SOFTWARE REQUIREMENTS SPECIFICATION

for

NetDog

Version 1.0

Prepared by Aswin Babu K

College of Engineering Trivandrum

September 2, 2018

Contents

Revision History

Name	Date	Reason For Changes	Version
Initial Release	01-09-18	Initial release of document	1.0

1 Introduction

1.1 Purpose

The purpose of this document is to describe in detail, the requirements for the "NetDog" Project. The document explains various features of the system and its constraints. The document is intented for the end user to determine if the software covers the required features and for the development team for impementing the first version of the system.

1.2 Project Scope

The NetDog project aims to make it easy for administrators to manage local PCs on a network. It allows remote powering up and down of PCs without worrying about the IP address of client machines. The system natively supports encrypted transfer of files over the network without third party protocols. Futher, the system will be completely pluggable, allowing administrators to easily extend the system by using third party extensions or writing their own.

1.3 References

The following are the main off the shelf components that has been used to implement the project. Click on the link to direct the browser to each project's homepage

Python 3 language
Sockets Library
PyCrypto Library
Standard logging
Netifaces library

2 Overall Description

2.1 Product Perspective

The netdog project started with the aim of designing a program which can easily bring computers up and down remotely. The idea came from the realization that, quite a few computers were left powered up when the college lab closes for the day. Thus the initial name ”Project Green”.

NetDog has a client server architecture. The server is responsible for issuing commands to the clients and is to be used by the administrator of the network. The client application is to be run on machines on the network to be administered.

After installing NetDog server or client on a machine, a unique public-private key pair for the machine is generated which is then used for uniquely identifying the machine and securing data transmission between the client and server.

Once the server program is up and running, it listens on the port 1337 for connections from clients. Once the client program is up, it starts listening on port 1994. These ports serve dual purpose of facilitating communication and allowing the identification of server and clients from the rest of the machines on the network. Both NetDog server and client are daemons. They are system services which remain in memory and automatically starts during system boot.

When a client starts for the first time, it looks for active servers on the network. When it finds one, it starts the pairing procedure. During the pairing process, the client sends its hostname and public-key. The server in turn provides the client with it’s public key. These keys are then used for identification and encrypted communication between machines.

The client and server uses public key encryption to identify and secure the communication between them. The network admininstrator can issue commands from the server machine which will then be sent to all the clients on the network. A client can also contact the server occasionally, for example if the client detects undseriable network traffic or if the system is overheating.

NetDog is capable of shutting down all the clients on the network at once by remotely executing the shutdown command. It is also able to power up systems which support remote Wake-On-LAN feature, by sending magic packets.

2.2 Product Functions

The initial release of NetDog will have the following features

Bring all computers on the network up and down remotely
Scheduled power up and down of computers
Execute commands/scripts remotely on machines
Copy files to remote machines without third party protocols
Track and identify clients through IP changes
Secure client-server communication using public key encryption
List all machines on the network which are not clients (detect intruders)
Alert if daemon on a client is not running
Centralized logging of all data regarding clients
Track power-on power-off times and user login history
command line and web interface

2.3 Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

2.4 Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

2.5 User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>

2.6 Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

3 External Interface Requirements

3.1 User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

3.2 Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

3.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

3.4 Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

4 System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

4.1 System Feature 1

<Don't really say "System Feature 1." State the feature name in just a few words.>

4.1.1 Description and Priority

<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

4.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

4.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate when necessary information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1: REQ-2:

4.2 System Feature 2 (and so on)

5 Other Nonfunctional Requirements

5.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

5.5 Business Rules

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

6 Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

6.1 Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

6.2 Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

6.3 Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>