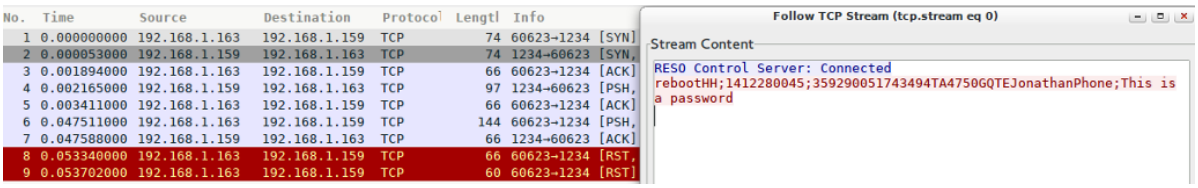


Figure 25: Captured unencrypted traffic between the mobile app and the RESO Control Server

Figure 26 shows the same message (albeit with a different timestamp) sent after encryption. As the message is not understandable (it appears as a set of numbers) without decryption it would be highly unlikely an attacker could successfully modify the encrypted message in order to attack the system.

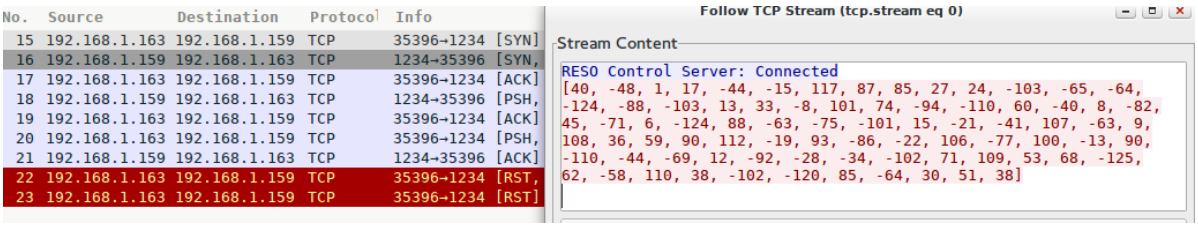


Figure 26: Traffic between the mobile app and the RESO Control Server, following encryption

22.3 Replay attack defence

The term "replay attack" describes malicious activity where data is captured and later resent (or replayed) to the original destination. Doing so would cause a repeat of the earlier action. To test the control server defended against such an attack a reboot instruction was captured and resent and the control server correctly identified the message was stale, halting any further processing.

```
Socket has been made
Waiting for connections...
Plaintext is: rebootHH;1412280277;359290051743494TA4750GQTEJonathanPhone;password
ALERT: Message was stale
```

Output from the control server on receiving replayed data

22.4 Push notifications

As there is no network monitoring system to provide data to the RESO Control Server the lab setup uses a script to send a push notification to RESO. This was received by the handset as shown in figure 27 which shows the "toast" (pop up message) and status bar notification. Both message types are used because the "toast" message is temporary whereas a status bar notification persists until cleared by the user. For clarity the screenshot has been annotated.

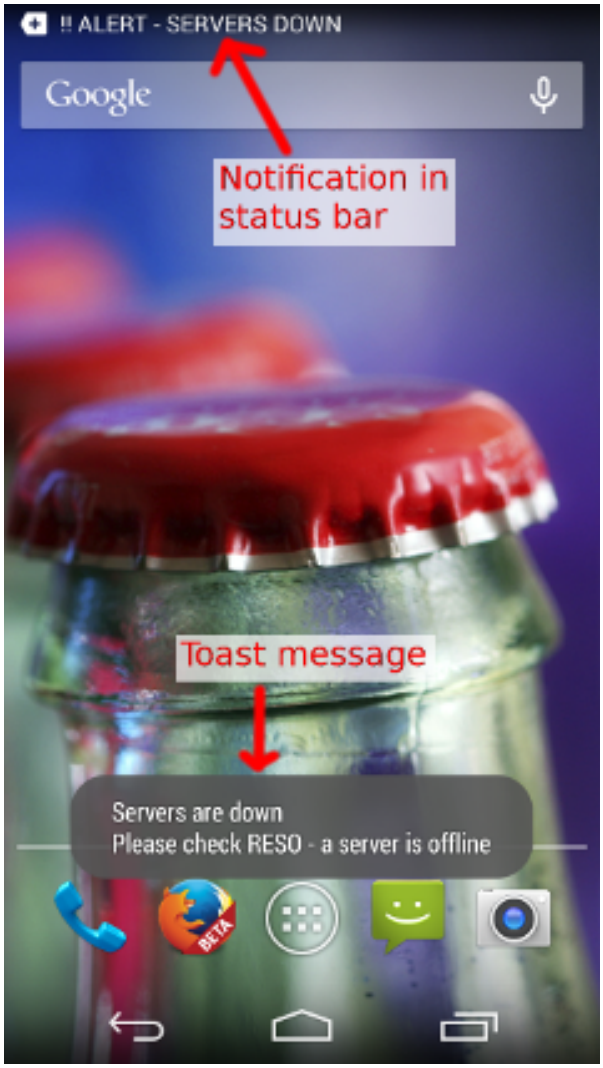


Figure 27: RESO: Push notification received by the phone

22.5 Protecting against unauthorised devices

When an unauthorised device attempts to send a send an instruction the control server rejects the message:

Socket has been made
Waiting for connections...
Plaintext is: startHH;1412281053;353115054807879HT25GTV04188;password
Instruction received from an unauthorised device identified as 353115054807879HT25GTV04188

Output from the control server on receiving message from an unauthorised device

22.6 Protecting against unauthorised users

In the event an employee must be locked out of the system his password is disabled. Trying to use that password results in the server ignoring the request:

Socket has been made
Waiting for connections...
Plaintext is: startHH;1412281132;359290051743494TA4750GQTEJonathanPhone;baduser2
Instruction received from an authorised device (359290051743494TA4750GQTEJonathanPhone) but invalid password specified

Output from the control server on receiving an invalid password

Part VI

Conclusions and evaluation

23 Key contributions

This project has designed, and partially implemented, a complete solution for remotely remedying a number of problems without the need for a traditional (laptop / computer based) network connection. Despite this solution needing further development it does prove such an implementation is possible.

A comparison of existing solutions has also been produced and can be used for further research by other projects. Although only a minor contribution it may be of benefit to those who follow.

24 Reflection on achievements related to aims and objectives

As has been shown in this report, the current state of RESO meets the requirements:

- Work on multiple hardware platforms
- Allow secure communications between the mobile device and monitoring server
- Permit an administrator to securely instigate corrective action
- Use *push notifications* to reduce the impact on battery life
- Be extensible to support multiple systems

The initial specification for secure communication only contained 2 elements, an instruction and timestamp, however, as the solution was developed it was noted how insecure this would be. Instead the specification has been expanded to also include a device identifier and user password allowing administrators to maintain control of their infrastructure.

It should be noted that further work is required to make the project commercially viable.

Research questions answered by this project were:

1. If an IT Manager can monitor a system remotely, can it also be influenced remotely?
2. Do IT administrators feel there is a benefit to using mobile device apps to monitor the health of their network?
3. What features would be required for a mobile app, should it be deemed one was desirable?

This research has answered these questions and shown it is possible to control a system remotely in a secure fashion. By using encryption, timestamps and obfuscation this project has successfully enabled an administrator control a system with a low risk of problems. Research conducted by interview has shown a mobile app is considered beneficial so long as it contains a mechanism for initiating remedial action.

25 Comparison to the research field

Various projects have shown it is not uncommon to extend a third-party monitoring system, for example Nagios, to provide additional functionality (Issariyapat et al., 2012) (Katsaros et al., 2011) and this project is in the same category. While RESO doesn't extend a specific monitoring system it requires an underlying system in order to function, however, interfacing with additional systems was outside the scope of this project. RESO was the only project found to provide a mechanism for remote control.

Buevich et al. have used remote systems to control electricity grids in rural areas (Buevich et al., 2014) and, despite not being an obvious comparison to this work, there are parallels. For example, Buevich et al. acknowledge the need for security in the system so encrypt exchanges between remote nodes and the control system. In their work a house's electrical supply is adjusted by a networked smart meter - clearly if the system can be tampered with their could be damaging consequences.

Using Android devices for remote control appears to be a growing trend with devices used to control CNC milling machines on the LAN (Truong and Vu, D-L., 2012) and for home automation (Gurek et al., 2013). In work by Gurek et al. it is noted their design elected

to use Android's `SharedPreferences` storage to persistently store user settings between program executions, as does this author's work. Concerningly, neither of these authors mention encryption or security in their papers which could lead to abuse of the systems if they are not securely implemented.

26 General strengths and weaknesses

RESO has advanced the remote network control field by allowing specific tasks to be conducted at a single button press on a mobile device. Previous solutions required a network connection and an interactive shell or web based control panel.

A weakness of RESO is that every version of the app uses the same encryption key (clearly this would need to be customisable should it become a commercial product). With each handset within a company having the same encryption key it would not be difficult to decrypt communications after an employee went rogue and adjust them accordingly. This could be improved by having individual keys and implementing an asymmetric mechanism.

The control server, as it stands, is another weakness as it requires significant improvement to become a viable product. This is discussed in more detail in section 18.5.

There are two key points of failure within this solution - the control server and the health information distribution. For RESO to be effective it must be able to download network health data in the form of an XML document. In the event this document is not available, for example due to a DoS attack, the app would be rendered useless. Alternatively, DNS poisoning could be used to direct the administrator's device to another server, providing them a different health file entirely. Depending on what information was present in the "rogue" XML file the administrator could be caused to reboot a system incorrectly which would lead to a DoS of the company's systems. Distribution of the health data via an encrypted or signed mechanism could help mitigate this risk so long as RESO alerted the administrator should an incorrectly signed document be received (and ideally the data should be discarded).

Similarly the control server could be targeted as part of the DoS attack. In this situation the administrator would be unable to take corrective action, should it be needed. As the control server is written in Java, which receives regular updates from its vendor for the purpose of security patching, it's possible a Java upgrade would stop the control server from functioning.

27 Commercial value

Should RESO become a viable, commercial product its uptake would depend, in part, on the legal obligations of organisations. For example, for government and local government who are connected to the Public Sector Network (PSN) would not be able to use the solution until 2 Factor Authentication (2FA) was implemented. 2FA on externally accessible systems is a strong recommendation (United Kingdom. CESG, 2012).

As RESO doesn't provide its own network health data it's necessary for existing network monitoring solutions to provide data for processing. As shown in section 8 a multitude of solutions exist so support would be needed from the respective software vendors to make them compatible with RESO. Open Source projects such as Nagios could have this provided as part of the RESO production process but this is not an option for closed source, commercial software.

Reputation is an important factor in the adoption of new software and as RESO and its publisher have no prior reputation companies may be hesitant to implement such a solution. Until RESO has been proven both useful and safe adoption would be slow.

28 General conclusions

This project has reviewed existing technologies and products for use in monitoring network health and found there is no single product suitable to cover all situations. Surveys and interviews with Information Technology industry professionals found the preferred method to be notified of problems by email although if an app were produced which allowed corrections to be made it would be useful. It was found there was no product capable of issuing remedial instructions so RESO was produced as a solution to this problem. It was also shown that it is necessary to authenticate both the device and the user of the device in order for the system to be as secure as possible.

29 Further work

This solution requires further development in order to become a polished commercial product. As discussed in section 18.5 the current configuration doesn't scale and this would need to

be addressed early on to bring the application out of the lab and into generic use.

Passwords are not currently protected as the system administrator is able to see the remote user's password supplied in the decrypted message on the control server. Preventing this would be a matter of Salting and hashing the password before it leaves the handset and then comparing it to a value stored on the control server. A more secure approach would be a move to using One Time Password (OTP) provided by a hardware token. The very nature of such passwords will make the system more secure as a captured password is useless, plus the user would need access to the hardware token. Tokens are widely available and common place in businesses where 2 factor authentication is used to secure access to other systems. Facilitating this additional step would require the RESO Control server to be made compatible with the token vendor's authentication system as token codes are mathematically generated, often using time as part of the seed.

While in the lab there was only one remote user that would be connecting to the control server. This is not realistic in an actual deployment scenario so the Control Server would need adjusting to make it capable of receiving multiple simultaneous connections. With multiple system administrators comes the possibility that multiple instructions will be sent to the control server in order to remedy a problem; in this case it would be necessary to notify other administrators that corrective action was already being taken (and by whom).

Authorised users and devices need to be definable in an administrator friendly way, perhaps by use of a GUI, and to be stored in an accessible format. A database would be appropriate for this application with tables for users, devices and a link table defining which users can use which devices. A suggested database entity relationship diagram can be found in figure 28 and foreign key constraints should be used to ensure the referential integrity of data. Additionally commands and their related servers would need defining and later distributing as noted in section 18.5.

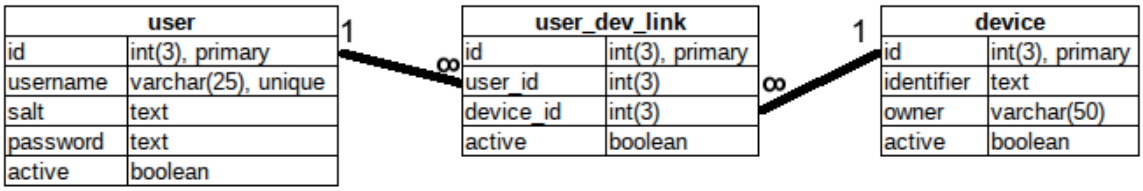


Figure 28: Suggested (basic) database entity relationship diagram

The control server component would require full testing to reduce the likelihood it would be compromised when attacked by a third party. Following a code review it would be worth while publishing the control server to the Internet within a honey pot. As the honey pot is attacked an analysis can be performed on the attacks in order to harden the software prior to releasing it in a production setting. Such a testing setup is described in figure 29.

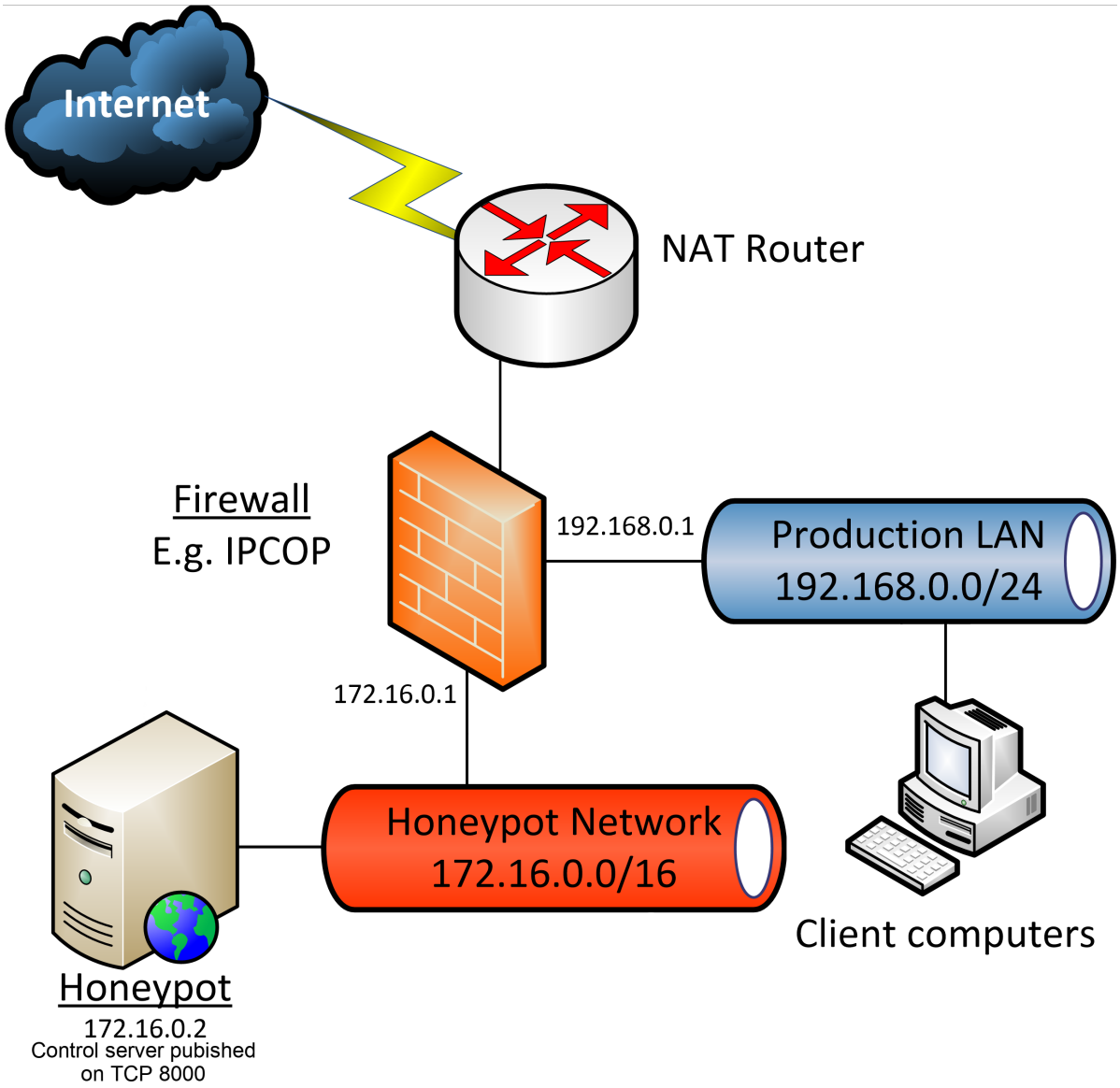


Figure 29: Lab network showing an appropriately isolated honey pot network.

Finally, there is no documentation for the application so this would need developing so administrators were able to deploy and maintain the system.

References

ACDX (2008, November). Digital signature diagram. [Online] Available at: http://upload.wikimedia.org/wikipedia/commons/2/2b/Digital_Signature_diagram.svg [Accessed 31 March 2014].

Atkinson, C. and Hummel, O. (2012). Iterative and incremental development of component-based software architectures. In *Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering*, CBSE '12, New York, NY, USA, pp. 77–82. ACM.

Beck, K. (2002). *Test driven development: by example* (2 ed.). Boston, MA, USA: Addison-Wesley.

Beck, K. et al. (2001). Manifesto for agile software development. [Online] Available at: http://agilemanifesto.org/ [Accessed: 1 November 2014].

Bramley, J. (2014). *Questions on your company's monitoring system*. [Interview - transcript with author]. 20 January 2014.

Bryan, A. and McDermott, J. (2011). Learning tree international course 468, system and network security: A comprehensive introduction. Course.

Buevich, M., Schnitzer, D., Escalada, T., Jacquiau-Chamski, A., and Rowe, A. (2014). Fine-grained remote monitoring, control and pre-paid electrical service in rural microgrids. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IPSN '14, Berlin, Germany, 15-17 April 2014, Piscataway, NJ, USA, pp. 1–12. IEEE Press.

Burke, J., McDonald, J., and Austin, T. (2000). Architectural support for fast symmetric-key cryptography. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX. Cambridge, Massachusetts, USA, 2000, New York, NY, USA, pp. 178–189. ACM.

Cassidy, A. (2014). *Questions on your company's monitoring system*. [Interview - transcript with author]. 20 January 2014.

Cockburn, A. (2008, May). Using both incremental and iterative development. *STSC Crosstalk 21*(5), 27–30.

Cody-Kenny, B., Guerin, D., Ennis, D., Carbajo, S.R, Huggard, M., and Mc Goldrick, C. (2009). Performance evaluation of the 6LoWPAN protocol on MICAz and TelosB Motes. In *Proceedings of the 4th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, PM2HW2N '09. Tenerife, Canary Islands, Spain, 2009, New York, NY, USA, pp. 25–30. ACM.

de Chaves, S.A., Uriarte, R.B., and Westphall, C.B. (2013). Toward an architecture for monitoring private clouds. In *Communications Magazine*, Volume 49(12), pp. 130–137. IEEE.

Excirial and Renier, M (2009). Test driven development graphic. [Online] Available at: http://upload.wikimedia.org/wikipedia/commons/9/9c/Test-driven_development.PNG [Accessed: 1 November 2014].

Fernandes, B. (2014). A simple example of using aes encryption in java and c. [Online] Available at: https://gist.github.com/bricef/2436364 [Accessed: 22 November 2014].

Flick, U. (2007). *Designing qualitative research* (1 ed.). London, England: Sage Publications Ltd.

GFI Max (2012). Ccr technology group — maxfocus remotemanagement — [case study]. [Online] Available at: http://www.youtube.com/v/uWfsMdLMHNE&rel=0&autoplay=1 [Accessed: 23 November 2014].

Glasser, B. G. and Strauss, A.L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Transaction.

Glenn, J. C. (2010). *Handbook of Research Methods* (2010 ed.). Oxford, England: Oxford Book Company.

Google (2014). Android NDK. [Online] Available at: http://developer.android.com/tools/sdk/ndk/index.html [Accessed: 27 April 2014].

Gurek, A., Gur, C., Gurakin, C., Akdeniz, M., Metin, S.K., and Korkmaz, I. (2013, Dec). An android based home automation system. In *10th International Conference on High Capacity Optical Networks and Enabling Technologies (HONET-CNS), 2013*, Turkey Magosa, Cyprus , 11-13 December 2013, pp. 121–125. IEEE Press.

HMKCode (2014, March). Android google cloud messaging tutorial. [Online] Available at: http://hmkcode.com/android-google-cloud-messaging-tutorial/ [Accessed: 22 November 2014].

Hunt, A. and Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master* (16 ed.). London, England: Addison-Wesley.

Issariyapat, C., Pongpaibool, P., Mongkolluksame, S., and Meesublak, K. (2012). Using Nagios as a groundwork for developing a better network monitoring system. In *Technology Management for Emerging Technologies (PICMET)*, 2012 Proceedings of PICMET '12. Vancouver, BC, Canada, July 29 2012-Aug. 2 2012, pp. 2771–2777. IEEE.

Katsaros, G., Kubert, R., and Gallizo, G. (2011). Building a service-oriented monitoring framework with REST and Nagios. In *Services Computing (SCC)*, 2011 IEEE International Conference. Washington, DC, USA, 4-9 July 2011, pp. 426–431. IEEE.

Knibbs, S. (2014). *Comment from survey on monitoring systems*. [Survey]. January 2014.

Lapan, S., Quartaroli, M.T., and Riemer, F.J. (Eds.) (2012). *QUALITATIVE RESEARCH An Introduction to Methods and Designs* (1 ed.). San Francisco, USA: Jossey-Bass.

Larman, C. and Basili, V.R. (2003, June). Iterative and incremental developments. a brief history. *Computer 36*(6), pp. 47–56.

Massie, M. L., Chun, B.N., and Culler, D.E. (2004). The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing 30*(7), 817 – 840.

MIT (2007). Kerberos v5 system administrator's guide. [Online] Available at: http://web.mit.edu/Kerberos/krb5-1.5/krb5-1.5.4/doc/krb5-admin/Clock-Skew.html [Accessed 20 November 2014].

Nagios Core Development Team (2010, August). Nagios core version 3.x documentation. [Online] Available at: http://nagios.sourceforge.net/docs/nagioscore-3-en.pdf [Accessed: 1 November 2014].

Neuman, C., Hartman, S., and Raeburn, K. (2005, July). Rfc 4120: The kerberos network authentication service (v5). [Online] Available at: tools.ietf.org/html/rfc4120 [Accessed 20 November 2014].

Parnerkar, A., Guster, D., and Herath, J. (2003, October). Secret key distribution protocol using public key cryptography. *J. Comput. Sci. Coll. 19*(1), 182–193.

Royce, W. W. (1970). Managing the development of large software systems. In *Preceedings, IEEE WESCON*, Los Angeles, California, USA 25th - 28th August 1970, pp. 1–9. IEEE.

Russinovich, M. E. and Solomon, D.A. (2009). *Windows Internals* (Fifth ed.). Redmond, Washington: Microsoft Press.

Simmons, G. J. (1979, December). Symmetric and asymmetric encryption. *ACM Comput. Surv. 11*(4), 305–330.

Smith, S. (2012). What is cloud computing? In *Cloud computing, moving IT out of the office*, Chapter 1, pp. 2–7. Swindon, England: British Informatics Society Limited.

Sun, J., Hoke, E., Strunk, J.D., Ganger, G.R., and Faloutsos,C. (2006). Intelligent system monitoring on large clusters. In *Proceedings of the 3rd International Workshop on Data Management for Sensor Networks (DMSN06)*, Seoul, South Korea, 11 September 2006, pp. 47–52. ACM.

Tang, L., Li, T., Schwartz, L., Pinel, F., and Grabarnik, G.Y. (2013). An integrated framework for optimizing automatic monitoring systems in large IT infrastructures. In *KDD '13 Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, Chicago, Illinois, USA, 11-14 August 2013, pp. 1249–1257. ACM.

Truong, N.-V. and Vu, D-L. (2012, Nov). Remote monitoring and control of industrial process via wireless network and android platform. In *2012 International Conference on Control, Automation and Information Sciences (ICCAIS)*, ICCAIS 2012. Ho Chi Minh City, Vietnam , 26-29 November 2012, pp. 340–343. IEEE Press.

United Kingdom. CESG (2012). *CESG Architectural Patterns Mobile Remote Endpoint Devices at RESTRICTED*. United Kingdom. CESG.

Wang, Z., Murmuria, R., and Stavrou, A. (2012). Implementing and optimizing an encryption filesystem on android. In *Proceedings of the 2012 IEEE 13th International Conference on Mobile Data Management*, MDM '12. Bengaluru, Karnataka, India ,23-26 July 2012, Washington, DC, USA, pp. 52–62. IEEE Computer Society.

Washam, M. (2012). Automating windows azure virtual machines with powershell. [Online] Available at: http://michaelwasham.com/2012/06/08/automating-windows-azure-virtual-machines-with-powershell/ [Accessed: 22 November 2014].

Wu, M., Zhang, Z., and Li, Y. (2013). Application research of Hadoop resource monitoring system based on Ganglia and Nagios. In *4th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, 23-25 May 2013, pp. 684–688.

Part VII

Appendix

A Survey questions

1. How would you describe your job role?
    - IT Manager with overall responsibility for systems
    - In-house IT support engineer/technician
    - IT support engineer providing services to 3rd parties
    - Financial director
    - Other (please specify)
2. If known, please state the cost to your business in the event of your IT system failing for 1 hour
3. Similarly, if known, please state the cost to your business in the event of your IT system failing for 1 business day
4. Please indicate network monitoring systems with which you have experience
    - Big Brother
    - Cacti
    - Ganglia
    - GFI Max Remote Management
    - Icinga
    - Nagios
    - openView
    - ParMon
    - Smokeping
    - Solarwinds Orion
    - Splunk
    - Tivoli
    - Other (please specify)

5. When the monitoring system detects a problem, is a support ticket automatically raised in your helpdesk/management system?
    - Yes (immediately)
    - Yes (after a delay)
    - No
    - Other (please specify)
6. If yes to question 5, what percentage would you estimate were false positives? (a false positive in this case is a ticket which was created by a transient problem, found to be already resolved)
    - Less than 25%
    - 25 - 50%
    - Over 50%
    - Over 75%
    - Unsure
7. In the event of a problem, how would you prefer to be alerted to problems (please place in order of preference)?
    - Email
    - SMS text message
    - Pager message
    - Phone call (automated)
    - Notification via desktop software (i.e. on a laptop or desktop computer)
    - Notification via smartphone app (not an SMS)
    - I prefer to wait until my users report the problem

B Interview questions

These questions formed the basis of interviews conducted as part of this project, however, additional questions were posed as appropriate to the interviewee.

1. Could you briefly outline the size of the system you monitor? How many servers, clients, other devices?
2. What Operating Systems are present in your environment?
3. Do you have any automated systems in place to monitor the system?
    - If so, which one(s)?
    - How do you receive alerts from these monitoring systems?
    - What benefits do you see from your automated monitoring system?
    - Are there cost implications (in both having the system, e.g. license fees, and not having the system, e.g. lost revenue)? That is, are the costs of your remote monitoring system offset by cost savings elsewhere?
    - What were the deciding factors when choosing your monitoring system?
    - Of the monitoring system you use, what features do you value the most? Which features do you wish were present?
4. What would be your preferred alerting method from a monitoring system? Why?
5. Do you think smart-phones could be more utilised by system administrators, and if so how? (Specifically in regards to monitoring system health but also in general)

C XML Specification

- The XML document is contained within the <maintag> element and may contain numerous <server> elements
- Within the <server> element are 3 children:
  - <name>
    (String) The server's name, for example DNS01
    </name>
  - <ip>
    (String) An IPv4 address
    </ip>
  - <state>
    (String) State can be one of either up or down
    </state>

```
<maintag>
   <server>
                <name>SERVERNAME</name>
                        <ip>IP_ADDRESS</ip>
                        <state>[up | down]</state>
        </server>
</maintag>
```

Listing 9: XML Specification for data exchange with RESO

D RESO App source code

Source code for the RESO app can be found here. There are a number of files, some in XML and some Java. The structure of the source follows the requirements for an Android app and can be seen in Figure 30. Note there are a number of `drawable` directories - these house graphics required for varying screen resolutions. Additionally the `raw` directory contains the sound file `serverdownalert.ogg`.

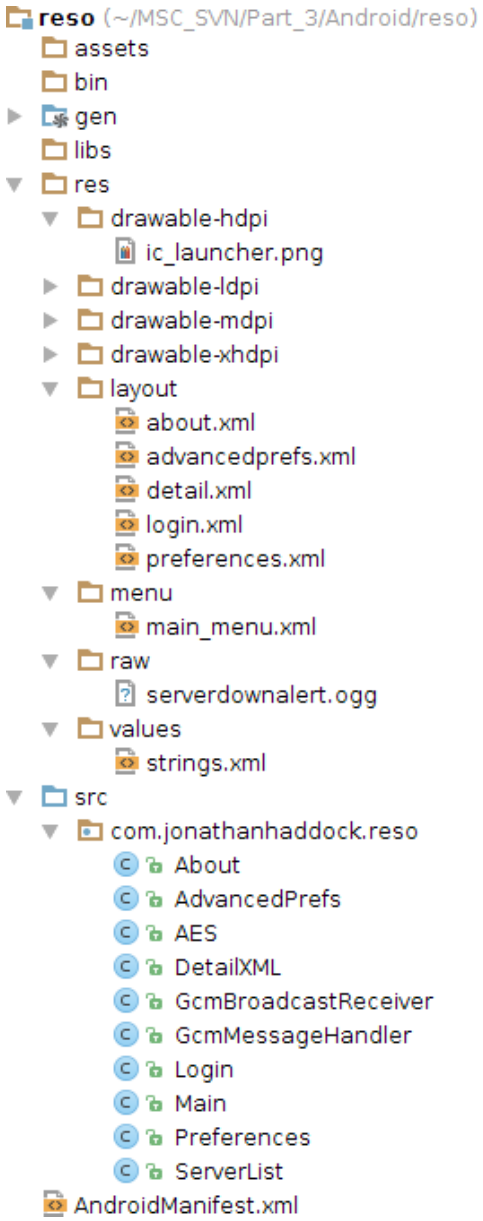


Figure 30: File structure of the RESO app, as shown in JetBrains IntelliJ v13.1

D.1 XML: RESO App, AndroidManifest.xml

All activities and permissions must be defined in `AndroidManifest.xml`, it's not possible to switch to an activity that's not defined in the file.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3            package="com.jonathanhaddock.reso"
4            android:versionCode="1"
5            android:versionName="1.0">
6      <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="19" />
7      <uses-permission android:name="android.permission.INTERNET" />
8      <uses-permission android:name="android.permission.VIBRATE" />
9      <uses-permission android:name="android.permission.READ_PHONE_STATE" />
10     <uses-permission android:name="android.permission.GET_ACCOUNTS" />
11     <uses-permission android:name="android.permission.WAKE_LOCK" />
12     <uses-permission android:name="com.google.android.c2dm.permission.
           RECEIVE" />
13     <permission android:name="com.jonathanhaddock.reso.permission.
           C2D_MESSAGE"
14                 android:protectionLevel="signature" />
15     <uses-permission android:name="com.jonathanhaddock.reso.permission.
           C2D_MESSAGE" />
16     <application android:label="@string/app_name" android:icon="@drawable/
           ic_launcher">
17         <activity android:name=".Login"
18                   android:label="@string/activityLogin">
19             <intent-filter>
20                 <category android:name="android.intent.category.LAUNCHER" /
                       >
21                 <action android:name="android.intent.action.MAIN" />
22             </intent-filter>
23         </activity>
24         <activity android:name=".Main"
25                   android:label="@string/app_name">
26         </activity>
27         <activity android:name=".About"
28                   android:label="@string/activityAbout" />
29         <activity android:name=".DetailXML"
30                   android:label="@string/activityDetail" />
31         <activity android:name=".ServerList"
32                   android:label="@string/activityServerList" />
33         <activity android:name=".Preferences"
34                   android:label="@string/activityPreferences" />
35         <activity android:name=".AdvancedPrefs"
36                   android:label="@string/activityAdvancedPrefs" />
37         <receiver
38                 android:name=".GcmBroadcastReceiver"
39                 android:permission="com.google.android.c2dm.permission.
                       SEND">
40             <intent-filter>
```

```
41                  <action android:name="com.google.android.c2dm.intent.
                        RECEIVE" />
42                  <category android:name="com.jonathanhaddock.reso" />
43              </intent-filter>
44          </receiver>
45          <service android:name=".GcmMessageHandler" />
46      </application>
47  </manifest>
```

Listing 10: XML: RESO App - AndroidManifest.xml

D.2 XML: RESO App, strings.xml

It is recommended that text strings in Android are stored in `strings.xml`; these can then be referenced within layouts. This technique is particularly useful for internationalisation where multiple translations may be available as the source code doesn't need to be adjusted.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="blank"></string>
    <string name="app_name">RESO</string>
    <string name="activityLogin">RESO - login</string>
    <string name="activityAbout">About</string>
    <string name="activityDetail">Detail</string>
    <string name="activityServerList">ServerList</string>
    <string name="activityPreferences">Preferences</string>
    <string name="activityAdvancedPrefs">Advanced Prefs</string>
    <!--For menus-->
    <string name="menuAbout">About</string>
    <string name="menuOverview">Overview</string>
    <string name="menuRefreshServerList">Refresh Server list</string>
    <string name="menuPreferences">Preferences</string>
<!--For login screen-->
    <string name="labelEnterPassword">Enter password:</string>
    <string name="labelLoginError">Please try again</string>
    <string name="labelBlankPass">Please enter a password</string>
    <string name="hintPassword">Type your password</string>
    <string name="buttonLogin">Login</string>
    <!--For details-->
    <string name="buttonReboot">Reboot</string>
    <string name="buttonStart">Start</string>
    <!-- For preferences -->
    <string name="labelServerAddress">Health Server address:</string>
    <string name="labelControlServerAddress">Control Server address:</
        string>
    <string name="labelPrefsServerAddress">Current Health server address:<
        /string>
    <string name="labelCurrentControlServerAddress">Current Control server
         address:</string>
    <string name="hintServerAddress">Type server address</string>
    <string name="hintServerPort">Port</string>
    <string name="buttonRegisterGCM">Register for GCM</string>
    <string name="buttonSavePrefs">Save preferences</string>
    <string name="buttonAdvancedPrefs">Advanced preferences</string>
    <string name="labelRememberLogin">Remember login</string>
    <string name="labelSendData">Enable send comms with server?</string>
    <string name="labelBlank"></string>
    <string name="labelControlIsHealth">Address same as health server</
        string>
    <string name="labelUserDeviceId">User specified device ID:</string>
    <string name="labelUserDeviceIdHelp">You can optionally provide a
        device ID. Care should be taken to ensure this is unique. Use
```

```
        with caution.</string>
41      <string name="labelDeviceIMEI">Device IMEI:</string>
42      <string name="labelDeviceSerial">Device Serial:</string>
43      <!--For about-->
44      <string name="labelAcknowledgments">The author would like to thank
            online communities such as StackOverflow and Freenode for providing
             assistance during the development of this app.</string>
45  </resources>
```

Listing 11: XML: RESO App - strings.xml

D.3 Java: RESO App, About.java

`About.java` is a simple example showing an activity which simply calls a view. It, and most other Java classes, extend the `Main` class providing access to additional variables and methods (see Section D.14).

```
1  package com.jonathanhaddock.reso;
2
3  import android.os.Bundle;
4
5  public class About extends Main {
6      @Override
7      public void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          setContentView(R.layout.about);
10     }
11 }
```

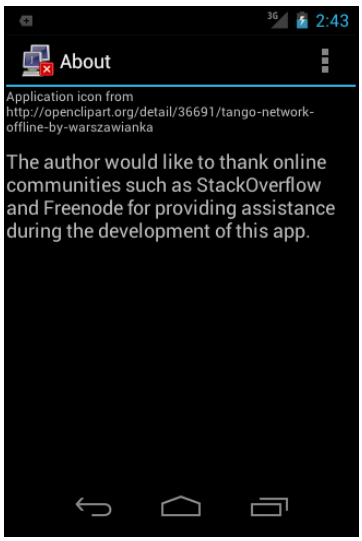
Listing 12: Java: RESO App - About.java

D.4 XML: RESO App, about.xml

This XML file defines the layout used in the About activity. There are two `TextView` elements, one which has the text entered directly (line 9) and another which pulls in text from the `strings.xml` file (line 18). Being a *relative* layout the order of elements on screen is described by its position relative to another. Line 17 is an example.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                  android:layout_width="match_parent"
4                  android:layout_height="match_parent">
5      <TextView
6              android:id="@+id/labelIcon"
7              android:layout_width="wrap_content"
8              android:layout_height="wrap_content"
9              android:text="Application icon from \nhttp://openclipart.org/
                   detail/36691/tango-network-offline-by-warszawianka"
10             />
11     <TextView
12             android:id="@+id/labelAcknowledgments"
13             android:layout_width="wrap_content"
14             android:layout_height="wrap_content"
15             android:layout_centerHorizontal="true"
16             android:layout_below="@id/labelIcon"
17             android:layout_marginTop="10dp"
18             android:text="@string/labelAcknowledgments"
19             android:textSize="8pt"
20             />
21 </RelativeLayout>
```

Listing 13: XML: RESO App - about.xml



D.5 XML: RESO App, advancedprefs.xml

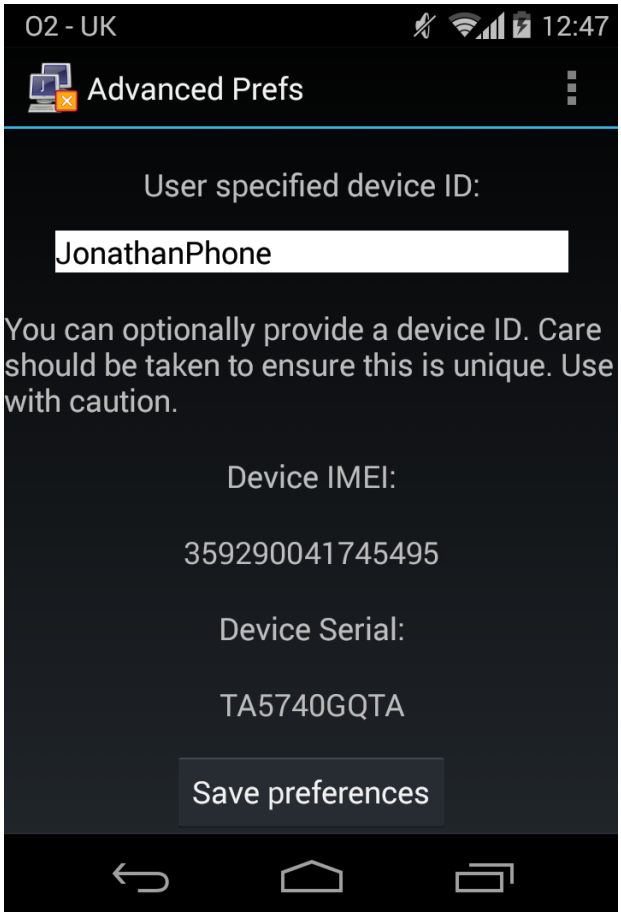
The advanced preferences activity allows users to view their device IMEI and serial number, which the administrator requires, and also the option of providing a friendly device ID. The XML below shows two `TextView` elements with blank text (lines 60-69 and 80-89). These are populated by code in `AdvancedPrefs.java`.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                  android:layout_width="match_parent"
4                  android:layout_height="match_parent"
5                  android:orientation="vertical"
6                  android:overScrollMode="ifContentScrolls"
7                  android:scrollbarStyle="insideOverlay"
8                  android:scrollbars="vertical"
9          >
10     <ScrollView
11             android:layout_width="fill_parent"
12             android:layout_height="wrap_content">
13         <RelativeLayout
14                 android:layout_width="match_parent"
15                 android:layout_height="wrap_content">
16
17             <TextView
18                     android:id="@+id/labelUserDeviceId"
19                     android:layout_width="wrap_content"
20                     android:layout_height="wrap_content"
21                     android:layout_centerHorizontal="true"
22                     android:layout_marginTop="20dp"
23                     android:text="@string/labelUserDeviceId"
24                     android:textSize="8pt"
25                     />
26             <EditText
27                     android:id="@+id/inputUserDeviceId"
28                     android:layout_width="300dp"
29                     android:layout_height="wrap_content"
30                     android:layout_centerHorizontal="true"
31                     android:layout_below="@id/labelUserDeviceId"
32                     android:layout_marginTop="15dp"
33                     android:cursorVisible="true"
34                     android:background="#FFF"
35                     android:textSize="8pt"
36                     android:textColor="#000"
37                     android:maxLength="250"
38                     android:inputType="text">
39             </EditText>
40             <TextView
41                     android:id="@+id/labelUserDeviceIdHelp"
42                     android:layout_width="wrap_content"
43                     android:layout_height="wrap_content"
44                     android:layout_centerHorizontal="true"
```

```
45                  android:layout_below="@id/inputUserDeviceId"
46                  android:layout_marginTop="20dp"
47                  android:text="@string/labelUserDeviceIdHelp"
48                  android:textSize="8pt"
49                  />
50          <TextView
51                  android:id="@+id/labelDeviceIMEI"
52                  android:layout_width="wrap_content"
53                  android:layout_height="wrap_content"
54                  android:layout_centerHorizontal="true"
55                  android:layout_below="@id/labelUserDeviceIdHelp"
56                  android:layout_marginTop="20dp"
57                  android:text="@string/labelDeviceIMEI"
58                  android:textSize="8pt"
59                  />
60          <TextView
61                  android:id="@+id/labelDeviceIMEIValue"
62                  android:layout_width="wrap_content"
63                  android:layout_height="wrap_content"
64                  android:layout_centerHorizontal="true"
65                  android:layout_below="@id/labelDeviceIMEI"
66                  android:layout_marginTop="20dp"
67                  android:text="@string/blank"
68                  android:textSize="8pt"
69                  />
70          <TextView
71                  android:id="@+id/labelDeviceSerial"
72                  android:layout_width="wrap_content"
73                  android:layout_height="wrap_content"
74                  android:layout_centerHorizontal="true"
75                  android:layout_below="@id/labelDeviceIMEIValue"
76                  android:layout_marginTop="20dp"
77                  android:text="@string/labelDeviceSerial"
78                  android:textSize="8pt"
79                  />
80          <TextView
81                  android:id="@+id/labelDeviceSerialValue"
82                  android:layout_width="wrap_content"
83                  android:layout_height="wrap_content"
84                  android:layout_centerHorizontal="true"
85                  android:layout_below="@id/labelDeviceSerial"
86                  android:layout_marginTop="20dp"
87                  android:text="@string/blank"
88                  android:textSize="8pt"
89                  />
90          <Button
91                  android:id="@+id/buttonSavePrefs"
92                  android:layout_width="wrap_content"
93                  android:layout_height="wrap_content"
94                  android:layout_centerHorizontal="true"
95                  android:layout_below="@id/labelDeviceSerialValue"
96                  android:layout_marginTop="15dp"
```

```
97                      android:text="@string/buttonSavePrefs"
98                      android:onClick="savePrefs"
99                      />
100         </RelativeLayout>
101     </ScrollView>
102 </RelativeLayout>
```

Listing 14: XML: RESO App - advancedprefs.xml



D.6 Java: RESO App, AdvancedPrefs.java

The `onCreate()` method defines the view and populates on screen elements with current information: lines 23-25 create objects for the updates which require updating while the remainder of the method obtains the relevant information and updates the text value of the objects.

A second class, `savePrefs()`, saves the user specified device ID to Shared Preferences storage before displaying a pop-up message (a toast) and returning the user to the server list.

```
1  package com.jonathanhaddock.reso;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.content.SharedPreferences;
6  import android.os.Build;
7  import android.os.Bundle;
8  import android.telephony.TelephonyManager;
9  import android.view.View;
10 import android.widget.EditText;
11 import android.widget.TextView;
12 import android.widget.Toast;
13
14 public class AdvancedPrefs extends Main {
15
16     public void onCreate(Bundle savedInstanceState) {
17         /**
18          * Define the view and then update on screen boxes with current
                values.
19          */
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.advancedprefs);
22
23         EditText inputUserDeviceId = (EditText) findViewById(R.id.
               inputUserDeviceId);
24         TextView labelDeviceIMEIValue = (TextView) findViewById(R.id.
               labelDeviceIMEIValue);
25         TextView labelDeviceSerialValue = (TextView) findViewById(R.id.
               labelDeviceSerialValue);
26
27         SharedPreferences sharedPrefUserDeviceId = getSharedPreferences("
               deviceIdFromUser", 0);
28         String deviceIdFromUser = sharedPrefUserDeviceId.getString("
               deviceIdFromUser", "");
29
30         inputUserDeviceId.setText(deviceIdFromUser);
31         TelephonyManager telephonyManager = (TelephonyManager)
               getSystemService(Context.TELEPHONY_SERVICE);
```

```
32          labelDeviceIMEIValue.setText(telephonyManager.getDeviceId());
33          labelDeviceSerialValue.setText(Build.SERIAL);
34      }
35
36      public void savePrefs(View view) {
37          /**
38           * Once "save preference" has been tapped write the values
39           */
40          Intent intent = new Intent(AdvancedPrefs.this, ServerList.class);
41          EditText inputUserDeviceId = (EditText) findViewById(R.id.
                inputUserDeviceId);
42          String userDeviceId = inputUserDeviceId.getText().toString();
43          SharedPreferences sharedPrefUserDeviceId = getSharedPreferences("
                deviceIdFromUser", 0);
44          SharedPreferences.Editor editorPrefUserDeviceId =
                sharedPrefUserDeviceId.edit();
45          if (!userDeviceId.isEmpty()) {
46              editorPrefUserDeviceId.putString("deviceIdFromUser",
                    userDeviceId);
47          }
48
49          editorPrefUserDeviceId.commit();
50
51          /** Toast to say the preference has been saved then go back to
                ServerList: */
52          Toast.makeText(getApplicationContext(), "Preferences saved", Toast
                .LENGTH_LONG).show();
53          startActivity(intent);
54      }
55  }
```

Listing 15: Java: RESO App - AdvancedPrefs.java

D.7 Java: RESO App, AES.java

`AES.java` provides no visible output to the user - it's not an Android activity. Methods within `AES.java` are called from elsewhere in the app to encrypt messages prior to transmission to the control server. The `decrypt()` method is not used in the current RESO app. This code uses contributions from Fernandes (2014).

```
1  package com.jonathanhaddock.reso;
2
3  import javax.crypto.Cipher;
4  import javax.crypto.spec.IvParameterSpec;
5  import javax.crypto.spec.SecretKeySpec;
6
7  /**
8   * From: https://gist.github.com/bricef/2436364, Brice Fernandes (bricef)
9   * NOTE: key and plaintext must both be multiples of 16 or padded with \0
        unless you use AES/CBC/PKCS5Padding
10  * NOTE: IV must also be 16 bytes long
11  */
12 public class AES {
13     static String IV = "ABCDEFGHIJKLMNOP";
14     static String encryptionKey = "0123456789abcdef0123456789abcdef";
15
16     public static byte[] aesmain(String providedPlaintext) {
17         /**
18          * Take plain text from elsewhere in the app and then encrypt it.
19          */
20         try {
21             byte[] cipher = encrypt(providedPlaintext, encryptionKey);
22             return cipher; //Cipher is of type byte[]
23         } catch (Exception e) {
24             e.printStackTrace();
25         }
26         return null; //was encString
27     }
28
29     public static byte[] encrypt(String plainText, String encryptionKey)
           throws Exception {
30         /**
31          * Main encryption function
32          */
33         Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
34         SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF
               -8"), "AES");
35         cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(IV.
               getBytes("UTF-8")));
36         return cipher.doFinal(plainText.getBytes("UTF-8"));
37     }
38
39     public static String decrypt(byte[] cipherText, String encryptionKey)
           throws Exception {
```

```
40          /**
41           * Main decryption function - currently unused
42           */
43          Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
44          SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF
                -8"), "AES");
45          cipher.init(Cipher.DECRYPT_MODE, key, new IvParameterSpec(IV.
                getBytes("UTF-8")));
46          return new String(cipher.doFinal(cipherText), "UTF-8");
47      }
48  }
```

Listing 16: Java: RESO App - AES.java

D.8 XML: RESO App, detail.xml

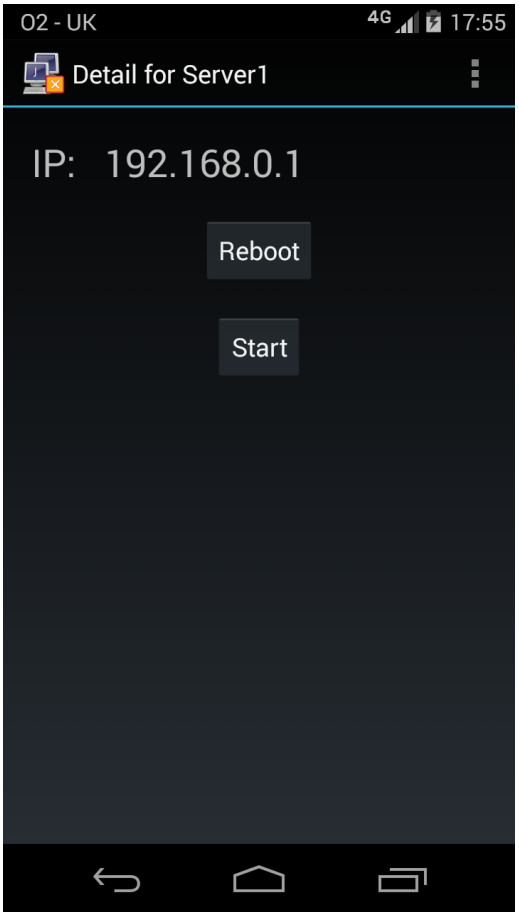
In the current build of RESO this view merely shows text as hard-coded in `DetailXML.java`, discussed in section D.9. This layout provides some additional summary information (the server IP) and two buttons - one to reboot and the other to start the server in question.

As a precaution, tapping either button will prompt the user to confirm their actions. Again, this is part of `DetailXML.java`.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent">
    <TextView
            android:id="@+id/labelIP"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginLeft="20dp"
            android:textSize="12pt"
            android:text="IP:"
            />
    <TextView
            android:id="@+id/labelIPValue"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginLeft="20dp"
            android:layout_toRightOf="@id/labelIP"
            android:textSize="12pt"
            android:text="--incomplete--"
            />
    <Button
            android:id="@+id/buttonReboot"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginLeft="20dp"
            android:layout_below="@id/labelIP"
            android:layout_centerHorizontal="true"
            android:text="@string/buttonReboot"
            android:onClick="rebootServer"
            />
    <Button
            android:id="@+id/buttonStartServer"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginLeft="20dp"
            android:layout_below="@id/buttonReboot"
```

```
42                  android:layout_centerHorizontal="true"
43                  android:text="@string/buttonStart"
44                  android:onClick="startServer"
45                  />
46  </RelativeLayout>
```

Listing 17: XML: RESO App - detail.xml



D.9 Java: RESO App, DetailXML.java

As mentioned, the current build of RESO doesn't extract detailed server information from the provided XML file - this functionality would need adding before the solution became a viable product. Instead, hard-coded entries are inserted into `TextView` elements of the layout defined in `detail.xml` (lines 26-28). This file shows examples to *to do* comments to provide guidance as the source is developed.

Methods `rebootServer()` (lines 31-71) and `startServer()` (lines 73-112) are virtually identical so should be refactored into a single method, potentially in a different Java class altogether. Doing so would reduce the likelihood of errors when updating the code.

Sending instructions to the control server is performed by the method `contactControlServer()` which in turn calls `AES,aesmain()` to obtain an encrypted string prior to transmission. This functionality should be split into a separate class as it isn't logical to be present in the detail view.

```
1  package com.jonathanhaddock.reso;
2
3  import android.app.AlertDialog;
4  import android.content.DialogInterface;
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.text.InputType;
8  import android.view.View;
9  import android.widget.EditText;
10 import android.widget.TextView;
11 import android.widget.Toast;
12 import java.io.IOException;
13 import java.io.PrintWriter;
14 import java.net.Socket;
15 import java.net.UnknownHostException;
16 import java.util.Arrays;
17
18 public class DetailXML extends Main {
19     private String toastMessage = "Problem rebooting server";
20
21     @Override
22     public void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.detail);
25         //TODO: Parse the XML into an array, then lookup the relevant
               server by key (serverID)
26         TextView labelIPValue = (TextView) findViewById(R.id.labelIPValue)
               ;
27         setTitle("Detail for Server1"); /** Sets the activity title */
28         labelIPValue.setText("192.168.0.1"); /** Sets the IP TextView */
29     }
```

```
30
31      public void rebootServer(View view) {
32          /** To reboot a server from the "Reboot" button:*/
33          AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(
                this);
34          final EditText userPassword = new EditText(this);
35          userPassword.setInputType(InputType.TYPE_CLASS_TEXT | InputType.
                TYPE_TEXT_VARIATION_PASSWORD);
36          userPassword.setHint("Password");
37          userPassword.setLines(1);
38          alertDialogBuilder.setView(userPassword);
39
40          /** set dialog title */
41          alertDialogBuilder.setTitle("Reboot?");
42
43          /** set dialog message */
44          alertDialogBuilder
45                  .setMessage("Are you sure you want to reboot this server?"
                        )
46                  .setCancelable(false)
47                  .setPositiveButton("Yes", new DialogInterface.
                        OnClickListener() {
48                      public void onClick(DialogInterface dialog, int id) {
49                          String[] rebootServer = contactControlServer("
                                rebootHH", userPassword.getText().toString());
50                          if (rebootServer[0] != null && rebootServer[1] !=
                                null) {
51                              toastMessage = rebootServer[1];
52                          } else {
53                              toastMessage = "Problem rebooting server, 
                                    possible connection problem";
54                          }
55                          Toast rebootingServer = Toast.makeText(
                                getApplicationContext(), toastMessage, Toast.
                                LENGTH_LONG);
56                          rebootingServer.show();
57
58                          Intent intent = new Intent(DetailXML.this,
                                ServerList.class);
59                          startActivity(intent);
60                      }
61                  })
62                  .setNegativeButton("No", new DialogInterface.
                        OnClickListener() {
63                      public void onClick(DialogInterface dialog, int id) {
64                          // if this button is clicked, just close the
                                dialog box and do nothing
65                          dialog.cancel();
66                      }
67                  });
68
69          // create alert dialog and show it
```

```
70          alertDialogBuilder.create().show();
71      }
72
73      public void startServer(View view) {
74          /** To start a server from the "Start" button:*/
75          AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(
                this);
76          final EditText userPassword = new EditText(this);
77          userPassword.setInputType(InputType.TYPE_CLASS_TEXT | InputType.
                TYPE_TEXT_VARIATION_PASSWORD);
78          userPassword.setHint("Password");
79          userPassword.setLines(1);
80          alertDialogBuilder.setView(userPassword);
81          // set dialog title
82          alertDialogBuilder.setTitle("Start this server?");
83
84          // set dialog message
85          alertDialogBuilder
86                  .setMessage("Are you sure you want to switch on this
                        server?")
87                  .setCancelable(false)
88                  .setPositiveButton("Yes", new DialogInterface.
                        OnClickListener() {
89                      public void onClick(DialogInterface dialog, int id) {
90                          String[] responseContactControlServer =
                                contactControlServer("startHH", userPassword.
                                getText().toString());
91                          if (responseContactControlServer[0] != null &&
                                responseContactControlServer[1] != null) {
92                              toastMessage = responseContactControlServer
                                    [1];
93                          } else {
94                              toastMessage = "Problem starting server,
                                    possible connection problem";
95                          }
96                          Toast startingServer = Toast.makeText(
                                getApplicationContext(), toastMessage, Toast.
                                LENGTH_LONG);
97                          startingServer.show();
98
99                          Intent intent = new Intent(DetailXML.this,
                                ServerList.class);
100                         startActivity(intent);
101                     }
102                 })
103                 .setNegativeButton("No", new DialogInterface.
                        OnClickListener() {
104                     public void onClick(DialogInterface dialog, int id) {
105                         /** if this button is clicked, just close the
                                dialog box and do nothing */
106                         dialog.cancel();
107                     }
```

```
108                 });
109
110         // create alert dialog and show it
111         alertDialogBuilder.create().show();
112     }
113
114     public String[] contactControlServer(String instruction, String
        userPassword) {
115         //TODO: Set status to be something that's relevant, maybe make an
            array (status,status detail)
116         String[] response = new String[2];
117         if (!mainSendData()) {
118             response[0] = "fail";
119             response[1] = "Control server comms disabled";
120             return response;
121         }
122
123         //Connect to the control server:
124         try {
125             Socket s = new Socket(mainControlServerAddress(),
                mainControlServerPort());
126             if (s.isConnected()) {
127                 //Encrypt the instruction:
128                 String message = instruction + ";" + (System.
                    currentTimeMillis() / 1000) + ";" + deviceIdentifier()
                    + ";" + userPassword;
129                 byte[] encString = AES.aesmain(message);
130                 PrintWriter out = new PrintWriter(s.getOutputStream(),
                    true);
131                 out.println(Arrays.toString(encString)); //was message
132                 out.close();
133                 //TODO: Make it so the status is only SUCCESS if the
                    server has received the instruction:
134                 s.close();
135                 response[0] = "ok";
136                 response[1] = "Action successful";
137             }
138         } catch (UnknownHostException e) {
139             // TODO: If the host isn't known, find some way to say so
140             e.printStackTrace();
141         } catch (IOException e) {
142             // TODO: If this occurs state that an "unknown error has
                occurred"
143             e.printStackTrace();
144         }
145         return response;
146     }
147 }
```

Listing 18: Java: RESO App - DetailXML.java

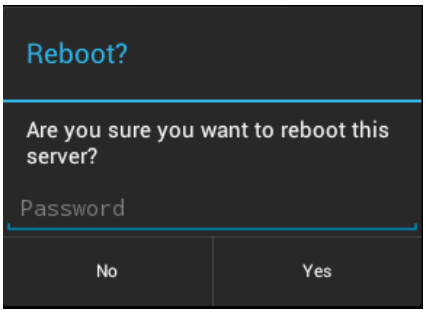


Figure 31: RESO: `DetailXML.java` offers a confirmation prompt prior to rebooting

D.10 Java: RESO App, GcmBroadcastReceiver.java

In order for GCM to function it's necessary for a service to be running which will receive messages and pass to a separate handler. This class has no graphical output, merely listening in the background for appropriate GCM messages.

This code includes contributions from an online tutorial (HMKCode, 2014).

```
1  package com.jonathanhaddock.reso;
2
3  import android.app.Activity;
4  import android.content.ComponentName;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.support.v4.content.WakefulBroadcastReceiver;
8
9  public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {
10     @Override
11     public void onReceive(Context context, Intent intent) {
12         // Explicitly specify that GcmMessageHandler will handle the
               intent.
13         ComponentName comp = new ComponentName(context.getPackageName(),
14                 GcmMessageHandler.class.getName());
15
16         // Start the service, keeping the device awake while it is
               launching.
17         startWakefulService(context, (intent.setComponent(comp)));
18         setResultCode(Activity.RESULT_OK);
19     }
20 }
```

Listing 19: Java: RESO App - GcmBroadcastReceiver.java

D.11 Java: RESO App, GcmMessageHandler.java

The message handler is passed the message from the broadcast receiver and acts on it accordingly. In this example the device vibrates (lines 42-43), an alarm sound is played (lines 45-46), a toast is displayed (seen by the user if they're looking at their device when the alert is received)(lines 54-60) and a notification is placed on the notification bar (lines 62-73). Other than the toast and the notification there is no visible output.

This code includes contributions from an online tutorial (HMKCode, 2014).

```
1  package com.jonathanhaddock.reso;
2
3  import android.app.IntentService;
4  import android.app.Notification;
5  import android.app.NotificationManager;
6  import android.app.PendingIntent;
7  import android.content.Context;
8  import android.content.Intent;
9  import android.media.MediaPlayer;
10 import android.os.Bundle;
11 import android.os.Handler;
12 import android.os.Vibrator;
13 import android.widget.Toast;
14 import com.google.android.gms.gcm.GoogleCloudMessaging;
15
16 public class GcmMessageHandler extends IntentService {
17
18     String mes, messageText;
19     private Handler handler;
20
21     public GcmMessageHandler() {
22         super("GcmMessageHandler");
23     }
24
25     @Override
26     public void onCreate() {
27         super.onCreate();
28         handler = new Handler();
29     }
30
31     @Override
32     protected void onHandleIntent(Intent intent) {
33         Bundle extras = intent.getExtras();
34
35         GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);
36         // The getMessageType() intent parameter must be the intent you
               received in your BroadcastReceiver.
37         String messageType = gcm.getMessageType(intent);
38
39         mes = extras.getString("title");
```

```
40          messageText = extras.getString("message");
41
42          Vibrator v = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE)
                ;
43          v.vibrate(1000);
44
45          MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.
                serverdownalert);
46          mediaPlayer.start(); // no need to call prepare(); create() does
                that for you
47
48          showToast();
49          placeNotification();
50          cleanMediaPlayer(mediaPlayer);
51          GcmBroadcastReceiver.completeWakefulIntent(intent);
52      }
53
54      public void showToast() {
55          handler.post(new Runnable() {
56              public void run() {
57                  Toast.makeText(getApplicationContext(), mes + "\n" +
                        messageText, Toast.LENGTH_LONG).show();
58              }
59          });
60      }
61
62      public void placeNotification() {
63          String subject = "Servers down";
64          String title = "!! ALERT - SERVERS DOWN";
65          String body = "Please check RESO, one or more servers are down";
66          Intent notifyIntent = new Intent(GcmMessageHandler.this,
                ServerList.class);
67          NotificationManager NM;
68          NM = (NotificationManager) getSystemService(Context.
                NOTIFICATION_SERVICE);
69          Notification notify = new Notification(android.R.drawable.
                stat_notify_more, title, System.currentTimeMillis());
70          PendingIntent pending = PendingIntent.getActivity(
                getApplicationContext(), 0, notifyIntent, 0);
71          notify.setLatestEventInfo(getApplicationContext(), subject, body,
                pending);
72          NM.notify(0, notify);
73      }
74
75      public void cleanMediaPlayer(MediaPlayer mediaPlayer) {
76          mediaPlayer.reset();
77          mediaPlayer.release();
78          mediaPlayer = null;
79      }
80  }
```

Listing 20: Java: RESO App - GcmMessageHandler.java

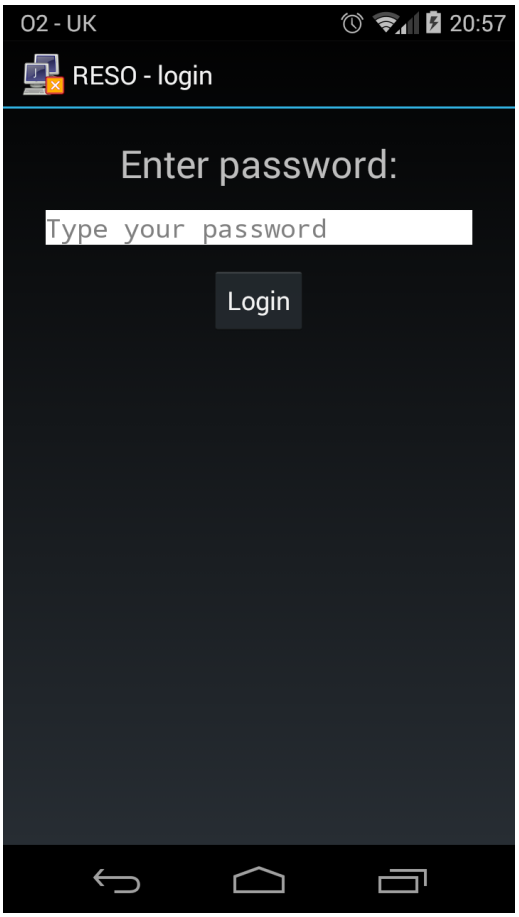
D.12 XML: RESO App, login.xml

The login form for the app only accepts a password and is used to prevent access to additional app functions rather than to authenticate the user to the system. A "hint" is paced in the text box to act as an additional prompt to the user (although there's also a clear heading above the only text box on the page). An empty `TextView` exists beneath the button which is populated with help / error text by `Login.java`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent">
    <TextView
            android:id="@+id/labelEnterPassword"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="20dp"
            android:text="@string/labelEnterPassword"
            android:textSize="12pt"
              />
    <EditText
            android:id="@+id/inputPassword"
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_below="@id/labelEnterPassword"
            android:layout_marginTop="15dp"
            android:cursorVisible="true"
            android:background="#FFF"
            android:textSize="8pt"
            android:textColor="#000"
            android:password="true"
            android:maxLength="25"
            android:inputType="textPassword"
            android:hint="@string/hintPassword">
           <requestFocus />
    </EditText>
    <Button
            android:id="@+id/buttonLogin"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_below="@id/inputPassword"
            android:layout_marginTop="15dp"
            android:text="@string/buttonLogin"
            android:onClick="login"
            />
    <TextView
            android:id="@+id/labelPasswordError"
            android:layout_width="wrap_content"
```

```
44                  android:layout_height="wrap_content"
45                  android:layout_centerHorizontal="true"
46                  android:layout_below="@id/buttonLogin"
47                  android:layout_marginTop="15dp"
48                  android:text="@string/blank"
49                  android:textSize="12pt"
50                  />
51  </RelativeLayout>
```

Listing 21: XML: RESO App - login.xml



D.13 Java: RESO App, Login.java

`Login.java` handles "authentication" from the login form. Firstly, the preference to "remember login" is checked and if true the server list is shown immediately. Otherwise the login is processed up to a maximum of 4 times before the app exits.

Clearly this mechanism is far from perfect. Firstly, the password is hard-coded, a separate preference could be included to allow users to set their own passwords in a commercial app. Secondly, the result of the logon is not used anywhere else so if the user could instead call the ServerList activity (rather than logon)

```
1  package com.jonathanhaddock.reso;
2
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.content.SharedPreferences;
6  import android.os.Bundle;
7  import android.view.View;
8  import android.widget.EditText;
9  import android.widget.TextView;
10
11 public class Login extends Activity {
12     @Override
13     public void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.login);
16         /** If RememberLogin is set, skip to the overview
17          * Note - this isn't secure but does offer a user preference
18          */
19         Intent intent;
20         SharedPreferences sharedPrefRememberLogin = getSharedPreferences("
               rememberLogin", 0);
21         boolean rememberLogin = sharedPrefRememberLogin.getBoolean("
               rememberLogin", false);
22         if (rememberLogin) {
23             intent = new Intent(Login.this, ServerList.class);
24             startActivity(intent);
25         }
26     }
27
28     @Override
29     public void onBackPressed() {
30         /** Ensure that pressing back does nothing by leaving this method
               blank */
31     }
32
33     private int loginCounter = 0;
34
35     public void login(View view) {
```

```
36          Intent intent = new Intent(Login.this, ServerList.class);
37          EditText inputPassword = (EditText) findViewById(R.id.
                inputPassword);
38          TextView errorText = (TextView) findViewById(R.id.
                labelPasswordError);
39
40          if (inputPassword.getText().toString().length() > 0) {
41              if (loginCounter >= 3) {
42                  System.exit(0);
43              } else if (inputPassword.getText().toString().equals("password
                    ")) {
44                  /** If the password is as hard-coded above */
45                  startActivity(intent);
46              } else {
47                  /** Increment the login counter, update the help/error
                        text, blank the password box */
48                  loginCounter++;
49                  String labelLoginError = this.getResources().getString(R.
                        string.labelLoginError);
50                  errorText.setText(labelLoginError + " " + (4-loginCounter)
                        + " - attempts remaining");
51                  inputPassword.setText("");
52              }
53          } else {
54              /** If no password was entered don't increment the counter,
                    just start again */
55              String labelBlankPass = this.getResources().getString(R.string
                    .labelBlankPass);
56              errorText.setText(labelBlankPass);
57              inputPassword.requestFocus();
58          }
59      }
60  }
```

Listing 22: Java: RESO App - Login.java

D.14 Java: RESO App, Main.java

The `Main` class isn't an Android activity but is extended by the classes which are - it has no graphical output of its own. Main serves the following purposes:

- Creates app-wide variables for user preferences, reducing duplicate code within the project (lines 13-44)
- Defines the device identifying characteristics such as serial number and IMEI (lines 46-68)
- Creates a menu which is placed in the top right of each activity (lines 70-75))
- Specifies routing for each menu item when it's selected (lines 77-97)

```
1  package com.jonathanhaddock.reso;
2
3  import android.app.Activity;
4  import android.content.Context;
5  import android.content.Intent;
6  import android.content.SharedPreferences;
7  import android.os.Build;
8  import android.telephony.TelephonyManager;
9  import android.view.Menu;
10 import android.view.MenuItem;
11
12 public class Main extends Activity {
13     /**
14      * Get some variables which will be needed most places:
15      * Server Address
16      */
17     public String mainServerAddress() {
18         SharedPreferences sharedPref = getSharedPreferences("serverAddress
               ", 0);
19         return sharedPref.getString("serverAddress", "");
20     }
21
22     /**
23      * Control Server Address
24      */
25     public String mainControlServerAddress() {
26         SharedPreferences sharedPrefControlServerAddress =
               getSharedPreferences("controlServerAddress", 0);
27         return sharedPrefControlServerAddress.getString("
               controlServerAddress", "");
28     }
29
30     /**
```

```
31          * Control Server Port
32          */
33         public int mainControlServerPort() {
34             SharedPreferences sharedPrefControlServerPort =
                   getSharedPreferences("controlServerPort", 0);
35             return sharedPrefControlServerPort.getInt("controlServerPort", 0);
36         }
37
38         /**
39          * Allow control server comms?
40          */
41         public boolean mainSendData() {
42             SharedPreferences sharedPrefSendData = getSharedPreferences("
                   sendData", 0);
43             return sharedPrefSendData.getBoolean("sendData", false);
44         }
45
46         /**
47          * Get "unique" device information:
48          */
49         public String deviceIdentifier() {
50             String deviceId = "FAKE_ID";
51             TelephonyManager telephonyManager = (TelephonyManager)
                   getSystemService(Context.TELEPHONY_SERVICE);
52             if (!telephonyManager.getDeviceId().isEmpty()) {
53                 /** If there is an IMEI, use it */
54                 deviceId = telephonyManager.getDeviceId();
55             }
56
57             if (!Build.SERIAL.isEmpty()) {
58                 /** If there is a device serial number return that */
59                 deviceId = deviceId + Build.SERIAL;
60             }
61
62             SharedPreferences sharedPrefUserDeviceId = getSharedPreferences("
                   deviceIdFromUser", 0);
63             if (!sharedPrefUserDeviceId.getString("deviceIdFromUser", "").
                   isEmpty()) {
64                 /** Get the user's specified device ID: */
65                 deviceId = deviceId + sharedPrefUserDeviceId.getString("
                       deviceIdFromUser", "");
66             }
67             return deviceId;
68         }
69
70         @Override
71         public boolean onCreateOptionsMenu(Menu menu) {
72             /** Place a menu on each activity */
73             getMenuInflater().inflate(R.menu.main_menu, menu);
74             return true;
75         }
76
```

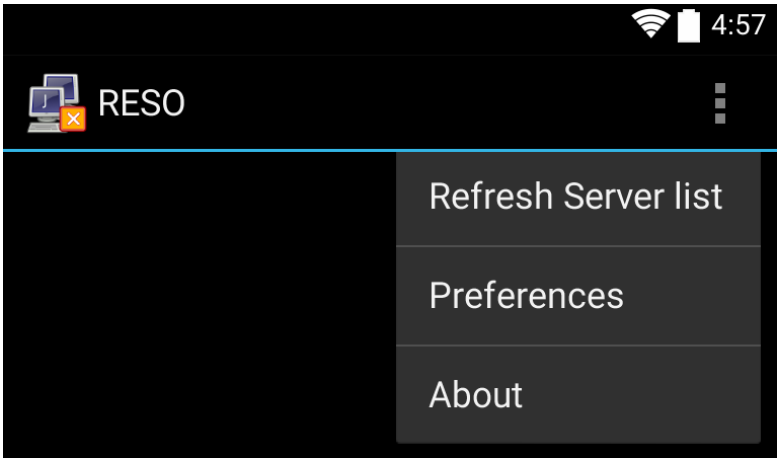
```
77      @Override
78      public boolean onOptionsItemSelected(MenuItem item) {
79          /** How to route each menu choice */
80          Intent intent;
81          switch (item.getItemId()) {
82              case R.id.menuServerList:
83                  intent = new Intent(Main.this, ServerList.class);
84                  break;
85              case R.id.menuPreferences:
86                  intent = new Intent(Main.this, Preferences.class);
87                  break;
88              case R.id.menuAbout:
89                  intent = new Intent(Main.this, About.class);
90                  break;
91              default:
92                  return super.onOptionsItemSelected(item);
93          }
94          startActivity(intent);
95          return true;
96      }
97  }
```

Listing 23: Java: RESO App - Main.java

D.15 XML: RESO App, main_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menuServerList"
          android:title="@string/menuRefreshServerList"
          android:orderInCategory="450"
          android:showAsAction="never"
            />
    <item android:id="@+id/menuPreferences"
          android:title="@string/menuPreferences"
          android:orderInCategory="460"
          android:showAsAction="never"
            />
    <item android:id="@+id/menuAbout"
          android:title="@string/menuAbout"
          android:orderInCategory="500"
          android:showAsAction="never"
            />
</menu>
```

Listing 24: XML: RESO App - main_menu.xml



D.16 XML: RESO App, preferences.xml

Arguably one of RESO's more complex layouts, `preferences.xml` has multiple `EditText` and `CheckBox` elements. Hints, as shown on line 39, give direction to the user advising what entry should be placed in each box and saves the need for further `TextView` elements. This file also shows in-line XML comments to give clarity.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                  android:layout_width="match_parent"
4                  android:layout_height="match_parent"
5                  android:orientation="vertical"
6                  android:overScrollMode="ifContentScrolls"
7                  android:scrollbarStyle="insideOverlay"
8                  android:scrollbars="vertical"
9          >
10     <!--Health server-->
11     <ScrollView
12             android:layout_width="fill_parent"
13             android:layout_height="wrap_content">
14         <RelativeLayout
15                 android:layout_width="match_parent"
16                 android:layout_height="wrap_content">
17             <TextView
18                     android:id="@+id/labelServerAddress"
19                     android:layout_width="wrap_content"
20                     android:layout_height="wrap_content"
21                     android:layout_centerHorizontal="true"
22                     android:layout_marginTop="20dp"
23                     android:text="@string/labelServerAddress"
24                     android:textSize="8pt"
25                     />
26             <EditText
27                     android:id="@+id/inputServerAddress"
28                     android:layout_width="300dp"
29                     android:layout_height="wrap_content"
30                     android:layout_centerHorizontal="true"
31                     android:layout_below="@id/labelServerAddress"
32                     android:layout_marginTop="15dp"
33                     android:cursorVisible="true"
34                     android:background="#FFF"
35                     android:textSize="8pt"
36                     android:textColor="#000"
37                     android:maxLength="250"
38                     android:inputType="text"
39                     android:hint="@string/hintServerAddress">
40             </EditText>
41             <TextView
42                     android:id="@+id/labelPrefsServerAddress"
43                     android:layout_width="wrap_content"
44                     android:layout_height="wrap_content"
```

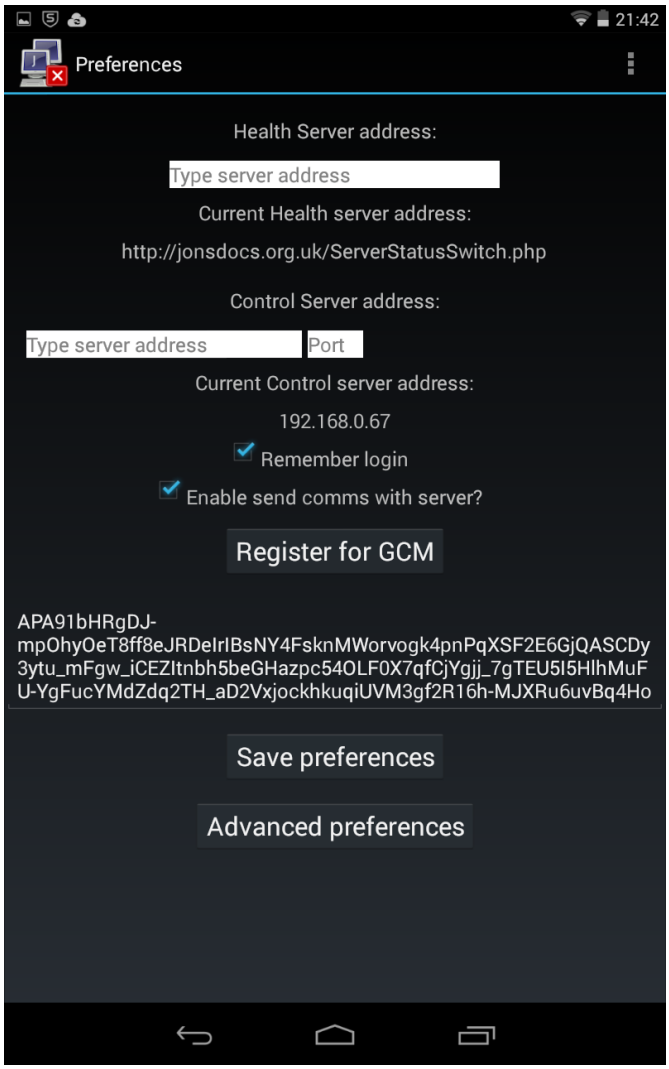
```
45                  android:layout_centerHorizontal="true"
46                  android:layout_below="@id/inputServerAddress"
47                  android:layout_marginTop="10dp"
48                  android:text="@string/labelPrefsServerAddress"
49                  android:textSize="8pt"
50                  />
51          <TextView
52                  android:id="@+id/labelPrefsServerAddressValue"
53                  android:layout_width="wrap_content"
54                  android:layout_height="wrap_content"
55                  android:layout_centerHorizontal="true"
56                  android:layout_below="@id/labelPrefsServerAddress"
57                  android:layout_marginTop="10dp"
58                  android:text="@string/labelBlank"
59                  android:textSize="8pt"
60                  />
61          <!--Control server :-->
62          <TextView
63                  android:id="@+id/labelControlServerAddress"
64                  android:layout_width="wrap_content"
65                  android:layout_height="wrap_content"
66                  android:layout_centerHorizontal="true"
67                  android:layout_below="@id/labelPrefsServerAddressValue
                        "
68                  android:layout_marginTop="20dp"
69                  android:text="@string/labelControlServerAddress"
70                  android:textSize="8pt"
71                  />
72          <EditText
73                  android:id="@+id/inputControlServerAddress"
74                  android:layout_width="250dp"
75                  android:layout_height="wrap_content"
76                  android:layout_below="@id/labelControlServerAddress"
77                  android:layout_marginTop="15dp"
78                  android:layout_marginLeft="20dp"
79                  android:cursorVisible="true"
80                  android:background="#FFF"
81                  android:textSize="8pt"
82                  android:textColor="#000"
83                  android:maxLength="250"
84                  android:inputType="text"
85                  android:hint="@string/hintServerAddress">
86          </EditText>
87          <EditText
88                  android:id="@+id/inputControlServerPort"
89                  android:layout_width="50dp"
90                  android:layout_height="wrap_content"
91                  android:layout_centerHorizontal="true"
92                  android:layout_below="@id/labelControlServerAddress"
93                  android:layout_toRightOf="@id/
                        inputControlServerAddress"
94                  android:layout_marginTop="15dp"
```

```
95                  android:layout_marginLeft="5dp"
96                  android:cursorVisible="true"
97                  android:background="#FFF"
98                  android:textSize="8pt"
99                  android:textColor="#000"
100                 android:maxLength="5"
101                 android:inputType="number"
102                 android:hint="@string/hintServerPort">
103         </EditText>
104         <TextView
105                 android:id="@+id/labelControlPrefsServerAddress"
106                 android:layout_width="wrap_content"
107                 android:layout_height="wrap_content"
108                 android:layout_centerHorizontal="true"
109                 android:layout_below="@id/inputControlServerPort"
110                 android:layout_marginTop="10dp"
111                 android:text="@string/labelCurrentControlServerAddress
                        "
112                 android:textSize="8pt"
113                 />
114         <TextView
115                 android:id="@+id/labelPrefsControlServerAddressValue"
116                 android:layout_width="wrap_content"
117                 android:layout_height="wrap_content"
118                 android:layout_centerHorizontal="true"
119                 android:layout_below="@id/
                        labelControlPrefsServerAddress"
120                 android:layout_marginTop="10dp"
121                 android:text="@string/labelBlank"
122                 android:textSize="8pt"
123                 />
124         <!--Remember logins / enable comms-->
125         <TextView
126                 android:id="@+id/labelRememberLogin"
127                 android:layout_width="wrap_content"
128                 android:layout_height="wrap_content"
129                 android:layout_centerHorizontal="true"
130                 android:layout_below="@id/
                        labelPrefsControlServerAddressValue"
131                 android:layout_marginTop="10dp"
132                 android:text="@string/labelRememberLogin"
133                 android:textSize="8pt"
134                 />
135         <CheckBox
136                 android:id="@+id/checkboxRememberLogin"
137                 android:layout_width="wrap_content"
138                 android:layout_height="wrap_content"
139                 android:layout_toLeftOf="@id/labelRememberLogin"
140                 android:layout_below="@id/
                        labelPrefsControlServerAddressValue"
141                 />
142         <TextView
```

```
143                     android:id="@+id/labelSendData"
144                     android:layout_width="wrap_content"
145                     android:layout_height="wrap_content"
146                     android:layout_centerHorizontal="true"
147                     android:layout_below="@id/labelRememberLogin"
148                     android:layout_marginTop="10dp"
149                     android:text="@string/labelSendData"
150                     android:textSize="8pt"
151                     />
152             <CheckBox
153                     android:id="@+id/checkboxSendData"
154                     android:layout_width="wrap_content"
155                     android:layout_height="wrap_content"
156                     android:layout_toLeftOf="@id/labelSendData"
157                     android:layout_below="@id/labelRememberLogin"
158                     />
159             <Button
160                     android:id="@+id/buttonRegisterGoogleCloudMessaging"
161                     android:layout_width="wrap_content"
162                     android:layout_height="wrap_content"
163                     android:layout_centerHorizontal="true"
164                     android:layout_below="@id/checkboxSendData"
165                     android:layout_marginTop="15dp"
166                     android:text="@string/buttonRegisterGCM"
167                     android:onClick="getGCMRegId"
168                     />
169             <EditText
170                     android:id="@+id/inputGoogleCloudMessagingRegId"
171                     android:layout_width="wrap_content"
172                     android:layout_height="wrap_content"
173                     android:layout_centerHorizontal="true"
174                     android:layout_below="@id/
                            buttonRegisterGoogleCloudMessaging"
175                     android:layout_marginTop="20dp"
176                     android:editable="false"
177                     android:textSize="8pt"
178                     />
179             <Button
180                     android:id="@+id/buttonSavePrefs"
181                     android:layout_width="wrap_content"
182                     android:layout_height="wrap_content"
183                     android:layout_centerHorizontal="true"
184                     android:layout_below="@id/
                            inputGoogleCloudMessagingRegId"
185                     android:layout_marginTop="15dp"
186                     android:text="@string/buttonSavePrefs"
187                     android:onClick="savePrefs"
188                     />
189             <Button
190                     android:id="@+id/buttonAdvancedPrefs"
191                     android:layout_width="wrap_content"
192                     android:layout_height="wrap_content"
```

```
193                     android:layout_centerHorizontal="true"
194                     android:layout_below="@id/buttonSavePrefs"
195                     android:layout_marginTop="15dp"
196                     android:text="@string/buttonAdvancedPrefs"
197                     android:onClick="advancedPrefs"
198                     />
199         </RelativeLayout>
200     </ScrollView>
201 </RelativeLayout>
```

Listing 25: XML: RESO App - preferences.xml



D.17 Java: RESO App, Preferences.java

This file is responsible for writing preferences to `SharedPreferences` storage allowing preferences to be preserved across app closures and device reboots. Method `getGCMRegId()` is responsible for registering with GCM to allow the device to receive push notifications.

This code includes contributions from an online tutorial (HMKCode, 2014) to register for a GCM ID. Lines 128-154 register the device before placing the ID in `SharedPreferences` storage.

```
1  package com.jonathanhaddock.reso;
2
3  import android.content.Intent;
4  import android.content.SharedPreferences;
5  import android.os.AsyncTask;
6  import android.os.Bundle;
7  import android.view.View;
8  import android.widget.*;
9  import com.google.android.gms.gcm.GoogleCloudMessaging;
10 import java.io.IOException;
11
12 public class Preferences extends Main {
13     GoogleCloudMessaging gcm;
14     String regid;
15     String PROJECT_NUMBER = "898826977374"; // Used with GCM to identify
           the app/project
16
17     public void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.preferences);
20 //TODO: Get the control and health server details from Main rather than
       determining them again from shared prefs!!
21         //Make a objects for each visible element of the Activity
22         TextView labelPrefsServerAddress = (TextView) findViewById(R.id.
               labelPrefsServerAddressValue);
23         EditText inputServerAddress = (EditText) findViewById(R.id.
               inputServerAddress);
24         CheckBox checkboxRememberLogin = (CheckBox) findViewById(R.id.
               checkboxRememberLogin);
25         CheckBox checkboxSendData = (CheckBox) findViewById(R.id.
               checkboxSendData);
26         EditText inputControlServerAddress = (EditText) findViewById(R.id.
               inputControlServerAddress);
27         TextView labelPrefsControlServerAddressValue = (TextView)
               findViewById(R.id.labelPrefsControlServerAddressValue);
28         EditText inputControlServerPort = (EditText) findViewById(R.id.
               inputControlServerPort);
29         EditText inputGCMRegid = (EditText) findViewById(R.id.
               inputGoogleCloudMessagingRegId);
30
```

```
31          //Get the preference object:
32          SharedPreferences sharedPrefServerAddress = getSharedPreferences("
                serverAddress", 0);
33          SharedPreferences sharedPrefRememberLogin = getSharedPreferences("
                rememberLogin", 0);
34          SharedPreferences sharedPrefSendData = getSharedPreferences("
                sendData", 0);
35          SharedPreferences sharedPrefControlServerAddress =
                getSharedPreferences("controlServerAddress", 0);
36          SharedPreferences sharedPrefControlServerPort =
                getSharedPreferences("controlServerPort", 0);
37          SharedPreferences prefGCMRegId = getSharedPreferences("GCMRegId",
                0);
38
39          //Get the value for each preference
40          String serverAddress = sharedPrefServerAddress.getString("
                serverAddress", "");
41          boolean rememberLogin = sharedPrefRememberLogin.getBoolean("
                rememberLogin", false);
42          boolean sendData = sharedPrefSendData.getBoolean("sendData", false
                );
43          String controlServerAddress = sharedPrefControlServerAddress.
                getString("controlServerAddress", "");
44          int controlServerPort = sharedPrefControlServerPort.getInt("
                controlServerPort", 0);
45          String GCMRegId = prefGCMRegId.getString("GCMRegId", "");
46
47          //Place values in relevant on screen boxes:
48          labelPrefsServerAddress.setText(serverAddress);
49          inputServerAddress.setText(serverAddress);
50          labelPrefsControlServerAddressValue.setText(controlServerAddress);
51          inputControlServerAddress.setText(controlServerAddress);
52          inputControlServerPort.setText(String.valueOf(controlServerPort));
53          inputGCMRegid.setText(GCMRegId);
54
55          if (rememberLogin) {
56              checkboxRememberLogin.setChecked(true);
57          }
58          if (sendData) {
59              checkboxSendData.setChecked(true);
60          }
61      }
62
63      public void savePrefs(View view) {
64          Intent intent = new Intent(Preferences.this, ServerList.class); //
                Where to go once preferences saved
65          EditText inputServerAddress = (EditText) findViewById(R.id.
                inputServerAddress);
66          String serverAddress = inputServerAddress.getText().toString();
67          CheckBox rememberLogin = (CheckBox) findViewById(R.id.
                checkboxRememberLogin);
68          CheckBox sendData = (CheckBox) findViewById(R.id.checkboxSendData)
```

```
        ;
69      EditText inputControlServerAddress = (EditText) findViewById(R.id.
            inputControlServerAddress);
70      String controlServerAddress = inputControlServerAddress.getText().
            toString();
71      EditText inputControlServerPort = (EditText) findViewById(R.id.
            inputControlServerPort);
72      int controlServerPort = Integer.parseInt(inputControlServerPort.
            getText().toString());
73
74      // Create object of SharedPreferences.
75      SharedPreferences prefServerAddress = getSharedPreferences("
            serverAddress", 0);
76      SharedPreferences prefRememberLogin = getSharedPreferences("
            rememberLogin", 0);
77      SharedPreferences prefSendData = getSharedPreferences("sendData",
            0);
78      SharedPreferences sharedPrefControlServerAddress =
            getSharedPreferences("controlServerAddress", 0);
79      SharedPreferences sharedPrefControlIsHealth = getSharedPreferences
            ("controlIsHealth", 0);
80      SharedPreferences sharedPrefControlServerPort =
            getSharedPreferences("controlServerPort", 0);
81      //now get an Editor so we can write to these preferences
82      SharedPreferences.Editor editorServerAddress = prefServerAddress.
            edit();
83      SharedPreferences.Editor editorRememberLogin = prefRememberLogin.
            edit();
84      SharedPreferences.Editor editorSendData = prefSendData.edit();
85      SharedPreferences.Editor editorControlServerAddress =
            sharedPrefControlServerAddress.edit();
86      SharedPreferences.Editor editorControlServerPort =
            sharedPrefControlServerPort.edit();
87      SharedPreferences.Editor editorControlIsHealth =
            sharedPrefControlIsHealth.edit();
88      //put your value
89      if (!serverAddress.isEmpty()) {
90          editorServerAddress.putString("serverAddress", serverAddress);
91      }
92
93      // Find the value of the checkbox:
94      if (rememberLogin.isChecked()) {
95          editorRememberLogin.putBoolean("rememberLogin", true);
96      } else {
97          editorRememberLogin.putBoolean("rememberLogin", false);
98      }
99      if (sendData.isChecked()) {
100         editorSendData.putBoolean("sendData", true);
101     } else {
102         editorSendData.putBoolean("sendData", false);
103     }
104
```

```
105         if (!controlServerAddress.isEmpty()) {
106             editorControlServerAddress.putString("controlServerAddress",
                    controlServerAddress);
107             editorControlServerPort.putInt("controlServerPort",
                    controlServerPort);
108         }
109
110         //commits your edits
111         editorServerAddress.commit();
112         editorRememberLogin.commit();
113         editorSendData.commit();
114         editorControlServerAddress.commit();
115         editorControlServerPort.commit();
116         editorControlIsHealth.commit();
117
118         //Toast to say the preference has been saved:
119         Toast.makeText(getApplicationContext(), "Preferences saved", Toast
                .LENGTH_LONG).show();
120         startActivity(intent);
121     }
122
123     public void advancedPrefs(View view) {
124         // Allows the app to move to the Advanced Preferences screen
125         startActivity(new Intent(Preferences.this, AdvancedPrefs.class));
126     }
127
128     public void getGCMRegId(View view) {
129         //Get a GCM registration ID and write it to preferences
130         new AsyncTask<Void, Void, String>() {
131             @Override
132             protected String doInBackground(Void... params) {
133                 try {
134                     if (gcm == null) {
135                         gcm = GoogleCloudMessaging.getInstance(
                                getApplicationContext());
136                     }
137                     regid = gcm.register(PROJECT_NUMBER);
138                 } catch (IOException ex) {
139                     //Do nothing
140                 }
141                 return regid;
142             }
143
144             @Override
145             protected void onPostExecute(String regId) {
146                 SharedPreferences prefGCMRegId = getSharedPreferences("
                        GCMRegId", 0);
147                 SharedPreferences.Editor editorGCMRegId = prefGCMRegId.
                        edit();
148                 editorGCMRegId.putString("GCMRegId", regId);
149                 editorGCMRegId.commit();
150                 EditText inputGCMRegid = (EditText) findViewById(R.id.
```

```
                    inputGoogleCloudMessagingRegId);
151                 inputGCMRegid.setText(regId);
152             }
153         }.execute(null, null, null);
154     }
155 }
```

Listing 26: Java: RESO App - Preferences.java

D.18 Java: RESO App, ServerList.java

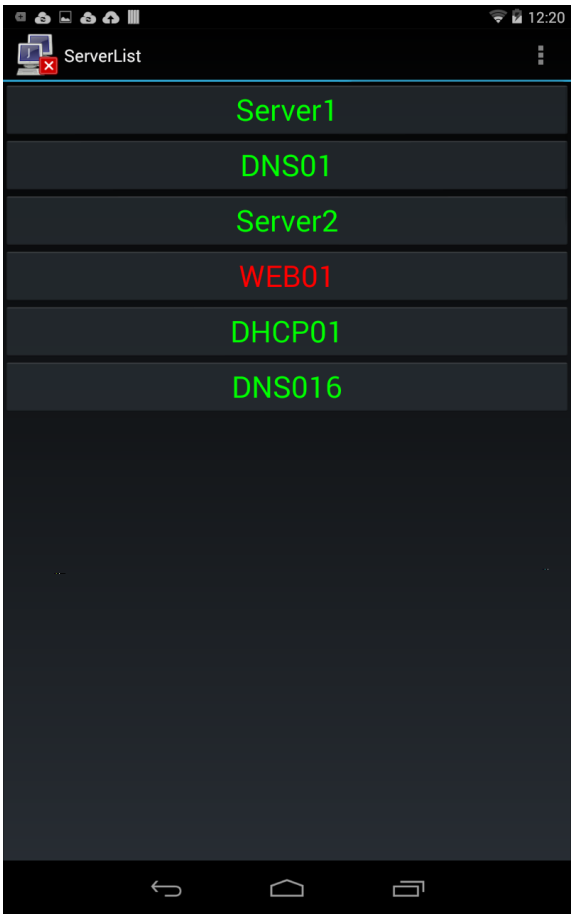
`ServerList.java` downloads health data in the form of XML (see Section C) and parses it to dynamically generate a view. In the event a server is found to be down a sound is played. A `try/catch` mechanism is used in order to prevent the app from crashing in the event the XML file is unobtainable.

```
1  package com.jonathanhaddock.reso;
2
3  import android.content.Intent;
4  import android.graphics.Color;
5  import android.media.MediaPlayer;
6  import android.os.Bundle;
7  import android.os.StrictMode;
8  import android.view.View;
9  import android.widget.Button;
10 import android.widget.LinearLayout;
11 import android.widget.TextView;
12 import android.widget.Toast;
13 import org.w3c.dom.*;
14 import org.xml.sax.InputSource;
15 import javax.xml.parsers.DocumentBuilder;
16 import javax.xml.parsers.DocumentBuilderFactory;
17 import java.net.URL;
18
19 public class ServerList extends Main {
20     boolean serversDown;
21
22     public void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.
               Builder().permitAll().build();
25         StrictMode.setThreadPolicy(policy); //Bypasses requirement to
               background the network operation - testing only
26
27         if (mainServerAddress().equals("")) {
28             // If there is no health server address set, display a toast
                   to say so and go to the preferences activity
29             Toast.makeText(getApplicationContext(), "Please set server
                   address", Toast.LENGTH_SHORT).show();
30             Intent intent = new Intent(ServerList.this, Preferences.class)
                   ;
31             startActivity(intent);
32         }
33
34         /** Create a new layout to display the view */
35         LinearLayout layout = new LinearLayout(this);
36         layout.setOrientation(LinearLayout.VERTICAL);
37         /** Create a new array to display the results */
38         Button name[];
39
```

```
40          try {
41              // If we've been able to get the XML file, parse it.
42              URL url = new URL(mainServerAddress());
43              DocumentBuilderFactory dbf = DocumentBuilderFactory.
                    newInstance();
44              DocumentBuilder db = dbf.newDocumentBuilder();
45
46              Document doc = db.parse(new InputSource(url.openStream()));
47              doc.getDocumentElement().normalize();
48
49              NodeList nodeList = doc.getElementsByTagName("server");
50
51              // Assign textview array length by arraylist size
52              name = new Button[nodeList.getLength()];
53
54              for (int i = 0; i < nodeList.getLength(); i++) {
55
56                  Node node = nodeList.item(i);
57
58                  name[i] = new Button(this);
59
60                  Element fstElmnt = (Element) node;
61                  NodeList nameList = fstElmnt.getElementsByTagName("name");
62                  Element nameElement = (Element) nameList.item(0);
63                  nameList = nameElement.getChildNodes();
64                  name[i].setText((nameList.item(0)).getNodeValue());
65                  name[i].setTextSize(24);
66                  name[i].setId(i);
67                  name[i].setOnClickListener(nameListener);
68
69                  NodeList stateList = fstElmnt.getElementsByTagName("state"
                        );
70                  Element stateElement = (Element) stateList.item(0);
71                  stateList = stateElement.getChildNodes();
72                  String serverState = stateList.item(0).getNodeValue();
73                  if (serverState.equals("down")) {
74                      serversDown = true;
75                      name[i].setTextColor(Color.parseColor("#FF0000"));
76                  } else {
77                      name[i].setTextColor(Color.parseColor("#00FF00"));
78                  }
79
80                  layout.addView(name[i]);
81              }
82          } catch (Exception e) {
83              //If there was a problem obtaining the XML, state so rather
                    than throwing an error
84              TextView errorText = new TextView(this);
85              errorText.setText("Problem␣contacting␣server␣or␣no␣server␣
                    address␣set");
86              errorText.setTextSize(24);
87              errorText.setTextColor(Color.parseColor("#FF0000"));
```

```
88              layout.addView(errorText);
89          }
90
91          /** Display the layout built by above Java code */
92          setContentView(layout);
93          if (serversDown) {
94              //Play a sound: See http://developer.android.com/guide/topics/
                    media/mediaplayer.html and http://www.tutorialspoint.com/
                    android/android_mediaplayer.htm
95              MediaPlayer.create(this, R.raw.serverdownalert).start();
96          }
97      }
98
99      View.OnClickListener nameListener = new View.OnClickListener() {
100         public void onClick(View arg0) {
101             Intent intent = new Intent(ServerList.this, DetailXML.class);
102             startActivity(intent);
103         }
104     };
105 }
```

Listing 27: Java: RESO App - ServerList.java



E Java: RESO Control Server

This appendix contains source code for the RESO Control Server.

E.1 Java: RESO Control Server, Main.java

`Main.java` listens on a port for the purposes of receiving messages from the remote mobile device.

```
1  package com.resoServerSide;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.ServerSocket;
8  import java.net.Socket;
9
10 public class Main {
11     //Code from http://content.gpwiki.org/index.php/Java:Tutorials:
           Simple_TCP_Networking
12     public static void main(String[] args) throws Exception {
13         Main.openSocket();
14     }
15
16     public static void openSocket() throws Exception {
17         ServerSocket serverSocket = null;
18         try {
19             serverSocket = new ServerSocket(1234);
20             System.out.println("Socket has been made");
21         } catch (IOException e) {
22             System.err.println("Could not listen on port: 1234.");
23             System.exit(1);
24         }
25
26         Socket clientSocket = null;
27         try {
28             System.out.println("Waiting for connections...");
29             clientSocket = serverSocket.accept();
30         } catch (IOException e) {
31             System.err.println("Accept failed.");
32             System.exit(1);
33         }
34
35         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
               true);
```

```
36          BufferedReader in = new BufferedReader(new InputStreamReader(
                clientSocket.getInputStream()));
37
38          String message;
39          message = in.readLine();
40          if (message != null && message.length() > 0) {
41              long timeStamp = System.currentTimeMillis() / 1000;
42
43              String plainText = "";
44              try {
45                  plainText = AES.decrypt(unwrap(message), AES.encryptionKey
                        );
46              } catch (IOException e) {
47                  //Do nothing
48                  System.out.println("JAH: Junk message received");
49              }
50
51              //Split the message on ;
52              CharSequence delimeter = ";";
53              if (plainText.contains(delimeter)) {
54                  String[] splitMessage = plainText.split(";");
55
56                  long messageTimestamp = Long.parseLong(splitMessage[1]);
57                  if (timeStamp - messageTimestamp > 5) {
58                      System.out.println("ALERT: Message was stale");
59                      out.close();
60                      in.close();
61                      clientSocket.close();
62                      serverSocket.close();
63                      Main.openSocket();
64                  } else {
65                      //If the message isn't stale, continue processing:
66                      //Check for valid device ID:
67                      if (splitMessage[2].equals("359290051743494
                            TA4750GQTEJonathanPhone")) {
68                          System.out.println(splitMessage[0]);
69
70                          //Check for a valid password:
71                          if (splitMessage[3].equals("password")) {
72                              processCommand(splitMessage[0]);
73                          } else {
74                              System.out.println("JAH: Instruction received
                                    from an authorised device (" + splitMessage
                                    [2] + ") but invalid password specified");
75                          }
76                      } else {
77                          System.out.println("JAH: Instruction received from
                                an unauthorised device identified as: " +
                                splitMessage[2]);
78                      }
79                  }
80              } else {
```

```
81                  System.out.println("Malformed message");
82                  out.close();
83                  in.close();
84                  clientSocket.close();
85                  serverSocket.close();
86                  Main.openSocket();
87              }
88          }
89          out.close();
90          in.close();
91          clientSocket.close();
92          serverSocket.close();
93          Main.openSocket();
94      }
95
96      public static String processCommand(String command) {
97          String status = "Could not process command";
98          ProcessBuilder systemCommand = new ProcessBuilder("");
99
100         if (command.equals("exit")) {
101             System.exit(0);
102         } else if (command.equals("rebootHH")) {
103             systemCommand = new ProcessBuilder("bash", "-c", "VBoxManage -
                    q controlvm Ubuntu14-04LTS_Desktop32 reset");
104         } else if (command.equals("startHH")) {
105             systemCommand = new ProcessBuilder("bash", "-c", "VBoxManage -
                    q startvm Ubuntu14-04LTS_Desktop32");
106         }
107
108         try {
109             systemCommand.start();
110         } catch (IOException e) {
111             //Do nothing
112             System.out.println("There was an exception...");
113         }
114         return status;
115     }
116
117     private static byte[] unwrap(String wrappedMessage) {
118         wrappedMessage = wrappedMessage.substring(1, wrappedMessage.length
                ()-1);
119         String[] arrWrappedMessage = wrappedMessage.split(", ");
120         byte[] unwrappedMessage = new byte[arrWrappedMessage.length];
121         for (int i = 0; i < unwrappedMessage.length; i++) {
122             unwrappedMessage[i] = Byte.parseByte(arrWrappedMessage[i]);
123         }
124         return unwrappedMessage;
125     }
126 }
```

Listing 28: Java: RESO Control Server - Main.java

E.2 Java: RESO Control Server, AES.java

`AES.java` contains the functions required for encrypting and decrypting messages sent from the mobile device. This code uses contributions from Fernandes (2014).

```
1  package com.resoServerSide;
2
3  /* Modified from: https://gist.github.com/bricef/2436364
4   * NOTE: key and plaintext must both be multiples of 16 or padded with \0
        unless you use AES/CBC/PKCS5Padding
5   * NOTE: IV must also be 16 bytes long */
6
7  import javax.crypto.spec.SecretKeySpec;
8  import javax.crypto.spec.IvParameterSpec;
9  import javax.crypto.Cipher;
10
11 public class AES {
12     static String IV = "ABCDEFGHIJKLMNOP"; //Initialisation vector
13     static String encryptionKey = "0123456789abcdef0123456789abcdef";
14
15     public static byte[] encrypt(String plainText, String encryptionKey)
           throws Exception {
16         Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
17         SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF
               -8"), "AES");
18         cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(IV.
               getBytes("UTF-8")));
19         return cipher.doFinal(plainText.getBytes("UTF-8"));
20     }
21
22     public static String decrypt(byte[] cipherText, String encryptionKey)
           throws Exception {
23         Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
24         SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF
               -8"), "AES");
25         cipher.init(Cipher.DECRYPT_MODE, key, new IvParameterSpec(IV.
               getBytes("UTF-8")));
26         return new String(cipher.doFinal(cipherText), "UTF-8");
27     }
28 }
```

Listing 29: Java: RESO Control Server - AES.java

F Glossary

Acronyms

2FA 2 Factor Authentication.

ASP Application Service Provision.

DMZ De-Militarised Zone.

DoS Denial of Service.

FOSS Free Open Source Software.

GCM Google Cloud Messaging.

hypervisor Software that allows a computer to run multiple operating systems, simultaneously, on the same hardware. Examples of hypervisors are VMware ESXi and Microsoft Hyper-V..

IID Iterative and Incremental Development.

IMEI International Mobile Equipment Identity.

Managed Solution Provider Managed Solution Providers offer services to clients who have elected to outsource a service. An example would be a company which monitors a client's IT systems health..

NDK Native Development Kit.

NMS Network Monitoring System.

OS Operating System.

OTP One Time Password.

PSN Public Sector Network.

Salting The process of taking a user provided password and adding an additional secret prior to hashing. As a result, it is significantly more difficult to brute force the hashed password *so long as the salt remains private*.

SDK Software Development Kit.

SNMP Simple Network Management Protocol.

TDD Test Driven Development.

VCS Version Control System.

VM Virtual Machine.